Arthur de Senna Rocha

Desenvolvimento de uma Inteligência Artificial para aprender a jogar jogos em Allegro

Brasil

6 de novembro de 2020

Arthur de Senna Rocha

Desenvolvimento de uma Inteligência Artificial para aprender a jogar jogos em Allegro

Trabalho de Conclusão de Curso II apresentado como requisito parcial à obtenção de título de bacharel em Engenharia de Sistemas pela Escola de Engenharia da Universidade Federal de Minas Gerais.

Universidade Federal de Minas Gerais – UFMG

Escola de Engenharia

Engenharia de Sistemas

Orientador: Pedro Olmo Stancioli Vaz De Melo

Brasil

6 de novembro de 2020

Histórico de Revisões

Versão (xx)	Data (dd/mm/yyyy)	Autor	Descrição
1.0	14/11/2019	Arthur de Senna Rocha	Texto inicial da monografia TCC I
1.1	04/12/2019	Arthur de Senna Rocha	Texto final da monografia TCC I
2.0	03/04/2020	Arthur de Senna Rocha	Adiciona Visão Geral TCC II
2.1	21/08/2020	Arthur de Senna Rocha	Atualiza Cronograma (ERE)
2.2	17/10/2020	Arthur de Senna Rocha	Texto inicial da monografia TCC II
2.3	6/11/2020	Arthur de Senna Rocha	Texto final da monografia TCC II

Resumo

O uso de inteligência artificial (IA) e de algoritmos de *machine learning* possibilita que máquinas aprendam com experiências, se ajustem à novas entradas de dados e performem tarefas como seres humanos. Com essas tecnologias, os computadores podem ser treinados para cumprir tarefas específicas ao processar grandes quantidades de dados e reconhecer padrões nesses dados. O presente trabalho se propõe a desenvolver uma IA capaz de aprender a jogar diferentes jogos, desde que se tenha acesso ao código fonte e feito em Allegro. Para isso, será implementado um algoritmo de *Deep Reinforcement Learning*, abordagem que consiste em fornecer ao sistema parâmetros relacionados ao seu estado e uma recompensa positiva ou negativa com base em suas ações. Nenhuma regra sobre o jogo é dada e, inicialmente, a IA não tem informações sobre o que precisa fazer. A única informação passada para a IA são os comandos básicos do jogo. O objetivo do sistema é descobrir e elaborar uma estratégia para maximizar a pontuação - ou a recompensa. Diferente de muitas IAs que focam na solução de um único problema, a proposta deste projeto é elaborar uma IA que seja genérica e capaz solucionar e elaborar estratégias para uma variedade de situações diferentes.

Palavras-chave: deep learning, allegro, inteligência artificial, jogos digitais, machine learning.

Abstract

The use of Artificial intelligence (AI) and machine learning algorithms enables computers to learn from experience, adjust to new data inputs, and perform tasks as human beings. With these technologies, computers can be trained to perform specific tasks by processing large amounts of data and recognizing patterns in that data. The present work aims to develop an AI capable of learning how to play different games, provided that it has access to the source code and the game is made in Allegro. For this, a Deep Reinforcement Learning algorithm will be implemented, which provides the system with parameters related to its state and a positive or negative reward based on its actions. No rules about the game are given and initially, the AI has no information on what it needs to do. The only information passed to AI is the basic commands of the game. The goal of the system is to discover and devise a strategy to maximize its score - or the reward. Unlike many AIs that focus on solving a single problem, the purpose of this project is to design a generic AI that can solve and develop a strategy for a variety of different situations.

Keywords: deep learning, allegro, artificial intelligence, video games, machine learning.

Lista de ilustrações

Lista de abreviaturas e siglas

ALE	*Allegro Learning Enviroment*
ANNs	*Artificial Neural Networks*
ASR	*Automatic Speech Recognition*
DL	*Deep Learning*
DQN	*Deep Q-network*
FIFO	*First in, first out*
GMM	*Gaussian Mixture Model*
HMM	*Hidden Markov Model*
IA	Inteligência Artificial
MDP	*Markov Decision Process*
ML	*Machine Learning*
NPC	*Non-player Character*
PCG	*Procedural Content Generation*
PNL	Processamento de Linguagem Natural
RL	*Reinforcement Learning*

Sumário

1 Introdução

A inteligência artificial (IA) vem ganhando manchetes no mundo todo, sendo anunciada tanto como uma salvação econômica quanto como precursora de desintegração social (ROBU et al., 2019). Quando computadores programáveis foram concebidos pela primeira vez, as pessoas se perguntavam se essas máquinas poderiam se tornar inteligentes, mais de cem anos antes de uma ser construída (MENABREA et al., 1843). Hoje, a inteligência artificial é um campo com inúmeras aplicações práticas e tópicos de pesquisa ativos. Buscamos softwares inteligentes para automatizar o trabalho de rotina, entender a fala ou as imagens, fazer diagnósticos em medicina e apoiar a pesquisa científica (GOODFELLOW; BENGIO; COURVILLE, 2016).

A IA adiciona inteligência a produtos existentes. Na maioria dos casos, a inteligência artificial não é vendida como uma aplicação individual. Pelo contrário, produtos já existentes são aprimorados com funcionalidades de IA, de maneira parecida como a Siri foi adicionada aos produtos da *Apple*. Automação, plataformas de conversa, robôs e aparelhos inteligentes podem ser combinados com grandes quantidades de dados para aprimorar diversas tecnologias para casa e escritório, de inteligência em segurança à análise de investimentos.

A maioria dos exemplos de IA sobre os quais se ouve falar hoje – de computadores mestres em xadrez a carros autônomos – dependem de *deep learning* e processamento de linguagem natural (PNL) (RODRIGUES, 2017). Treinar um agente para superar os jogadores humanos e otimizar sua performance pode nos ensinar como otimizar diferentes processos em uma grande variedade de situações. Foi o que o *DeepMind* do Google fez com seu popular *AlphaGo* e seu sucessor *AlphaZero*, vencendo os campeões mundiais em Go, xadrez e shogi, e obtendo resultados de performance nunca antes vistos.

1.1 Motivação

Técnicas de aprendizado de máquina e algoritmos de *deep learning* (DL) têm consistentemente melhorado a capacidade de um computador de fornecer reconhecimento de padrões e previsões cada vez mais precisas. Além disso, sistemas de DL são consistentemente aplicados com sucesso a conjuntos de aplicações cada vez mais amplos.

Ao mesmo tempo em que a escala e a precisão das redes neurais aumentaram, a complexidade das tarefas que podem ser resolvidas também cresceu significativamente. Uma conquista importante de sistemas de DL é a sua extensão ao domínio da aprendizagem por reforço ou *reinforcement learning* (RL) (SUTTON; BARTO, 2018). No contexto do

aprendizado por reforço, um agente autônomo deve aprender a executar uma tarefa por tentativa e erro, sem nenhuma orientação do operador humano.

Além do valor para pesquisa em múltiplas áreas da ciência, muitas dessas aplicações de aprendizado de máquina e *deep learning* são altamente lucrativas. O aprendizado de máquina hoje é usado por muitas empresas de tecnologia, incluindo *Google, Microsoft, Facebook*, IBM, *Baidu, Apple, Adobe, Netflix*.

Diante à crescente presença de sistemas que utilizam técnicas de *deep learning* no dia-a-dia, nota-se o grande potencial do investimento em pesquisa, modelagem de novos problemas e estudo de técnicas de aprendizado de máquina. Uma interessante aplicação desses sistemas está na área de jogos digitais. A indústria de videogames tem testemunhado um enorme crescimento, graças, em boa parte, ao incrível aumento no poder da computação em termos de representações visuais. Seja no controle de personagens não-jogadores (NPC), ou para a geração de conteúdo processual (PCG), são inúmeras as potenciais aplicações dessas técnicas em jogos digitais. O potencial dessas ferramentas de obter uma vantagem competitiva no mercado, ou simplesmente fornecer uma melhor experiência para o usuário é, no mínimo, instigante. Nesse contexto, a modelagem de novos problemas, implementação de soluções utilizando técnicas de *deep learning* e investimento na área, torna-se uma relevante contribuição para o estado da arte.

1.2 Objetivos

O presente trabalho tem como objetivo geral propor o desenvolvimento de uma IA capaz de aprender a jogar diferentes jogos, desde que se tenha acesso ao código fonte e feito em Allegro. Para isso, será implementado um algoritmo utilizando *Deep Reinforcement Learning* (DRL), abordagem que consiste em fornecer ao sistema parâmetros relacionados ao seu estado e uma recompensa positiva ou negativa com base em suas ações. Nenhuma regra sobre o jogo é dada e, inicialmente, a IA não tem informações sobre o que precisa fazer. A única informação passada para a IA são os comandos básicos do jogo. O objetivo do sistema é descobrir e elaborar uma estratégia para maximizar a pontuação - ou a recompensa.

Os objetivos mais específicos deste trabalho são:

1. Revisão da literatura do problema;

2. Descrição e modelagem do problema;

3. Proposta de critérios adicionais que possibilitem estimar outras características das possíveis soluções do projeto, tais como performance, confiabilidade, entre outras;

4. Modelagem de um ou mais jogos que atendam aos requisitos, para validação do sistema;

5. Proposta de um algoritmo de *deep learning* para a solução do problema.

Vale apena ressaltar que a ideia de se implementar um sistema que possa ser adaptado para uma grande variedade de cenários ou jogos, sugere uma ferramenta que possa ser aplicada não só na indústria de videogames, mas em diversas áreas da ciência. Uma situação ou problema do mundo real poderia, por exemplo, ser modelada na forma de um jogo. Nesse caso, a ferramenta utilizada poderia ser aplicada para maximizar sua pontuação. Essa pontuação, por sua vez, seria modelada dentro do jogo de forma a se aproximar do resultado ideal. Dessa forma, o sistema seria capaz de desenvolver estratégias para solucionar problemas e qualquer área da ciência.

Perante o exposto, a implementação de algoritmos que utilizam o aprendizado de máquina de forma a serem aplicados em diferentes cenários, apresenta um potencial de propor novas estratégias e otimizar sistemas já existentes, melhorar a qualidade do produto final e a experiência do usuário, além de proporcionar uma vantagem competitiva no mercado.

1.3 Descrição do problema

O campo da inteligência artificial é capaz de solucionar, com certa facilidade, problemas que são intelectualmente muito difíceis para os serem humanos, mas relativamente diretos para os computadores - problemas que podem ser descritos por uma lista de regras formais e matemáticas. Tarefas abstratas e formais que estão entre os empreendimentos mentais mais difíceis para um ser humano estão entre os mais fáceis para um computador.

Ironicamente, o grande desafio à inteligência artificial provou estar em resolver tarefas fáceis de executar para um ser humano. Problemas que parecem automáticos, que resolvemos intuitivamente, como reconhecer palavras faladas ou rostos em imagens. Os computadores há muito conseguem derrotar até o melhor jogador de xadrez humano (HSU, 2002), mas apenas recentemente começaram a alcançar algumas das habilidades dos seres humanos comuns, como reconhecer objetos ou fala.

A vida cotidiana de uma pessoa requer uma imensa quantidade de conhecimento sobre o mundo. A grande quantidade de informação desses cenários torna inviável a codificação de todas as regras do sistema e, por isso, o computador tem uma grande dificuldade para solucionar esses problemas. Além disso, grande parte desse conhecimento é subjetivo e intuitivo e, portanto, difícil de articular de maneira formal. Os computadores precisam capturar esse mesmo conhecimento para se comportarem de maneira inteligente.

Um dos principais desafios da inteligência artificial é como obter esse conhecimento informal em um computador.

As dificuldades enfrentadas por sistemas que dependem de conhecimento codificado sugerem que os sistemas de IA necessitam da capacidade de adquirir seu próprio conhecimento, extraindo padrões de dados brutos. Esse recurso é conhecido como aprendizado de máquina ou *machine learning* (ML). A introdução do aprendizado de máquina permitiu que os computadores resolvessem problemas que envolvem o conhecimento sobre o mundo real e tomassem decisões mais subjetivas.

O problema proposto nesse trabalho é o de implementar uma IA que, utilizando algoritmos de *deep reinforcement learning*, seja capaz de aprender e desenvolver estratégias para jogar diferentes jogos digitais. Os requisitos do sistema podem ser resumidos pelos seguintes critérios:

1. O sistema receberá, inicialmente, somente os comandos básicos do jogo. Nenhuma regra sobre o jogo é dada e, inicialmente, o agente não tem nenhuma informação sobre o que precisa fazer;

2. O agente deve ser capaz de elaborar uma estratégia para maximizar sua pontuação e que alcance resultados consideravelmente superiores aos de uma abordagem aleatória e próximos aos de um agente humano;

3. O sistema deverá ser capaz de lidar com cenários aleatórios, onde os obstáculos mudam a cada partida, e não aleatórios, onde os obstáculos são "fixos" e a dificuldade varia de acordo com o progresso no jogo;

4. O sistema deve ser generalizado para que possa ser aplicado à diferentes cenários e treinado para jogar diferentes jogos digitais.

