UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

JONATAN AMARAL DA SILVA

APRENDIZADO POR REFORÇO NO AMBIENTE DE JOGOS

CAXIAS DO SUL

2024

JONATAN AMARAL DA SILVA

APRENDIZADO POR REFORÇO NO AMBIENTE DE JOGOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Carine G. Webber

CAXIAS DO SUL

2024

JONATAN AMARAL DA SILVA

APRENDIZADO POR REFORÇO NO AMBIENTE DE JOGOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 04/07/2024

BANCA EXAMINADORA

Carine G. Webber
Universidade de Caxias do Sul - UCS

Elisa Boff
Universidade de Caxias do Sul - UCS

Marcos Casa
Universidade de Caxias do Sul - UCS

Este trabalho é dedicado aos apaixonados por tecnologia.

AGRADECIMENTOS

Agradeço aos meus pais, Elias Ocanha da Silva e Lizete Amaral da Silva, que sempre me incentivaram a crescer e dar o meu melhor em todos os aspectos da vida

Aos meus amigos e colegas, que estiveram presentes nos momentos bons e ruins para me apoiar.

Agradeço à Professora Carine G. Webber, responsável pela orientação deste trabalho e que me apoiou em cada etapa.

“Sem sonhos, a vida não tem brilho. Sem metas, os sonhos não têm alicerces. Sem prioridades, os sonhos não se tornam reais. Sonhe, trace metas, estabeleça prioridades e corra riscos para executar seus sonhos. Melhor é errar por tentar do que errar por se omitir!”

Augusto Cury

RESUMO

A Inteligência artificial (IA) tem se tornado uma presença cada vez mais constante em nosso cotidiano, com aplicações que abrangem desde *chatbot*s e carros autônomos até reconhecimento de imagem e jogos. A IA tem sido objeto de estudo e reflexões desde a década de 1950, quando Turing propôs o Teste de Turing, também conhecido como Jogo da Imitação. Este teste provocou um debate sobre a capacidade das máquinas de pensar e os limites da inteligência computacional. Nesse contexto, os jogos surgem como um domínio intrigante para a IA, oferecendo um ambiente rico para exploração. Com regras definidas e de fácil compreensão, assim dispensando a necessidade de um especialista, permitindo a participação de pessoas comuns em experimentos e treinos. Dentre os diferentes paradigmas de aprendizado em IA, o Aprendizado por Reforço (AR) se destaca. Este tipo de aprendizado tem a capacidade de aprender diretamente através da interação com o ambiente, sem a necessidade de amostragem. Essa habilidade de tomar decisões em ambientes incertos é uma vantagem significativa, especialmente no ambiente complexo dos jogos eletrônicos. Um exemplo notável de algoritmo de AR é o Q-Learning. Este algoritmo se destaca por não exigir um modelo prévio do ambiente, permitindo um aprendizado direto a partir da experiência acumulada. Para implementação de modelo de IA capaz de aprender a jogar o jogo Pong de forma autônoma, foi utilizado o algoritmo Dueling Deep Q-networks (DDQN) na linguagem pyhton, que é uma evolução do Deep Q-networks (DQN), que combina o Q-Learning e redes neurais profundas em conjunto com a biblioteca gymnasium. Como resultado o modelo chega ao ponto de sempre ganhar, conseguindo atingir a pontuação final máxima de 21 pontos e equiparando aos melhores resultados dos trabalhos relacionados. Esses avanços na aplicação de IA e AR em jogos eletrônicos não apenas demonstram o potencial dessas técnicas, mas também abrem caminho para novas possibilidades de pesquisa e desenvolvimento, contribuindo significativamente para o avanço contínuo da IA e AR nesse campo dinâmico.

Palavras-chave: Inteligência Artificial. Artificial Intelligence. Aprendizado de Máquina. Machine learning. Aprendizado por reforço; Reinforcement learning. Jogos. Games. Atari. Pong. Gymnasium. Python.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ALE Arcade Learning Environment

AM Aprendizado de Máquina

API Application Programming Interface

AR Aprendizado por Reforço

DQN Deep Q-networks

DDQN Dueling Deep Q-networks

elem. elemento

GPU Graphics Processing Unit

IA Inteligência artificial

IoT Internet das Coisas

MCTS Monte Carlo Tree Search

MDP Processo de decisão de Markov

RMSProp Root Mean Squared Propagation

Simple Simulated Policy Learning

SUMÁRIO

1 INTRODUÇÃO

A IA pode ser definida como o ramo da ciência em que se ocupa da automação do comportamento inteligente (LUGER, 2013). No entanto a IA não é muito bem definida ou compreendida. Embora a maioria das pessoas reconheça o comportamento inteligente quando o vê, é difícil definir a IA de forma específica o suficiente para avaliar um programa de computador como inteligente, mantendo ao mesmo tempo a complexidade da mente humana. Por isso simplesmente podemos defini-la como uma coleção de problemas e metodologias estudadas pelos pesquisadores de IA (LUGER, 2013).

Em 1956, John McCarthy, juntamente com Marvin Minsky, Nathaniel Rochester e Claude Shannon, organizou a *Dartmouth Summer Research Project on Artificial Intelligence* em *Dartmouth College*, localizado em *Hanover*, *New Hampshire*. Reunindo diversos pesquisadores de renome para explorar o que seria mais tarde denominado por IA (MCCARTHY *et al.*, 1955). A Figura 1 apresenta alguns dos cientistas que participaram da conferência em *Dartmouth College*.

> Propomos que um estudo de inteligência artificial com a duração de 2 meses e 10 pessoas seja realizado durante o verão de 1956 no Dartmouth College em Hanover, New Hampshire. O estudo deverá prosseguir com base na conjectura de que todos os aspectos da aprendizagem ou qualquer outra característica da inteligência podem, em princípio ser descrito com tanta precisão que uma máquina possa ser feita para simulá-lo. Uma tentativa será feita para descobrir como fazer as máquinas usarem linguagem, formar abstrações e conceitos, resolver tipos de problemas agora reservados aos humanos e melhorarem. Achamos que um avanço significativo pode ser feito em um ou mais desses problemas se um grupo cuidadosamente selecionado de cientistas trabalhar nele juntos por um verão (MCCARTHY *et al.*, 1955) .

Durante o *workshop*, o termo IA foi cunhado pela primeira vez e direções como métodos simbólicos foram iniciadas (MCCARTHY *et al.*, 1956). Muitos dos participantes fizeram contribuições-chave para a IA, como Marvin Minsky, que desenvolveu um dos primeiros simuladores de rede neural, e John McCarthy, que inventou a linguagem de programação LISP. Este evento é amplamente considerado o marco fundador do campo da IA (MCCARTHY *et al.*, 1956).

Como Silva *et al.* (2018) explica, com o passar do tempo, diversas vertentes de pesquisa na área da IA surgiram, incluindo a abordagem biológica, que explorava o desenvolvimento de conceitos inspirados nas redes neurais humanas. Durante a década de 1960, o campo adquiriu oficialmente o nome de Inteligência Artificial, e os pesquisadores começaram a considerar a possibilidade de máquinas executarem tarefas complexas, como o raciocínio, de maneira semelhante aos seres humanos. Após um período de estudos, na década de 1980, o estudo das redes neurais ganhou força e, nos anos 1990, experimentou um notável avanço, solidificando-se como a base central das investigações em IA. Foi definido que o sistema de IA não apenas armazena e manipula dados, mas também adquire, representa e manipula conhecimento. De forma que

Figura 1 – À esquerda Marvin Minsky, Claude Shannon, Ray Solomonoff e outros cientistas no Dartmouth Summer Research Project on Artificial Intelligence

Fonte: https://www.cantorsparadise.com/the-birthplace-of-ai-9ab7d4e5fb00

um sistema de IA pode ter a capacidade de deduzir novos conhecimentos a partir do conhecimento existente, bem como a resolução de problemas complexos não quantitativos por meio de métodos de representação e manipulação (SILVA *et al.*, 2018).

Uma das diversas aplicações da IA são os jogos. Os jogos fazem parte de nossas vidas e, à medida que eles se tornam mais avançados, a IA também avança para desafiar os jogadores humanos. A história da IA e a história dos jogos estão ligadas. Um marco importante foi o famoso Teste de Turing proposto por Alan Turing em 1950 (TURING, 1950).

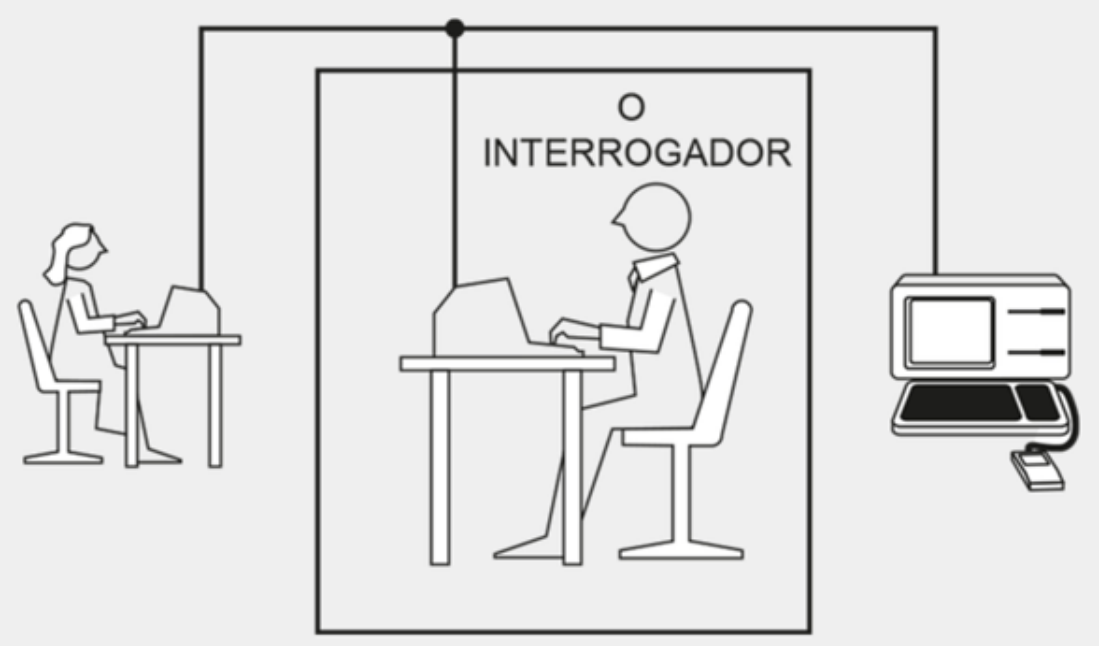
Alan Turing introduziu o conceito do "Jogo de Imitação" ou "Teste de Turing". Com o objetivo de responder à pergunta "As máquinas podem pensar?". Turing descreve o jogo da seguinte forma (TURING, 1950):

É jogado com três pessoas, um homem (A), uma mulher (B) e um interrogador (C) , que pode ser de qualquer sexo. O interrogador fica em uma sala separada dos outros dois. O

objetivo para o interrogador é determinar qual dos dois é o homem e qual é a mulher. Ele os conhece pelos rótulos X e Y, e no final do jogo ele diz que “X é A e Y é B” ou “X é B e Y é A” (TURING, 1950).

Agora, o que aconteceria se uma máquina assumisse o papel de A neste jogo? O interrogador cometeria erros na mesma proporção que quando o jogo é disputado entre um homem e uma mulher? Estas perguntas nos levam a uma nova forma de abordar a questão original: “As máquinas podem pensar?” (TURING, 1950). A figura 2 ilustra a situação proposta pelo Teste de Turing, onde um interrogador fica isolado e se comunica com uma máquina e uma pessoa, sem saber se está falando com uma ou com outra.

Figura 2 – Situação proposta pelo Teste de Turing



Fonte: Retirado de Carla Oliveira

Para passar no teste de Turing, um computador precisaria pelo menos das seguintes habilidades (SANTOS, 2021):

1. Processamento de linguagem natural: habilidade de se comunicar de forma natural com o idioma em questão.
2. Representação do conhecimento: precisa ser capaz de ter conhecimento e armazená-lo em algum lugar.
3. Raciocínio automatizado: precisa ser capaz de fazer raciocínio com base no armazenado do conhecimento.
4. Aprendizado de máquina: precisa ser capaz de aprender com o seu ambiente.

Ao questionar se uma máquina pode enganar um interrogador humano a ponto de ser confundida com um ser humano, Turing nos incentivou a explorar os limites da capacidade das máquinas e a nos esforçarmos para criar sistemas de IA cada vez mais sofisticados (TURING, 1950).

Com a publicação do Teste de Turing, a área de jogos atraiu o interesse de diversos cientistas que exploraram como os jogos poderiam contribuir para o campo de IA. As pesquisas iniciais foram sobre busca em espaço de estados, e frequentemente utilizavam jogos de tabuleiro comuns, como xadrez ou damas. A natureza dos jogos, que são atividades físicas ou intelectuais regidas por um conjunto de regras, facilita a geração de espaços de busca. Isso ajuda a eliminar ambiguidades e reduzir a complexidade de problemas menos estruturados (LUGER, 2013).

Os jogos possuem a incrível habilidade de gerar vastos espaços de busca, podendo se tornar complexos labirintos que exigem o uso de técnicas avançadas para avaliar múltiplas alternativas possíveis. Estas técnicas são conhecidas como heurísticas e desempenham um papel fundamental na pesquisa em IA (LUGER, 2013). O fato de a grande maioria das pessoas ter alguma experiência com jogos mais simples, como jogos de tabuleiro, torna mais fácil projetar e testar a eficácia das heurísticas. Não há a necessidade de um especialista, diferente de outras áreas como a matemática e a medicina. Por esses motivos, os jogos propiciam um domínio rico para o estudo da busca heurística (LUGER, 2013).

Uma das sub-áreas da IA que se destaca no desenvolvimento de jogos é o Aprendizado de Máquina (AM), que consiste em treinar algoritmos para aprender com dados e melhorar seu desempenho em tarefas específicas. No contexto de jogos, o AM pode ser usado para ensinar programas a aprenderem com a experiência, assim como um indivíduo aprende as regras e estratégias de um esporte por meio da observação e experiência de vários jogos (LUGER, 2013).

Uma das subáreas da IA que se destaca no desenvolvimento de jogos é o AM, que consiste em treinar algoritmos para aprender com dados e melhorar seu desempenho em tarefas específicas. No contexto de jogos, o AM pode ser usado para ensinar programas a aprenderem com a experiência, assim como um indivíduo aprende as regras e estratégias de um esporte por meio da observação e experiência de vários jogos (LUGER, 2013).

À medida que o agente continua a interagir com seu ambiente e a receber *feedback* através deste sistema de recompensas e penalidades, ele começa a desenvolver uma estratégia, ou política, que maximiza sua recompensa total ao longo do tempo. Este processo pode ser visto como um ciclo de aprendizado, onde cada sucesso reforça o que foi aprendido e informa as ações futuras do agente. Este método de aprendizado é particularmente útil em situações que envolvem uma série de decisões sequenciais e interações complexas, como é o caso dos jogos. Através da AR, um agente pode aprender a navegar pelas regras de um jogo e a tomar decisões que aumentam suas chances de vitória (LUGER, 2013).

Com o AR, à medida que os movimentos dos agentes se tornam mais precisos, seus

resultados melhoram, criando um ciclo de aprendizado onde o sucesso ou a falha são reforçados com o que foi aprendido. Isso demonstra como o AM pode ser aplicado para melhorar o desempenho de agentes de IA em atividades baseadas em regras e interações complexas, como os jogos (LUGER, 2013).

1.1 OBJETIVO GERAL E ESPECÍFICOS

O objetivo deste estudo é explorar a aplicação e avaliação do aprendizado por reforço no contexto de jogos, visando o desenvolvimento de um modelo de IA.

Os objetivos específicos deste trabalho são:

1. Implementar um modelo de aprendizado por reforço.
2. Examinar como o modelo aprende, através de suas interações com o ambiente do jogo.
3. Avaliar a eficácia do método de aprendizado por reforço no problema escolhido na área de jogos.
4. Comparar a solução e o modelo desenvolvido com outros trabalhos do estado da arte.

1.2 ORGANIZAÇÃO DO DOCUMENTO

Este trabalho de conclusão está organizado em 4 capítulos.

1. O primeiro capítulo é definido pela introdução explicando conceitos iniciais do trabalho.
2. O segundo capítulo apresenta um estudo sobre a área de aprendizado de máquina, em especial do aprendizado por reforço e os trabalhos relacionados.
3. O terceiro capítulo é descrita a proposta de implementação, detalhando os materiais, ferramentas e métodos necessários.
4. Por fim, o quarto capítulo apresenta as considerações finais, apresenta uma síntese do trabalho realizado e o cronograma para o TCC 2.

2 APRENDIZADO POR REFORÇO APLICADO NA ÁREA DE JOGOS

Os jogos representam uma área popular de pesquisa em IA. Embora os jogos de tabuleiro tenham sido fundamentais na pesquisa de IA desde o início, os videogames tornaram-se cada vez mais a principal forma de testar e introduzir novos algoritmos nas últimas décadas. Simultaneamente, os videogames evoluíram, tornando-se mais diversos e complexos. Alguns deles incorporam avanços em IA para controlar personagens não-jogadores, gerar conteúdo ou adaptar-se aos jogadores (YANNAKAKIS *et al.*, 2018).

2.1 O QUE É APRENDIZADO DE MÁQUINA

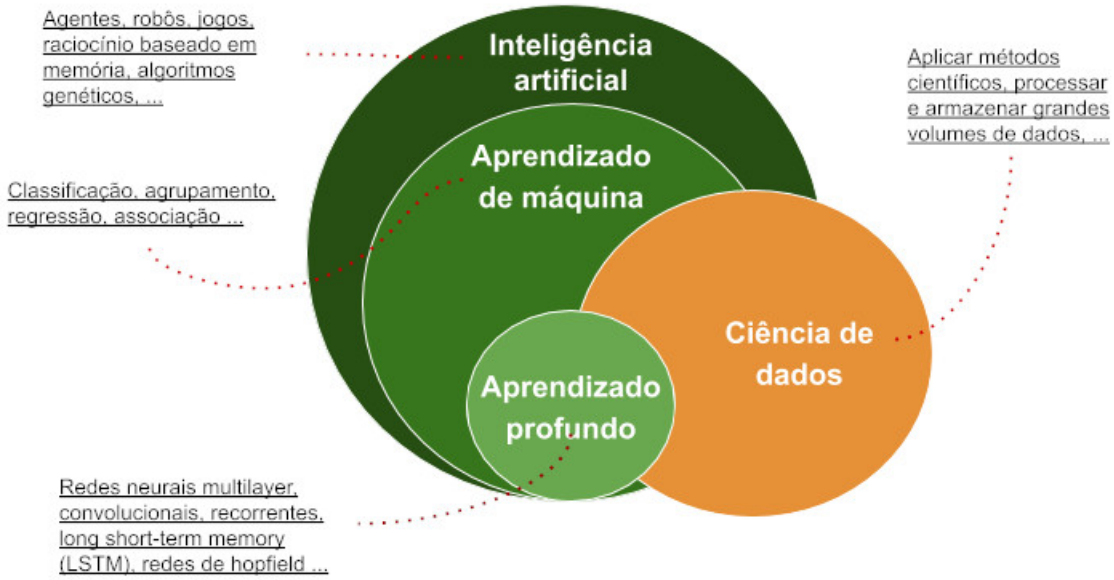
O AM é uma área fundamental da IA, que se concentra no desenvolvimento de algoritmos e modelos estatísticos que permitem que sistemas aprendam e melhorem automaticamente com base em dados. Esta abordagem, junto com a *Ciência de Dados*, campo focado na coleta, limpeza, análise e interpretação de dados, desempenha um papel essencial na revolução da IA, permitindo que sistemas adquiram conhecimento e tomem decisões com base em informações. A relação entre IA, AM, aprendizado profundo e ciência de dados é apresentada na Figura 3, ilustrando quais campos fazem parte da IA, seja como subáreas ou atuando de forma multidisciplinar, como é o caso da ciência da computação.

Com o crescente volume de dados gerados por empresas e indivíduos, surge a possibilidade de utilizá-los como fonte de experiência em processos computacionais (MITCHELL *et al.*, 2007). É nesse contexto de grande volume de dados e necessidade de processamento eficiente que surge o AM. O AM é visto como um campo interdisciplinar da ciência da computação e estatística, focado no desenvolvimento de algoritmos e modelos estatísticos para que os sistemas aprendam e melhorem seu comportamento automaticamente com base em dados, sem a necessidade de serem explicitamente programados (ALPAYDIN; ETHEM, 2020). O AM trabalha em conjunto com o campo da Ciência de Dados e inclui o *aprendizado profundo* como subárea, técnica que permite que sistemas aprendam e melhorem seu comportamento automaticamente com base em dados, sem programação explícita.

O AM também pode ser definido como, o estudo de algoritmos computacionais que melhoram seu desempenho automaticamente, através da experiência adquirida pela interação com dados. Esses algoritmos aprendem a reconhecer padrões complexos e tomar decisões com base em dados, tornando-se cada vez mais precisos à medida que são alimentados com mais informações (MITCHELL *et al.*, 2007).

O AM tem uma ampla gama de aplicações empresariais, trazendo melhorias significativas em várias frentes. Como visto no artigo de Crockett (2023), um dos casos mais notáveis

Figura 3 – Relação entre inteligência artificial, aprendizado de máquina, aprendizado profundo e ciência de dados.