De modo a garantir a factibilidade da implementação do sistema, algumas restrições devem ser acatadas. Por exemplo, além de haver a necessidade de se conhecer os comandos básicos do jogo, o sistema precisa ser capaz de obter informações atualizadas sobre o estado do jogo em que se encontra. No caso deste trabalho, foram definidas as seguintes restrições:

1. O sistema deve ter acesso ao código fonte do jogo no qual será aplicado;

2. O sistema deverá ter acesso à pontuação do jogo;

3. O jogo deverá ter sido implementado em *Allegro*;

4. O jogo deve ser 2D para garantir a viabilidade da implementação do sistema.

O acesso ao código fonte nos permite ter conhecimento dos comandos básicos do jogo, enquanto a biblioteca *Allegro* fornece rotinas de baixo nível comumente necessárias na programação de jogos (HARGREAVES, 1990). Essas rotinas, por serem fáceis de manipular, auxiliarão na implementação de um sistema de aprendizado.

O desafio nesse projeto é criar e treinar uma rede neural convolucional capaz de aprender políticas através de pixels brutos em ambientes complexos por meio de um algoritmo de *deep reinforcement learning*. O objetivo principal é implementar um agente que seja capaz de aprender a jogar o maior número de jogos possíveis sem conhecimento prévio do ambiente. Em outras palavras, o sistema deverá ser genérico e o agente não receberá nenhuma informação prévia sobre um jogo específico.

1.4 Revisão da literatura

Apesar de se falar sobre *deep learning* como uma emocionante nova tecnologia, este tem uma história longa e rica, mas apresentando diversos nomes, os quais refletem diferentes pontos de vista filosóficos. Em termos gerais, ocorreram três ondas de desenvolvimento com níveis de popularidade variados: DL conhecido como *cybernetics* nas décadas de 1940 a 1960, DL conhecido como *connectionism* entre as décadas de 1980 e 1990 e o ressurgimento atual sob o nome de aprendizado profundo ou *deep learning* a partir de 2006 (GOODFELLOW; BENGIO; COURVILLE, 2016).

Alguns dos primeiros algoritmos de aprendizado que são reconhecidos hoje pretendiam ser modelos computacionais de aprendizado biológico, isto é, modelos de como o aprendizado acontece ou pode acontecer no cérebro. Como resultado, um dos nomes que o DL passou é o de *artificial neural networks* (ANNs). No entanto, o termo moderno "*deep learning*" vai além da perspectiva neurocientífica da atual geração de modelos de aprendizado de máquina. Ele apela a um princípio mais geral de aprendizado de vários níveis de composição, que podem ser aplicados em estruturas de aprendizado de máquina que não são necessariamente inspiradas em neurônios.

Uma das muitas contribuições do DL está no reconhecimento de fala (Nassif et al., 2019). Até recentemente, os de reconhecimento automático de fala (ASR) combinavam principalmente modelos ocultos de Markov (HMMs) e modelos de mistura gaussianos (GMM). Com a introdução de redes neurais e, posteriormente, modelos de DL cada vez maiores e mais profundos e conjuntos de dados muito maiores, a precisão do reconhecimento foi dramaticamente aprimorada usando redes neurais para, eventualmente, substituir GMMs na tarefa de associar recursos acústicos a fonemas (GOODFELLOW; BENGIO; COURVILLE, 2016).

O *deep learning* também contribuiu para outras ciências. As redes convolucionais modernas para reconhecimento de objetos e visão computacional fornecem um modelo de

processamento visual com diversas aplicações na medicina (YEUNG et al., 2019; AFRAZ DANIEL L.K. YAMINS, 2014). O *deep learning* também fornece ferramentas úteis para processar grandes quantidades de dados e fazer previsões úteis em campos científicos. Ele tem sido usado com sucesso para prever como as moléculas irão interagir, a fim de ajudar as empresas farmacêuticas a projetar novos medicamentos (DAHL; JAITLY; SALAKHUTDINOV, 2014), a procurar partículas subatômicas (BALDI; SADOWSKI; WHITESON, 2014), e para o processamento de linguagem natural (YOUNG et al., 2018). Espera-se que o DL apareça em cada vez mais campos científicos no futuro.

Pesquisas recentes em IA deram origem a técnicas poderosas para o *deep reinforcement learning*. Na combinação de aprendizado de representação com comportamento orientado por recompensas, o DRL parece ter um interesse inerente à psicologia e neurociência. Um argumento contra essa abordagem foi o de que os procedimentos de aprendizado por DRL exigem grandes quantidades de dados de treinamento, sugerindo que esses algoritmos podem diferir fundamentalmente daqueles subjacentes ao aprendizado humano. Embora essa preocupação se aplique à onda inicial de técnicas de RL profunda, o trabalho subsequente de IA estabeleceu métodos que permitem que os sistemas de RL profunda aprendam mais rápida e eficientemente (BOTVINICK et al., 2019).

A IA em jogos digitais possui algumas peculiaridades (YANNAKAKIS, 2012; MILLINGTON; FUNGE, 2009), que a distinguem da IA clássica, especialmente porque, em muitos casos, ela deve lidar com aplicativos em tempo real e não necessariamente precisa otimizar resultados. Ela pode ser explorada para muitos propósitos, que podem ser coletados em três macro-categorias principais: ajudar na jogabilidade, melhorar a imersão do jogador no mundo do jogo (também simular a psicologia dos agentes que representam os personagens que não jogam - NPCs) e apoiar o trabalho de designers de jogos e níveis (Piergigli et al., 2019). Entre as técnicas de IA mais difusas, podemos contar aquelas usadas para gerar procedimentalmente conteúdos (Karavolos; Liapis; Yannakakis, 2018; RIPAMONTI et al., 2017) e aquelas destinadas a apoiar o sistema de tomada de decisão dos agentes artificiais (Ripamonti et al., 2017).

O aprendizado de máquina e as redes neurais são aplicadas aos jogos há muito tempo, mas seu uso recentemente conheceu um interesse renovado e aborda uma ampla variedade de tópicos. No entanto, o uso dessas técnicas para treinar agentes em ambientes complexos, com várias ações simultâneas possíveis é um resultado bastante desafiador a ser alcançado (Piergigli et al., 2019).

O *DeepMind* do Google desenvolveu o *Deep Q-network* (DQN), uma arquitetura de rede neural, que demonstrou ser capaz de aprender políticas de controle no nível humano em vários jogos diferentes do Atari 2600 (MNIH et al., 2015). Os DQNs aprendem a estimar os valores Q (função de valor da ação do estado) de selecionar cada ação do estado atual do jogo. Como a função de valor da ação do estado é uma representação suficiente

da política do agente, um jogo pode ser jogado selecionando a ação com o valor Q máximo em cada etapa do tempo. Dessa forma, aprendendo políticas de pixels em tela bruta a ações, essas redes têm demonstrado desempenho avançado em vários jogos do Atari 2600. Vale ressaltar que a mesma rede pode ser usada em várias tarefas sem nenhuma alteração e que o aprendizado é de ponta a ponta, dos valores brutos dos pixels aos valores Q, sem a necessidade de intervenção humana. Os DQNs também foram estendidos para obter melhor desempenho em jogos ainda mais complexos (DWIBEDI; VEMULA, 2016).

Um dos feitos mais notáveis nesse contexto, realizado também pelo *DeepMind*, é o da implementação da IA conhecida como *AlphStar*. Essa inteligência artificial alcançou uma classificação de grande mestre depois de ter sido lançada nos servidores europeus do jogo *StarCraft II*, ficando entre os $0,15\%$ dos 90.000 jogadores da região. O domínio do *StarCraft* emergiu como um importante desafio para a pesquisa em inteligência artificial, devido ao seu status icônico e duradouro entre os mais difíceis e-sports profissionais e sua relevância para o mundo real em termos de complexidade bruta e desafios multi-agente. Ao longo de uma década e inúmeras competições, os agentes mais fortes simplificaram aspectos importantes do jogo, utilizaram capacidades sobre-humanas ou empregaram subsistemas artesanais. Apesar dessas vantagens, nenhum agente anterior chegou perto de igualar a habilidade geral dos melhores jogadores de *StarCraft* (VINYALS et al., 2019). Tudo isso torna os resultados obtidos pelo AlphaStar ainda mais impressionantes: a IA foi classificada no nível grande mestre e acima de $99,8\%$ dos jogadores humanos classificados oficialmente.

1.5 Organização do trabalho

Este trabalho está estruturado em sete capítulos. O **Capítulo 1** consiste em uma breve introdução ao tema do projeto e uma análise da literatura do problema. O **Capítulo 2** apresenta uma contextualização do problema nos âmbitos social, ambiental e econômico. O **Capítulo 3** discorre a abordagem proposta para o problema e descreve as ferramentas que serão utilizadas. O **Capítulo 4** apresenta comentários finais sobre o trabalho desenvolvido no TCC I e descreve a proposta para o Trabalho de Conclusão de Curso II. O **Capítulo 5** constitui o início da segunda parte do projeto e descreve a modelagem matemática do problema além de uma dissertação sobre a implementação do sistema. O **Capítulo 6** apresenta os resultados obtidos e realiza uma discussão sobre os mesmos. Por fim, o **Capítulo 7** apresenta as conclusões finais do trabalho assim como possíveis propostas de continuidade.

2 Contextualização em Humanidades

Nos últimos anos, houve um progresso significativo na solução de problemas desafiadores em diversos campos, utilizando algoritmos de *deep reinforcement learning*. Como consequência, o RL experimentou um crescimento dramático na atenção e no interesse da comunidade científica.

Do ponto de vista econômico, são inúmeros os usos de *deep learning* no mercado. Desde ferramentas que melhoram a precisão dos sensores de precipitação por satélite e concentrando-se na redução do viés e dos alarmes falsos (TAO et al., 2016), à agentes que permitem que diferentes dispositivos eletrônicos interpretem dados de multimídia não estruturados e reajam de maneira inteligente aos eventos do usuário e do ambiente (Tang et al., 2017), o DL tem se tornado cada vez mais presente e essencial para a sociedade. Grandes setores e empresas na área da tecnologia não existiriam sem o uso dessas ferramentas.

Em relação aos impactos sociais do DL podemos mencionar a sua utilização para estimar as características socioeconômicas de regiões de 200 cidades dos Estados Unidos usando 50 milhões de imagens de cenas de rua reunidas com carros do *Google Street View* (GEBRU et al., 2017). O DL também teve impactos em diversas áreas da ciência, desde pesquisa em física de partículas (BALDI; SADOWSKI; WHITESON, 2014), à medicina (Nassif et al., 2019).

É interessante ressaltar que apesar de todos os benefícios oferecidos pela IA, alguns indivíduos notáveis como o famoso físico Stephen Hawking, e o líder da Tesla e da SpaceX Elon Musk, sugerem que a IA pode ser potencialmente muito perigosa. De fato, existem muitos aplicativos de IA que tornam nossa vida cotidiana mais conveniente e eficiente. São os aplicativos de IA que desempenham um papel crítico para garantir a segurança que Musk, Hawking e outros estavam preocupados quando proclamaram sua hesitação sobre a tecnologia. Por exemplo, se a IA for responsável por garantir a operação de nossa rede elétrica, de uma usina nuclear ou outro sistema de alto risco, e a IA for invadida ou tiver seus objetivos desalinhados com os nossos, isso poderá resultar em danos enormes (MARR, 2018).

Apesar de todo o medo ao redor dessa nova tecnologia, muitos argumentam que os mesmos são exagerados e que os benefícios oferecidos são muito maiores que os potenciais riscos, desde que sejam gerenciados adequadamente. O crescimento de pesquisas em DRL revelam seu grande potencial e benefícios para a sociedade. Reproduzir e comparar os trabalhos existentes existente e julgar com precisão as melhorias oferecidas por novos métodos é vital para sustentar esse progresso.

No contexto de jogos digitais, treinar um agente para superar os jogadores humanos e otimizar sua pontuação pode nos ensinar como otimizar processos diferentes em uma variedade de subcampos diferentes e intrigantes (COMI, 2018). Os impactos econômicos e sociais que essas técnicas podem oferecer são diversos.