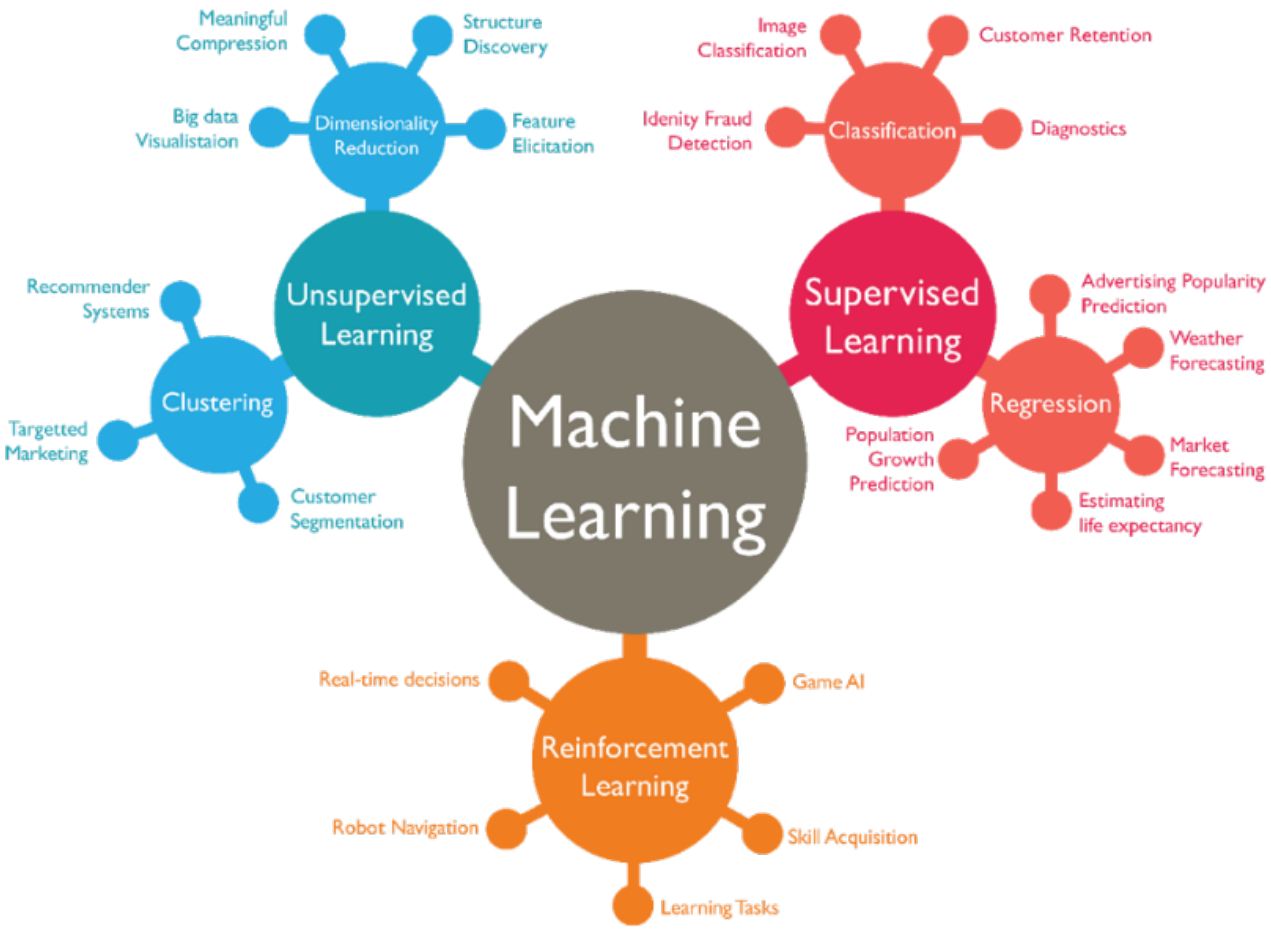
Fonte: Adaptado de https://www.serpro.gov.br/menu/noticias/noticias-2019/democratizando-a-inteligencia-artificial.

na segurança é a detecção de fraudes, bancos e emissores de cartões de crédito utilizam o AM para identificar transações potencialmente fraudulentas, protegendo assim os clientes contra atividades não autorizadas. Outra aplicação é a vigilância por vídeo, onde o AM é utilizado para aprimoramento de sistemas de reconhecimento facial, permitindo que a identificação de criminosos conhecidos, e comportamentos anômalos que possam representar ameaças à segurança pública. A 4 mostra uma série de aplicações possíveis a partir dos ramos e sub-ramos do AM.

O AM também é fundamental para a análise em tempo real de dados em constante atualização, como *feeds* de redes sociais e transações de vendas online. Os *feeds* de redes sociais são fluxos contínuos de atualizações de usuários nas redes sociais, enquanto os *insights* são entendimentos ou descobertas valiosas obtidas a partir da análise de dados. O AM permite a identificação imediata desses *insights* e problemas emergentes. Outro cenário importante onde o AM é utilizado é a manutenção preditiva, impulsionada pela Internet das Coisas (IoT), que usa dados históricos de equipamentos para prever falhas, possibilitando reparos proativos e minimizando impactos nas operações (CROCKETT, 2023).

Os algoritmos de AM podem ser categorizados em tarefas preditivas e descritivas (FACELI; LORENA; GAMA, 2021). Nas tarefas preditivas, os algoritmos são utilizados para criar modelos capazes de fazer previsões, com base em um conjunto de dados de treinamento que contém rótulos conhecidos.Os rótulos são informações desejadas para cada registro em um conjunto de dados. Por exemplo, em um conjunto de dados que contém características de diferentes tipos de feijão, o rótulo identifica a espécie de feijão associada a essas características. Esses modelos,

Figura 4 – Casos de uso da IA.



Fonte: Retirado de <https://www.researchgate.net/publication/358935730_Applications_of_Machine_Learning_in_Alloy_Catalysts_Rational_Selection_and_Future_Development_of_Descriptors>

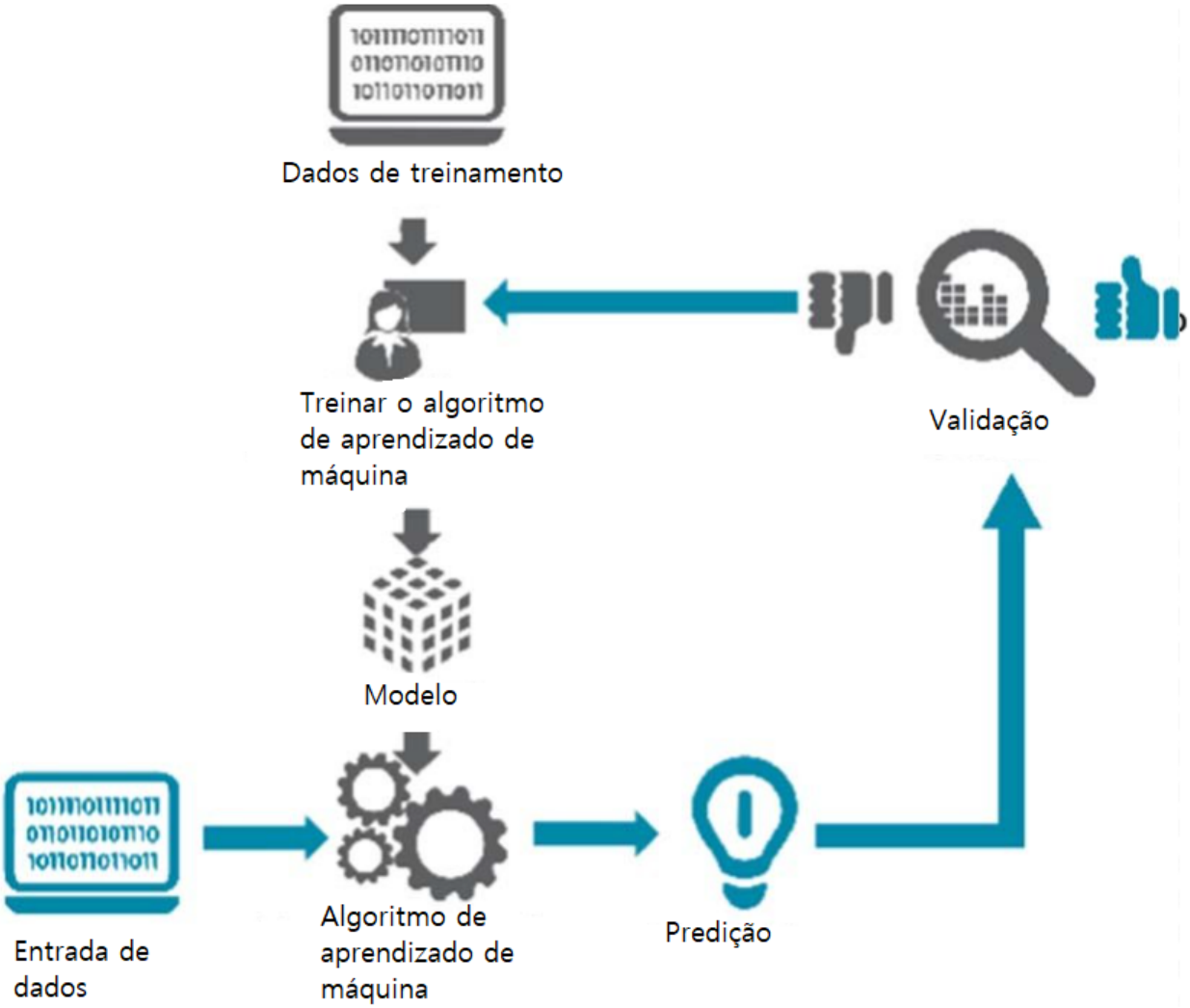
que geralmente seguem o paradigma de aprendizado supervisionado, utilizam um "supervisor externo" que fornece informações sobre os resultados corretos durante o processo de treinamento. Um exemplo prático desse cenário é o uso de algoritmos de AM para prever o estado de saúde de um paciente com base em seus sintomas, com informações previamente rotuladas (FACELI; LORENA; GAMA, 2021).

Por outro lado, nas tarefas descritivas, o objetivo principal não é a previsão, mas a descoberta de padrões e estruturas subjacentes nos dados. Os algoritmos de AM associados a essas tarefas seguem o paradigma do aprendizado não supervisionado, o que significa que não recebem orientações explícitas sobre rótulos ou categorias durante o treinamento. Em vez disso, esses algoritmos exploram os próprios dados para identificar naturalmente agrupamentos, associações ou regularidades. Exemplos típicos de tarefas descritivas incluem a identificação de comportamentos de compras similares, a descoberta de regras de associação, entre outros padrões que revelam conexões frequentes entre subconjuntos de atributos em conjuntos de dados, como explicado por Faceli, Lorena e Gama (2021).

Os três principais tipos de AM são: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço (FACELI; LORENA; GAMA, 2021).

O aprendizado supervisionado, é uma abordagem de aprendizado de máquina em que um modelo é treinado a partir de exemplos rotulados, com o objetivo de aprender a mapear entradas para saídas corretas. No aprendizado supervisionado, o agente observa alguns pares de entrada e saída de exemplo e aprende uma função que mapeia da entrada para a saída, como definido por Russell e Norvig (2022). A Figura 5 mostra o fluxo do aprendizado supervisionado, a partir do dados de treinamento, um modelo é treinado, após é inserido uma entrada de dados, e então o modelo realiza uma predição que é validada por um agente externo.

Figura 5 – Esquema do aprendizado supervisionado.

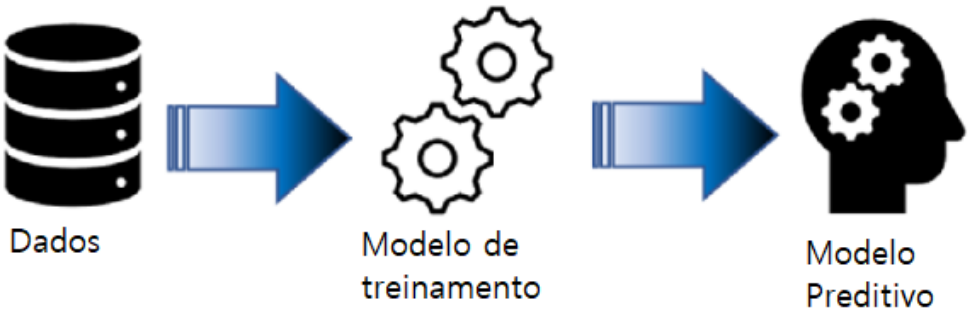


Fonte: Adaptado de <https://www.researchgate.net/publication/337517556_Intrusion_Detection_In_IoT_Using_Artificial_Neural_Networks_On_UNSW-15_Dataset>

O aprendizado não supervisionado, compreende os algoritmos que não recebem dados rotulados e busca encontrar padrões e estruturas nos dados de entrada. No aprendizado não supervisionado, o agente adquire uma compreensão dos padrões nos dados de entrada, mesmo sem receber orientação direta. Uma das tarefas predominantes nessa categoria é o agrupamento,

que envolve identificar possíveis agrupamentos significativos entre os exemplos de entrada, como explicado por Russell e Norvig (2022). A Figura 6 mostra o esquema do aprendizado não supervisionado, onde o modelo é treinado a partir dos dados de treinamento, mas não há uma validação feita por um agente externo.

Figura 6 – Esquema do aprendizado não supervisionado.



Fonte: Adaptado de <https://www.researchgate.net/publication/358518962_AI-Based_Modeling_Techniques_Applications_and_Research_Issues_Towards_Automation_Intelligent_and_Smart_Systems>

No AR, o agente é treinado através de uma série de reforço, que podem ser recompensas ou penalidades. O aprendizado ocorre através da interação com um ambiente dinâmico, no qual o agente deve realizar uma sequência de ações e pode ser recompensado ou penalizado em cada ação, dependendo do resultado. A seção seguinte detalha o AR, foco do estudo desenvolvido neste trabalho.

2.2 O QUE É APRENDIZADO POR REFORÇO

A aprendizagem por reforço é uma subárea da IA que se concentra em como um agente pode aprender a tomar decisões ideais para alcançar um objetivo, dada uma série de ações possíveis e um ambiente incerto. Diferentemente de outras formas de aprendizado, como o aprendizado supervisionado e não supervisionado, o aprendizado por reforço não depende de dados de treinamento rotulados ou de encontrar padrões ocultos nos dados. Em vez disso, o agente aprende a tomar decisões ideais por meio de um processo de tentativa e erro, onde cada ação tomada tem uma recompensa associada. O objetivo do agente é maximizar a soma total dessas recompensas ao longo do tempo, o que é conhecido como retorno (SUTTON; BARTO, 2018).

As ações podem afetar não apenas a recompensa imediata, mas também a ações subsequentes e, por meio dela, todas as recompensas subsequentes. Essas duas características, busca

por tentativa e erro e recompensa adiada, são as duas características mais importantes que distinguem o aprendizado por reforço (SUTTON; BARTO, 2018).

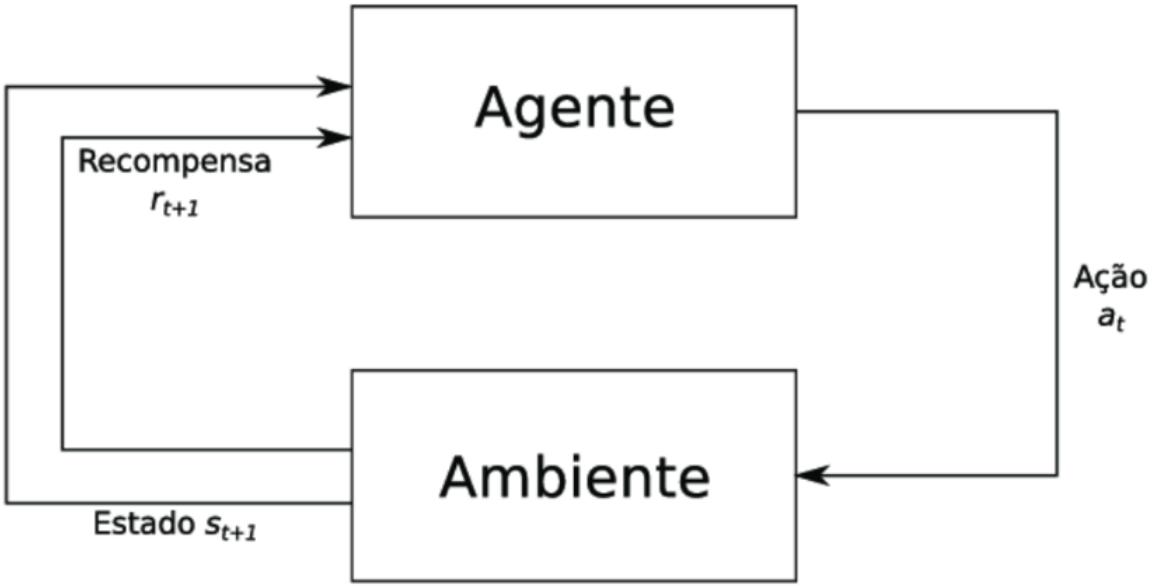
Figura 7 – Ambiente de estacionamento de automóveis.

Um exemplo prático de aprendizado por reforço é o sistema de direção autônoma desenvolvido pela *Waymo*, uma subsidiária da *Alphabet Inc*. A *Waymo* utiliza o AR para treinar seus veículos autônomos a navegar em ambientes complexos e imprevisíveis. O veículo, atuando como agente, interage com o ambiente (as ruas, outros veículos, pedestres, etc...) e aprende a tomar decisões de direção com base nas recompensas e penalidades que recebe. A figura 7 representa um ambiente de estacionamento para veículos autônomos, onde o estacionamento é o ambiente, o carro é o agente que precisa tomar as decisões, a partir da ação, uma transição de estados acontece, neste caso o carro vai de uma marcação no estacionamento pra outra. Se o veículo faz uma manobra segura, ele recebe uma recompensa positiva, se faz uma manobra perigosa, recebe uma penalidade. Com o tempo, o veículo aprende a dirigir de forma mais segura e eficiente (JEBESSA *et al.*, 2022) .

Um agente, conforme definido por Russell e Norvig (2022), é qualquer entidade capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente através de atuadores. A interação do agente com o ambiente é fundamental para suas operações. Além do agente e do ambiente, um sistema de AR possui quatro subelementos principais: uma política, um sinal de recompensa, uma função de valor e, opcionalmente, um modelo do ambiente. A política define o comportamento do agente, o sinal de recompensa define o objetivo, e a função de valor ajuda na avaliação das ações do agente (SUTTON; BARTO, 2018).

Um agente, seja hardware ou software, possui quatro propriedades principais (SANTOS, 2021):

Figura 8 – Diagrama do aprendizado por reforço.



1. Autonomia: os agentes operam sem a intervenção direta de humanos, ou outros, e têm algum tipo de controle sobre suas ações e seu estado interno.

2. Habilidade social: os agentes interagem com outros agentes (e possivelmente humanos) por meio de algum tipo de linguagem de sua comunicação.

3. Reatividade: os agentes percebem seu ambiente – que pode ser o mundo físico, um usuário por meio de uma interface gráfica, uma coleção de outros agentes, a internet ou talvez todos esses elementos combinados – e respondem em tempo hábil às mudanças que ocorrem nele.

4. Proatividade: os agentes não agem simplesmente em resposta ao seu ambiente; eles são capazes de exibir um comportamento direcionado a um objetivo, tomando a iniciativa.

A política define a estratégia de comportamento de um agente em um momento específico. Ela corresponde aos estados percebidos do ambiente e às ações a serem tomadas pelo agente. A política pode ser uma função simples ou envolver processos computacionais mais complexos, como busca, e pode ser estocástica, especificando probabilidades para ações (SUTTON; BARTO, 2018).

Um modelo do ambiente pode ser determinístico ou estocástico, introduzindo aleatoriedade em seus resultados. Esses modelos ajudam o agente a prever as consequências de suas ações. O sinal de recompensa define o objetivo do aprendizado por reforço. O agente busca maximizar a soma total de recompensas futuras. Isso é conhecido como retorno. Em geral, o retorno é uma soma ponderada das recompensas futuras (SUTTON; BARTO, 2018).

A função de valor é uma previsão do retorno esperado, dada uma ação específica em um estado específico. A função de valor ajuda o agente a avaliar a qualidade de suas ações e a escolher a melhor ação a ser tomada. A função de valor pode ser aprendida por meio de um processo de atualização iterativa, onde o agente ajusta suas estimativas de valor com base nas recompensas recebidas e nas novas informações obtidas (SUTTON; BARTO, 2018).

Em resumo, a AR é uma abordagem poderosa para a resolução de problemas de decisão sequencial, onde um agente deve aprender a tomar uma série de decisões para maximizar um objetivo a longo prazo. A AR tem sido aplicada com sucesso em uma variedade de domínios, desde jogos até robótica e carros autônomos, e continua a ser uma área de pesquisa ativa na IA. Uma das principais vantagens do AR é sua capacidade de generalizar a partir de experiências passadas para lidar com situações nunca antes vistas. Isso torna o AR particularmente útil para tarefas onde o ambiente pode mudar ou ser inicialmente desconhecido para o agente.

2.2.1 Método de Markov

O Processo de decisão de Markov (MDP) é usado para modelar situações em que ações precisam ser executadas sequencialmente em ambientes incertos. Essa incerteza pode se manifestar tanto no resultado das ações quanto no estado atual do ambiente. Por exemplo, um jogador de xadrez deve escolher a melhor jogada a cada turno, mas não tem certeza como o adversário vai responder. O MDP permite calcular a melhor política de decisão, ou seja, a regra que indica qual ação tomar em cada estado, considerando as probabilidades dos valores resultantes das ações (PELLEGRINI; WAINER, 2007) .

Os processos são denominados *de Markov* pois seguem a *propriedade de Markov*, onde o impacto de uma ação em um estado é determinado unicamente pela ação e pelo estado atual do sistema. Eles são referidos como processos *de decisão* porque representam a capacidade de um agente de intervir no sistema de forma periódica através da execução de ações (PELLEGRINI; WAINER, 2007).

Um exemplo típico de uma situação que pode ser representada como um MDP envolve um sistema com vários estados, ações que podem alterar o estado do sistema e a capacidade de perceber, ainda que indiretamente, os resultados de cada ação executada. Resolver o problema em questão implica encontrar uma política ótima que, a cada momento, determina qual ação tomar, com o objetivo de maximizar a recompensa esperada ou minimizar o custo esperado (PELLEGRINI; WAINER, 2007).

Para ilustrar, considere o jogo de xadrez. Em um jogo de xadrez, cada posição no tabuleiro pode ser considerada um *estado*, e cada movimento possível é uma *ação*. O resultado de cada ação (movimento) é incerto porque depende da resposta do adversário. O objetivo do jogador é encontrar a melhor estratégia ou *política* que maximiza a chance de ganhar o jogo, considerando todas as possíveis respostas do adversário.

O MDP é um modelo matemático que descreve situações em que decisões devem ser tomadas em sequência, e o resultado de cada decisão não é claro para o tomador de decisões. Por exemplo, um jogador de xadrez deve escolher a melhor jogada a cada turno, mas não sabe como o adversário vai responder. O processo de markov permite calcular a melhor política de decisão, ou seja, a regra que indica qual ação tomar em cada estado, considerando as probabilidades dos valores resultantes das ações (PELLEGRINI; WAINER, 2007).

A definição de um processo de decisão de Markov segundo Pellegrini e Wainer (2007, p. 3).

> Um MDP é uma tupla (S,A,T,R), onde:
>
> - S é um conjunto de estados em que o processo pode estar;
> - A é um conjunto de ações que podem ser executadas em diferentes épocas de decisão;
> - $T : S \times A \times S \rightarrow [0, 1]$ é uma função que dá a probabilidade de o sistema passar para um estado s' $\in S$, dado que o processo estava em um estado $s \in S$ e o agente decidiu executar uma ação $a \in A$ (denotada $T(s|s, a)$);
> - $R : S \times A \rightarrow R$ é uma função que dá o custo (ou recompensa) por tomar uma decisão $a \in A$ quando o processo está em um estado $s \in S$.
>
> Pode-se também definir, para cada estado $s \in S$, um conjunto de ações possíveis naquele estado A_s. Assim, o conjunto de todas as ações poderia ser $\cup_{s \in S} A_s$, a fim simplificar é utilizado apenas "A".Os conjuntos S e A podem ser finitos ou infinitos.
>
> O horizonte de um MDP é o número de épocas de decisão disponíveis para a tomada de decisões. Definido z para denotar o horizonte de um MDP.O horizonte pode ser finito (quando há um número fixo de decisões a tomar), infinito (quando a tomada de decisão é feita repetidamente, sem a possibilidade de parada) ou indefinido (semelhante ao horizonte infinito, mas com a possibilidade do processo parar se chegar a algum estado que tenha sido marcado como final (PELLEGRINI; WAINER, 2007).

Como Pellegrini e Wainer (2007) explica, o processo de tomada de decisões em um contexto de MDP envolve várias etapas. O tomador de decisões inicia verificando o estado atual do sistema, representado por s. Em seguida, consulta uma política, geralmente denotada como π, que fornece orientações sobre qual ação tomar em um determinado estado. Assim, com base na política, o tomador de decisões seleciona uma ação, representada por a. A figura 8 ilustra o funcionamento de um sistema modelo como processo de MDP.

Essa ação escolhida pode afetar o ambiente, provocando uma mudança no estado atual do sistema. Após a execução da ação, o tomador de decisões verifica o novo estado resultante, preparando-se para a próxima iteração de decisão (PELLEGRINI; WAINER, 2007).

A cada época de decisão, o tomador de decisões segue uma regra específica para escolher a próxima ação. Uma regra de decisão simples pode ser expressa como um mapeamento direto dos estados para ações, representado como $d : S \rightarrow A$, onde S é o conjunto de estados e A é o conjunto de ações possíveis. O conjunto de todas as regras de decisão, uma para cada época de decisão, é chamado de *política* (PELLEGRINI; WAINER, 2007).

A política pode ser total quando regras de decisão estão definidas para todos os estados do MDP, ou parcial quando se aplicam apenas a estados específicos. Em relação às épocas de decisão, ela pode ser classificada como estacionária, quando a ação recomendada permanece constante, independentemente da época em questão, ou não-estacionária, quando a ação depende da época de decisão (PELLEGRINI; WAINER, 2007).

Além disso, uma política pode ser determinística, o que significa que, em cada estado, ela está sempre associada a uma única ação. Por outro lado, também pode ser não-determinística, que inclui elementos de aleatoriedade ou estocasticidade, onde um estado pode ser mapeado em um conjunto de ações, sendo que cada ação tem uma probabilidade de ser escolhida. No caso de não-determinística, a regra de decisão é uma função $d_k : S \times A \rightarrow [0, 1]$ (PELLEGRINI; WAINER, 2007).

Uma política, ainda por ser classificada como *Markoviana*, quando a ação depende apenas do estado corrente. E de forma contrária, é classificada como *não-Markoviana* quando a escolha depende do histórico de ações e dos estados até o momento. No caso de *não-Markoviana*, as regras de decisão são definidas como funções $d_k : H_k \rightarrow A$, sendo H_k o histórico de ações e estados até o momento (PELLEGRINI; WAINER, 2007).

Para comparar duas políticas, é necessário um critério de desempenho. Os critérios de desempenho ou otimização mais conhecidos para um MDP são:

- A recompensa média por época de decisão: é a média das recompensas recebidas a cada passo do processo. Esse critério é adequado para processos que não têm um fim definido, ou seja, que são contínuos, $\frac{1}{z}\sum_{k=0}^{z-1} r_k$;
- A recompensa esperada total: é a soma total das recompensas que um agente espera receber ao longo do tempo, considerando todas as recompensas futuras sem qualquer desconto, $E[\sum_{k=0}^{z-1} r_k]$;
- A recompensa total descontada: é a soma das recompensas recebidas ao longo do processo, multiplicadas por um fator de desconto γ que varia entre 0 e 1. Esse fator serve para dar mais peso às recompensas imediatas do que às futuras, pois há mais incerteza sobre o que acontecerá no futuro. Esse critério é adequado para processos que têm um fim definido $E[\sum_{k=0}^{z-1}\gamma^k r_k]$;

Uma regra de decisão para um MDP, em uma época de decisão k é uma função $dk : S \rightarrow A$, que determina a ação a ser executada, dado o estado do sistema. Uma política para um MDP é uma sequência de regras de decisão $\pi = d_0, d_1...1d_{z-1}$ uma para cada época de decisão (PELLEGRINI; WAINER, 2007).

A função valor de uma política π para um MDP $M = (S, A, T, R)$ é uma função $V^\pi : S \rightarrow R$, tal que $V^\pi\ (s)$ dá o valor esperado da recompensa para esta política, de acordo com o critério de otimalidade (PELLEGRINI; WAINER, 2007).

$$V_k^{\pi}(s) = R(s, d_k(s)) + \gamma \sum_{s' \in S} T(s, d_k(s), s') V_{k+1}^{\pi}(s')$$

onde $V \overset{\pi}{z} = 0$ para todo s, visto após a última época de decisão não há mais ações que possam gerar recompensas. Quando não houver ambiguidade, V será usado ao invés de V^{π}.

Seja $M = (S, A, T, R)$ um MDP com horizonte z. Uma função valor V^* é ótima para M se $\forall U \in \nu$, $\forall s \in S$, $V_k(s) > U_k$ para todo $k \in N$ tal que $0 \leq k < z$ (PELLEGRINI; WAINER, 2007)..A função que satisfaz esse critério, chamada de função valor ótima, pode ser descrita como :

$Q(s, a) = R(s, a) + \sum_{s' \in S} T(s' \mid s, a) \cdot V^*(s')$

2.2.2 Método Q-Learning

O Q-learning, proposto por Watkins em 1989, é uma abordagem sem modelo que capacita os agentes a aprenderem e agirem de forma otimizada em domínios Markovianos. Essa técnica de aprendizado por reforço permite que os agentes aprendam diretamente a partir da interação com o ambiente, coletando dados sobre as recompensas recebidas e os estados alcançados, sem depender de um modelo explícito do ambiente (WATKINS, 1992).

O processo de aprendizado do Q-learning envolve um agente tentando realizar ações em estados específicos, avaliando as consequências em relação às recompensas imediatas e de longo prazo, com base em estimativas do valor do estado para o qual é levado. Ao experimentar todas as ações em todos os estados repetidamente, o agente aprende quais ações são as melhores para cada estado, tornando-se uma técnica fundamental no campo do AR (WATKINS, 1992).

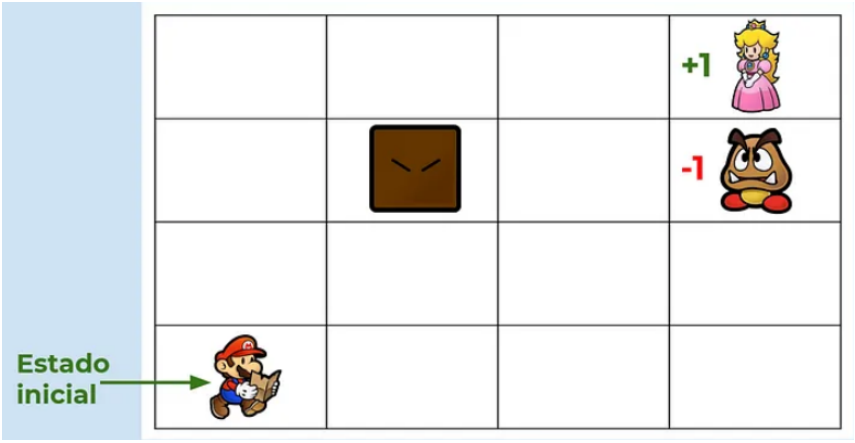
O algoritmo Q-Learning mantém uma tabela chamada tabela Q, que é atualizada para cada par estado-ação com o valor Q. Esse valor Q representa o nível de recompensa associado a cada ação tomada em um determinado estado, ou seja, a tabela Q armazena a expectativa da recompensa ao realizar uma ação em um determinado estado. A Figura 9 mostra um exemplo de uma tabela Q, onde cada ação representa uma direção, contendo um valor Q para par estado-ação, por exemplo, ao escolher realizar a ação sul quando esta no estado 328, a recompensa é de-2.30108105.

A Figura 10 representa um ambiente em um jogo de tabuleiro, cada quadrado representa um estado, o personagem Mário é o agente, chegar até a princesa concede uma recompensa positiva e chegar no personagem goomba concede uma recompensa negativa. Para cada estado existe um conjunto de ações, a Figura 11 mostra o conjunto de ações disponíveis para cada um dos estados. Por fim a Figura 12 mostra qual a expectativa de recompensa para cada par de estado-ação, armazenado na tabela Q.

Figura 9 – Exemplo de uma tabela Q

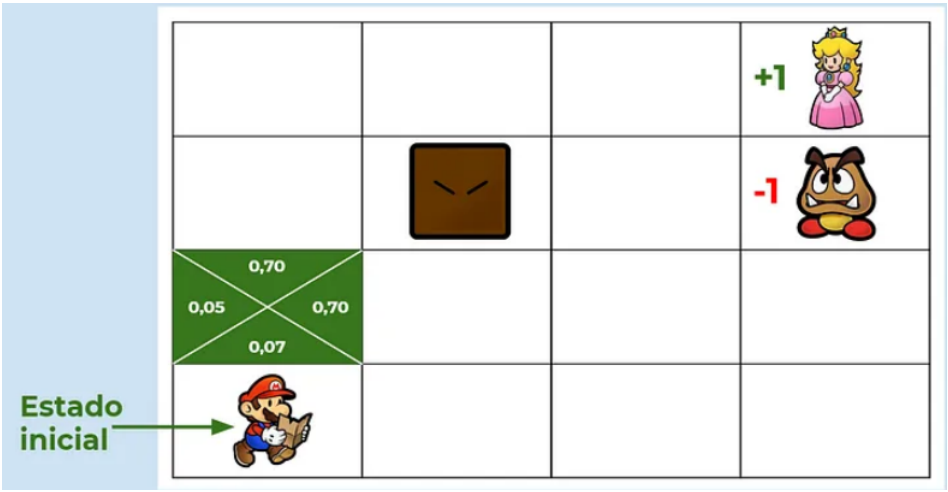
Tabela - q		Ações					
		Sul (0)	Norte (1)	Leste (2)	Oeste (3)	Embarque(4)	Desembarque(5)
Estados	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Figura 10 – Representação de um ambiente, cada quadrado representado um estado



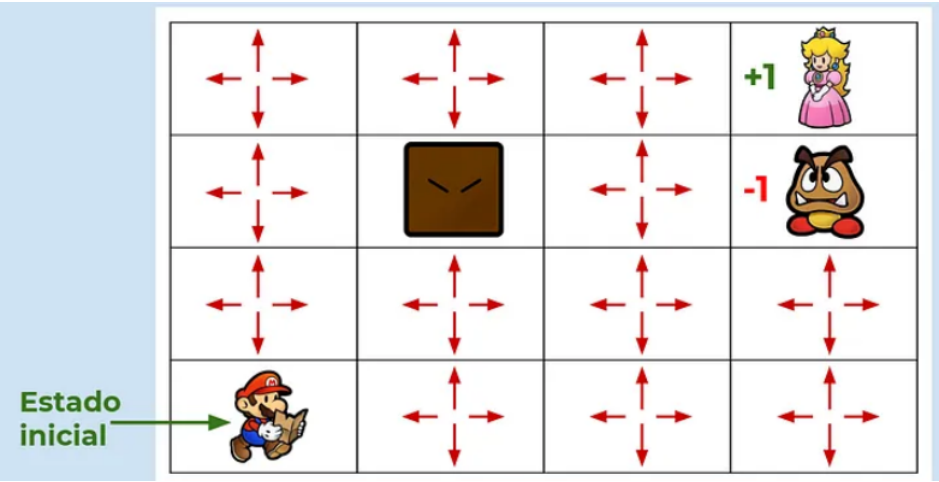
Fonte: Retirado de https://paulovasconcellos.com.br/explicando-deep-reinforcement-learning-com-super-mario-ao-inves-de-matematica-4c77392cc733

Figura 11 – Conjunto de ações possíveis para cada estado



Fonte: Retirado de https://paulovasconcellos.com.br/explicando-deep-reinforcement-learning-com-super-mario-ao-inves-de-matematica-4c77392cc733

Figura 12 – Recompensa por estado-ação



Fonte: Retirado de https://paulovasconcellos.com.br/explicando-deep-reinforcement-learning-com-super-mario-ao-inves-de-matematica-4c77392cc733

A função Q do Q-Learning é representada por :

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

onde:

- a_t é a ação do momento t.
- s_t é o estado do momento t.
- $Q(s_t, a_t)$ é o valor atual da recompensa estimada para o movimento a_t no estado s_t.
- γ é a constante de desconto.
- α é a constante de fator de aprendizado.
- r_{t+1} é a recompensa do estado $t + 1$.
- $\max_{a'} Q(s_{t+1}, a')$ é a expressão que calcula o valor máximo da função de valor de ação Q para o estado s_{t+1}, representando a ação que maximiza o valor da função Q nesse estado específico.

As etapas do Q-learning são:

1. Inicialize $Q(s, a)$ arbitrariamente.
2. Repita (para cada episódio).
3. Inicialize s.
4. Repita para cada passo do episódio.

5. Escolha $\alpha \in A(s)$.
6. Execute a ação α.
7. Observe os valores s' e r.
8. $Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$
9. $s < -s'$.
10. até que s seja terminal.

Figura 13 – Funcionamento procedural do Q-Learning.

Inicializar $Q(s, a), \forall s \in S, a \in A(s)$ aleatoriamente e
$Q(estadoFinal, -) = 0$;
foreach *episódio* **do**
  Inicializar S
  repeat
    Escolher A de S usando política derivada de Q
    Realização ação A, observar R e S'
    $Q(S, A) = Q(S, A) + \alpha[R + \gamma max_a Q(S', a) - Q(S, A)]$
    $S = S'$
  until *S é um estado final*;
end

2.3 REDES NEURAIS

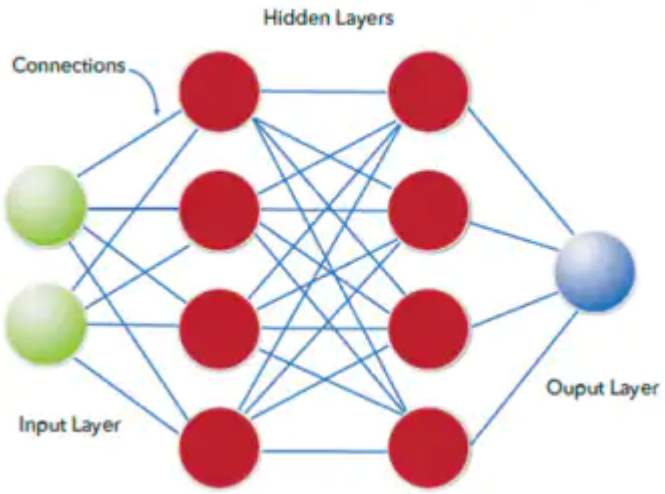
Uma rede neural artificial (RNA) é um modelo computacional inspirado no sistema nervoso biológico, que consiste em neurônios artificiais organizados em camadas interconectadas (GOODFELLOW; BENGIO; COURVILLE, 2016). Cada neurônio recebe entradas ponderadas, aplica uma função de ativação e transmite um sinal para os neurônios na camada seguinte, permitindo assim a aprendizagem de representações complexas dos dados (RUSSELL; NORVIG, 2016).

As RNAs são capazes de aprender a partir de exemplos, ajustando automaticamente os pesos das conexões entre neurônios para melhorar seu desempenho em tarefas específicas (HAYKIN, 2009). Esse processo de aprendizagem é geralmente realizado por algoritmos de otimização, como o gradiente descendente, que minimiza uma função de perda através da propagação do erro (NIELSEN, 2015).

Existem diferentes arquiteturas de RNAs, como as redes feedforward, onde a informação se move em uma direção, sem ciclos (BISHOP, 2006). Já as redes recorrentes permitem conexões retroalimentadas, sendo adequadas para tarefas sequenciais, como processamento de linguagem natural (GRAVES; FERNÁNDEZ; SCHMIDHUBER, 2012).

A figura 14 exemplifica uma rede neural simples, que inclui uma camada de entrada, outra de saída (ou alvo) e, entre elas, uma camada oculta. As camadas são conectadas através de nós e essas conexões formam uma "rede".

Figura 14 – Ilustração de uma rede neural



Fonte: Retirado de <Fonte:https://www.sas.com/pt_br/insights/analytics/neural-networks.html>

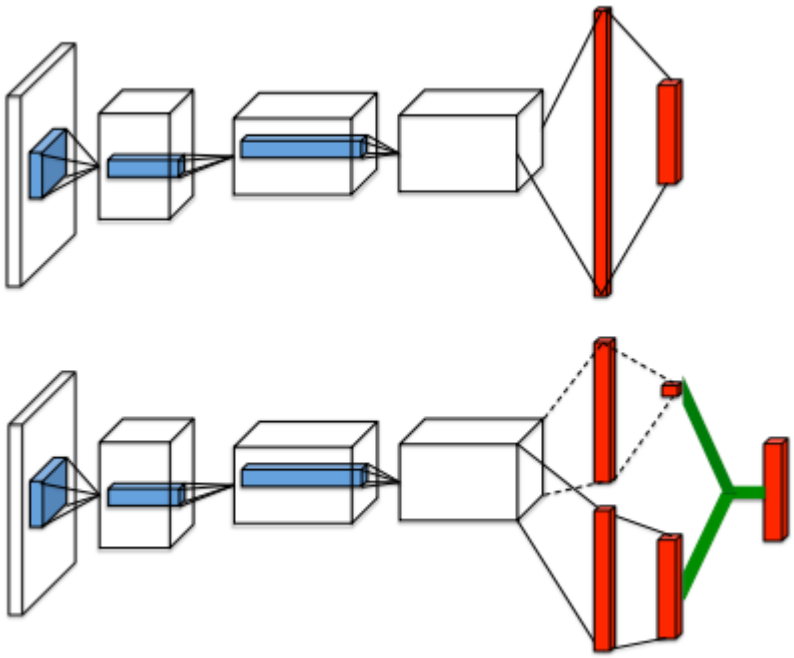
Em jogos, redes neurais são frequentemente utilizadas para reconhecimento de padrões em imagens, como captura de tela de um ambiente virtual. Por exemplo, em jogos de estratégia em tempo real, uma RNA pode analisar pixels da tela para identificar unidades inimigas ou aliadas, tomando decisões baseadas nessa análise (SCHMIDHUBER, 2015). Esta capacidade de processar informações visuais em tempo real torna as RNAs valiosas para melhorar a jogabilidade e a inteligência artificial de personagens não jogáveis (NPCs).

2.3.1 Double Deep Q-Network

DDQN é uma variante do algoritmo de AR chamado DQN. A DDQN aborda um problema comum encontrado no DQN conhecido como *overestimation bias* (viés de superestimação), onde o algoritmo tende a superestimar os valores Q durante o aprendizado (WANG *et al.*, 2016). A arquitetura de duelo é uma estrutura inovadora que consiste em dois fluxos distintos, representando as funções de valor e vantagem. Esses fluxos compartilham um módulo comum de aprendizado de recursos convolucionais. As duas correntes são combinadas por meio de uma camada de agregação especial para produzir uma estimativa da função de valor de ação de estado Q conforme ilustrado na Figura 15 (WANG *et al.*, 2016).

Essa rede de duelo é essencialmente uma rede Q única com dois fluxos, que serve como uma alternativa à popular rede Q de fluxo único em algoritmos existentes. A rede de duelo gera

Figura 15 – A rede Q de fluxo único (topo) e a rede Q de duelo (fundo) são diferenciadas pela última ter dois fluxos para estimar separadamente o valor do estado e as vantagens para cada ação, que são combinadas por um módulo de saída para produzir os valores Q para cada ação.



Fonte: Retirado de <Fonte:https://arxiv.org/pdf/1511.06581v3>

automaticamente estimativas separadas da função de valor de estado e da função de vantagem, sem necessidade de supervisão extra (WANG *et al.*, 2016).

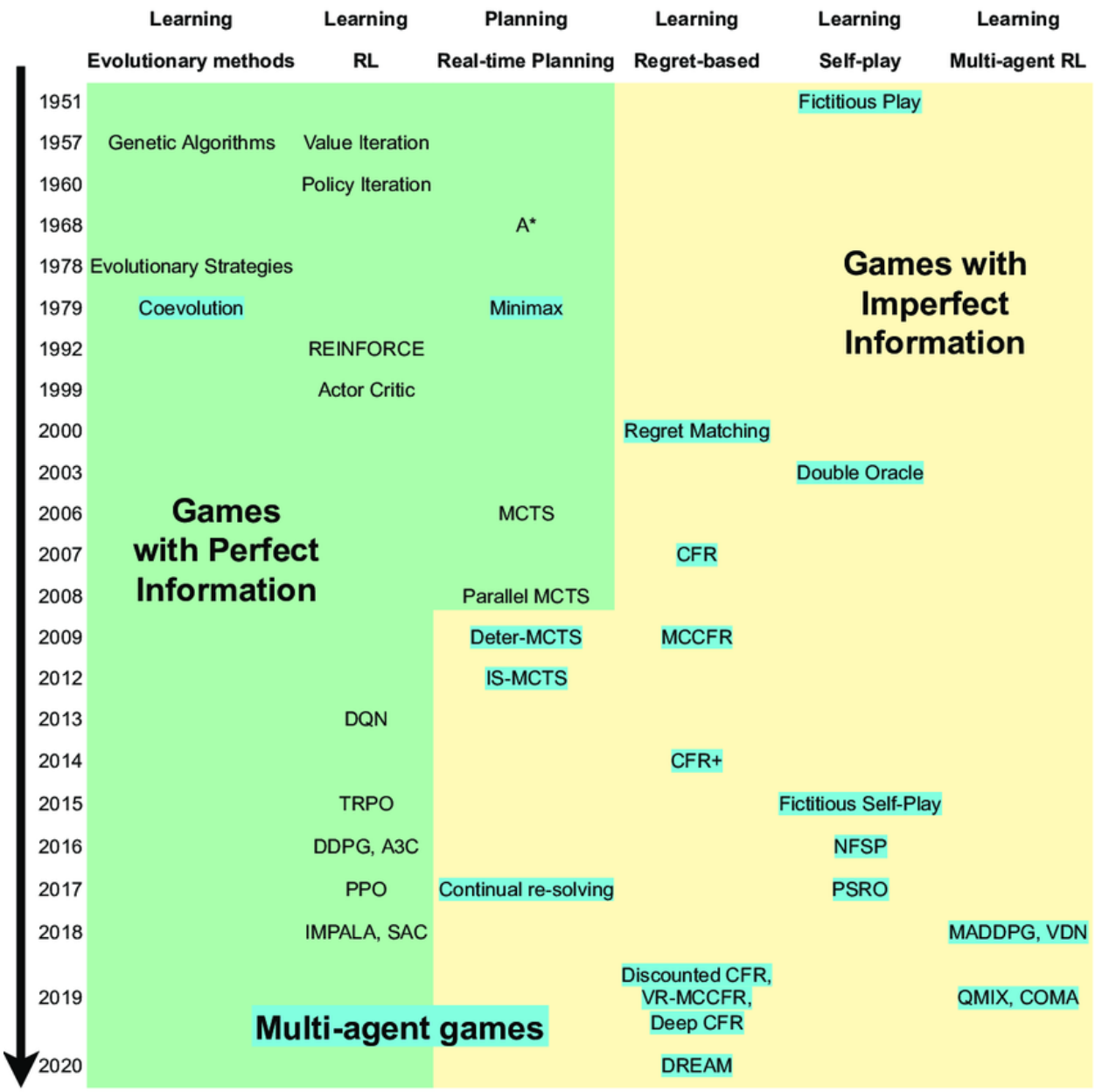
2.4 APRENDIZAGEM EM JOGOS

A IA tem sido uma parte integral dos videogames desde a década de 1950, proporcionando comportamentos inteligentes e responsivos, especialmente em personagens não-jogáveis. Com o aumento da complexidade dos jogos nas últimas décadas, a IA teve que evoluir para criar personagens que respondem de maneiras imprevisíveis (LIMA, 2014).

Conforme ilustrado na linha do tempo na Figura 16, a complexidade crescente dos jogos levou ao desenvolvimento de novos algoritmos de IA. Inicialmente, o foco estava em otimizar o comportamento de agentes individuais. No entanto, com o tempo, a necessidade de lidar com ambientes de jogo mais complexos e imprevisíveis levou ao desenvolvimento de técnicas mais sofisticadas.