Situações do mundo real são muitas vezes complexas e apresentam problemas com um número muito grande de variáveis. Para tais problemas, encontrar a melhor solução pode ser um desafio muito grande para algoritmos de otimização tradicionais. Uma vez que se tenha um sistema capaz de aprender e elaborar estratégias para diferentes cenários, é fácil modelar problemas que possam ser resolvidos pelo mesmo. No caso, uma IA que possa aprender a jogar e a otimizar estratégias para maximizar a pontuação de um jogo, pode ser aplicada em um jogo que simule uma situação real e encontrar a melhor resposta ou solução para um dado problema.

Imagine, por exemplo, um jogo que simule o trânsito em uma cidade, e a pontuação desse jogo é calculada de acordo com a elaboração das rotas de ônibus, as quais devem alcançar o maior número de áreas da cidade e minimizar o tempo de cada trajeto. A IA proposta seria, idealmente, capaz de encontrar a melhor organização possível dessas rotas. Na área da biomedicina e química, poderíamos modelar um jogo que simule o comportamento de uma célula cancerígena, e a IA teria o objetivo de encontrar o tratamento mais efetivo para a doença. Um jogo que simule condições extremas de temperaturas, ambiente e terreno, poderia ser aplicado ao sistema e a IA poderia propor a modelagem das máquinas que iriam se adaptar melhor à dadas condições. Essas máquinas, por sua vez, poderiam ser utilizadas em diversas expedições espaciais ou de alta profundidade, por exemplo. O sistema proposto, portanto, poderia idealmente ser aplicado para quaisquer cenários ou jogos, os quais podem ser modelados para serem mais ou menos complexos, de forma a melhor atender a necessidade do usuário.

Em resumo, o DL já é utilizado com sucesso em diversas áreas da ciência, otimizando e solucionando diferentes problemas. Ao propor um sistema que seja flexível e capaz de se adaptar à diferentes cenários, seria capaz de unificar muitas dessas ferramentas em uma única. A mesma ferramenta poderia ser aplicada nas diferentes situações mencionadas anteriormente e propor soluções para inúmeros problemas, melhorar produtos já existentes e otimizar processos no mundo real.

3 Abordagem Proposta

No contexto de jogos digitais, treinar um agente para superar os jogadores humanos e otimizar sua pontuação pode nos ensinar como otimizar processos diferentes em uma variedade de subcampos intrigantes (COMI, 2018). Uma solução proposta na literatura, obtendo ótimos resultados, e que tem como objetivo treinar um computador pra aprender e desenvolver estratégias para jogar diferentes jogos, é o *deep reinforcement learning* (DRL).

No presente trabalho é proposto a implementação de uma inteligência artificial que, utilizando um algoritmo de *deep reinforcement learning*, seja capaz de aprender a jogar diferentes jogos e desenvolver estratégias para maximizar sua pontuação.

Diante das peculiaridades e restrições do problema discutidos em **1.3**, a biblioteca *Allegro* foi escolhida como a base para a implementação dos jogos que serão apresentados ao sistema. O *Allegro* é uma biblioteca multiplataforma destinada principalmente a jogos de vídeo e programação multimídia. A biblioteca fornece rotinas de baixo nível comumente necessárias na programação de jogos, como a criação de janelas, aceitação de entrada do usuário, carregamento de dados, desenho de imagens, reprodução de sons etc (HARGREAVES, 1990).

A IA será treinada apartir de capturas de tela em diferentes estado do jogo, e da pontuação obtida. Esses dados serão obtidos a partir de um "*Allegro Learning Enviroment*" (ALE), o qual consiste de uma ferramenta para o desenvolvimento de inteligência artificial em jogos implementados em *Allegro*. Seu objetivo é oferecer uma plataforma que facilite o desenvolvimento de algoritmos de ML para jogos em *Allegro*, o que irá auxiliar a implementação do sistema.

A **Figura 1** mostra a arquitetura da abordagem proposta. Inicialmente, a ALE irá extrair os comandos básicos do jogo para que o agente tenha conhecimento das limitações físicas do ambiente no qual ele será inserido. Uma vez que o treinamento seja iniciado, a ALE será responsável por obter as capturas de tela que conterão informações sobre o estado atual do jogo, assim como a pontuação obtida pelo agente. Com esses dados, o agente deverá elaborar uma política de decisão para tomar uma ação em cada estado. A ação tomada pelo agente será passada para o jogo, que irá atualizar o seu estado de acordo. Esse ciclo continua até o jogo ser finalizado (seja pelo sucesso ou falha do agente), e uma pontuação final ser obtida. O treinamento do agente consiste na repetição desse processo de modo que a IA, utilizando técnicas de *reinforcement learning*, seja capaz de elaborar uma estratégia para maximizar a pontuação final.

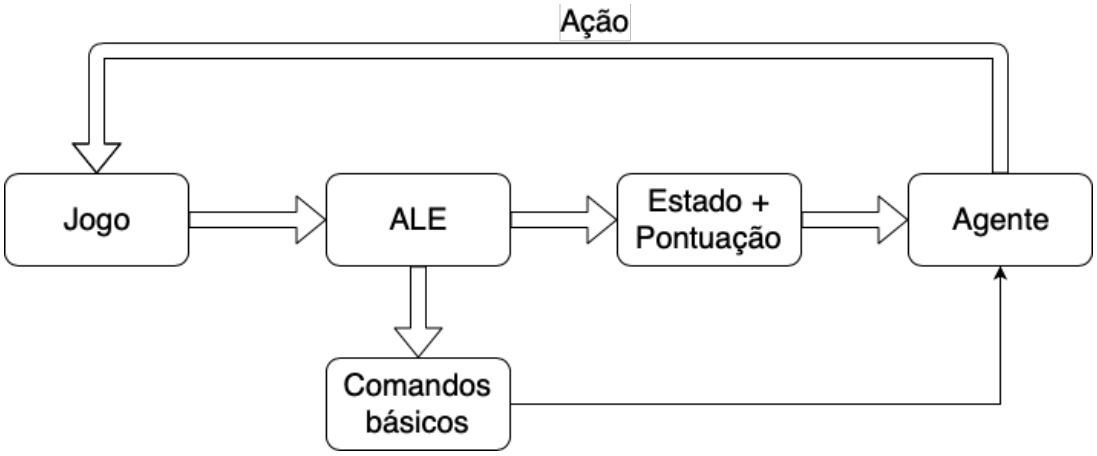

Figura 1: Arquitetura da abordagem proposta. A partir do código fonte do jogo, a ALE extrai os comandos básicos do jogo para que o agente tenha conhecimento de suas limitações físicas. Durante o processo de treinamento, para cada estado do jogo, a ALE passa as informações sobre o estado atual do jogo e a pontuação obtida até então. Com essas informações, o agente toma uma ação que irá influenciar o próximo estado do jogo

3.1 *Deep Learning*

O *deep learning* (DL) é uma área do aprendizado de máquina que propõe que os computadores aprendam com a experiência, se ajustem à novas entradas de dados e compreendam o mundo em termos de hierarquia de conceitos, sendo cada conceito definido por sua relação com conceitos mais simples. Ao reunir conhecimento a partir da experiência, essa abordagem evita a necessidade dos operadores humanos de especificar formalmente todo o conhecimento que o computador precisa. Além disso, a hierarquia de conceitos permite que o computador aprenda conceitos complexos, construindo-os a partir de conceitos mais simples. O *deep learning* apresenta grande poder e flexibilidade a nos permitir o treinamento de computadores para cumprir tarefas específicas ao processar grandes quantidades de dados e reconhecer padrões nesses dados.

A **Figura 2** mostra como um sitema de *deep learning* representa o conceito de imagem de uma pessoa combinando conceitos mais simples, como cantos e contornos, que por sua vez são definidos em termos de arestas.

O mapeamento de funções de um conjunto de pixels para uma identidade de objeto é uma tarefa complicada. O algoritmo de *deep learning* resolve essa dificuldade dividindo o mapeamento complicado desejado em séries de mapeamentos simples aninhados, cada um deles descrito por uma camada diferente do modelo. A entrada é apresentada na camada visível, em seguida, uma série de camadas ocultas extrai recursos cada vez mais abstratos da imagem. A camada de saída obtém a identidade de objeto abstrata a partir dos conceitos obtidos pelas camadas ocultas.

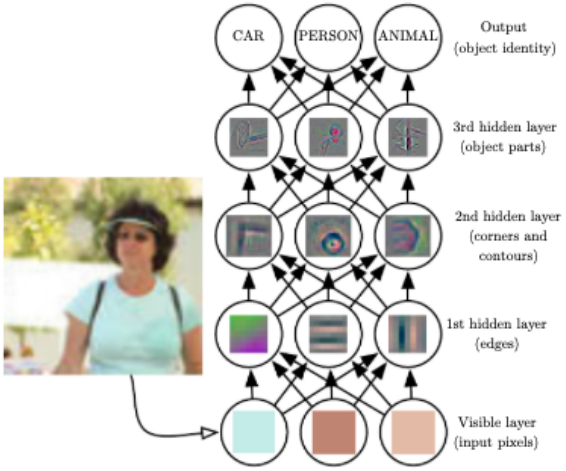

Figura 2: Ilustração de um modelo de aprendizado profundo. Cada camada é capaz de identificar dados de complexidade crescente a partir dos pixels pasados para a camada de entrada. Imagem retirada de (GOODFELLOW; BENGIO; COURVILLE, 2016)

O reconhecimento de imagens a partir da extração de padrões de pixels brutos, será crucial para o agente no processo de análise do estado atual do jogo. Como mencionado anteriormente, as informações do ambiente e estado atual do jogo serão extraídas a partir de capturas de tela em cada estado. A partir dessas capturas de tela, a IA deverá reconhecer padrões e identificar obstáculos, caminhos disponíveis e objetivos a serem alcançados dentro do jogo. A utilização de capturas de tela permite ao agente se adaptar à cenários onde os obstáculos e caminhos disponíveis são elaborados de forma aleatória, e sua disposição se altera a cada iteração do jogo.

3.2 *Reinforcement Learning*

O aprendizado por reforço ou *reinforcement learning* (RL) é uma abordagem computacional para entender e automatizar o aprendizado direcionado a objetivos e a tomada de decisões. O aprendizado por reforço distingue-se de outras abordagens computacionais por sua ênfase na aprendizagem de um agente apartir da interação direta com seu ambiente, sem exigir supervisão exemplar ou modelos completos do ambiente (SUTTON; BARTO, 2018).

Em algoritmos de *reinforcement learning*, o agente não é informado sobre quais ações executar, mas, em vez disso, deve descobrir quais ações geram mais recompensa, através de tentativa e erro. Em alguns casos mais interessantes, as ações podem afetar não apenas a recompensa imediata, mas também a próxima situação e, com isso, todas as recompensas subsequentes. Essas duas características - pesquisa por tentativa e erro e recompensa atrasada - são as duas características distintivas mais importantes do aprendizado por

reforço.

O aprendizado por reforço é diferente do aprendizado supervisionado, o tipo de aprendizado estudado na maioria das pesquisas atuais no campo do aprendizado de máquina. Aprendizado supervisionado é aprender com um conjunto de treinamento de exemplos rotulados fornecidos por um supervisor externo qualificado. O objetivo desse tipo de aprendizado é o sistema extrapolar ou generalizar suas respostas para que ele atue corretamente em situações não presentes no conjunto de treinamento. Este é um tipo importante de aprendizado, mas por si só não é adequado para aprender com a interação. Em problemas interativos, muitas vezes é impraticável obter exemplos do comportamento desejado que sejam corretos e representativos de todas as situações nas quais o agente precisa agir. Em um território desconhecido - onde se espera que a aprendizagem seja mais benéfica - um agente deve ser capaz de aprender com sua própria experiência.

O aprendizado por reforço também é diferente do que os pesquisadores de aprendizado de máquina chamam de aprendizado não supervisionado, que geralmente consiste em encontrar estruturas ocultas em coleções de dados não rotulados. Os termos aprendizado supervisionado e aprendizado não supervisionado parecem classificar exaustivamente os paradigmas de aprendizado de máquina, mas não o fazem. Embora se possa ficar tentado a pensar no aprendizado por reforço como um tipo de aprendizado não supervisionado, porque não se baseia em exemplos de comportamento correto, o aprendizado por reforço está tentando maximizar um sinal de recompensa em vez de tentar encontrar uma estrutura oculta. Descobrir a estrutura na experiência de um agente certamente pode ser útil no aprendizado por reforço, mas por si só não aborda o problema do aprendizado por reforço de maximizar um sinal de recompensa. Portanto, o aprendizado por reforço é considerado como um terceiro paradigma de aprendizado de máquina, ao lado de aprendizado supervisionado, aprendizado não supervisionado e talvez outros paradigmas (SUTTON; BARTO, 2018).