Os métodos evolutivos foram uma das primeiras abordagens para criar agentes de jogos. Eles funcionam através da busca e seleção aleatórias no espaço de parâmetros. No entanto, esses métodos podem ser ineficientes quando o espaço de política é muito grande (LU; LI, 2023).

Figura 16 – Linha do tempo ilustrando a evolução dos jogos em relação à IA.



Fonte: Retirado de https://www.researchgate.net/figure/Features-of-the-games-tackled-in-recent-milestones_tbl1_362701454

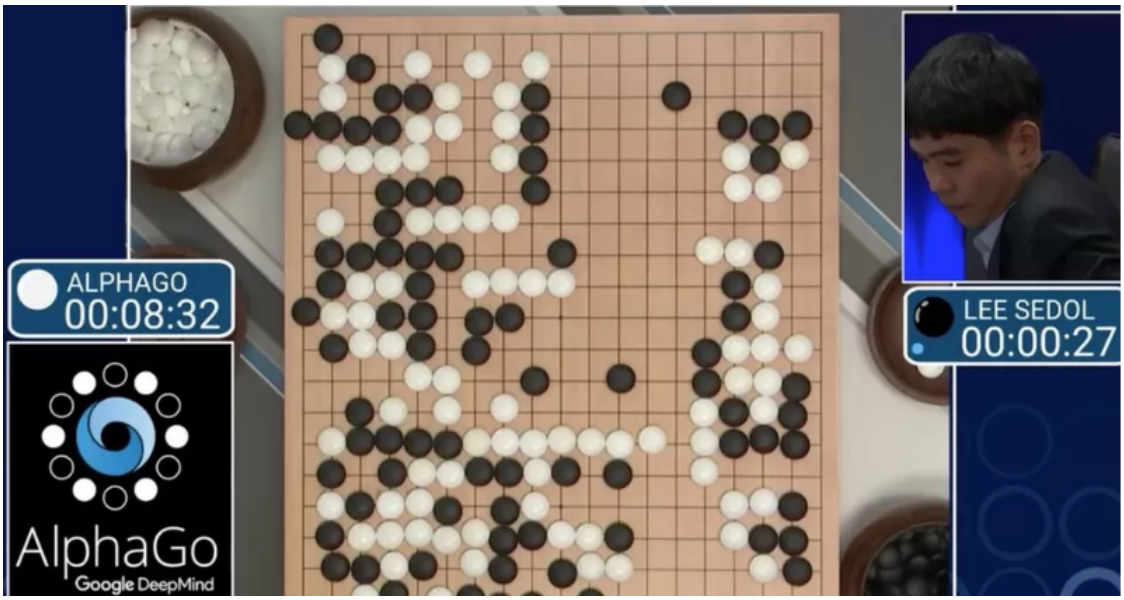
O AR surgiu como uma alternativa promissora, capaz de lidar com ambientes onde nem todas as informações sobre o estado do jogo estão disponíveis para todos os jogadores. Isso é particularmente útil em jogos com informação imperfeita, onde o agente deve aprender a tomar decisões com base em informações incompletas (LU; LI, 2023).

Nas últimas décadas, os pesquisadores começaram a explorar jogos mais complexos, como jogos de cartas e videogames, que são jogos com informações imperfeitas. Para esses jogos, foram desenvolvidos novos algoritmos e técnicas, como a pesquisa de árvore Monte Carlo e algoritmos de AR para vários agentes (LU; LI, 2023).

Hoje, os sistemas modernos de IA para jogos usam uma combinação de várias técnicas, em vez de depender de um único algoritmo. Uma IA típica para jogar um jogo específico geralmente envolve algum conhecimento prévio, que é então usado no jogo, combinado com planejamento e raciocínio em tempo real (LU; LI, 2023).

O IA também é utilizado na melhoria da jogabilidade e na personalização da experiência do usuário. Um exemplo notável é a criação do sistema *AlphaGo* pela *DeepMind*, uma empresa de IA do *Google*. O *AlphaGo* é um programa de computador pioneiro que alcançou marcos significativos no mundo do Go, um antigo e complexo jogo de tabuleiro estratégico conhecido como *Baduk* na Coreia e *Weiqi* na China. A Figura 17 mostra Lee Sedol jogando contra o *AlphaGo*, que se tornou o primeiro programa a derrotar um jogador humano profissional de Go e também o primeiro a vencer um campeão mundial de Go, sendo considerado por muitos como o jogador mais forte da história do Go, como explicado em Silver e Hassabis (2016). Essa conquista demonstra o potencial transformador do AM não apenas em jogos, mas também em uma variedade de outras aplicações.

Figura 17 – Lee Se-dol, campeão de GO jogando contra o AlphaGo.



Fonte: Retirado de <https://tecnoblog.net/noticias/computador-google-vence-campeao-go/>

A área de AM apresenta um vasto potencial em diversas aplicações. O AR, em especial, destaca-se como uma abordagem promissora, especialmente no caso de jogos eletrônicos. No futuro, espera-se que a IA avance em direção a jogos do mundo real, como jogos esportivos, devido à sua maior complexidade e ao potencial de inspirar estratégias inovadoras. Além disso, espera-se que a IA de jogos continue a se beneficiar do avanço da tecnologia computacional, permitindo o treinamento de agentes em jogos de maior complexidade

2.5 TRABALHOS RELACIONADOS

Com o intuito de realizar uma busca imparcial e analítica de trabalhos relacionados, foi conduzido um procedimento de revisão sistemática. O objetivo central da revisão sistemática é identificar o conhecimento científico previamente desenvolvido por outros pesquisadores. Para isso, foram considerados os critérios estabelecidos no guia de Sampaio e Mancini (2007), os quais orientaram a coleta e análise do material bibliográfico. O portal *semantic scholar* foi escolhido como fonte de pesquisa do material científico. Os seis passos estabelecidos foram:

1. Determinar uma questão de pesquisa
2. Identificação das palavras chaves
3. Filtragem inicial
4. Download do material
5. Análise superficial
6. Análise aprofundada

No passo 1, busca-se responder à pergunta: "Quais as técnicas e métodos de aprendizagem por reforço são utilizadas em jogos?". A motivação para realizar esta revisão sistemática é dupla. Primeiro, pretende-se identificar as técnicas mais promissoras e eficazes de aprendizagem por reforço em jogos, o que pode fornecer *insights* valiosos para pesquisadores e desenvolvedores de jogos. Segundo, ao sintetizar a literatura existente e assim, espera-se contribuir para o corpo de conhecimento na área de aprendizagem por reforço em jogos (SAMPAIO; MANCINI, 2007).

No Passo 2, houve a identificação das palavras-chave. As palavras-chave utilizadas na busca foram: 'artificial intelligence', 'reinforcement learning', 'game', 'atari' e 'pong'. Elas foram combinadas no filtro de pesquisa do portal, resultando em 704 resultados.

No Passo 3, foi realizada a filtragem inicial. Com o objetivo de reduzir a quantidade de resultados, foi selecionada a opção *Computer Science* no filtro *Fields of Study*, o período entre 2019 e 2023 no filtro *Date Range* e marcada a opção *Has PDF*. Desta forma, foram obtidos 173 resultados.

No Passo 4, foi feito o download do material bibliográfico. Dos resultados obtidos, somente 43 estavam disponíveis gratuitamente para download.

No Passo 5, foi realizada uma análise superficial. Após essa análise, restaram 12 resultados, dos quais 8 foram descartados pois não se adequavam aos critérios. Assim, 4 artigos continham objetivos e temas semelhantes, os quais serão apresentados nesta seção.

2.5.1 Utilização de Aprendizado Supervisionado e por Reforço na Automação do Clássico Jogo Atari 'Pong'

O trabalho foi escrito por WATERREUS, com o objetivo de desenvolver um programa de IA capaz de jogar o jogo *Pong* de forma autônoma, utilizando técnicas de AR como o aprendizado supervisionado e aprendizado por reforço. O autor descreve o processo de desenvolvimento do programa, incluindo a utilização de métodos de tentativa e erro e a disponibilização dos arquivos do programa no *GitHub* para futuras pesquisas. O trabalho também apresenta a organização da tese, com capítulos dedicados à revisão da literatura, métodos utilizados, resultados obtidos e conclusões.

O autor desenvolveu um ambiente do zero utilizando a linguagem *Python*, projetado para operar de maneira semelhante ao programa *Matlab Pong*, a fim de manter os resultados comparáveis entre as técnicas de aprendizado de máquina empregadas. A escolha de *Python* permitiu a utilização de bibliotecas amplamente usadas no AM, como *TensorFlow* e *Keras*.

Para o aprendizado supervisionado, foi utilizado redes neurais[1], sendo assim necessário realizar a coleta de dados para criação de uma base de dados. Foram coletados 40.000 registros e foi necessário jogar entre 60 a 70 jogos para coletar os dados. A análise do projeto revelou diferenças significativas nas precisões entre as raquetes esquerda e direita. A raquete esquerda, com 61,84% de precisão, enquanto a raquete da direita apresentou 72,82% de precisão. Essa diferença se dá ao fato de que o modelo é sensível às amostras, o que significa que pequenas variações nos dados de entrada podem levar a grandes variações nos resultados. Isso pode ter impactado o desempenho do programa, especialmente em termos de sua capacidade de prever com precisão a trajetória da bola.

Posteriormente, foi empregado o uso de uma DQN [2] para o AR, no entanto não convergia para um valor, ao contrário do que geralmente se espera de uma DQN. O autor sugere a necessidade de uma função aprimorada no programa DQN desenvolvido. Em média, as raquetes conseguiam realizar entre três e cinco rebatidas antes que qualquer uma das raquetes perdesse a bola. No entanto, nenhuma das raquetes alcançou um nível de desempenho onde se considerava capaz de desafiar um oponente humano.

2.5.2 Aprendizado por Reforço Baseado em Modelo para Atari

O trabalho realizado por Kaiser *et al.* (2019) teve como objetivo explorar como modelos de previsão de vídeo podem permitir que agentes resolvam jogos *Atari* com menos interações do que os métodos sem modelo. Para isso, foi apresentado o Simulated Policy Learning (Simple),

[1] Modelos computacionais inspirados no funcionamento do cérebro humano, consistem em camadas de 'neurônios' interconectados, processando informações para tarefas como reconhecimento de padrões, previsões e tomada de decisões

[2] Algoritmo do campo o aprendizado por reforço que combina os princípios das redes neurais profundas com o Q-Learning

um algoritmo de AR baseado em modelo. O Simple opera diretamente em observações de pixels brutos e aprende políticas eficazes para jogar jogos no ambiente de prendizado *Atari*. O modelo preditivo do Simple possui variáveis latentes estocásticas para lidar com ambientes altamente estocásticos. A abordagem usa o modelo como um simulador aprendido e aplica diretamente o aprendizado de políticas sem modelo para adquirir a política.

O ambiente utilizado foi o Arcade Learning Environment (ALE), que é uma plataforma desenvolvida para avaliar e comparar o desempenho de algoritmos e técnicas de inteligência artificial ao jogar jogos clássicos de fliperama. O Simple foi avaliado em vários jogos *Atari*, incluindo *Beam Rider*, *Breakout*, *Enduro*, *Freeway*, *Pong*, *Q*Bert*, *Seaquest* e *Space Invaders*. Em dois jogos, *Pong* e *Freeway*, o Simple foi capaz de alcançar a pontuação máxima. A métrica utilizada para avaliar o desempenho dos agentes foi quantidade de *steps* para atingir a pontuação máxima nos jogos.

Os resultados mostraram que o Simple superou os métodos de aprendizado profundo sem modelo de última geração em termos de eficiência de amostra na maioria dos jogos *Atari*.

2.5.3 Aprendizado Profundo para Jogabilidade em Tempo Real de Jogos Atari Utilizando Planejamento Offline de Busca de Árvore de Monte Carlo

O trabalho escrito por Guo *et al.* (2014) apresenta uma análise de agentes de jogos que combinam algoritmos de planejamento baseados em Monte Carlo Tree Search (MCTS) e redes neurais convolucionais [3]. O MCTS é utilizado para selecionar a melhor ação a ser tomada em um determinado estado do jogo, simulando várias jogadas futuras a partir desse estado e avaliando a qualidade dessas jogadas. Para isso, o MCTS utiliza uma árvore de busca para representar as possíveis jogadas e seus resultados, e emprega a amostragem para estimar a qualidade das jogadas.

Os agentes propostos foram avaliados em sete jogos *Atari*, *Breakout*, *Enduro*, *Pong*, *Q*bert*, *Seaquest*, *Space Invaders* e *Freeway*. A métrica utilizada para avaliar o desempenho dos agentes foi a pontuação obtida nos jogos.

Um dos principais resultados do estudo é que os agentes propostos superaram o desempenho do DQN em tempo real nos sete jogos *Atari* avaliados. Além disso, o estudo sugere que treinar uma rede neural convolucional para aprender um classificador que mapeia observações do jogo para ações é mais eficaz do que treinar a rede para aprender uma função de regressão que mapeia observações do jogo para valores de ação.

O estudo demonstrou que os agentes de jogos propostos, que combinam algoritmos de planejamento baseados emMCTS e redes neurais convolucionais, têm um desempenho signi-

[3] são um tipo especializado de arquitetura de redes neurais projetadas principalmente para processar dados em forma de grade, como imagens

ficativamente melhor do que os agentes baseados apenas em MCTS em jogos *Atari* em tempo real. Isso sugere que o aprendizado por reforço baseado em modelos preditivos estocásticos representa uma alternativa promissora e altamente eficiente ao aprendizado sem modelo.

2.5.4 Jogando Atari com Aprendizado Profundo por Reforço

O artigo escrito por Mnih *et al.* (2013) discute a criação de um agente de DQN com o objetivo de aprender a jogar o maior número possível de jogos *Atari 2600*. Cinco jogos são mencionados explicitamente para treinamento do agente: *Pong*, *Breakout*, *Space Invaders*, *Seaquest* e *Beam Rider*.

A plataforma utilizada para implementar os jogos *Atari 2600* é a ALE, que fornece aos agentes de aprendizado por reforço um ambiente completo. Além disso, utiliza-se de uma técnica chamada *timesteps*, que amostra aleatoriamente transições anteriores para suavizar a distribuição de treinamento.

Durante os experimentos, os agentes foram treinados em nos cinco jogos populares do *Atari* usando a mesma arquitetura de rede, algoritmo de aprendizado e configurações de hiperparâmetros. A estrutura de recompensa dos jogos foi modificada durante o treinamento, com recompensas positivas fixadas em 1, recompensas negativas fixadas em -1 e recompensas 0 inalteradas. Os agentes foram treinados por um total de 10 milhões de quadros usando o algoritmo Root Mean Squared Propagation (RMSProp), técnica de otimização baseada em gradiente usada no treinamento de redes neurais,

O desempenho dos agentes treinados usando a abordagem DQN foi comparado com outros métodos de aprendizado. A abordagem DQN superou os outros métodos em todos os cinco jogos, alcançando recompensas totais médias mais altas. O artigo introduz um novo modelo de aprendizado profundo para aprendizado por reforço que usa pixels brutos como entrada. O modelo DQN, é treinado com uma variante de Q-learning e é capaz de aprender políticas de controle diretamente a partir de entrada sensorial de alta dimensão.

2.5.5 Considerações finais obtidas do estudo dos trabalhos relacionados

Após a análise realizada, observou-se uma notável prevalência do aprendizado por reforço, seja em sua forma isolada ou combinada com outras abordagens. O aprendizado supervisionado, quando aplicado isoladamente, revelou-se sensível aos dados, impactando negativamente seu desempenho. Isso reforça a ampla exploração do Q-learning e suas variantes nos estudos revisados, justificando sua seleção para a proposta de estudo de caso.

Ademais, tornou-se evidente o impacto dos ambientes de simulação na qualidade dos resultados obtidos. Trabalhos que se utilizaram de plataformas mais abrangentes e sofisticadas alcançaram desempenhos superiores, atribuídos à riqueza e complexidade desses ambientes

mais completos. Essa constatação reforça a importância crucial de uma plataforma robusta e abrangente para experimentação e validação.

2.6 CONSIDERAÇÕES FINAIS

Com base no estudo realizado e apresentado neste capítulo, é possível destacar a AR como uma abordagem única dentro da IA devido à sua capacidade singular de aprendizado através da interação com o ambiente. Diferentemente de métodos supervisionados ou não supervisionados, onde os dados rotulados ou a validação dos resultados são fundamentais, a AR aprende a partir das consequências de suas ações, utilizando recompensas e punições para ajustar seu comportamento. Isso a torna excepcionalmente adaptável a ambientes dinâmicos e desconhecidos, como os encontrados nos jogos eletrônicos.

Para as abordagens supervisionadas a IA, que necessitam de uma grande quantidade de dados rotulados para identificar padrões, diferente do AR que permite explorar e aprender através da tentativa e erro, refinando suas estratégias continuamente. Essa característica é crucial em jogos, onde as condições e desafios podem ser imprevisíveis e variados. A capacidade de aprendizagem contínua e adaptativa é uma das principais vantagens do AR sobre outras técnicas de IA no contexto de jogos eletrônicos, onde a flexibilidade para lidar com a incerteza e a complexidade é fundamental.

Em cenários de jogos, onde decisões precisam ser tomadas em meio a incertezas e complexidades, a aplicação de técnicas de Aprendizado por Reforço (AR) capacita os agentes a aprenderem com o ambiente através de recompensas positivas ou negativas. Algoritmos como o Q-Learning demonstram alta compatibilidade com esses ambientes e se tornam uma opção viável a serem utilizados.

Por fim, o AR e suas técnicas associadas, como o Q-Learning, desempenham um papel central no progresso da IA. Sua aplicação nos jogos não apenas nos permite explorar e compreender melhor suas capacidades e limitações, mas também abre caminho para inúmeras aplicações futuras. Este estudo busca contribuir para a compreensão desses conceitos, e também impulsionar o avanço contínuo da AM e da AR, explorando o uso de suas técnicas no ambiente de jogos devido ao seu grande potencial e compatibilidade.

3 APLICANDO Q-LEARNING AO AMBIENTE DO JOGO PONG

Este estudo se concentrou na exploração, implementação e avaliação do AR, mais especificamente do algoritmo Q-Learning, no âmbito dos jogos. O objetivo principal deste estudo é desenvolver uma compreensão sobre as técnicas de AR e como elas podem ser aplicadas efetivamente em jogos. Busca-se alcançar este objetivo através da implementação de algoritmos de AR em um ambiente de jogo e a avaliação de seu desempenho.

A fim de apresentar o desenvolvimento do componente implementado, este capítulo aborda a proposta de desenvolvimento, a trajetória metodológica seguida, as ferramentas utilizadas para o desenvolvimento do modelo e os critérios de escolhas.

3.1 ESTUDO DE CASO

Em 1972, Allan Alcorn, um engenheiro eletrônico do norte da Califórnia, recebeu uma tarefa de Nolan Bushnell, o fundador da *Atari Inc.*, uma das empresas pioneiras no desenvolvimento de jogos eletrônicos. Bushnell pediu a Alcorn que criasse um jogo simples baseado no Magnavox Odyssey, apresentado na Figura 18, o primeiro console de videogame doméstico comercial da história. O objetivo era treinar as habilidades de Alcorn em engenharia de jogos. O que Bushnell não esperava era que Alcorn fosse criar um jogo tão divertido e inovador que se tornaria um dos marcos iniciais na história dos videogames: o famoso Pong (KENT, 2001).

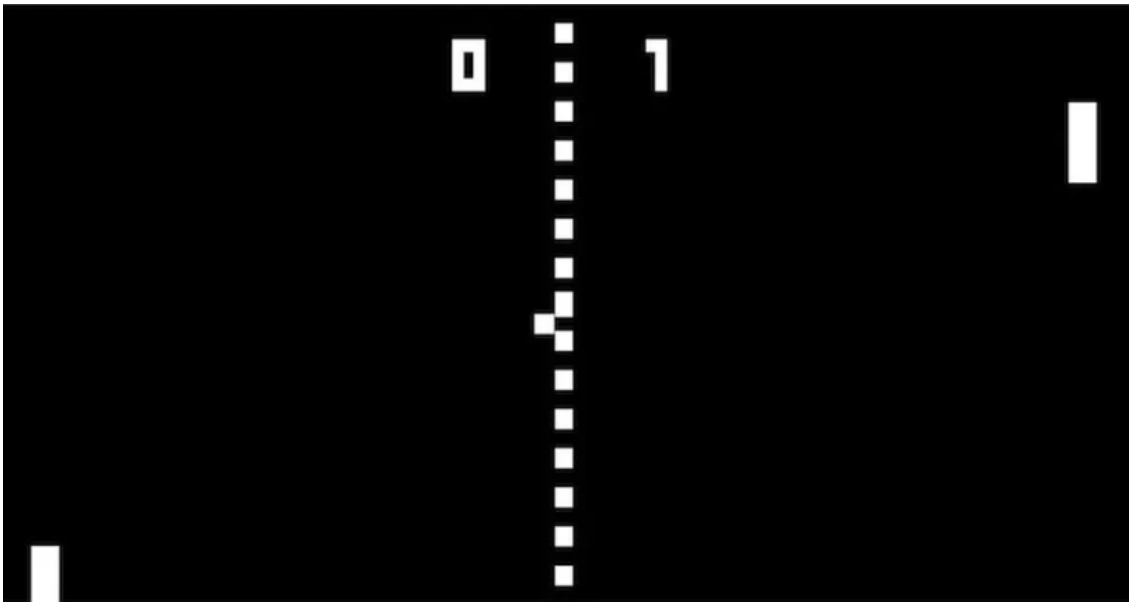
O jogo criado por Nolan consiste em dois jogadores controlando linhas brancas verticais em um fundo preto, estas linhas representam raquetes que podem ser movidas para esquerda e direita e são usadas para rebater um pequeno quadrado branco que simboliza a bola. A transmissão do jogo no protótipo acontecia através de uma televisão Hitachi em preto e branco que foi inserida em um armário de madeira com mais de um metro e meio de altura, que lembrava vagamente uma caixa de correio. O Processo de construção do protótipo levou levou quase três meses, visto que as placas de circuito ainda não estavam prontas e Alcorn teve que realizar todas as conexões eletrônicas por conta própria (KENT, 2001).

O projeto finalizado surpreendeu tanto Bushnell quanto Ted Dabney, o co-fundador da Atari e parceiro de Bushnell na época. Em vez de criar um simples exercício técnico, Alcorn havia desenvolvido um jogo que foi considerado viciante pelos jogadores e que se tornaria o produto principal da empresa. Foi nesse momento que Bushnell nomeou o jogo de Pong e introduziu algumas melhorias, como a adição de um sistema de pontuação, a utilização de moedas para poder jogar e um cartão de instruções simples com a mensagem: “Evite perder a bola para obter uma pontuação alta.” Para avaliar a viabilidade de mercado do jogo, Bushnell e Alcorn o instalaram em uma localização ao longo da rota dos fliperamas da Atari. O sucesso foi imediato e o jogo atraiu tantos jogadores que a máquina ficou cheia de moedas em pouco

Figura 18 – Magnavox Odyssey

Fonte: Retirado de <https://br.pinterest.com/pin/312155817893276083/>

Figura 19 – Pong



Fonte: Retirado de <https://www.techtudo.com.br/noticias/2016/03/conheca-pong-o-primeiro-videogame-lucrativo-da-historia.ghtml>

tempo e parou de funcionar, umas das primeiras versões da máquina pode ser vista na Figura 20. A Atari logo começou a produzir e distribuir o Pong em larga escala, iniciando uma nova era nos jogos eletrônicos (KENT, 2001).

O sucesso do Pong foi além do que qualquer um poderia ter imaginado. O jogo não apenas se tornou um sucesso instantâneo nos fliperamas, mas também desencadeou uma revolução na indústria dos videogames. A simplicidade do jogo, combinada com a sua jogabilidade competitiva, atraiu uma ampla gama de jogadores, muitos dos quais nunca haviam jogado um

videogame antes (KENT, 2001), a Figura 19 mostra como era a tela da versão original do PONG feita por Allan Alcorn.

Figura 20 – Gabinete de Pong

Fonte: Retirado de <https://pt.wikipedia.org/wiki/Pong>

O Pong gerou lucros significativos para a Atari na época e estabeleceu um novo padrão para os videogames. Ele demonstrou que os videogames poderiam ser uma forma de entretenimento popular e lucrativa, abrindo caminho para o desenvolvimento de uma indústria multibilionária. Além disso, o Pong ajudou a popularizar os fliperamas, que se tornaram um ponto de encontro social para jovens em todo o mundo (KENT, 2001). O impacto do Pong na indústria dos videogames não pode ser subestimado. Ele foi um dos primeiros jogos a alcançar um sucesso comercial significativo, e seu design simples mas envolvente se tornou um modelo para muitos jogos que se seguiram. O Pong transformou a Atari em uma das principais empresas de videogames do mundo, e também ajudou a moldar a indústria dos videogames como a conhecemos

hoje (KENT, 2001).

3.2 PERCURSO METODOLÓGICO

Esse trabalho é uma pesquisa de natureza exploratória, que busca compreender e aplicar técnicas de AR no contexto dos jogos. A escolha por um estudo exploratório é justificada pela natureza inovadora e dinâmica do campo de AR em jogos. A pesquisa exploratória permite uma abordagem flexível que pode se adaptar à medida que o conhecimento na área se desenvolve. Ela é realizada de modo que o pesquisador se torne mais próximo do mundo do seu objeto de pesquisa e forneça informações, além de orientar a elaboração das hipóteses do estudo em questão (ACADêMICO, 2018).

A metodologia adotada envolve várias etapas. Primeiro, uma revisão da literatura foi realizada para entender o estado atual do campo de AR em jogos. Em seguida, o algoritmo de AR foi implementado em um ambiente de jogo usando a plataforma *Gymnasium*. O desempenho desse algoritmo foi então avaliado usando várias métricas, como a pontuação final do jogo, o número de vitórias e o tempo necessário para completar o jogo. Sendo assim, o método seguido no trabalho compreendeu cinco etapas:

- Etapa 1: seleção da plataforma de desenvolvimento
- Etapa 2: definição e detalhamento do jogo a ser implementado
- Etapa 3: implementação do jogo com *Gymnasium*
- Etapa 4: avaliação dos resultados obtidos
- Etapa 5: comparação com outros trabalhos

3.2.1 Etapa 1: seleção da plataforma de desenvolvimento

A escolha das ferramentas é de grande importância para o sucesso do projeto e a eficiência de seus experimentos. A habilidade de selecionar um ambiente de desenvolvimento, linguagem e bibliotecas tem um impacto direto no desenvolvimento, nos resultados obtidos e no tempo de execução do experimento (ANDRIJAUSKAS, 2020). Para atender a esses requisitos, foram escolhidas as seguintes ferramentas: Google Colab, Gymnasium e a linguagem Python.

O Google Colaboratory (figura 21), também conhecido como Google Colab, é um ambiente interativo onde você pode criar e compartilhar notebooks com código, texto, imagens e muito mais, permitindo a escrita e execução diretamente no navegador, sem necessidade de qualquer configuração (GOOGLE, 2024). O Google Colab permite a execução de testes em qualquer computador, independentemente de sua configuração. Isso é particularmente útil, considerando que o treinamento de modelos de IA podem levar horas. A capacidade de iniciar o

treinamento em um computador e monitorar o progresso em outro local oferece uma grande flexibilidade, facilitando a experimentação e otimizando o tempo. Uma característica importante é que o Google Colab fornece ferramentas de integração com o GitHub para versionamento e repositório, é compatível com a linguagem python e fornece versões de acordo com a necessidade do usuário.

Figura 21 – Ambiente de desenvolvimento do O Google Colab.

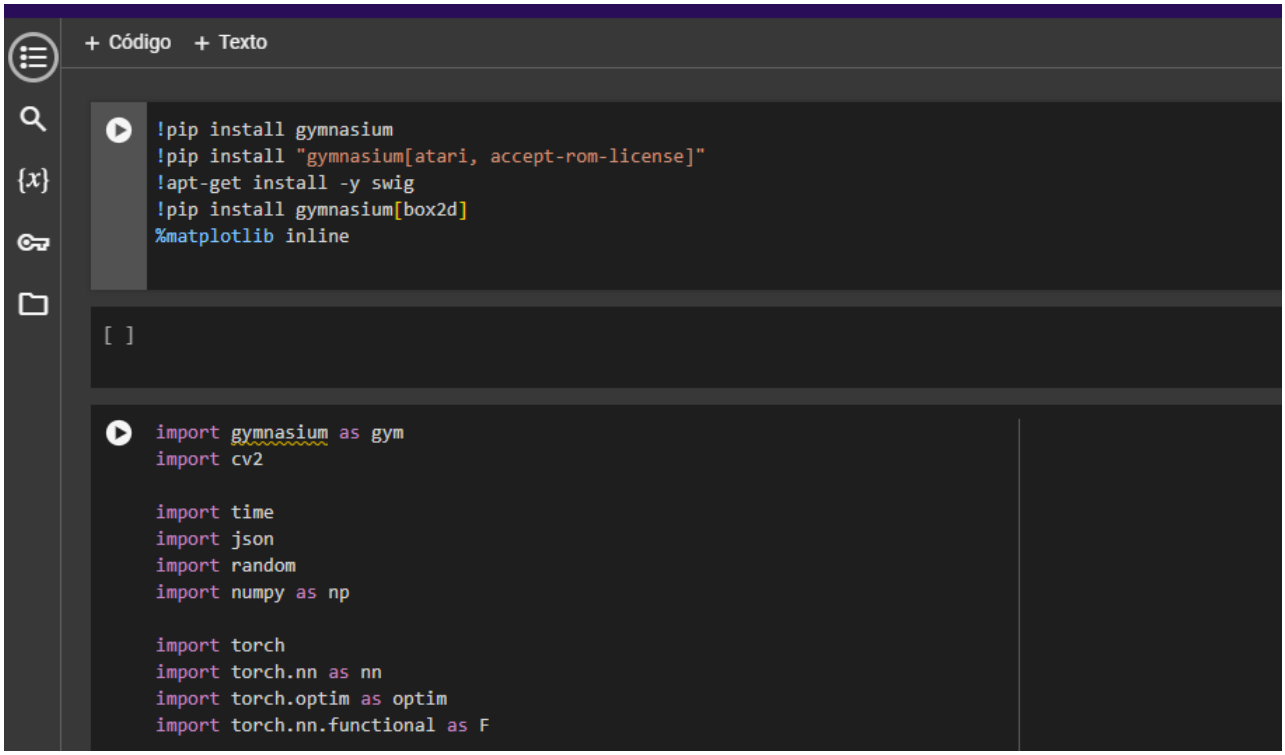


Figura 22 – Fonte: Criado pelo autor

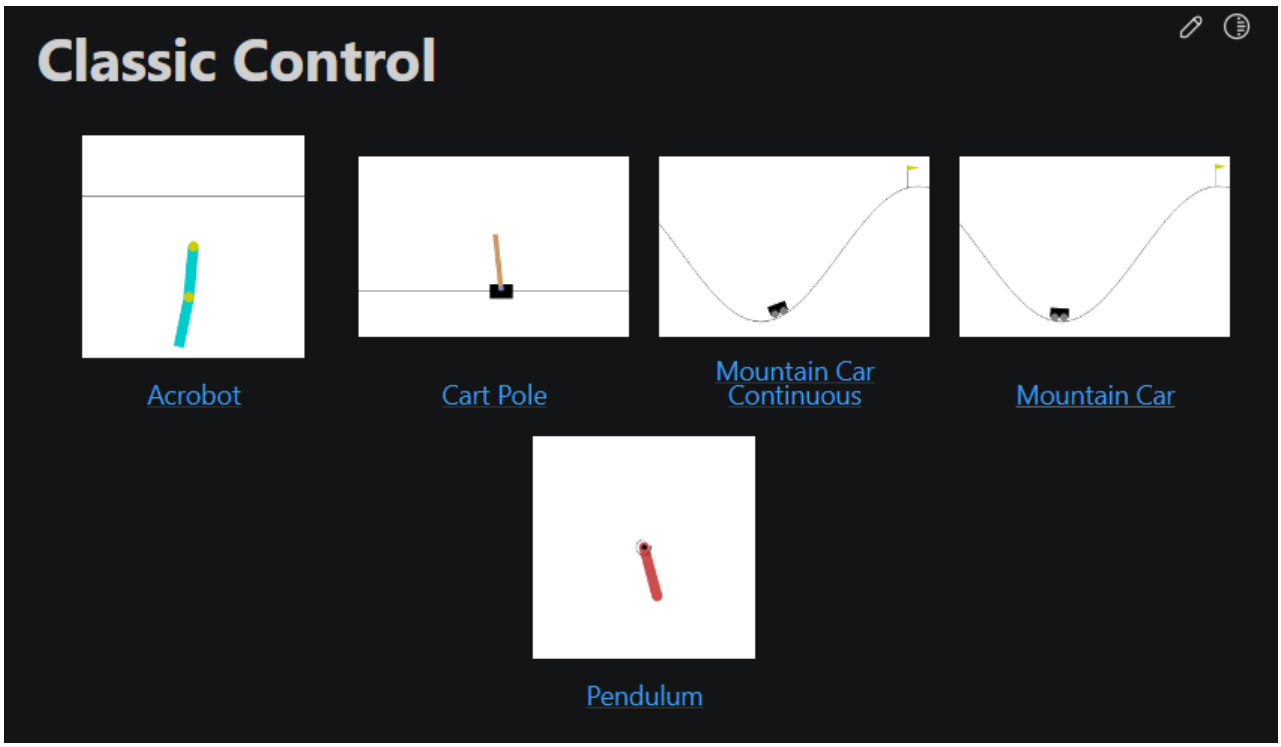
Outra ferramente importante é a Gymnasium, uma biblioteca que disponibiliza uma Application Programming Interface (API) que utiliza a linguagem Python, amplamente utilizada para IA e AR, possuindo uma vasta gama de bibliotecas e frameworks capazem de representar problemas de AR (LISBOA, 2021). A figura 23 ilustra uma parte da coleção de ambientes de jogos disponíveis na Gymnasium.

O Gymnasium oferece uma ampla variedade de ambientes de simulação, desde jogos simples até cenários complexos, todos projetados especificamente para o desenvolvimento e teste de agentes de aprendizado por reforço. Isso inclui implementações de ambientes comuns, como CartPole, Pendulum, Mountain Car, MuJoCo, Atari e muitos outros apresentados na Figura 24 gymnasium2023. Além disso, Gymnasium padroniza o espaço de trabalho e a interface dos ambientes, permitindo a comparação de resultados entre diferentes algoritmos e estudos. Também é compatível com outras ferramentas e frameworks para o aprendizado de máquina, como TensorFlow, PyTorch, Keras, Scikit-learn, entre outros. Isso a torna uma escolha versátil e poderosa para projetos de AR gymnasium2023.

Figura 23 – Amostra da coleção de ambientes de jogos disponíveis na Gymnasium.

Fonte: Retirado de https://gymnasium.farama.org/environments/classic$_{c}ontrol/$

Figura 24 – Ambientes da Gymnasium, mostrando Acrobot, CartPole, Mountain Car, Continuous Mountain Car, e Pendulum respectivamente.



Fonte: Retirado de https://opendatascience.com/up-your-game-with-openai-gym-reinforcement-learning/

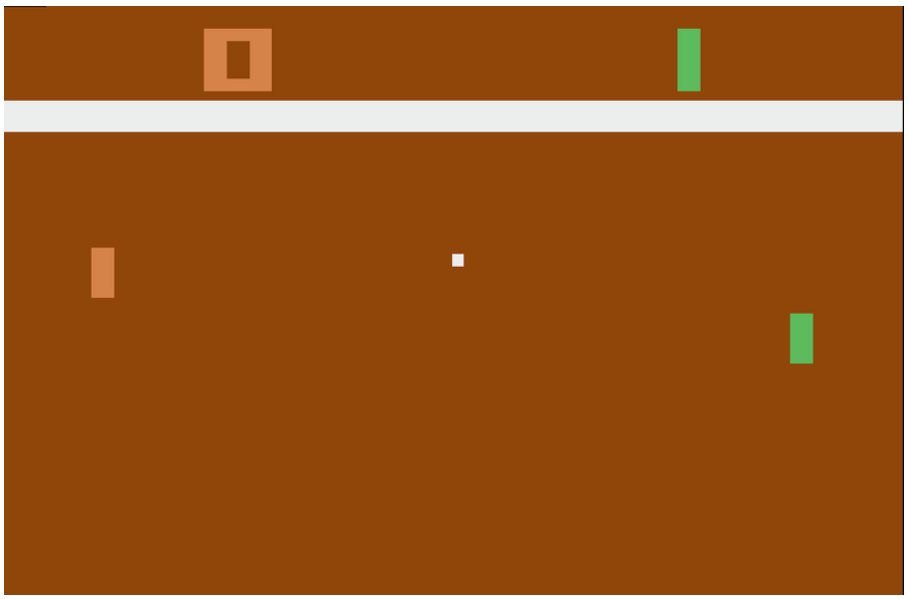
3.2.2 Etapa 2: definição e detalhamento do jogo a ser implementado

O jogo Pong, é um jogo simples, competitivo e desafiador, apreciado por todas as idades desde de seu lançamento. O jogo simula uma partida de tênis com dois retângulos representando as raquetes e um quadrado representando a bola. Cada raquete tem como objetivo impedir que a

bola passe por ela, rebatendo-a de volta para o campo do adversário. A bola ricocheteia quando atinge as bordas superior e inferior da tela, outro aspecto interessante do jogo é que o ângulo de rebatida da bola muda dependendo de qual parte da raquete a bola atinge, adicionando um elemento de imprevisibilidade e desafio ao jogo. Se a bola passar pela raquete do adversário e atingir a linha que a raquete está protegendo, o jogador marca um ponto. O jogo normalmente continua até que um dos jogadores alcance uma pontuação de 21 pontos, momento em que o jogo termina e o jogador com a maior pontuação é declarado o vencedor (KENT, 2001).

Com regras simples, o Pong oferece uma experiência de jogo emocionante e desafiadora que mantém os jogadores engajados e sempre buscando melhorar suas habilidades. É um exemplo perfeito de como um design de jogo simples pode resultar em uma experiência de jogo profundamente gratificante e divertida.

Figura 25 – Tela do jogo Pong disponibilizado pela Gymnasium.



Fonte: Retirado de https://gymnasium.farama.org/environments/atari/pong/pong

3.2.3 Etapa 3: implementação do jogo com *Gymnasium*

A *Gymnasium* é uma biblioteca abrangente que oferece uma variedade de recursos para facilitar o desenvolvimento e a comparação de algoritmos de AR. Ela contém uma série de bibliotecas, classes e métodos projetados para fornecer as ferramentas necessárias para a implementação de soluções de AR. Somente o que é utilizado na implementação deste trabalho é abordado, alguns termos, conceitos e informações que são utilizados precisam ser explicados.

- **action_space**: Refere-se ao ambiente virtual onde o agente pode observar o jogo. Podem ser definidos como:
  - *ram*: Retorna os 128 Bytes de RAM.

  - *rgb*: Retorna uma renderização RGB do jogo.
  - *grayscale*: Retorna uma renderização em escala de cinza do jogo.

- **frameskip** (*int* ou uma tupla de dois *ints*): Controla o frame skipping estocástico.
- **repeat_action_probability** (*float*): A probabilidade de que uma ação seja repetida, também chamada de sticky actions.

Para os ambientes do tipo Atari, uma lista de ações é disponibilizada após a criação do ambiente, o conjunto de ações disponíveis depende do jogo, a lista completa de ações para o Pong são descritas na tabela 1.

Valor	**Significado**	**Valor**	**Significado**
0	NOOP	1	FIRE
2	RIGHT	3	LEFT
4	RIGHTFIRE	5	LEFTFIRE

Tabela 1 – Ações disponíveis para o Pong

Dentre os métodos e propriedades do ambiente da *Gymnasium* destaca-se:

- *make*: Esta função cria um ambiente de AR. O id do ambiente desejado é passado como argumento e ela retorna uma instância do ambiente.
- *reset*: Esta função reinicia o estado do ambiente para um estado inicial e retorna a primeira observação. O retorno é um objeto que representa o estado inicial do ambiente.
- *step*: Esta função executa uma ação no ambiente. O retorno é uma tupla que contém a nova observação, a recompensa, um booleano indicando se o episódio terminou e um dicionário com informações adicionais.
- *action_space*: Esta é uma propriedade do ambiente que representa o espaço de todas as ações possíveis que um agente pode tomar em um determinado estado. O retorno é um objeto que descreve o formato e os limites das ações que um agente pode tomar.

Conforme demonstra o Algoritmo 1, a utilização básica da *Gymnasium* funciona como em um MDP, em um ciclo de ação-recompensa, apresentado na Figura 26. O ambiente é criado (linha 2). Um loop é iniciado (linha 5), neste loop um agente observa o estado do ambiente e toma uma ação (linha 6), e então recebe uma recompensa e observa o novo estado (linha 7). Este processo se repete até que um estado terminal seja alcançado (linha 9) ou o ambiente é reiniciado (linha 10) .

Algoritmo 1 – Criando loop ambiente-agente no Gymnasium

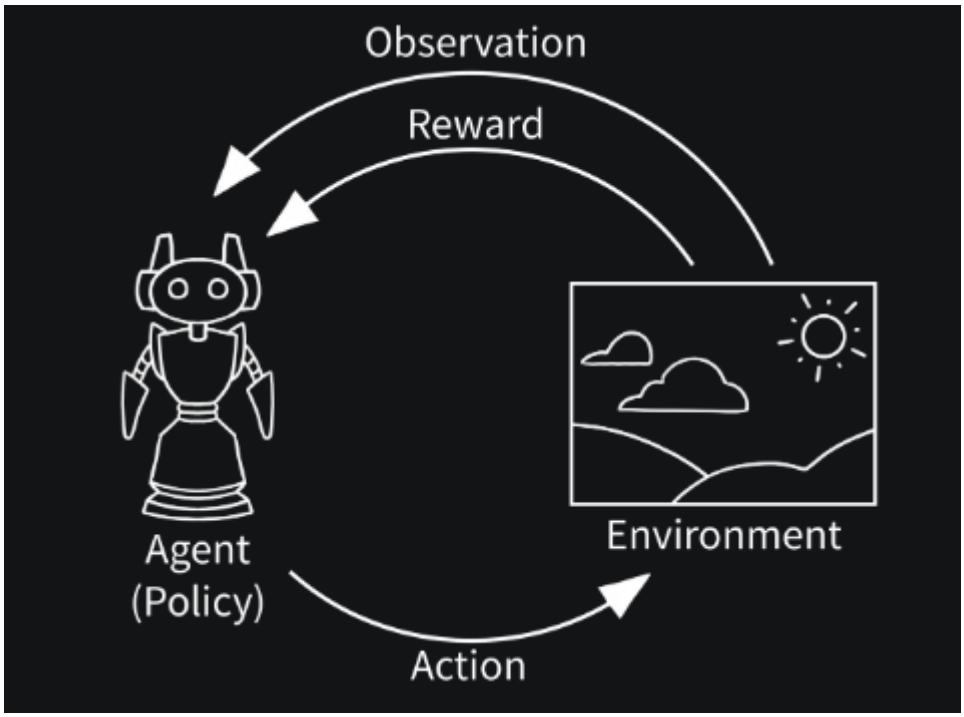
```
import gymnasium as gym
env = gym.make("Pong-v2", render_mode="human")
observation, info = env.reset()

for _ in range(1000):
    action = env.action_space.sample()
    info = env.step(action)

    if terminated or truncated:
        observation, info = env.reset()

env.close()
```

Figura 26 – Digrama agente-ambiente



Fonte: Retirado de https://gymnasium.farama.org/content/basic_usage/

Na implementação deste trabalho foi selecionado o id *PongDeterministic-v4* que possui o espaço de observação *RGB*, com o *frameskip* de 4 e o *repeat_action_probability* igual a 0, o que significa que o controle de aleatoriedade não é controlado pelo ambiente da *Gymnasium*.

A configuração do ambiente da *Gymnasium* é de grande importância, pois define as condições nas quais o agente de AR interage e realiza suas ações. Além disso, a configuração do ambiente pode influenciar significativamente o desempenho e a eficiência do algoritmo de AR. Portanto, projetar e ajustar adequadamente o ambiente da *Gymnasium* é crucial para maximizar o potencial de aprendizagem e melhorar sua capacidade de resolver problemas complexos em

tempo real. Para mais informações sobre a *Gymnasium* e seus recursos e como utilizá-los, é indicado consultar a documentação oficial disponível Foundation (2023).

3.2.4 Preparação do ambiente

No desenvolvimento deste projeto, tornou-se crucial a utilização da versão *Google Colab PRO*. A versão gratuita do *Google Colab* já oferece um ambiente de execução robusto com uma quantidade razoável de memória RAM e acesso a Graphics Processing Unit (GPU), mas as demandas computacionais e a complexidade do treinamento do modelo de IA exigiram recursos adicionais. A versão PRO disponibiliza maior quantidade de memória RAM e acesso a um maior número de unidades de processamento que dão acesso a uma GPU. Isso permitiu a utilização eficiente de núcleos tensores[1], possibilitando o processamento paralelo dos dados e acelerando significativamente o tempo de treinamento. Vale ressaltar que, apesar da necessidade de uma GPU, não é necessário um modelo extremamente potente, pois o treinamento precisa aguardar a resposta do ambiente da *gymnasium*.

Existe uma série de bibliotecas que são necessárias para o desenvolvimento, para adicionalas ao projeto, é utilizado o comando *!pip install*. Este comando é usado para instalar bibliotecas *Python* e utiliza o gerenciador de pacotes *pip*, que é uma ferramenta padrão para instalação e gerenciamento de pacotes em *Python*. As bibliotecas necessárias são listadas abaixo, o Algoritmo 2 refere-se ao código utilizado para instalação e importação destas bibliotecas:

- Gym: Biblioteca desenvolvida pela OpenAI que fornece uma variedade de ambientes de simulação para treinar algoritmos de aprendizado de máquina.
- Atari: Extensão para a Gym, necessária para trabalhar com ambientes de jogos Atari clássicos.
- WIG: Uma ferramenta que facilita a integração de código em C/C++ com outras linguagens de programação, como Python, utilizado pela biblioteca Box2D.
- Box2D: Extensão para a Gym, necessária para trabalhar com ambientes que utilizam a física do Box2D, um motor de física 2D.
- Matplotlib: Uma biblioteca de visualização de dados em Python, comumente utilizada para plotar gráficos e visualizações.