A **Figura 3** mostra um diagrama de aprendizagem por reforço relacionando o agente de aprendizado com o ambiente no qual ele é inserido. O ambiente representa o mundo pelo qual o agente se move. O ambiente nada mais é do que um sistema que toma o estado atual e a ação do agente como entrada e retorna como saída a recompensa do agente e seu próximo estado.

Ambientes podem ser modelados como funções que transformam uma ação executada no estado atual, no próximo estado e uma recompensa. Já os agentes podem ser modelados como funções que transformam o novo estado e recompensam na próxima ação. Podemos conhecer a função do agente, mas não podemos conhecer a função do ambiente. É uma caixa preta onde só vemos as entradas e saídas. O aprendizado por reforço representa a tentativa de um agente de aproximar a função do ambiente, para que possamos enviar ações para o ambiente de caixa preta que maximize as recompensas que ele distribui

(NICHOLSON, 2016).

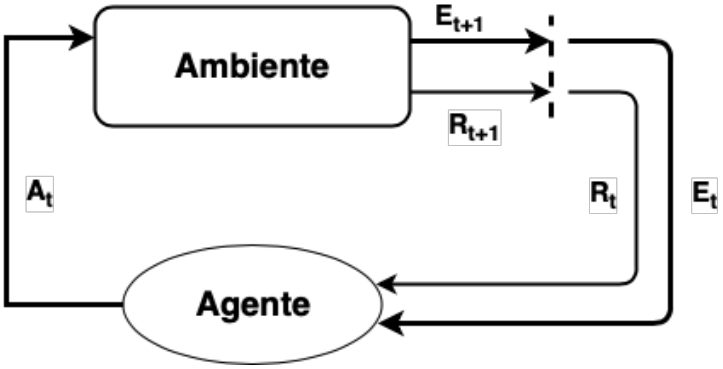

Figura 3: Diagrama de aprendizagem por reforço. No *loop* de *feedback* acima, os subscritos indicam as etapas de tempo t e $t+1$, cada uma das quais se refere a estados diferentes: o estado no momento t e o estado no momento $t+1$. A ação A_t de um agente é determinada por sua **política**, que por sua vez é uma função que depende do estado atual do sistema E_t. A política de um agente tem como objetivo maximizar a **função de valor** que é calculada utilizando o **sinal de recompensa** R_t. O ambiente se comporta como um sistema caixa preta que transforma uma ação executada no estado atual A_t, no próximo estado E_{t+1} e uma recompensa R_{t+1}

As escolhas de ação são feitas com base em julgamentos de valor. Buscamos ações que gerem estados de maior valor, e não de maior recompensa, porque essas ações obtêm a maior quantidade de recompensa a longo prazo. As recompensas são basicamente dadas diretamente pelo ambiente, mas os valores devem ser estimados e re-estimados a partir das sequências de observações que um agente faz ao longo de toda a sua vida útil.

Além do agente e do ambiente, é interessante ressaltar outros elementos importantes de um sistema de aprendizado por reforço: a **política**, o **sinal de recompensa** e a **função de valor**.

A **política** define a maneira que o agente deve se comportar em um determinado momento. Uma política é basicamente um mapeamento dos estados do ambiente para as ações a serem tomadas quando nesses estados. A política em casos mais simples ter a forma de uma função simples ou uma tabela de pesquisa, enquanto em casos mais complexos pode envolver cálculos mais extensivos. Em geral, as políticas podem ser estocásticas, especificando probabilidades para cada ação.

Um **sinal de recompensa** define o objetivo de um problema de aprendizado por reforço. Em cada etapa, o ambiente envia ao agente de aprendizado por reforço um único número que funciona como uma recompensa para o agente. O único objetivo do agente é maximizar a recompensa total que recebe a longo prazo. O sinal de recompensa define, portanto, quais são os eventos bons e ruins para o agente. O sinal de recompensa é a base principal para alterar a política - se uma ação selecionada pela política for seguida por

uma baixa recompensa, a política poderá ser alterada para selecionar outra ação nessa situação no futuro. Em geral, os sinais de recompensa podem ser funções estocásticas do estado do ambiente e das ações tomadas.

Enquanto o sinal de recompensa indica o que é bom em um sentido imediato, uma **função de valor** especifica o que é bom a longo prazo. O valor de um estado representa a quantidade total de recompensa que um agente pode esperar acumular no futuro, a partir desse estado. Enquanto as recompensas determinam a conveniência imediata e intrínseca dos estados ambientais, os valores indicam a conveniência a longo prazo dos estados após levar em conta os estados que provavelmente seguirão e as recompensas disponíveis nesses estados. Por exemplo, um estado sempre pode gerar uma recompensa imediata baixa, mas ainda tem um valor alto porque é seguido regularmente por outros estados que produzem recompensas altas. Ou o contrário poderia ser verdade.

O *deep reinforcement learning* (DRL) é uma abordagem do *deep learning* que, em contraste a abordagens mais tradicionais como o aprendizado supervisionado e não supervisionado, utiliza as técnicas de aprendizagem por reforço para treinar o agente. Essa abordagem consiste em fornecer ao sistema parâmetros relacionados ao seu estado e uma recompensa positiva ou negativa com base em suas ações. Nenhuma regra sobre o jogo é dada e, inicialmente, o agente não tem nenhuma informação sobre o que precisa fazer. O objetivo do sistema é descobrir e elaborar uma estratégia para maximizar sua pontuação - ou recompensa.

3.3 *Allegro*

O *Allegro* é uma biblioteca multiplataforma destinada principalmente a jogos de vídeo e programação multimídia. A biblioteca fornece rotinas de baixo nível comumente necessárias na programação de jogos, como a criação de janelas, aceitação de entrada do usuário, carregamento de dados, desenho de imagens, reprodução de sons etc (HARGREAVES, 1990). Algumas outras características da biblioteca que facilitam a implementação de jogos são:

- Suportada em Windows, Linux, Mac OS, iPhone e Android;
- API intuitiva e amigável, utilizável em C, C ++ e em muitas outras linguagens;
- Bitmap acelerado por hardware e suporte a desenho gráfico primitivo (via OpenGL ou Direct3D);
- Suporte de gravação de áudio;
- Carregamento e desenho de fontes;

- Reprodução de vídeo.

A implementação de uma ferramenta de aprendizado voltada para jogos em *Allegro*, é facilitada pelo fato de o *Allegro* ser simples e amigável, o que permite a extração dos comandos básicos do jogo a partir do seu código fonte, funcionalidade essencial para a implementação de uma ferramenta genérica.

Outro motivo que levou à escolha da biblioteca como requisito importante do projeto, é a funcionalidade de controle de *frame rates*. Um jogo que possua uma alta taxa de quadros, ou *frames per second* (FPS), apresenta uma saída com um grande número de informação em um intervalo curto de tempo. Esse alto fluxo de informações pode sobrecarregar a IA, ou até mesmo apresentar perda de informação (*frame drops*), o que pode resultar em um treinamento ineficiente em uma arquitetura com baixo poder computacional. Ao reduzir o FPS do jogo, podemos diminuir a velocidade com que os estados do jogo são atualizados, diminuindo a quantidade de informação que deve ser tratada e permitindo o desenvolvimento de uma IA em arquiteturas com menor poder computacional.

3.4 Aplicação de DRL em um *Allegro Learning Enviroment*

O *Arcade Learning Enviroment* é uma ferramenta de software que oferece uma interface para interagir com ambientes de jogos Atari 2600 emulados. Seu objetivo é oferecer uma plataforma que facilite o desenvolvimento de agentes de aprendizado para aprender a jogar jogos Atari. Essa ferramenta também fornece uma camada de manipulação de jogos que transforma cada jogo em um problema padrão de aprendizado por reforço, identificando a pontuação acumulada e se o jogo terminou. (BELLEMARE et al., 2012)

Inspirado na plataforma descrita acima, este trabalho visa a utilização de um *Allegro Learning Enviroment*, que funcionaria de forma semelhante ao *Arcade Learning Enviroment*, com a distinção de que o primeiro seria uma plataforma voltada para jogos implementados exclusivamente em *Allegro*.

O *Allegro Learning Enviroment* (ALE) terá como base a ferramenta implementada por (SILVA, 2019), que oferece um ambiente facilitador ao estudo de soluções de IA aplicada em jogos. Essa ferramenta fornece funcionalidades como a exportação dos comandos básicos de um jogo, que precisam ser passados para o agente para que o mesmo tenha conhecimento dos limites físicos do ambiente no qual está inserido. Isso permite que o pesquisador não fique limitado a um jogo existente, mas possa usar qualquer jogo que ele tenha acesso ao código fonte e feito em *Allegro*. Outra funcionalidade fornecida pelo ALE, é a possibilidade de se extrair a pontuação e obter capturas de tela em cada estado do jogo, informações que devem ser fornecidas para o treinamento do agente.

Para o treinamento do agente, serão utilizados capturas da tela em cada estado do jogo, obtidas pelo ALE. A partir dessas imagens serão extraídas as informações do estado atual do jogo (posição do jogador, obstáculos, etc), de forma a determinar qual a melhor ação do agente para a situação na qual ele se encontra. A utilização de capturas de tela como entradas para o agente permite que a IA seja treinada para situações em que hajam obstáculos gerados de forma aleatória. A partir dessas imagens, o agente deverá ser capaz de identificar tais obstáculos, sua localização e a melhor maneira de lidar com os mesmos.

Em relação aos jogos nos quais o agente será treinado, a proposta é de inicializar o treinamento com jogos mais simples (e.g. *Snake, Frogger, Agar.io*). Se possível, neste trabalho ou em trabalhos posteriores, a ideia é de se utilizar diferentes jogos de diferentes complexidades para avaliar o potencial do sistema. Os jogos serão obtidos de fontes de código aberto disponíveis *online* ou, caso seja necessário, serão implementados com os requisitos necessário para o projeto.

4 Comentários finais TCC I e Proposta para o TCC II

O problema proposto nesse trabalho é o de implementar uma IA que, utilizando algoritmos de *deep reinforcement learning*, seja capaz de aprender e desenvolver estratégias para jogar diferentes jogos digitais. A IA proposta deverá ser genérica, ou seja, capaz de aprender a jogar diferentes jogos, desde que se tenha acesso ao codigo fonte e que sejam implementados em *Allegro*. Para auxiliar na implementação do sistema será utilizado um *Allegro Learning Enviroment*, plataforma que irá facilitar a implementação da ferramenta para o treinamento do agente.

Diante disso, utilizando técnicas de DRL existentes, espera-se produzir uma IA que seja flexível e que possa ser adaptada para diferentes cenários. Neste sentido, espera-se uma IA que seja genérica e capaz de ser treinada para diversos jogos. Por fim, será feita uma análise crítica dos resultados e uma comparação dos mesmos com trabalhos semelhantes realizados por outras entidades.

4.1 Proposta TCC 2

Este trabalho tem como continuidade o desenvolvimento do Trabalho de Conclusão de Curso II, onde haverá um maior detalhamento sobre a modelagem matemática do problema, além de especificadas as decisões de implementação da ferramenta elaborada, bem como uma análise dos resultados.

A abordagem, além do que já foi exposto, consistirá na implementação da rede neural proposta, utilizando os algoritmos DRL mencionados para o treinamento do agente. Também serão implementados (se necessário), diferentes jogos em *Allegro* para a validação do sistema. Por fim, o agente será treinado em jogos simples e de baixa complexidade e serão apresentados os resultados obtidos para avaliar o potencial do sistema. Em trabalhos futuros, a ferramenta poderá também ser utilizada para o treinamento em jogos de diferentes complexidades, ampliando ainda mais o alcance do sistema.

4.2 Cronograma TCC 2

Atividades	Meses			
	Agosto	Setembro	Outubro	Novembro
Levantamento bibliográfico	X			
Pesquisa e implementação da rede neural proposta do trabalho	X			
Aplicação do estudo realizado no assunto do TCC a ser desenvolvido	X			
Entrega da Visão Geral do Trabalho	21/08/2020			
Desenvolvimento do trabalho e implementação da rede neural proposta do trabalho	X	X		
Elaboração do corpo principal do TCC		X	X	
Entrega do FomulárioPonto de Controle		11/09/2020		
Marcação da defesa		25/09/2020		
Ajustes finais e conclusão to trabalho implementado			X	
Emissão da versão inicial do TCC			17/10/2020	
Preparação do material referente à apresentação do TCC			X	
Apresentação oral para banca examinadora			Semana de 26 a 30/10	
Ajuste no material relativo ao trabalho escrito				06/11/2020

5 Modelagem e Implementação

Como mencionado no **Capítulo 3**, a proposta do presente trabalho consiste na implementação de um algoritmo de *deep reinforcement learning* para treinar um agente que seja capaz de aprender a jogar um jogo em *Allegro*. A inspiração para o mesmo vem do trabalho realizado pelo *Deep Mind* e publicado no artigo (MNIH et al., 2013), onde foi implementado uma IA capaz de jogar diferentes jogos Atari 2600. Assim, será implementado um sistema semelhante voltado para jogos em *Allegro*.

O atual capítulo consiste em uma contextualização do ambiente e sistema a ser implementado, seguido de uma elaboração da modelagem matemática da abordagem proposta no **Capítulo 3** e, por fim, uma discussão sobre como o sistema foi implementado.

5.1 *Contextualização*

5.1.1 O Jogo

O ambiente escolhido para treinamento do modelo foi o jogo *Frogger*. A escolha do mesmo foi feita tendo em vista sua simplicidade, tendo em vista as limitações de implementação (**Seção 5.3.5**). A **Figura 4** mostra a posição inicial do jogo utilizado. O agente controla o quadrado verde iniciado no centro inferior da tela, e tem como objetivo alcançar o topo da tela sem colidir com nenhum obstáculo, os quais são iniciados com tamanho e posições aleatórias.

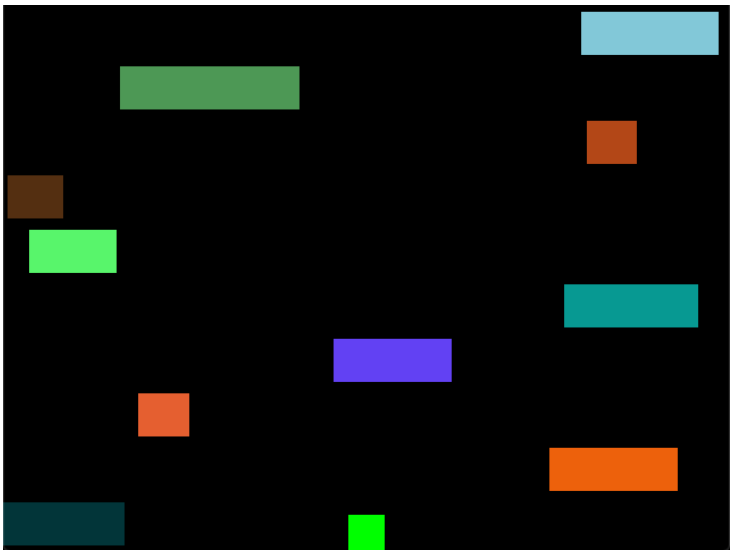

Figura 4: Exemplo do jogo *Frogger* utilizado. O jogador controla o quadrado verde no centro inferior da tela, enquanto os outros retângulos coloridos são os obstáculos

As condições de parada, ou seja, os estados que constituem os estados finais do jogo são:

- Estados onde ocorra uma colisão do agente com o ambiente;
- Estado onde o agente atravessou o topo da tela (uma posição acima da última linha observável do jogo).

Por fim, as recompensas para ações durante o jogo foram definidas inicialmente da seguinte forma:

- $r = -0.05$ caso o agente realize uma ação que o mantenha na mesma linha que se encontrava previamente. Essa recompensa negativa foi estipulada com o objetivo a desmotivar o agente a permanecer longos períodos de tempo sem progredir;
- $r = 1$ caso o agente realize uma ação que o aproxime verticalmente de seu objetivo;
- $r = -1$ caso o agente realize uma ação que o distancie verticalmente de seu objetivo;
- $r = -1$ caso o agente realize uma ação que o leve a colidir com algum obstáculo (independente da direção que se movimentou);
- $r = 10$ caso o agente alcance seu objetivo.

5.1.2 *Allegro Learning Enviroment*

Conforme descrito no **Capítulo 3**, será utilizada um *Allegro Learning Enviroment* (ALE) que funcionará como intermédio entre o jogo e a IA. Essa ferramente tem como base a ferramente implementada por (SILVA, 2019), que foi modificada para atender as necessidades especificas do sistema atual. O ALE terá as seguintes funções principais:

- Fornecer os valores de $\mathcal{A} = \{1, \cdots, K\}$, que representa o conjunto de ações possíveis para o agente;
- Determinar a posição da tela do jogo e realizar a captura das imagens que servirão como observações de estado para o agente;
- Executar um novo jogo para cada episódio de treinamento;
- Executar as ações estabelecidas pelo agente;
- Processar os dados do jogo incluindo: observação de cada instante de tempo, calcular a recompensa do atual estado do jogo, informação sobre quando um episódio é finalizado.

5.2 Modelagem Matemática

Como elaborado anteriormente, no aprendizado por reforço é desenvolvido um agente que irá interagir com um ambiente ε, nesse caso o jogo em *Allegro*, a partir de uma sequência de ações, observações e recompensas. Em cada etapa de tempo, o agente seleciona uma ação do conjunto de ações legais do jogo, $\mathcal{A} = \{1, \cdots, K\}$. A ação é executada, modificando o estado do ambiente e pontuação do jogo. O estado interno do jogo não é observado pelo agente, este observa apenas uma imagem $x_t \in \mathbb{R}^d$, que é um vetor de valores de pixel brutos que representam a tela do estado atual do jogo. Além disso, o agente recebe uma recompensa r que representa a alteração na pontuação do jogo.

Em outras palavras, um agente explora um jogo, e é treinado tentando maximizar as recompensas nesse jogo. Este ciclo é ilustrado na **Figura 5**.

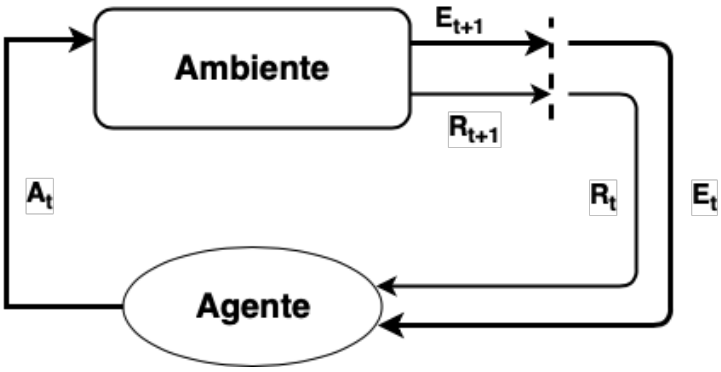

Figura 5: Diagrama de aprendizagem por reforço elaborada melhor no **Capítulo 3**

É importante ressaltar que a pontuação do jogo pode depender de toda a sequência anterior de ações e observações. O *feedback* sobre uma ação só pode ser recebido depois de decorridos múltiplos de intervalos de tempo. Uma vez que o agente apenas observa as imagens da tela atual, a análise do estado do jogo em que se encontra pode ser mal-representada, ou seja, é difícil para o agente compreender totalmente a situação em que se encontra apenas da tela atual x_t. Para solucionar esse problema, considera-se como um estado s_t do jogo, uma sequência de ações e observações $s_t = (x_{t-n}, a_{t-n}, \cdots, a_{t-1}, x_t)$, as quais serão utilizadas para treinar o agente, fornecendo-o um melhor contexto do estado em que se encontra. Esse formalismo dá origem a um processo de decisão de Markov (MDP), no qual cada sequência é um estado distinto. Como resultado, pode-se aplicar métodos de aprendizado por reforço padrão para MDPs, simplesmente usando a sequência completa s_t como a representação do estado no tempo t.

Conforme descrito em (MNIH et al., 2013), o objetivo do agente é interagir com o jogo, selecionando ações de uma forma que maximize recompensas futuras. É feita a suposição padrão de que as recompensas futuras são descontadas por um fator de γ por

intervalo de tempo, e que o retorno descontado futuro é definido por:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} \cdot r_{t'} \tag{5.1}$$

onde T é o intervalo de tempo em que o jogo termina.

A função de valor de ação ótima $Q^*(s, a)$ pode ser definida como o máximo retorno esperado alcançável de uma estratégia, depois de ver a sequência s e se tomar alguma ação a:

$$Q^*(s, a) = max_\pi(\mathbb{E}[R_t|s_t = s, a_t = a, \pi]) \tag{5.2}$$

onde π é uma política que mapeia sequências para ações e $\mathbb{E}$ é a função de retorno esperado para um estado s dado uma ação a.

A função de valor de ação ótima obedece a identidade da equação de Bellman. Essa se baseia na seguinte intuição: se o valor ótimo $Q^*(s_{t+1}, a_{t+1})$ da sequência s_{t+1} na próxima etapa de tempo for conhecido para todas as ações possíveis ações a_{t+1}, então a estratégia ótima para o estado s_t consiste em selecionar a ação a_t que maximize o valor esperado futuro:

$$Q^*(s_t, a_t) = r + \gamma \cdot max(Q^*(s_{t+1}, a_{t+1})|\forall a_{t+1}) \tag{5.3}$$

A ideia básica por trás de muitos algoritmos de aprendizagem por reforço é estimar a função de valor de ação, usando a equação de Bellman como uma atualização iterativa. Assim, dado um fator de aprendizagem α, o valor de $Q(s, a)$, é atualizado durante o treinamento da seguinte forma:

$$Q_{i+1}(s_t, a_t) = Q(s_t, a_t) + \alpha[r + \gamma \cdot max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{5.4}$$

sendo que a subtração de $\gamma \cdot max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$ por $Q(s_t, a_t)$ é realizada para normalizar a atualização.

Essa atualização dos valores da função de valor Q, permite que o algoritmo convirja para a função de ação ótima $Q_i \rightarrow Q^*$, com $i \rightarrow \infty$ (SUTTON; BARTO, 1998). Na prática, essa abordagem é totalmente impraticável, pois a função valoração é estimada separadamente para cada sequência, sem nenhuma generalização. Em vez disso, é comum usar um aproximador de função para estimar a função de valor de ação, $Q(s, a; \theta_t) \approx Q^*(s, a)$. Este aproximador toma a forma de uma rede neural, conhecida como uma *Q-network*, onde θ_t refere-se aos parâmetros da rede neural (ou seja, os pesos e bias da rede). Assim, se os pesos são atualizados após cada passo de tempo, então chegamos ao conhecido algoritmo de *Q-learning* (WATKINS; DAYAN, 1992). O valor de θ_t por sua vez pode ser atualizado conforme a **Equação 5.5**:

$$\theta_t = \theta_t + \alpha(r + \gamma \cdot max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))\nabla_\theta Q(s_t, a_t) \tag{5.5}$$

Por fim, considerando a iteração de treinamento do agente usando *Q-learning*, a política de seleção de ação é baseada apenas no valor máximo de Q em qualquer estado dado. É concebível que, dada a inicial natureza estocástica do ambiente, o agente inicialmente tome decisões "ruins". No entanto, como o agente nunca explorou ações melhores, ele pode interpretar que as decisões ruins tomadas inicialmente são boas e, daquele momento em diante, tomar somente aquelas decisões. Da mesma forma, mesmo que o agente tome ações iniciais "boas" ele pode acabar desenvolvendo uma política que prenda o agente em ótimos locais, ou seja, a ação determinada pela política é boa a curto prazo, mas a longo prazo ela é inferior ou até ruim. Esta política de seleção de ação é chamada de política gulosa.

Para evitar esses problemas, o agente deve explorar o maior número possível de estados e ações. Infelizmente, para problemas com um número muito grande de estados, explorar todos os estados e ações possíveis se torna impraticável. A política *epsilon-greedy* na aprendizagem por reforço é basicamente a mesma que a política gulosa, exceto que há um valor épsilon que define uma probabilidade $1 - \epsilon$ de uma ação ser escolhida aleatoriamente ao invés de definida pelo atual modelo. Dessa forma, o algoritmo força o agente a explorar diversos estados e ações, mesmo que estes não sejam a princípio recomendados pela função de valor Q. Dado um número suficientemente grande de ações, o agente terá explorado uma quantidade de estados suficiente para determinar uma política satisfatória. Essa abordagem permite também que o valor de ϵ seja atualizado e reduzido com o tempo de forma a permitir que o algoritmo se concentre mais em explorar as melhores soluções que encontrou.