Algoritmo 2 – Instalação e importação das biblitecas necessárias

```
1
2 !pip install gymnasium
```

[1] Núcleos tensores são estruturas matemáticas usadas para definir operações entre espaços vetoriais, essenciais na teoria dos tensores para generalizar conceitos como derivadas e integração em espaços de dimensões superiores, fundamentais também na física para descrever interações complexas entre sistemas físicos.

```
!pip install "gymnasium[atari,accept-rom-license]"
!apt-get install -y swig
!pip install gymnasium[box2d]

import gymnasium as gym
import cv2
import time
import json
import random
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from collections import deque
```

Para implementar e gerenciar o algoritmo de forma eficaz, é necessário definir um conjunto de variáveis usadas para sua configuração. Isso permite que o algoritmo apresente comportamentos diferentes conforme os valores atribuídos a essas variáveis de controle.

- **ENVIRONMENT**: String utilizada para informar o ID que será utilizado para criação do ambiente *Gymnasium*.
- **DEVICE**: Variável que armazena o dispositivo selecionado (GPU ou CPU) para uso posterior no código. Isso direciona os cálculos da rede neural para o dispositivo apropriado, garantindo uma execução eficiente.
- **SAVE_MODELS**: Booleano que indica se o modelo em treinamento deve ser salvo.
- **MODEL_PATH**: String que contém o diretório onde os modelos treinados serão salvos ou de onde os modelos pré-treinados serão carregados.
- **LOAD_MODEL_FROM_FILE**: Booleano que indica se um modelo previamente treinado deve ser carregado.
- **TRAIN_MODEL**: Booleano que indica se o modelo deve ser treinado.
- **BATCH_SIZE**: Inteiro que define o tamanho do lote para o treinamento.
- **MAX_EPISODE**: Inteiro que define o número máximo de episódios para o treinamento.
- **MAX_STEP**: Inteiro que indica o número máximo de passos em cada episódio.
- **MAX_MEMORY_LEN**: Inteiro que define o tamanho máximo da memória de repetição, ou seja, o número máximo de experiências que podem ser armazenadas na memória.

- **MIN_MEMORY_LEN**: Inteiro que define o número mínimo de experiências que deve ser acumulada antes de começar o treinamento.
- **GAMMA**: Float que define a taxa de desconto usada no cálculo da recompensa futura.
- **ALPHA**: Float que define a taxa de aprendizado do algoritmo.
- **EPSILON_DECAY**: Float que define a taxa de decaimento de épsilon por episódio.
- **RENDER_GAME_WINDOW**: Booleano que indica se a tela do jogo deve ser renderizada.

Algoritmo 3 – Declaração e atribuição de valores para as váriaveis

```
1
2  ENVIRONMENT = "PongDeterministic-v4"
3  RENDER_GAME_WINDOW = True
4
5  TRAIN_MODEL = False
6  BATCH_SIZE = 64
7  MAX_EPISODE = 901
8  MAX_STEP = 100000
9  GAMMA = 0.97
10 ALPHA = 0.0005
11 EPSILON_DECAY = 0.99
12
13 DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
14
15 MAX_MEMORY_LEN = 50000
16 MIN_MEMORY_LEN = 40000
17
18 MODEL_PATH = "./pong-"
19 LOAD_MODEL_FROM_FILE = True
20 LOAD_FILE_EPISODE = 900
```

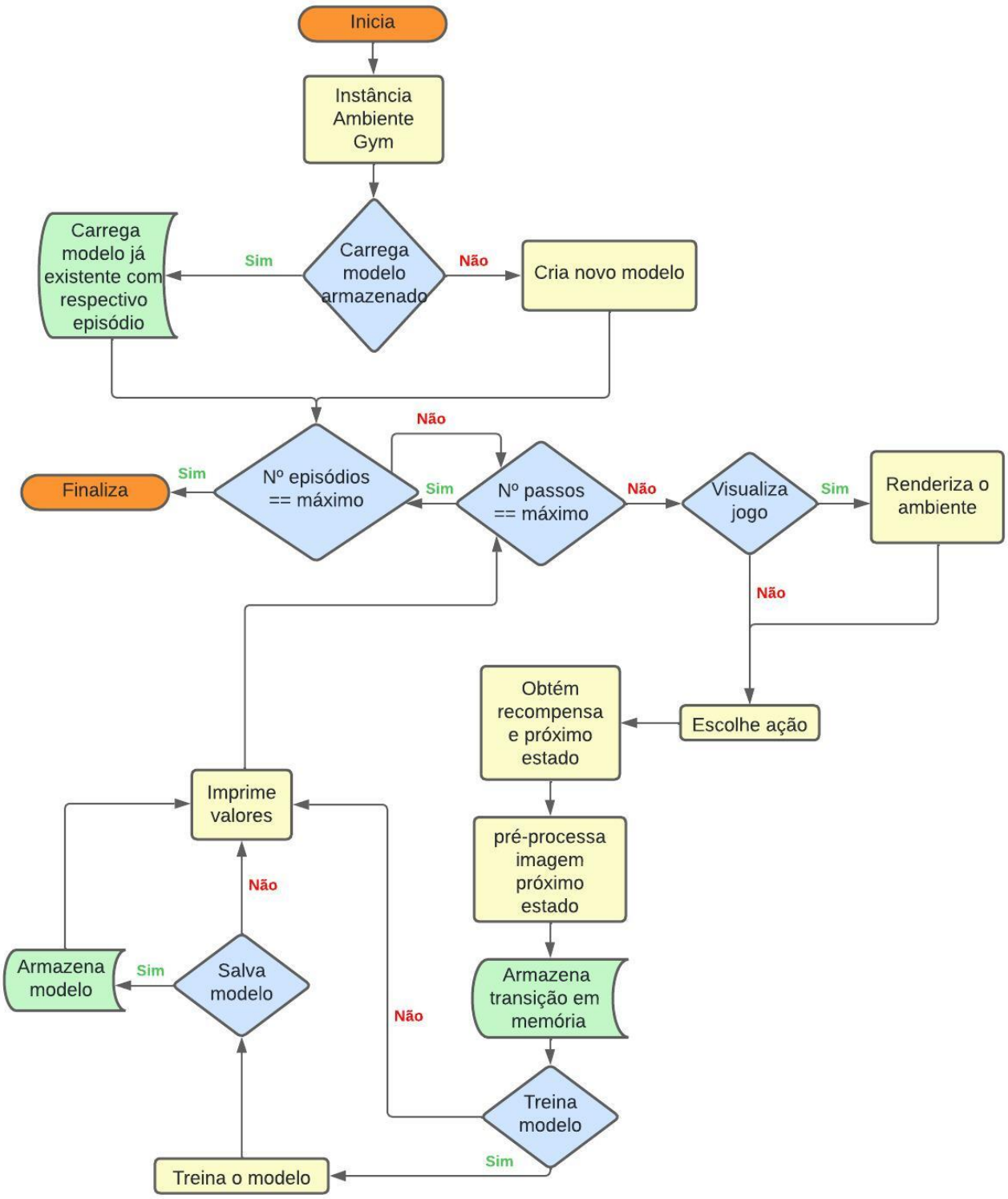
3.2.5 Desenvolvimento do modelo

Na criação e treinamento de um modelo de IA que emprega técnicas de AR para jogar Pong foi Inicialmente proposto a utilização exclusiva do Q-learning. No entanto, sua limitação em lidar com espaços de ação discretos e de alta dimensionalidade, como no Pong, tornou-se evidente. A construção de uma tabela Q, onde cada entrada representa o valor de uma ação para um par estado-ação específico, torna-se impraticável em ambientes complexos devido ao grande número de estados e ações possíveis.

Para superar a alta dimensionalidade foi utilizado o DDQN, uma extensão do DQN que utiliza redes neurais profundas para aproximar a função Q. Ela permite que o agente aprenda

diretamente a partir de dados brutos de entrada, como imagens de jogos. O fluxo do algoritmo pode ser visto na Figura 27.

Figura 27 – Fluxograma do algoritmo Deep-Q-Network

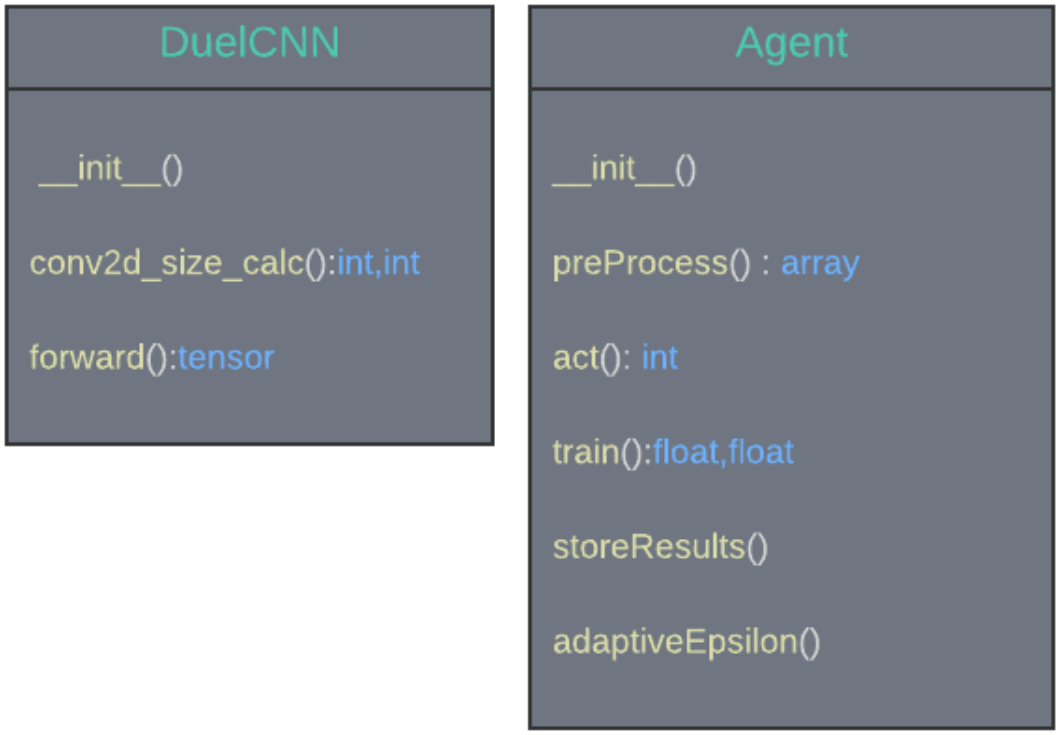


Fonte: O autor(2024)

Duas classes principais são utilizadas na implementação deste projeto: a classe *Du-*

elCNN e a classe *Agent*. Ambas as classes, juntamente com seus respectivos métodos, são ilustradas na Figura 28.

Figura 28 – Diagrama de classes



Fonte: O autor(2024)

A classe *DuelCNN* é uma implementação da arquitetura de Rede Neural Convolucional de Duelo. Ela herda de nn.Module, que é a classe base para todos os módulos de rede neural no PyTorch. Descreve-se, a seguir, cada um dos métodos desta classe:

- *__init__(self, h, w, output_size)*: Este é o construtor da classe. Ele inicializa a rede neural com três camadas convolucionais e duas camadas lineares para cada uma das funções de vantagem e valor do estado. As dimensões da imagem de entrada e o número de ações possíveis são passados como argumentos.
- *conv2d_size_calc(self, w, h, kernel_size=5, stride=2)*: Este método calcula o tamanho da saída de uma camada convolucional com base no tamanho da entrada, no tamanho do kernel e no passo. Ele é usado durante a inicialização para calcular o tamanho da entrada para as camadas lineares.
- *forward(self, x)*: Este método implementa a passagem para frente através da rede. Ele recebe um tensor de entrada x, passa-o através das camadas convolucionais e das camadas

lineares para as funções de vantagem e valor do estado, e retorna um tensor de valores Q para todas as ações possíveis.

A implementação completa da classe *DuelCNN* pode ser encontrada no Algoritmo 4 (Apêndice A).

A classe *Agent* é responsável pela implementação do agente de aprendizado por reforço. Detalha-se, a seguir, os seus métodos:

- *__init__(self, environment)*: Este é o construtor da classe. Ele inicializa o agente com o ambiente de jogo, define os tamanhos de estado e ação com base no ambiente, inicializa os parâmetros para o pré-processamento de imagens, define os hiper-parâmetros para o aprendizado por reforço, cria a memória de repetição, inicializa os modelos online e alvo para o algoritmo DDQN e define o otimizador.
- *preProcess(self, image)*: Este método processa as imagens antes de serem alimentadas na rede neural. Ele converte a imagem para escala de cinza, corta a imagem para remover partes desnecessárias, redimensiona a imagem para o tamanho alvo e normaliza a imagem.
- *act(self, state)*: Este método decide qual ação o agente deve tomar com base no estado atual. Ele usa uma abordagem de exploração versus exploração, onde o agente escolhe aleatoriamente uma ação com probabilidade epsilon (exploração) e escolhe a ação que maximiza o valor Q com probabilidade 1 - epsilon (exploração).
- *train(self)*: Este método treina a rede neural usando a memória de repetição. Ele amostra um lote de transições da memória, calcula os valores Q para os estados atuais e próximos, calcula o valor Q alvo usando a equação de Bellman, calcula a perda como a diferença quadrática entre o valor Q selecionado e o valor Q esperado, e atualiza os pesos da rede para minimizar a perda.
- *storeResults(self, state, action, reward, nextState, done)*: Este método armazena os resultados de uma etapa do ambiente na memória de repetição.
- adaptiveEpsilon(self): Este método atualiza o valor de epsilon após cada etapa. À medida que o agente ganha mais experiência, epsilon é reduzido para que o agente explore menos e explore mais.

A implementação completa da classe *Agent* pode ser encontrada no algoritmo 5 (Apêndice A).

É importante como funciona o treinamento das redes neurais. O DDQN utiliza redes neurais convolucionais para aprender a função Q, que estima os valores de ação para um dado

estado em um ambiente de aprendizado por reforço. A implementação do O DDQN consiste em duas principais componentes neurais: a Rede de Valor (*Value Network*) e a Rede de Vantagem (*Advantage Network*), que são combinadas para prever os valores Q.

A estrutura da rede neural utilizada neste algoritmo é definida pela classe DuelCNN, que herda as funcionalidades da biblioteca PyTorch. A DuelCNN é inicializada com três camadas convolucionais seguidas de camadas totalmente conectadas, projetadas para processar as entradas do ambiente e gerar saídas que representam os valores Q esperados para cada ação possível.

- **Camadas Convolucionais:** As primeiras três camadas (conv1, conv2 e conv3) são convolucionais, projetadas para extrair características importantes das imagens de entrada. Cada camada aplica filtros para capturar padrões espaciais nas imagens, reduzindo gradualmente a dimensionalidade das representações.
- **Normalização por Lotes (*Batch Normalization*):** Após cada camada convolucional, a normalização por lotes (bn1, bn2 e bn3) normaliza as ativações das camadas anteriores, melhorando a estabilidade e acelerando o treinamento da rede.
- **Camadas Lineares:** Após a etapa de convolução, os dados são achatados (*flattened*) e passam por duas redes totalmente conectadas separadas:
  - **Rede de Valor (V):** A primeira rede linear (Vlinear1) recebe a entrada achatada e gera representações do valor de estado (*state value*). A segunda camada (Vlinear2) produz um único valor de vantagem (advantage value).
  - **Rede de Vantagem (A):** A segunda rede linear (Alinear1) também recebe a entrada achatada e gera representações da vantagem potencial de cada ação. A camada final (Alinear2) produz os valores Q para cada ação possível.
- **Função de Ativação:** Entre as camadas lineares, a função de ativação Leaky ReLU (Alrelu e Vlrelu) é aplicada para introduzir não linearidades, permitindo que a rede aprenda relações complexas entre as características extraídas.

Durante a execução, imagens do ambiente são pré-processadas pelo agente (`Agent`), que as converte para escala de cinza, redimensiona para um tamanho padrão e aplica uma sequência de quatro imagens para capturar informações temporais. Essas imagens são então alimentadas na rede DuelCNN para prever os valores Q associados a cada ação.

Durante o treinamento, a rede DuelCNN é otimizada usando o algoritmo Adam (KINGMA; BA, 2015), que ajusta os pesos das conexões para minimizar a diferença entre os valores Q preditos e os valores Q esperados, calculados usando a fórmula de *Bellman*. Isso permite que a rede aprenda a melhor política de ação para maximizar recompensas futuras no ambiente de aprendizado por reforço.

O fluxo do programa inicia na função *main*, que atua como o ponto de partida. O fluxo específico que é seguido durante a execução detalhado, pode ser visualizado na Figura 27.

1. O ambiente do jogo é inicializado e o agente é criado.
2. Se um modelo pré-treinado estiver disponível, o agente carrega os pesos desse modelo e continua o treinamento a partir do episódio onde o modelo parou. Caso contrário, o treinamento começa do zero.
3. Para cada episódio até o máximo número de episódios:
    - O tempo de início do episódio é registrado e o ambiente é reiniciado.
    - O estado inicial é pré-processado e empilhado para formar o estado inicial completo.
    - Para cada passo até o máximo número de passos:
        - Se a opção de renderização estiver ativada, o estado atual do ambiente é renderizado visualmente.
        - O agente escolhe uma ação com base no estado atual e executa essa ação no ambiente.
        - O agente recebe o próximo estado, a recompensa, e a informação se o episódio terminou.
        - O próximo estado é pré-processado e empilhado com os estados anteriores para formar o próximo estado completo.
        - A transição (estado, ação, recompensa, próximo estado, feito) é armazenada na memória de repetição do agente.
        - O estado é atualizado para o próximo estado.
        - Se a opção de treinamento estiver ativada, o agente treina o modelo online usando um lote de transições da memória de repetição.
        - A cada 1000 passos, o valor de epsilon do agente é adaptativamente diminuído.
        - Se o episódio terminou, o agente registra as estatísticas do episódio, salva o modelo se necessário e passa para o próximo episódio.

A implementação integral da função *Main* está disponível para consulta no Algoritmo 6 (Apêndice A).

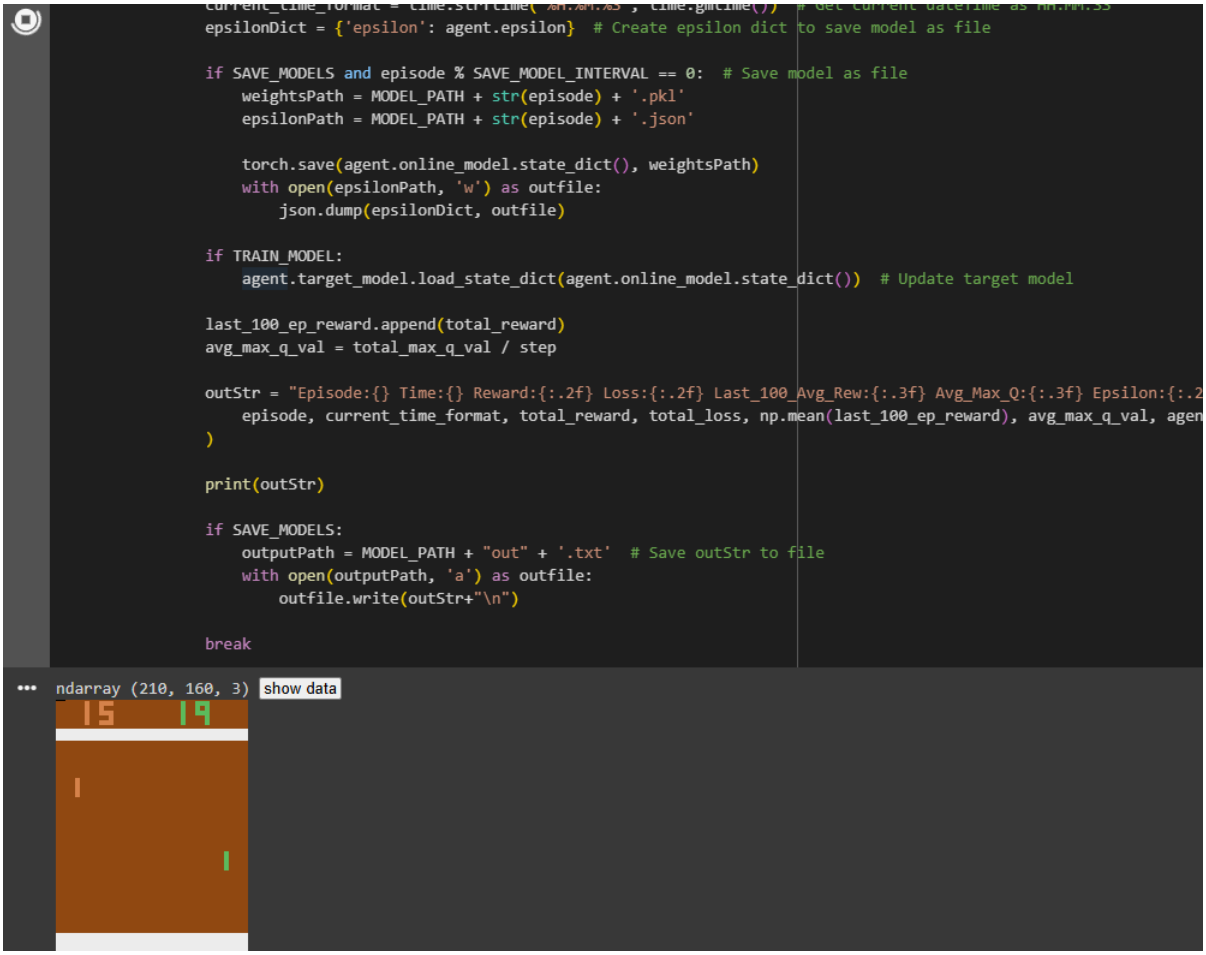
A etapa inicial consistiu na validação da biblioteca *Gymnasium*. Testes foram conduzidos para assegurar que os métodos disponíveis estavam operando corretamente e que os retornos correspondiam ao esperado. Dessa maneira, adquiriu-se um entendimento mais aprofundado da *Gymnasium* e de seu funcionamento.

Posteriormente, foram examinadas implementações de outros autores. Isso auxiliou na definição de um ponto de partida para os valores dos hiperparâmetros, tais como a taxa de

aprendizado, o fator de desconto e a taxa de decaimento do epsilon. A partir desses valores iniciais, uma série de experimentos foi realizada para validar e ajustar os hiperparâmetros, visando otimizar o desempenho do agente.

Durante a execução dos testes, o desempenho do agente é calculado e os dados são exibidos em tempo real na tela, permitindo o acompanhamento do treinamento. A Figura 30 apresenta uma série de informações por episódio. Caso um experimento falhasse ou produzisse resultados inconsistentes ou insatisfatórios, os dados eram analisados, os hiperparâmetros ajustados e os testes repetidos, possibilitando a identificação da melhor combinação. A visualização do jogo pela implementação pode ser vista na Figura 29.