5.3 Implementação

A modelagem descrita na **Seção 5.2**, fornece uma modelagem da rede neural proposta. O sistema nada mais é do que uma *Deep Q Network* (DQN), que é constituída por uma *Deep Neural Network* (GOODFELLOW; BENGIO; COURVILLE, 2016) que utiliza a técnica de *Q-learning*, onde cada nó de saída da rede corresponde à uma possível ação do agente. No entanto, um simples algoritmo usando o método descrito não converge para uma solução satisfatória. Nessa seção serão vistas algumas modificações para o algoritmo que o fará obter melhores resultados.

5.3.1 *Experience Replay*

Foi aplicada a técnica conhecida como *Experience Replay* (LIN, 1992), onde as experiências do agente em cada passo de tempo, são armazenadas em um *replay buffer*, que nada mais é do que uma lista de tuplas, cada tupla contendo o estado atual, a ação tomada, a recompensa obtida, e o próximo estado alcançado (s_t, a_t, r_t, s_{t+1}). Para treinar a DQN, recupera-se aleatoriamente um pequeno lote do *replay buffer*, o qual é usado como

dado de treinamento. A **Figura 6** mostra o processo descrito.

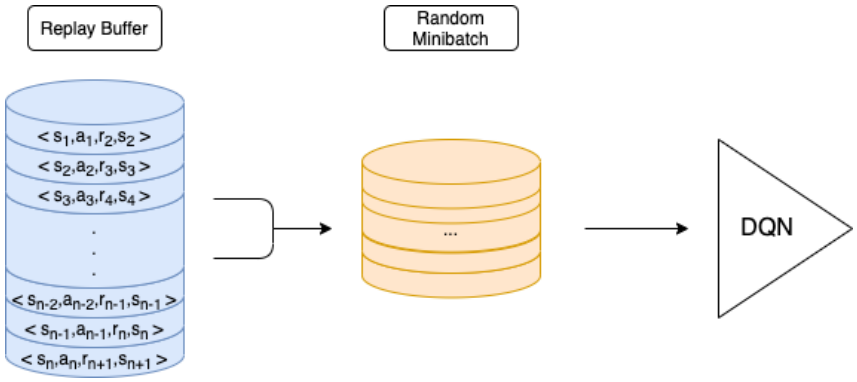

Figura 6: Exemplo do processo de seleção de dados de treinamento. O histórico de estados, ações e recompensas obtido é salvo em um *replay buffer*. Para treinar o agente, o algoritmo seleciona aleatoriamente um conjunto de amostras (*minibatch*) de tamanho pré-definido que servirão como dados de treinamento da rede

O *buffer* implementado dessa forma é uma fila FIFO (*First in, first out*), ou seja, quando ele atingir seu tamanho máximo, as novas entradas irão substituir as entradas mais antigas de forma a sempre manter o conjunto de dados mais atualizado. Além disso, antes de se inicializar o treinamento, o *buffer* é inicializado com um tamanho mínimo, inicializado com amostras obtidas por um agente com política de ação aleatória.

O objetivo da alocação dos dados dessa forma é de fornecer à rede uma melhor distribuição de informação para o aprendizado. Um algoritmo de aprendizado simples recebe os dados de treinamento na mesma ordem em que são observados. Isso pode introduzir ao agente padrões ou correlações indesejadas, que podem afetar o processo de aprendizado do algoritmo. Assim, ao passar as observações de forma aleatória, a eficiência do algoritmo é melhorada.

5.3.2 Pré-processamento de Imagens

Para treinar uma rede neural apartir de capturas de tela em cada instante do jogo, os dados de cada observação são extraídos dos pixels brutos exibidos. Um problema de utilizar imagens estáticas como observações é que isso limita a quantidade de informação que pode ser obtida do estado do jogo: não é possível determinar a velocidade ou direção na qual um objeto está se movendo. Para contornar essa situação, pode-se redefinir um estado como sendo um conjunto de n instantes de tempo, como mencionado na **Seção 5.2**.

Dado que a tela do jogo representa uma janela de 640×480 pixels, pode-se converter isso para um vetor de pixels RGB com dimensões $640 \times 480 \times 3$. Uma vez definido o estado como um conjunto de $n = 4$ instantes, temos um vetor de entrada para rede neural de $640 \times 480 \times 3 \times 4 = 3686400$. Considerando que cada iteração do algoritmo deverá

analisar um vetor desse tamanho, o custo computacional do treinamento do modelo é extremamente alto. Felizmente o vetor de entrada pode ser reduzido consideravelmente removendo algumas informações desnecessárias. Primeiramente, do ponto de vista do agente, a variação de cores do jogo é irrelevante. Sendo assim, pode-se converter a imagem para cinza, reduzindo o tamanho da matriz da imagem original de $640 \times 480 \times 3$ para 640×480. Por fim, podemos reduzir o tamanho dessa imagem, tendo em vista que informações redundantes de uma tela grande não são necessárias para treinamento do modelo. A **Figura 7** mostra o tratamento de uma imagem antes de ela ser adicionada ao estado que servirá de entrada para a rede.

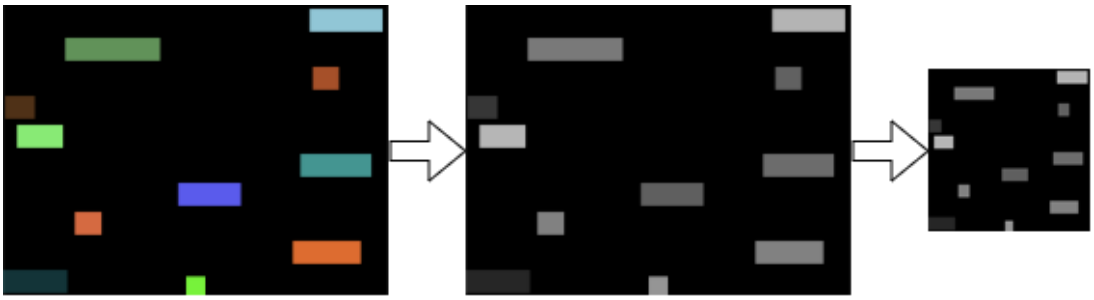

Figura 7: Processamento das capturas de tela realizado antes das mesmas serem passadas como entradas para o treinamento da rede. Primeira imagem na esquerda é um exemplo da captura original, seguindo da mesma imagem transformada para cinza e por fim a imagem reduzida que será convertida para um vetor de pixels a ser passado como entrada para o modelo

Com isso, considerando uma redução para uma imagem final de 84×84 pixels, obtém-se um estado formado por um vetor de tamanho $84 \times 84 \times 4 = 28224$. Apesar do vetor final ainda apresentar um tamanho considerável, foi alcançado uma redução de 99.23% na entrada do modelo, aliviando consideravelmente o custo computacional do sistema.

5.3.3 *Dueling Q Network*

O problema do aprendizado por DQN está ligado à definição da sua função de valor. Da **Equação 5.3** temos:

$$Q^*(s_t, a_t) = r + \gamma \cdot max(Q^*(s_{t+1}, a_{t+1})|\forall a_{t+1})$$

O problema da equação acima surge com o valor máximo. Esta parte da equação deve estimar o valor das recompensas para ações futuras se a ação a for tomada a partir do estado atual s_t. O problema é que em muitos ambientes, a existência de um ruído aleatório, ou seja, uma variância no valor da recompensa, é inevitável. Assim, à medida que um agente explora um ambiente, ele não está observando diretamente r ou r_{futuro}, mas algo como $r + \epsilon$, onde ϵ é o ruído ou o a variância da recompensa. Em tal ambiente, depois

de jogar o jogo repetidamente, esperaríamos que a rede aprendesse a fazer estimativas imparciais do valor esperado das recompensas $E[r]$. Com isso, basta à rede escolher as melhores ações para recompensas atuais e futuras, apesar da presença de ruído.

É aqui que a operação de máximo é um problema: ela produz estimativas tendenciosas das recompensas futuras, não as estimativas imparciais de que necessitamos para obter os melhores resultados. Considere o ambiente exemplificado pela **Figura 8** (THOMAS, 2020). O agente começa no estado A e em cada estado pode mover-se para a esquerda ou direita. Os estados C e E são estados terminais. O jogo termina quando esses pontos são alcançados. Os valores de r são as recompensas que o agente recebe ao fazer a transição de um estado para outro.

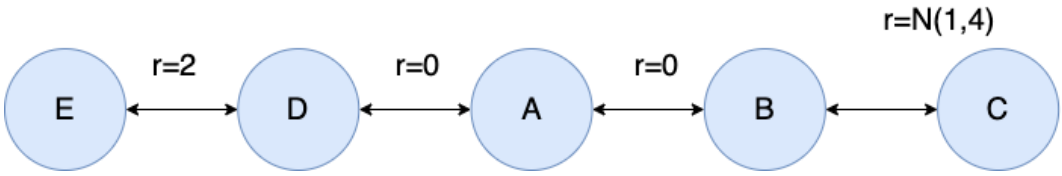

Figura 8: Exemplo de um jogo que causa bias em uma DQN. O agente começa no estado A e em cada estado pode mover-se para a esquerda ou direita. Os estados C e E são estados terminais. O jogo termina quando esses pontos são alcançados. Os valores de r são as recompensas que o agente recebe ao fazer a transição de um estado para outro. Figura extraída de (THOMAS, 2020)

Todas as recompensas são determinísticas, exceto as recompensas durante a transição dos estados B para C. As recompensas para essa transição é retirada aleatoriamente de uma distribuição normal com uma média de 1 e um desvio padrão de 4.

Sabemos as recompensas esperadas, $E[r]$ de qualquer ação de B para C é 1. No entanto, há uma variância associada à essas recompensas. Independentemente disso, em média, o agente deve aprender idealmente a sempre se mover para a esquerda de A, em direção a D e finalmente E, onde r sempre é igual a 2. No entanto, devido a função de máximo utilizada para calcular a função de valor, sempre é obtido o valor máximo dos sorteios aleatórios das recompensas e, com isso, o algoritmo tende a ser positivamente tendencioso e não dá uma indicação verdadeira dos valores esperados das recompensas para um movimento nesta direção (ou seja, 1). Como tal, um agente que usa a metodologia de *Q-learning* não escolherá a ação ideal de A (ou seja, mover para a esquerda), mas tenderá a mover para a direita.

A solução para este problema foi proposta por (HASSELT; GUEZ; SILVER, 2015), e consiste na implementação de duas redes neurais, mas somente uma rede primária é treinada à cada iteração, enquanto a segunda rede, chamada de rede objetivo, é treinada com menos frequência. Esse processo ajuda a estabilizar os pesos da rede objetivo.

A arquitetura conhecida como *Dueling Q*, é uma melhoria para a rede neural dupla.

Utiliza-se a mesma metodologia de uma rede alvo e primária, com atualizações periódicas ou combinação dos pesos da rede alvo com os pesos da rede primária. No entanto, ele incorpora dois conceitos importantes na arquitetura da rede. Estas são as funções de vantagem e valor:

- **Função de vantagem** $A(s, a)$: A função de vantagem é o benefício relativo de escolher uma determinada ação no estado s sobre as outras ações possíveis no mesmo estado.

- **Função de valor V(s)**: A função de valor é o valor de estar no estado s, independente dos benefícios relativos das ações nesse estado

A função $Q(s, a)$ se torna então, uma adição dessas duas funções:

$$Q(s, a) = V(s) + A(s, a) \tag{5.6}$$

A motivação de dividir essas duas funções explicitamente na arquitetura é que pode haver estados inerentemente bons ou ruins para o agente estar, independentemente do benefício relativo de quaisquer ações nesse estado. Por exemplo, em um determinado estado, todas as ações podem fazer com que o agente "morra"em um jogo - este é um estado inerentemente ruim de se estar, e não há necessidade de desperdiçar recursos computacionais tentando determinar a melhor ação neste estado . O inverso também pode ser verdadeiro. Idealmente, esta “divisão” em função de vantagem e função de valor deve ser aprendida implicitamente durante o treinamento. No entanto, a arquitetura *Dueling Q* torna essa divisão explícita, o que atua para melhorar o treinamento.

5.3.4 Arquitetura Final

De acordo com as especificações descritas acima, a arquitetura final do modelo pode ser observada na **Figura 9**. Pode-se observar que na arquitetura implementada, foram utilizadas camadas comuns de redes neurais convolucionais que realizam o processamento de imagens. A saída dessas camadas é então achatada e rede então se bifurca em um fluxo de função de valor $V(s)$ e um fluxo de função de vantagem $A(s, a)$. A saída desses fluxos separados é então agregada em uma camada especial, antes de finalmente produzir os valores Q da rede.

A partir da arquitetura descrita acima, os **Algoritmos 1** e **2** mostram o pseudo-código do sistema implementado. O **Algoritmo 1** mostra o corpo da função principal,

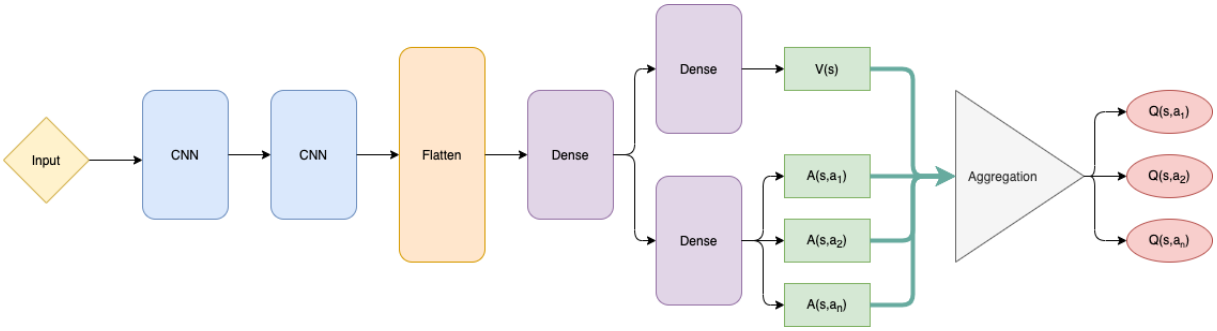

Figura 9: Diagrama da arquitetura final da rede implementada

enquanto o **Algoritmo 2** descreve a função de treinamento do modelo.

Algoritmo 1: Corpo Principal

```
Inicializa o ambiente
Inicializa a rede primária
Inicializa a rede objetivo
for episodio = 1:NUM_EPISODIOS do
    while jogo não termina do
        ação ← modelo.obtemAcao(estado,ϵ)
        proximaObservacao, recompensa, fim ← ambiente.executaAcao(ação)
        proximoEstado ← obtemEstado(estado,proximaObservacao)
        experienceReplay.adicionaTupla(estado,ação,recompensa,proximoEstado,fim)
        treinaModelo()
    end
end
```

Algoritmo 2: treinaModelo

```
< s_t, a_t, r_t, s_{t+1}, fim >batch ← experienceReplay.obtemBatch()
if fim then
    y ← r_t
else
    y ← r_t + γ · max_{a_{t+1}} Q(s_{t+1}, a_{t+1})
end
# Gradiente descendente
Δθ ← ∂/∂θ (y − Q(s, a))^2
Atualiza os pesos da rede
```

5.3.5 Limitações

A implementação do sistema apresentado enfrentou uma série de limitações que fogem do alcance do desenvolvedor. Os desafios encontrados durante a implementação,

assim como as soluções para circunver estes problemas, quando possível, são descritas abaixo.

5.3.5.1 Tempo de Treinamento e Limitações de Hardware

O principal obstáculo enfrentado foi o tempo de treinamento do sistema. Devido a complexidade e número de entradas do problema, o modelo pode levar dias ou semanas para alcançar uma política ótima. Como mencionado anteriormente, cada estado é formado por quatro imagens constituindo os quatro últimos instante do jogo. Mesmo após a conversão para cinza e compressão das imagens de 640×480 para 84×84 pixels, isso ainda deixa cada estado com um vetor de entrada de tamanho $84 \times 84 \times 4 = 28224$. Tudo isso considerando um tamanho de tela bem modesto e um jogo sem informações muito complexas (quando comparado com um jogo 3D por exemplo).

Esses problemas são agravados ainda mais devido às limitações do hardware onde o sistema foi implementado. A máquina utilizada possui as seguintes especificações:

- MacBook Air 2015;
- Processador: 1.6 GHz Dual-Core Intel Core i5;
- Memória: 4 GB 1600 MHz DDR3;
- Sistema Operacional: macOS Catalina | Windows 10.

Como pode ser observado, trata-se de uma máquina com poder computacional limitado, com relativamente pouca memória, um processador bastante simples e sem uma placa de vídeo dedicada. Todas essas limitações acabam levando à um tempo de treinamento ainda maior.

5.3.5.2 Integração do Sistema com o Jogo em *Allegro*

Como mencionado no **Capítulo 3** e em 5.1.2, a comunicação entre a IA, implementada em *Python*, e o jogo, implementado em C, é realizada através de um *Allegro Learning Enviroment* (ALE). No entanto, como o ALE também é implementado em *Python*, a integração dos dois sistemas apresenta algumas limitações.

Primeiramente, o cálculo da recompensa seria, idealmente, calculado pelo programa do jogo e passado para o ALE. No entanto, qualquer saída do programa em C só é recebida pelo ALE uma vez que o programa é finalizado ao fim de um episódio. Isso inviabiliza o calculo em tempo real de qualquer estado que não sejam os estados iniciais ou finais. Por esse motivo o calculo das recompensas de cada estado deve ser feito pela ALE, o qual é realizado através um simples registro da posição do agente no jogo.

Outro problema que surge da integração da ALE com o o jogo é o processamento das imagens e que, por sua vez, acaba exacerbando ainda mais o problema de tempo de treinamento descrito em 5.3.5.1. Ao iniciar um novo episódio, o programa em C leva alguns instantes para ser lançado e ter seu estado inicial renderizado. Portanto, para garantir que a observação do estado inicial do jogo seja realizada propriamente, a ALE aguarda $0.1s$ para garantir que o programa foi inicializado propriamente e $0.2s$ após o jogo ser concluído. Esse tempo torna-se considerável a medida que o número de episódios de treinamento aumentam. Um treinamento com dez mil episódios, por exemplo, recebem um aumento de cerca de 50 minutos.

Além disso, os jogos em *Allegro* devem ter seus dados processados manualmente. A **Figura 10** mostra um diagrama que indica todos os passos do processamento de dados necessários para gerar as informações de cada estado do jogo.

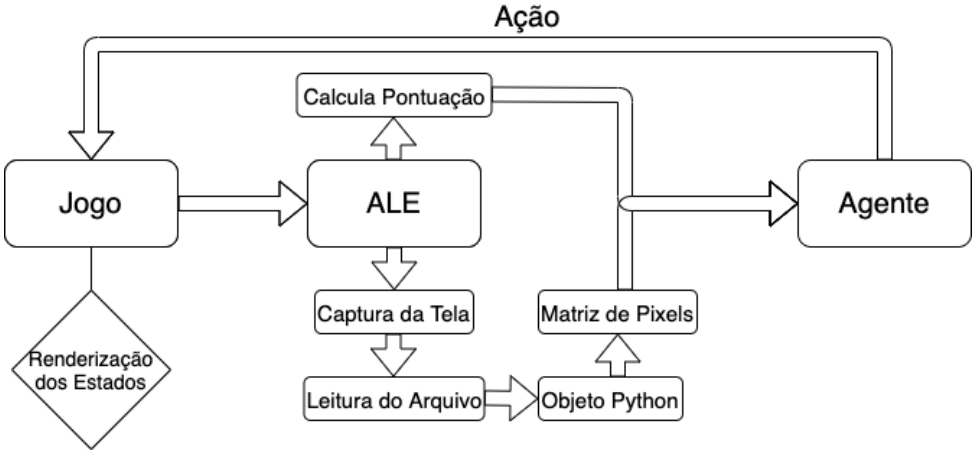

Figura 10: Diagrama mostrando como as informações de cada estado são processadas antes de serem passadas para a IA. Isso inclui executar o programa em C, transmitir os comandos de ação do agente para o jogo, renderizar as imagens em cada instante do jogo, realizar a captura de tela, acessar os arquivos onde essas imagens foram salvas, transformar os dados desse arquivo para um objeto em *Python*, transformar o objeto em uma matriz com os dados da imagem para somente então os mesmos serem passados para o modelo a ser treinado

Jogos em Atari 2600 que possuem um emulador em *Python* que são capazes de gerar um vetor de observação que contém os dados de cada instante do jogo sem a necessidade de renderizar as imagens do mesmo (BROCKMAN et al., 2016). O mesmo não pode ser dito para os jogos em *Allegro*. Para obter as observações dos estados do jogo, deve-se executar o programa em C, renderizar cada estado do jogo e realizar a captura de tela dos mesmos, e realizar o processamento dessas imagens antes desses dados serem passados para a IA. Todo o processo deve ser realizado para cada instante que deseja-se observar e calcular uma ação. Para evitar que o alto tempo de processamento dos dados leve a alguma perda de informações que prejudique o treinamento (como o agente permanecer inativo por um longo período de tempo enquanto calcula o próximo movimento), o jogo foi alterado de

forma que cada quadro (*frame*) do jogo seja atualizado somente quando o agente realizar uma ação. Esse formato de implementação acaba constituindo um *trade-off*: a perda de informações do agente é minimizada, mas o tempo de execução de cada episódio aumenta.

6 Análise dos Resultados

Nesta seção serão analisados os resultados obtidos após a execução do algoritmo implementado. É importante reafirmar que devido às limitações de hardware e à complexidade do problema, o algoritmo deve ser executado por um período de tempo longo (dias ou até semanas), para se obter um agente com uma política ótima. Como a execução por esse período de tempo é inviável, a análise de sua eficiência é feita executando o mesmo por períodos de tempo menores e observando as melhoras de performance obtidas.

A **Figura 11** mostra os resultados obtidos de um modelo treinado durante cerca de 3150 episódios, sendo que os primeiros 556 episódios utilizaram uma política aleatória para coletar amostras para o *buffer* de experiência. Os resultados são referentes aos episódios após o início do treinamento.

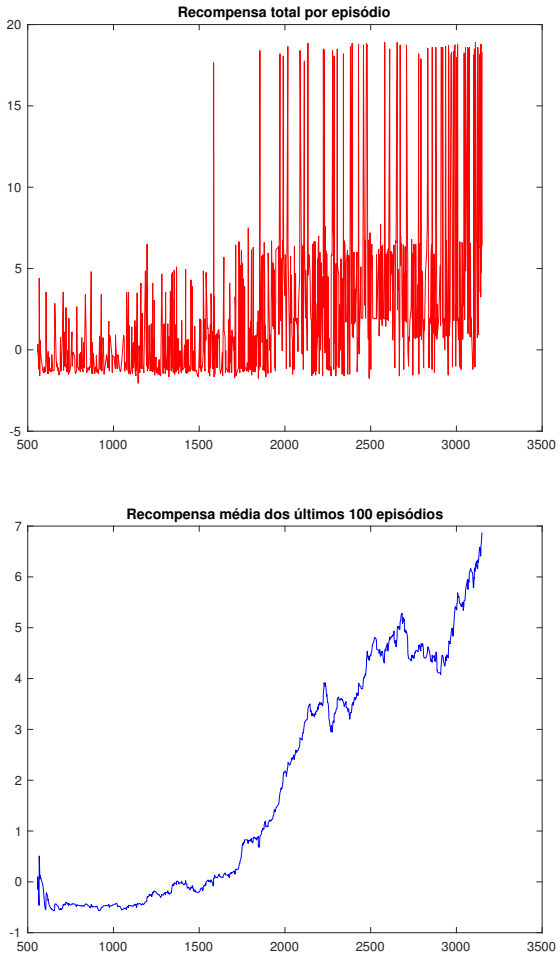

Figura 11: Resultados de um treinamento com 3150 episódios. A primeira imagem mostra as recompensas totais obtidas em cada episódio. A segunda indica a recompensa média dos 100 episódios anteriores

De acordo com as recompensas estabelecidas na **Seção 5.1.1**, a pontuação máxima possível a ser obtida por episódio é de $19pts$, sendo que qualquer pontuação > 10 indica uma vitória do agente para aquele episódio. Além disso, qualquer pontuação < 2 indica que o agente foi eliminado com pouco ou nenhum progresso no jogo.

Observando a **Figura 11**, podemos observar que a pontuação média dos primeiros 1500 episódios é negativa e próxima de zero, indicativos de uma política estocástica. Existem, mesmo nesse período inicial, alguns episódios no qual o agente foi capaz de obter algum progresso no jogo, mas no geral a sua performance foi ruim. Isso se dá ao fato de que, não somente a rede ainda não foi muito bem treinada, mas o valor de ϵ no início do treinamento é alto e próximo de 1, o que leva o agente a ter uma política próxima de uma política aleatória.