Figura 29 – Visualização do jogo pela implementação

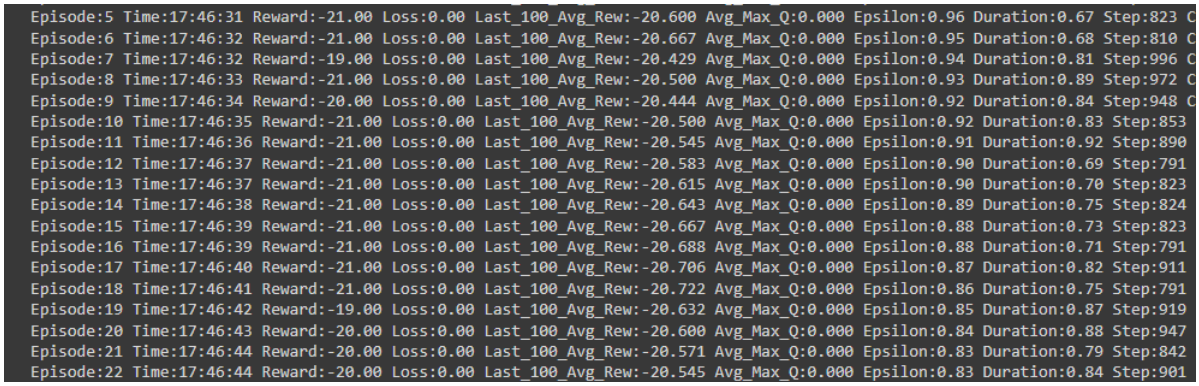


Fonte: O autor(2024)

Adicionalmente, foram conduzidos testes de jogabilidade, treinando o agente em diversas rodadas para verificar a coesão de todas as características e funcionalidades aprendidas. Isso permitiu observar como o agente aplicava o que havia aprendido em um ambiente de jogo real. A Tabela 2 apresenta combinações entre os parâmetros utilizados na implementação.

BATCH_SIZE: O tamanho do lote, definido como 64, é um valor comumente usado em implementações de aprendizado profundo. Este tamanho de lote oferece um equilíbrio entre

Figura 30 – Informações por episódio exibidas em tempo real durante o treinamento



Fonte: O autor(2024)

Variável	Valor 1	Valor 2	Valor 3	Valor 4	Valor 5	Melhores resultados
BATCH_SIZE	64	32	128	64	64	64
MAX_EPISODE	400000	300000	200000	100000	50000	100000
MAX_STEP	100000	50000	200000	100000	100000	100000
MAX_MEMORY_LEN	50000	40000	60000	30000	45000	50000
MIN_MEMORY_LEN	30000	20000	40000	10000	30000	40000
GAMMA	0.99	0.98	0.97	0.96	0.95	0.97
ALPHA	0.0005	0.0001	0.001	0.00025	0.00015	0.0001
EPSILON_DECAY	0.99	0.95	0.90	0.97	0.98	0.99

Tabela 2 – Valores das variáveis

eficiência computacional (permitindo o processamento de mais lotes em paralelo) e estabilidade de aprendizado (um lote maior fornece uma estimativa mais precisa do gradiente).

MAX_EPISODE e MAX_STEP: Os valores selecionados para esses parâmetros foram determinados com base no ponto em que a variação no aprendizado do agente se tornava insignificante, indicando a convergência do algoritmo. Isso sugere que o agente já havia aprendido uma política eficaz e que episódios ou passos adicionais não resultariam em melhorias significativas.

MAX_MEMORY_LEN e MIN_MEMORY_LEN: O parâmetro `MAX_MEMORY_LEN` estabelece o tamanho máximo da memória de repetição, isto é, o número máximo de experiências que podem ser armazenadas na memória. Seu valor foi ajustado e aumentado progressivamente até que não houvesse progresso que justificasse um aumento adicional. Da mesma forma, o parâmetro `MIN_MEMORY_LEN` foi ajustado para encontrar um valor que já tivesse uma quantidade significativa de experiências armazenadas, sem causar atrasos desnecessários no treinamento.

GAMMA: Este é o fator de desconto, que determina o quanto o agente valoriza as recompensas futuras em relação às imediatas. Um valor de 0.97 foi escolhido com base em implementações semelhantes e experimentação, onde se observou que o agente alcançava o melhor

desempenho. Isso indica que, para um jogo como Pong, é importante que o agente valorize as recompensas futuras, mas não ao ponto de desconsiderar completamente as recompensas imediatas. Isso é especialmente relevante no Pong, onde uma única ação (realizada a cada 4 quadros) não tem tanto impacto, e a ação que resultará na maior recompensa será realizada no futuro quando a bola chegar à raquete. Portanto, um valor de 0.97 para GAMMA permite que o agente leve em consideração essas recompensas futuras.

ALPHA: Valores muito altos para a taxa de aprendizado resultaram em oscilações maiores no aprendizado do agente e facilitaram a criação de máximos locais. Portanto, um valor mais baixo de 0.0001 foi escolhido para garantir a estabilidade do aprendizado e a convergência do algoritmo.

EPSILON_DECAY: O valor de EPSILON começa em 1 e é multiplicado por uma taxa de decaimento de 0.99 a cada passo, resultando em uma redução de 1% do seu valor atual a cada passo. Isso continua até que EPSILON atinja um valor mínimo de 0.05. Este processo de decaimento permite que o agente comece com uma estratégia de exploração ampla no início do treinamento, dando-lhe a oportunidade de experimentar uma variedade de ações. À medida que o treinamento progride e EPSILON decai, o agente gradualmente se torna mais focado em explorar o conhecimento que adquiriu, optando por ações que acredita serem mais benéficas com base em suas experiências passadas. Isso equilibra a necessidade de exploração e explotação ao longo do tempo, permitindo que o agente aprenda uma política eficaz para o ambiente.

3.2.6 Etapa 4: avaliação dos resultados obtidos

É importante esclarecer que neste experimento, o jogador 1 é o modelo de IA que está em treinamento, aprendendo a jogar o jogo através de um processo de tentativa e erro e evoluindo. Por outro lado, o jogador 2 é o jogador padrão fornecido pela biblioteca *Gymnasium*, o jogador 2 age como um adversário constante e segue uma política de jogo predefinida que não muda ao longo do tempo.

Para avaliar os resultados obtidos, utilizou-se as seguintes métricas:

Pontuação Final: Esta métrica representa a pontuação final a cada episódio. Quando o Jogador 1 marca um ponto, é adicionado +1 à pontuação final. De forma contrária, quando o Jogador 2 marca um ponto, a pontuação final recebe -1. Ao final de cada episódio, uma pontuação positiva implica em vitória do Jogador 1 e uma pontuação final negativa implica na vitória do Jogador 2. Por exemplo, uma pontuação final de 7 significa que o Jogador 1 marcou 21 pontos e o Jogador 2 marcou 14 pontos.

Convergência do modelo: Esta métrica representa o número de iterações necessárias para o modelo convergir, ou seja, o modelo chega a um ponto em que a função de perda não muda significativamente com iterações adicionais de treinamento, as atualizações nos parâmetros do modelo se tornam tão pequenas que o modelo é considerado treinado, ou "convergido". O valor

de *loss* ou perda quantifica o quão bem o modelo está se ajustando aos dados de treinamento. Na implementação, *loss* é calculado pela diferença entre o valor Q previsto pelo modelo online e o valor Q esperado, que é determinado pela função de Bellman. Além disso, um modelo que converge mais rapidamente pode ser considerado mais eficiente, economizando tempo e recursos computacionais.

Durante a fase de treinamento, realizou-se a coleta de dados com o objetivo de analisar o comportamento do modelo e avaliar o seu desempenho. Três conjuntos de dados foram selecionados, cada um variando o valor de *alpha* (taxa de aprendizado), sendo os valores 0.0001, 0.00025 e 0.0005.

Cada conjunto de dados é composto por colunas que incluem o número do episódio, a recompensa obtida, a média das recompensas dos últimos 100 episódios e a duração do episódio. Esses dados foram coletados durante a fase de treinamento, como ilustrado na Figura 30. Além disso, após a coleta inicial, foram calculados dados adicionais, como a pontuação de cada jogador, que podem ser visualizados na Tabela 3.

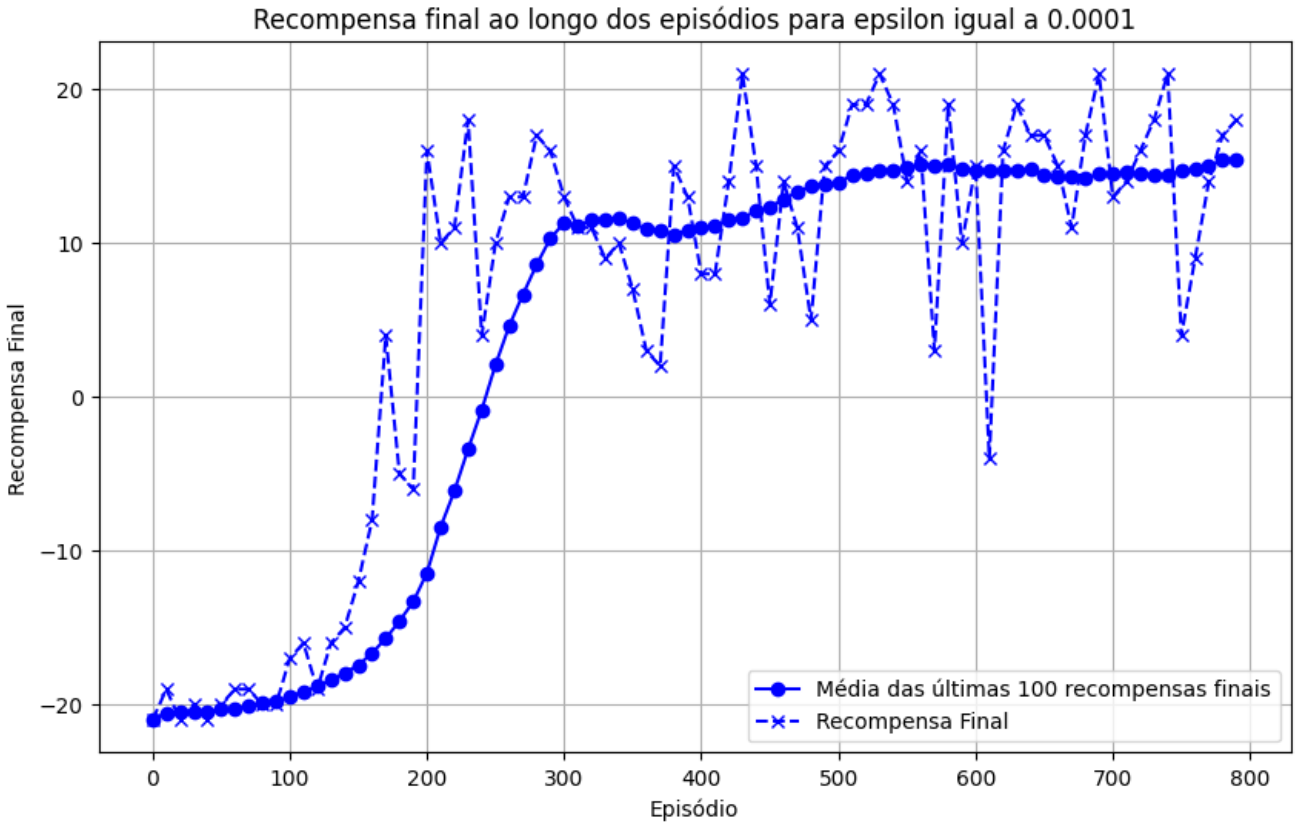
Campo	**Tipo**	**Descrição**
Episode	int64	Número do episódio
Time	int64	Tempo de execução do episódio
Reward	float64	Recompensa obtida
Loss	float64	Valor da função de perda
Last_100_Avg_Rew	float64	Média das recompensas dos últimos 100 episódios
Avg_Max_Q	float64	Valor médio da função Q máxima
Epsilon	float64	Valor do parâmetro epsilon
Duration	float64	Duração do episódio
Step	int64	Número de passos no episódio
CStep	int64	Contagem de passos cumulativos
p_player1	float64	Pontuação individual do jogador 1
p_player2	float64	Pontuação individual do jogador 2

Tabela 3 – Conjuntos de dados treinamento

Os gráficos apresentados nas figuras 31, 32 e 33 ilustram a evolução da recompensa final ao longo de 800 episódios para três conjuntos distintos. A partir desses gráficos, observa-se que a recompensa final aumenta progressivamente a cada episódio. A partir do episódio 170, a recompensa começa a se tornar positiva, indicando que o jogador 1 começou a vencer. Em um determinado momento, é possível notar um período no qual o jogador 1 vence consistentemente, sem variações significativas na vitória. Entre os valores de recompensa final alcançados, o valor máximo obtido pelo jogador 1 foi de 21 pontos, que é o valor máximo possível. A variação do *Loss* dos últimos 100 episódios é um indicador importante do desempenho do modelo. Para o conjunto com *alpha* de 0.0001 e 0.00025, a variação do *Loss* foi de -0,58, indicando uma diminuição no erro. No entanto, para o conjunto com *alpha* de 0.0005, a variação do *Loss* foi de 0,19, indicando um aumento no erro.

Essa variação do *Loss* também pode ser vista como um indicativo da convergência do modelo. A convergência refere-se ao ponto em que o modelo deixa de aprender e os parâmetros do modelo param de mudar. Uma pequena variação de *Loss* como a vista em todos os modelos sugere que o modelo está se aproximando da convergência. A tabela 4 mostra que todos os modelos convergiram, visto a pequena variação de *Loss* nos últimos 100 episódios. A taxa de aprendizado (*alpha*) desempenha um papel importante na velocidade de convergência. Modelos com taxas de aprendizado menores podem precisar de mais episódios para convergir, mas podem ser menos propensos a oscilações. Para obter conclusões robustas sobre a influência de cada parâmetro, uma análise estatística em várias execuções seria recomendada. Isso permitiria avaliar a consistência e a significância das diferenças observadas.

Figura 31 – Gráfico da pontuação final x episódios para taxa de aprendizado de 0.0001



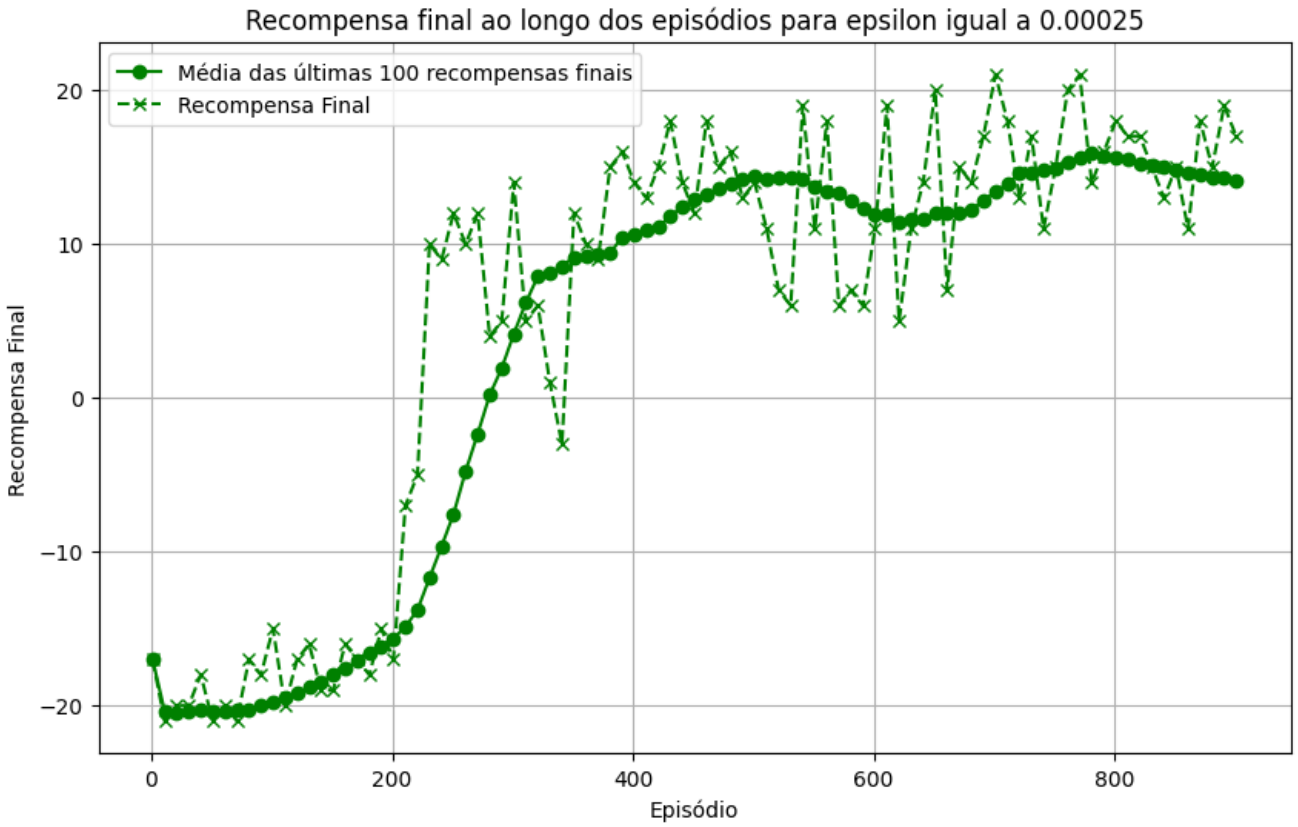
Fonte: O autor(2024)

Conjunto	Primeira vitória	Última derrota	Variação do Loss	Recompensa máxima
Alpha de 0.0001	episódio 170	episódio 610	-0,58	21
Alpha 0.00025	episódio 222	episódio 619	-0,58	21
Alpha 0.0005	episódio 171	episódio 436	0,19	21

Tabela 4 – Dados referentes a pontuação final dos conjuntos de treinamento

Em resumo, a análise dos resultados obtidos demonstrou que todos os experimentos foram bem-sucedidos, independentemente dos três valores de *alpha* utilizados. Os gráficos mos-

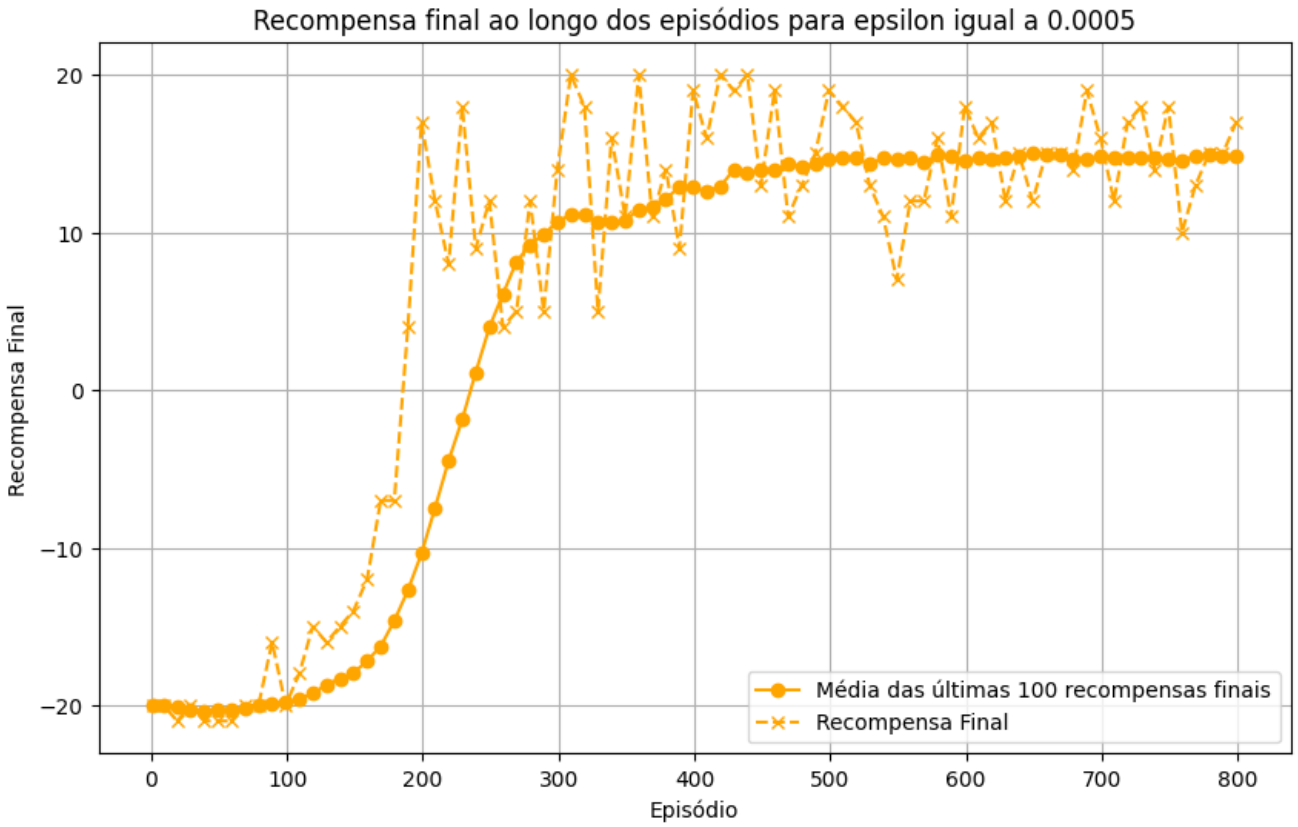
Figura 32 – Gráfico da pontuação final x episódios para taxa de aprendizado de 0.00025



Fonte: O autor(2024)

tram que os três modelos conseguiram atingir 21 pontos que é o valor máximo de recompensa, com uma média final de pontuação em torno de 15 pontos. Isso indica que os modelos foram capazes de aprender e evoluir eficientemente, alcançando altos níveis de desempenho e mantendo uma pontuação final robusta e consistente ao longo dos episódios. Embora a taxa de aprendizado (*alpha*) tenha tido um impacto significativo no desempenho e na eficiência do treinamento do modelo de IA, as diferenças na velocidade de convergência e na variação do *Loss* não impediram o sucesso dos experimentos. Esses resultados destacam a eficácia do processo de treinamento implementado, confirmando a robustez e a eficiência dos modelos de IA treinados.

Figura 33 – Gráfico da pontuação final x episódios para taxa de aprendizado de 0.0005



Fonte: O autor(2024)

3.2.7 Etapa 5: comparação com outros trabalhos

Nos trabalhos relacionados, observou-se a utilização de diferentes abordagens para o desenvolvimento de um modelo de IA capaz de jogar o *Pong*. WATERREUS empregou o aprendizado supervisionado e o AR, contudo, encontrou dificuldades com a convergência do modelo DQN, o que resultou em um desempenho que deixou a desejar e foi considerado insatisfatório. Por outro lado, Kaiser *et al.* (2019) propuseram o *Simulated Policy Learning* (Simple), que alcançou pontuações máximas em diversos jogos, incluindo *Pong*, onde obteve 21 pontos.

Guo *et al.* (2014) adotaram uma abordagem que combinava o planejamento baseado em *Monte Carlo Tree Search* (MCTS) com redes neurais convolucionais, atingindo também a pontuação máxima de 21 pontos. Mnih *et al.* (2013) introduziram o DQN, que se destacou nos jogos *Atari*, alcançando altas recompensas totais em diversos jogos de Atari e a pontuação máxima de 21 para o Pong.

Em contraste com essas abordagens, o presente trabalho utilizou um modelo de *Double Deep Q-Network* (DDQN) implementado em conjunto com o *Gymnasium*. Este modelo atingiu a pontuação máxima de 21 em *Pong*, demonstrando uma eficácia comparável aos métodos de Kaiser *et al.* (2019), Guo *et al.* (2014) e Mnih *et al.* (2013), e superando o desempenho do modelo de WATERREUS.

Título	**Pontuação Máxima**
Utilização de Aprendizado Supervisionado e por Reforço na Automação do Clássico Jogo Atari 'Pong' (WATERREUS)	Baixo desempenho
Aprendizado por Reforço Baseado em Modelo para Atari (Kaiser *et al.* (2019))	21
Aprendizado Profundo para Jogabilidade em Tempo Real de Jogos Atari Utilizando Planejamento Offline de Busca de Árvore de Monte Carlo (Guo *et al.* (2014))	21
Jogando Atari com Aprendizado Profundo por Reforço (Mnih *et al.* (2013))	21
Este trabalho	21

Tabela 5 – Comparação das Pontuações Máximas dos Trabalhos Relacionados

4 CONCLUSÕES

Neste capítulo apresenta-se uma síntese das atividades desenvolvidas durante o desenvolvimento deste trabalho, descreve também os resultados analisados e ideias para trabalhos futuros.

4.1 SÍNTESE DO TRABALHO

Este trabalho se propõe a explorar a implementação de um modelo de AR em jogos eletrônicos, com foco específico no jogo *Pong*, parte do ambiente *Atari* da plataforma *Gymnasium*. O estudo contextualiza a evolução e a relevância do AR neste domínio, destacando como a aplicação de AR em jogos eletrônicos tem sido um campo de constante desenvolvimento, impulsionado pela busca por agentes de IA cada vez mais proficientes e adaptáveis.

No escopo deste estudo, são abordados tópicos como IA, a Máquina de Turing, aprendizado de máquina, Processos Decisórios de Markov (MDP) e o algoritmo Q-Learning. Essa abordagem visa prover uma base teórica robusta para o entendimento do AR em jogos eletrônicos.

O objetivo principal desta pesquisa é examinar como o modelo de AR aprende através de suas interações com o ambiente do jogo *Pong*. A implementação específica do modelo é detalhada, destacando sua arquitetura, os hiperparâmetros-chave e a maneira como o agente interage com o ambiente para otimizar seu desempenho.

A eficácia do método de AR é avaliada no contexto do jogo *Pong*, analisando métricas específicas e comparando os resultados obtidos com outras abordagens do estado da arte. Observa-se que o algoritmo Q-Learning, seja usado isoladamente ou em combinação com outros algoritmos, demonstra-se eficaz na exploração e aprendizado no ambiente do jogo *Pong*.

No entanto, ao realizar essa implementação, são identificados desafios, como a generalização para outros jogos e questões de escalabilidade do modelo. Esses aspectos incentivam a discussão não apenas das conquistas, mas também das limitações e áreas para aprimoramento na aplicação de AR em jogos eletrônicos.

Além disso, aprofunda-se a discussão sobre plataformas como o *Gymnasium*, que desempenham um papel fundamental ao oferecer ambientes padronizados para o treinamento de agentes de AR. Esse tipo de plataforma é essencial para a implementação e teste de algoritmos de AR, facilitando a reprodutibilidade e comparação de resultados.

O primeiro passo envolveu a implementação de um modelo de AR, seguido por uma análise de como o modelo aprendeu através de suas interações com o ambiente do jogo selecionado, o *Pong*. Para esta tarefa, foi empregado o método DDQN, uma variação do Q-Learning,

para implementar o modelo de IA. O desenvolvimento foi conduzido na plataforma *Gymnasium*, que disponibiliza uma ampla gama de ambientes de jogos para treinamento e teste de algoritmos de AR.

A eficácia do método de AR foi avaliada no contexto do jogo escolhido, o *Pong*. Isso proporcionou uma compreensão mais aprofundada de como AR pode ser aplicado de maneira eficaz em ambientes de jogos. Além disso, a solução e o modelo desenvolvidos foram comparados com outros trabalhos de ponta na área, fornecendo uma visão valiosa sobre o desempenho do modelo em relação às abordagens existentes e destacando áreas para futuras melhorias e pesquisas.

Em suma, este trabalho contribui para o entendimento do AR em jogos eletrônicos, evidenciando o potencial do modelo DDQN no ambiente *Gymnasium* e no jogo *Pong*. A pesquisa destaca a importância de plataformas padronizadas como o *Gymnasium* para o avanço da área e sugere direções futuras para a superação dos desafios identificados. A validação do modelo proposto e a comparação com abordagens anteriores são etapas cruciais para a consolidação do conhecimento na aplicação de AR em jogos eletrônicos e para o avanço da IA de maneira geral.

4.2 CONTRIBUIÇÕES DO TRABALHO

Este trabalho apresenta várias contribuições significativas para o campo de AR em jogos eletrônicos. Primeiramente, é demonstrada a eficácia do algoritmo (DDQN) na aprendizagem e otimização do desempenho no jogo *Pong*, parte do ambiente *Atari* na plataforma *Gymnasium*. Através da análise detalhada da implementação do modelo, incluindo sua arquitetura e hiperparâmetros, foi possível entender como agentes de IA podem ser treinados para interagir de forma eficiente com o ambiente do jogo, destacando a capacidade do Q-learning em lidar com problemas complexos de tomada de decisão.

Além disso, este estudo enfatiza a importância de plataformas padronizadas como o *Gymnasium*, que oferecem ambientes controlados e consistentes para o desenvolvimento e teste de algoritmos de AR. A pesquisa não apenas valida o modelo implementado, mas também compara seu desempenho com abordagens do estado da arte, proporcionando uma visão crítica sobre as vantagens e limitações das diferentes metodologias.

Por fim, a identificação de desafios como a generalização para outros jogos e a escalabilidade do modelo fornece direções para futuras pesquisas, contribuindo para o avanço contínuo da IA e AR em contextos de jogos eletrônicos.

4.3 TRABALHOS FUTUROS

Para trabalhos futuros, algumas direções de pesquisa são sugeridas com base nas descobertas e limitações identificadas neste estudo. Primeiramente, seria interessante explorar a

aplicação do modelo DDQN em uma gama mais ampla de jogos eletrônicos dentro e fora da plataforma *Gymnasium*, a fim de avaliar sua capacidade de generalização e adaptabilidade a diferentes cenários de jogos.

Outro aspecto crucial para futuros estudos é a análise de algoritmos alternativos ou complementares ao DDQN. Comparações entre esses algoritmos podem revelar *insights* importantes sobre as melhores práticas para diferentes tipos de jogos e tarefas.

Além disso, sugere-se a utilização de métricas diferentes da pontuação final, a fim de avaliar o modelo de outras formas, proporcionando uma análise mais abrangente do desempenho e comportamento dos agentes de AR.

A variação nos resultados de treinamento é uma consideração importante. Para obter conclusões robustas sobre a influência de cada parâmetro, uma análise estatística em várias execuções seria recomendada. Isso permitiria avaliar a consistência e a significância das diferenças observadas.

REFERÊNCIAS

ACADêMICO, P. **Pesquisa Exploratória: exemplos, dicas, entenda o que é e como fazer**. 2018. Disponível em: <https://projetoacademico.com.br/pesquisa-exploratoria/>.

ALPAYDIN; ETHEM. **Introduction to machine learning**. [S.l.]: MIT press, 2020.

ANDRIJAUSKAS, K. **A importância da experimentação no ensino de ciências: uma revisão sistemática da literatura nacional na última década**. 2020. Universidade Tecnológica Federal do Paraná. Acesso em: 05 nov. 2023. Disponível em: <http://riut.utfpr.edu.br/jspui/bitstream/1/25648/1/importanciaexperimentacaoensinociencias.pdf>.

BISHOP, C. M. **Pattern Recognition and Machine Learning**. [S.l.]: Springer, 2006.

CROCKETT, E. **What is Machine Learning?** 2023. Disponível em: <https://www.datamation.com/big-data/what-is-machine-learning/>.

FACELI, K.; LORENA, A. C.; GAMA, J. **Inteligência Artificial - Uma Abordagem de Aprendizado de Máquina**. Grupo GEN, 2021. Acesso em: 03 set. 2023. ISBN 9788521637509. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788521637509/>.

FOUNDATION, T. F. **Basic Usage**. 2023. Gymnasium Farama. Acesso em: 05 nov. 2023. Disponível em: <https://gymnasium.farama.org/content/basic_usage/>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016.

GOOGLE. **Google Colaboratory**. 2024. <https://colab.research.google.com/>.

GRAVES, A.; FERNÁNDEZ, S.; SCHMIDHUBER, J. Supervised sequence labelling with recurrent neural networks. **Springer**, 2012.

GUO, X. *et al.* Deep learning for real-time atari game play using offline monte-carlo tree search planning. **Advances in Neural Information Processing Systems**, v. 27, 2014.

HAYKIN, S. **Neural Networks and Learning Machines**. [S.l.]: Prentice Hall, 2009.

JEBESSA, E. *et al.* Analysis of reinforcement learning in autonomous vehicles. In: **2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)**. [S.l.: s.n.], 2022. p. 0087–0091.

KAISER, L. *et al.* **Model-Based Reinforcement Learning for Atari**. 2019. Disponível em: <https://www.semanticscholar.org/paper/Model-Based-Reinforcement-Learning-for-Atari-Kaiser-Babaeizadeh/1fd4694e7c2d9c872a427d50e81b5475056de6bc>.

KENT, S. L. **The Ultimate History of Video Games: From Pong to Pokemon and Beyond**. [S.l.]: Prima Lifestyles, 2001.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **International Conference on Learning Representations**, 2015. Disponível em: <https://arxiv.org/abs/1412.6980>.

LIMA, I. **Inteligência Artificial**. E-book. Grupo GEN, 2014. Acesso em: 10 set. 2023. ISBN 9788595152724. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788595152724/>.

LISBOA, P. **Por que aprender Python? Motivos para investir nessa linguagem de programação**. 2021. Awari. Acesso em: 05 nov. 2023. Disponível em: <https://awari.com.br/por-que-aprender-python-motivos-para-investir-nessa-linguagem-de-programacao/>.

LU, Y.; LI, W. Técnicas e paradigmas em sistemas modernos de ia em jogos. v. 15, n. 8, p. 282, 2023. Disponível em: <https://www.mdpi.com/1999-4893/15/8/282>.

LUGER, G. F. **Inteligência Artificial - 6ª Edição**. 6ª. ed. [S.l.]: Editora Person, 2013. ISBN 9788581435503.

MCCARTHY, J. *et al.* **Dartmouth Summer Research Project on Artificial Intelligence**. 1956. <https://www.aaai.org/ojs/index.php/aimagazine/article/view/1904>.

____. **A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence**. 1955.

MITCHELL, T. M. *et al.* **Machine learning**. [S.l.]: McGraw-hill New York, 2007. v. 1.

MNIH *et al.* Playing atari with deep reinforcement learning. **ArXiv**, abs/1312.5602, 2013.

NIELSEN, M. **Neural Networks and Deep Learning**. [S.l.]: Determination Press, 2015.

PELLEGRINI, J.; WAINER, J. Processos de decisão de markov: um tutorial. **Nome da Revista**, Vol. 14, n. No. 2, p. 47, 2007.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. [S.l.]: Pearson, 2016.

____. **Artificial Intelligence: A Modern Approach**. Terceira edição. Pearson, 2022. ISBN 978-0136795654. Disponível em: <https://www.amazon.com/Artificial-Intelligence-Modern-Approach-4th/dp/0134610997/>.

SAMPAIO, R.; MANCINI, M. Estudos de revisão sistemática: um guia para síntese criteriosa da evidência científica. **Revista Brasileira de Fisioterapia**, v. 11, n. 1, p. 83–89, 2007.

SANTOS, M. H. dos. **Introdução à Inteligência Artificial**. E-book. [Digite o Local da Editora]: Editora Saraiva, 2021. Acesso em: 10 set. 2023. ISBN 9786559031245. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9786559031245/>.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. **Neural Networks**, 2015.

SILVA, F. M. *et al.* **Inteligência Artificial**. E-book. [Digite o Local da Editora]: Grupo A, 2018. Acesso em: 10 set. 2023. ISBN 9788595029392. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788595029392/>.

SILVER, D.; HASSABIS, D. **AlphaGo: Mastering the Ancient Game of Go**. 2016. Disponível em: <https://blog.research.google/2016/01/alphago-mastering-ancient-game-of-go.html>.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018.

TURING, A. M. Computing machinery and intelligence. **Mind**, LIX, n. 236, p. 433–460, 1950.

WANG, Z. *et al.* Dueling network architectures for deep reinforcement learning. **Proceedings of the 33rd International Conference on Machine Learning**, v. 48, p. 1995–2003, 2016. Disponível em: <https://arxiv.org/abs/1511.06581>.

WATERREUS, A. J. **Utilization of Supervised and Reinforcement Learning in the Control of Legged Robots**. 2023. Disponível em: <https://www.semanticscholar.org/paper/UTILIZATION-OF-SUPERVISED-AND-REINFORCEMENT-IN-THE-Waterreus-Bhounsule/ab3a8a9494e8ab2937fb4b118fc605df9256f06e>.

WATKINS, P. D. C. J. Technical note q-learning. **Kluwer Academic Publishers, Boston**, p. 14, 1992.

YANNAKAKIS *et al.* **Inteligência Artificial e Jogos**. [S.l.]: Springer, 2018. ISBN 978-3-319-63518-7.

APÊNDICE A – IMPLENTAÇÕES EM PYTHON

A.0.1 Classe DuelCNN

Algoritmo 4 – Implementação da classe DuelCNN na linguagem python

```
1
2  class DuelCNN(nn.Module):
3
4      def __init__(self, h, w, output_size):
5          super(DuelCNN, self).__init__()
6          self.conv1 = nn.Conv2d(in_channels=4,  out_channels=32,
7          kernel_size=8, stride=4)
8          self.bn1 = nn.BatchNorm2d(32)
9          convw, convh = self.conv2d_size_calc(w, h, kernel_size=8,
10         stride=4)
11         self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
12         kernel_size=4, stride=2)
13         self.bn2 = nn.BatchNorm2d(64)
14         convw, convh = self.conv2d_size_calc(convw, convh,
15         kernel_size=4, stride=2)
16         self.conv3 = nn.Conv2d(in_channels=64, out_channels=64,
17         kernel_size=3, stride=1)
18         self.bn3 = nn.BatchNorm2d(64)
19         convw, convh = self.conv2d_size_calc(convw, convh,
20         kernel_size=3, stride=1)
21         linear_input_size = convw * convh * 64
22         self.Alinear1 = nn.Linear(in_features=linear_input_size,
23         out_features=128)
24         self.Alrelu = nn.LeakyReLU()
25         self.Alinear2 = nn.Linear(in_features=128,
26         out_features=output_size)
27         self.Vlinear1 = nn.Linear(in_features=linear_input_size,
28         out_features=128)
29         self.Vlrelu = nn.LeakyReLU()
30         self.Vlinear2 = nn.Linear(in_features=128, out_features=1)
31
32     def conv2d_size_calc(self, w, h, kernel_size=5, stride=2):
33         next_w = (w - (kernel_size - 1) - 1) // stride + 1
34         next_h = (h - (kernel_size - 1) - 1) // stride + 1
35         return next_w, next_h
36
37     def forward(self, x):
38         x = F.relu(self.bn1(self.conv1(x)))
39         x = F.relu(self.bn2(self.conv2(x)))
40         x = F.relu(self.bn3(self.conv3(x)))
```

```
41          x = x.view(x.size(0), -1)
42          Ax = self.Alrelu(self.Alinear1(x))
43          Ax = self.Alinear2(Ax)
44          Vx = self.Vlrelu(self.Vlinear1(x))
45          Vx = self.Vlinear2(Vx) r
46          q = Vx + (Ax - Ax.mean())
47          return q
```

A.0.2 Agent

Algoritmo 5 – Implementação da classe Agent na linguagem python

```
1
2   class Agent:
3       def __init__(self, environment):
4
5           self.state_size_h = environment.observation_space.shape[0]
6           self.state_size_w = environment.observation_space.shape[1]
7           self.state_size_c = environment.observation_space.shape[2]
8           self.action_size = environment.action_space.n
9           self.target_h = 80
10          self.target_w = 64
11          self.crop_dim = [20, self.state_size_h, 0, self.state_size_w]
12          self.gamma = GAMMA
13          self.alpha = ALPHA
14          self.epsilon = 1
15          self.epsilon_decay = EPSILON_DECAY
16          self.epsilon_minimum = 0.05
17          self.memory = deque(maxlen=MAX_MEMORY_LEN)
18          self.online_model = DuelCNN(h=self.target_h, w=self.target_w,
19          output_size=self.action_size).to(DEVICE)
20          self.target_model = DuelCNN(h=self.target_h,
21          w=self.target_w, output_size=self.action_size).to(DEVICE)
22          self.target_model.
23          load_state_dict(self.online_model.state_dict())
24          self.target_model.eval()
25          self.optimizer = optim.Adam(self.online_model.parameters(),
26          lr=self.alpha)
27
28      def preProcess(self, image):
29
30          frame = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
31          frame = frame[self.crop_dim[0]:self.crop_dim[1],
32          self.crop_dim[2]:self.crop_dim[3]]
33          frame = cv2.resize(frame, (self.target_w, self.target_h))
34          frame = frame.reshape(self.target_w, self.target_h) / 255
35
```

```
36          return frame
37
38      def act(self, state):
39
40          act_protocol = 'Explore' if random.uniform(0, 1) <=
41                         self.epsilon else 'Exploit'
42
43          if act_protocol == 'Explore':
44              action = random.randrange(self.action_size)
45          else:
46              with torch.no_grad():
47                  state = torch.tensor(state, dtype=torch.float,
48                  device=DEVICE).unsqueeze(0)
49                  q_values = self.online_model.forward(state)
50                  action = torch.argmax(q_values).item()
51
52          return action
53
54      def train(self):
55
56          if len(agent.memory) < MIN_MEMORY_LEN:
57              loss, max_q = [0, 0]
58              return loss, max_q
59
60          state, action, reward, next_state, done =
61          zip(*random.sample(self.memory, BATCH_SIZE))
62          state = np.concatenate(state)
63          next_state = np.concatenate(next_state)
64          state = torch.tensor(state, dtype=torch.float,
65          device=DEVICE)
66          next_state = torch.tensor(next_state, dtype=torch.float,
67          device=DEVICE)
68          action = torch.tensor(action, dtype=torch.long,
69          device=DEVICE)
70          reward = torch.tensor(reward, dtype=torch.float,
71          device=DEVICE)
72          done = torch.tensor(done, dtype=torch.float,
73          device=DEVICE)
74          state_q_values = self.online_model(state)
75          next_states_q_values = self.online_model(next_state)
76          next_states_target_q_values = self.target_model(next_state)
77          selected_q_value = state_q_values.gather(1,
78          action.unsqueeze(1)).squeeze(1)
79          next_states_target_q_value =
80          next_states_target_q_values.gather(1,
81          next_states_q_values.max(1)
82          [1].unsqueeze(1)).squeeze(1)
```

```
83          expected_q_value = reward +
84          self.gamma * next_states_target_q_value
85          * (1 - done)
86          loss = (selected_q_value -
87          expected_q_value.detach()).pow(2).mean()
88          self.optimizer.zero_grad()
89          loss.backward()
90          self.optimizer.step()
91
92          return loss, torch.max(state_q_values).item()
93
94      def storeResults(self, state, action, reward, nextState, done):
95          self.memory.append([state[None, :], action, reward,
96          nextState[None, :], done])
97
98      def adaptiveEpsilon(self):
99          if self.epsilon > self.epsilon_minimum:
100             self.epsilon *= self.epsilon_decay
```

A.0.3 Main

Algoritmo 6 – Implementação da função main na linguagem python

```
1
2   def upload_file(file_types):
3       uploaded_files = files.upload()
4       filenames = list(uploaded_files.keys())
5       if filenames:
6           return filenames[0]
7       else:
8           return None
9
10  def save_model(agent, episode, model_path):
11      if not os.path.exists(model_path):
12          os.makedirs(model_path)
13
14      weightsPath = os.path.join(model_path, f'{episode}.pkl')
15      epsilonPath = os.path.join(model_path, f'{episode}.json')
16
17      torch.save(agent.online_model.state_dict(), weightsPath)
18      epsilonDict = {'epsilon': agent.epsilon}
19      with open(epsilonPath, 'w') as outfile:
20          json.dump(epsilonDict, outfile)
21      print(f"Model saved at episode {episode} in {model_path}")
22
23  if __name__ == "__main__":
24      environment = gym.make(ENVIRONMENT)
```

```
25      agent = Agent(environment)
26
27      if LOAD_MODEL_FROM_FILE:
28          print("Selecione o arquivo .pkl")
29          pkl_file_path = upload_file([("PKL Files", "*.pkl")])
30          print("Selecione o arquivo .json")
31          json_file_path = upload_file([("JSON Files", "*.json")])
32
33          if pkl_file_path and json_file_path:
34              agent.online_model.load_state_dict(torch.load(pkl_file_path))
35
36              with open(json_file_path, 'r') as outfile:
37                  model_params = json.load(outfile)
38
39              agent.epsilon = model_params.get('epsilon', agent.epsilon)
40              startEpisode = LOAD_FILE_EPISODE + 1
41          else:
42              print("Erro ao carregar arquivos.")
43      else:
44          startEpisode = 1
45
46      last_100_ep_reward = deque(maxlen=100)
47      total_step = 1
48      for episode in range(startEpisode, MAX_EPISODE):
49
50          startTime = time.time()
51          state = environment.reset()
52
53          state = agent.preProcess(state[0])
54
55
56          state = np.stack((state, state, state, state))
57
58          total_max_q_val = 0
59          total_reward = 0
60          total_loss = 0
61          for step in range(MAX_STEP):
62
63
64              action = agent.act(state)
65              next_state, reward, done, info,truncated =
66              environment.step(action)
67
68              if RENDER_GAME_WINDOW:
69                clear_output(True)
70                display(next_state);
71                time.sleep(0.05)
```

```
72
73              next_state = agent.preProcess(next_state)
74              next_state = np.stack((next_state, state[0], state[1], state[2]))
75              agent.storeResults(state, action, reward, next_state, done)
76
77              state = next_state
78
79              if TRAIN_MODEL:
80
81                  loss, max_q_val = agent.train()
82              else:
83                  loss, max_q_val = [0, 0]
84
85              total_loss += loss
86              total_max_q_val += max_q_val
87              total_reward += reward
88              total_step += 1
89              if total_step % 1000 == 0:
90                  agent.adaptiveEpsilon()
91
92              if done:
93                  currentTime = time.time()
94                  time_passed = currentTime - startTime
95                  current_time_format = time.strftime("%H:%M:%S",
96                  time.gmtime())
97                  epsilonDict = {'epsilon': agent.epsilon}
98
99
100                 if TRAIN_MODEL:
101                     agent.target_model.load_state_dict(
102                     agent.online_model.state_dict())
103
104                 last_100_ep_reward.append(total_reward)
105                 avg_max_q_val = total_max_q_val / step
106
107                 outStr = "Episode:{} Time:{} Reward:{:.2f} Loss:{:.2f}
108 Last_100_Avg_Rew:{:.3f} Avg_Max_Q:{:.3f} Epsilon:{:.2f}
109 Duration:{:.2f} Step:{} CStep:{}".format(
110                 episode, current_time_format, total_reward, total_loss,
111                 np.mean(last_100_ep_reward), avg_max_q_val, agent.epsilon,
112                 time_passed, step, total_step
113
114                 )
115
116                 print(outStr)
117
118                 break
```

```
119
120
121     weightsPath = MODEL_PATH + 'modelo.pkl'
122     epsilonPath = MODEL_PATH + 'modelo.json'
123
124     torch.save(agent.online_model.state_dict(), weightsPath)
125
126
127     epsilonDict = {'epsilon': agent.epsilon}
128     with open(epsilonPath, 'w') as outfile:
129         json.dump(epsilonDict, outfile)
130
131
132     files.download(weightsPath)
133     files.download(epsilonPath)
```