Uma vez que o valor de ϵ começa a reduzir e a rede neural é treinada por um tempo considerável, os resultados começam a melhorar. O agente obtém sua primeira vitória no episódio 1584, e sua performance é aprimorada com o passar do tempo. Eventualmente, o algoritmo obtém vitórias consideravelmente mais frequentes e alcança uma pontuação média próxima de $7pts$ ao final dos 3150 episódios. Mesmo após esse treinamento, nota-se que o agente ainda não é capaz de vencer o jogo em boa parte dos episódios e ocasionalmente ainda obtém pontuações próximas de zero. Isso pode ser explicado pelo fato de que a rede precisa ser treinada por mais tempo e continuar aprimorando sua política. Além disso, o algoritmo utilizado tem o menor valor de $\epsilon_{min} = 0.1$, isso faz que a qualquer momento, mesmo após o treinamento da rede, existe uma probabilidade de 10% de que o agente irá tomar uma ação aleatória.

A **Figura 12** mostra os quadros do episódio com a melhor pontuação obtida.

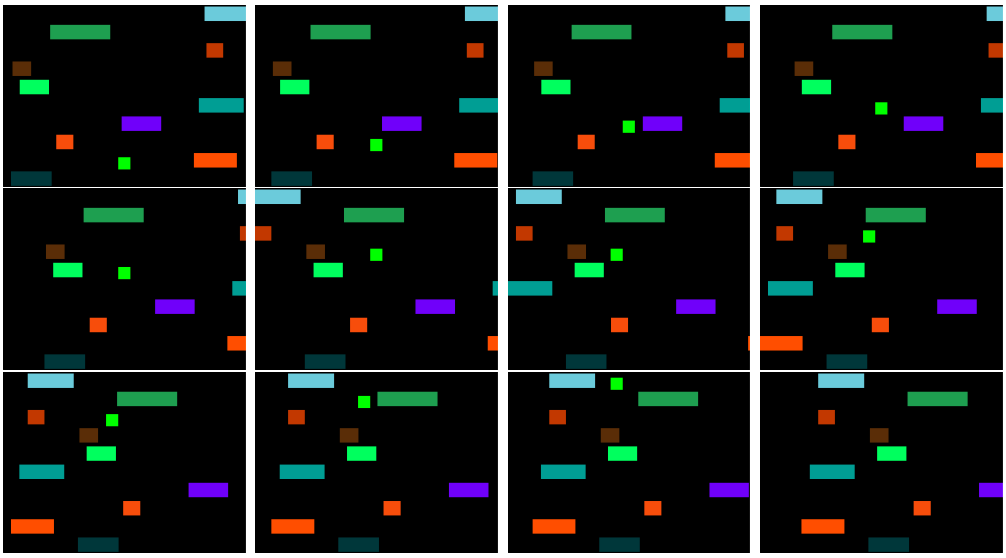

Figura 12: Quadros do episodio 2575 onde o agente obteve uma pontuação de 18.9 pts

A **Figura 13** mostra os resultados de uma segunda rede treinada com diferentes

parâmetros. É interessante notar que, mesmo a rede tendo sido treinada durante um número de episódios maior, a performance obtida não foi tão boa quanto a da primeira rede. Isso reflete a alteração dos valores dos parâmetros utilizados para essa iteração do algoritmo. Os principais parâmetros alterados foram o tamanho do *buffer* de memória, o valor de ϵ e a frequência com o qual ϵ é atualizado.

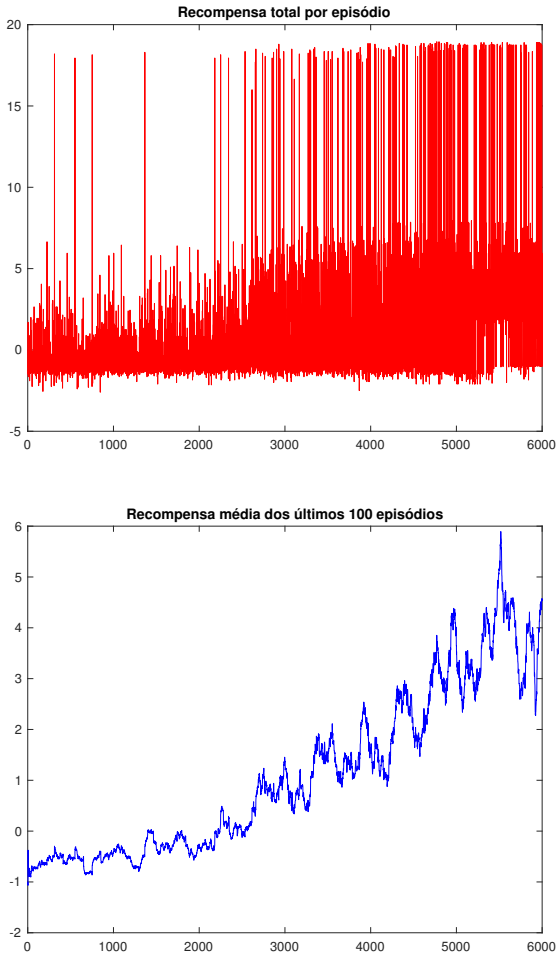

Figura 13: Resultados de um treinamento com 6000 episódios. A primeira imagem mostra as recompensas totais obtidas em cada episódio. A segunda indica a recompensa média dos 100 episódios anteriores

O tamanho máximo do *buffer* de memória foi aumentado em 10 vezes, enquanto o seu valor mínimo foi dobrado. Isso permite uma coleta maior de observações para treinamento da rede mas, em contrapartida, aumenta o número de episódios que serão executados antes de se iniciar o treinamento do agente. Além disso, o aumento do tamanho máximo do *buffer* de memória faz com que a sobreposição dos dados do mesmo demore a ocorrer, tendo em vista que o *buffer* irá armazenar 10 vezes mais dados antes de atingir seu volume máximo. Já o valor mínimo de ϵ foi definido com $\epsilon_{min} = 0.05$, em vez de $\epsilon_{min} = 0.1$, e o número de passos a serem dados, após o inicio do treinamento, para que ϵ varie de $1 \rightarrow \epsilon_{min}$ foi de 20000 da primeira rede, para 50000 na segunda. Isso permite uma

maior exploração do agente e por um período mais prolongado, mas aumenta o número de episódios necessários para que o agente reduza sua política estocástica.

Com isso, espera-se que a segunda rede obtenha melhores resultados a longo prazo, devido à sua maior exploração dos estados do jogo e maior quantidade e dados pra treinamento. No entanto, a curto prazo, a segunda rede deve demorar mais para alcançar bons resultados, o que explica os valores obtidos na **Figura 13**.

7 Conclusões

A inteligência artificial e o aprendizado de máquina são ferramentas muito poderosas e com inúmeras aplicações práticas. A IA busca fornecer softwares que sejam capazes de realizar atividades como seres humanos para automatizar e otimizar o trabalho de rotina. Diante do grande potencial da IA, além do crescimento exponencial de pesquisa que a área vêm sofrendo, grandes empresas no mercado estão investindo na tecnologia, seja para propor novos serviços ou aprimorar produtos existentes e garantir uma vantagem competitiva no mercado. Situações do mundo real são muitas vezes complexas e apresentam problemas com um número muito grande de variáveis, o que dificulta a solução utilizando algoritmos de otimização tradicionais. Nesse contexto, treinar um agente em jogos digitais para superar os jogadores humanos e otimizar sua pontuação pode nos ensinar como otimizar processos variados com múltiplas aplicações. Uma vez que se tenha uma IA que possa aprender a jogar e a otimizar estratégias para maximizar a pontuação de um jogo, pode-se facilmente implementar um jogo que simule uma situação real e aplicar o sistema para que este encontre a melhor resposta ou solução para um dado problema.

Com isso em mente, foi implementada uma *Deep Q Network* com o objetivo de treinar um agente capaz de aprender e desenvolver estratégias para jogar jogos digitais a partir de capturas de tela obtidas em tempo real. Para extrair as informações necessárias do jogo foi implementado um *Allegro Learning Enviroment*, que tem como base a ferramenta implementada por (SILVA, 2019), modificada para conter funcionalidades semelhantes ao emulador de Atari 2600 (BROCKMAN et al., 2016).

Os resultados obtidos, apesar de limitados, são promissores. O agente foi capaz de alcançar uma política claramente superior à uma abordagem aleatória. Com isso, dado um ambiente de treinamento mais propício, com hardware mais apropriado para executar algoritmos de *machine learning* e com um tempo de execução maior, espera-se que a rede eventualmente seja capaz de alcançar resultados equivalentes ou até superiores aos obtidos por um ser humano. Outro fator a se considerar é o fato de que, apesar de o sistema implementado ter sido testado somente em um ambiente, em teoria o mesmo sistema pode ser aplicado para outros jogos sem a necessidade de grandes alterações.

A maior limitação do sistema implementado se encontra na comunicação entre o jogo e o ALE. Em um cenário ideal, seria desejado uma comunicação em tempo real entre a ALE, em *Python*, e o programa, em C. Apesar de ter sido encontrado soluções para muitos dos problemas causados por essa interação, essas soluções deixam espaço para erro e tendem a aumentar o custo computacional do algoritmo. Outra grande desvantagem do sistema é a necessidade de renderizar as imagens para extrair os dados de observação

do instante atual do jogo. Essa limitação resulta em um tempo de treinamento muito maior quando comparado à redes neurais treinadas para jogos Atari 2600 que utilizam um emulador em *Python* (BROCKMAN et al., 2016).

7.1 Proposta de Continuidade

O sistema implementado e a análise dos resultados obtidos fornecem algumas possibilidades de aprimoramento do sistema e aplicação em novos cenários. A principal proposta de continuidade se encontra na execução do sistema em períodos longos de tempo para analisar os resultados obtidos por uma rede treinada propriamente após pelo menos 1 milhão de episódios. Somente com uma rede extensamente treinada será possível analisar com precisão o potencial do sistema.

Com a ideia de aprimorar o sistema atual, a maior potencial para tal se encontra no desenvolvimento de uma comunicação em tempo real entre o jogo em C e o ALE. Caso implementado, um sistema com essas características seria capaz de otimizar o processo de treinamento, reduzindo a margem para erros e possibilitando um sistema mais genérico. Além disso, potencialmente poderia-se implementar uma transmissão dos dados de estado entre o jogo e o agente sem a necessidade da renderização das imagens para cada instante de tempo, o que reduziria consideravelmente o tempo de execução do algoritmo.

Por fim, propõe-se aplicar o sistema implementado para diferentes jogos, com mecânicas distintas, de forma a analisar o potencial de aplicação do mesmo em cenários variados.

Referências

AFRAZ DANIEL L.K. YAMINS, J. J. D. A. Neural mechanisms underlying visual object recognition. *Cold Spring Harb Symp Quant*, Cold Spring Harbor Laboratory Press; all rights reserved, 2014. Disponível em: <https://doi.org/10.1101/sqb.2014.79.024729>. Acesso em: 2 ago 2019. Citado na página 18.

BALDI, P.; SADOWSKI, P.; WHITESON, D. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, v. 5, n. 1, p. 4308, 2014. Disponível em: <https://doi.org/10.1038/ncomms5308>. Citado 2 vezes nas páginas 18 e 21.

BELLEMARE, M. G. et al. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. Disponível em: <http://arxiv.org/abs/1207.4708>. Citado na página 29.

BOTVINICK, M. et al. Reinforcement learning, fast and slow. *Trends in Cognitive Sciences*, Elsevier, v. 23, n. 5, p. 408–422, 2019/10/17 2019. Disponível em: <https://doi.org/10.1016/j.tics.2019.02.006>. Citado na página 18.

BROCKMAN, G. et al. *OpenAI Gym*. 2016. Citado 3 vezes nas páginas 44, 51 e 52.

COMI, M. *How to teach AI to play Games: Deep Reinforcement Learning*. 2018. Disponível em: <https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a>. Acesso em: 8 out 2019. Citado 2 vezes nas páginas 22 e 23.

DAHL, G. E.; JAITLY, N.; SALAKHUTDINOV, R. *Multi-task Neural Networks for QSAR Predictions*. 2014. Citado na página 18.

DWIBEDI, D.; VEMULA, A. 2016. Disponível em: <https://pdfs.semanticscholar.org/179d/04d9da112c16b6fa5310c273d66de65e5768.pdf>. Acesso em: 18 ago 2019. Citado na página 19.

GEBRU, T. et al. Using deep learning and google street view to estimate the demographic makeup of neighborhoods across the united states. *Proceedings of the National Academy of Sciences*, National Academy of Sciences, v. 114, n. 50, p. 13108–13113, 2017. ISSN 0027-8424. Disponível em: <https://www.pnas.org/content/114/50/13108>. Citado na página 21.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado 4 vezes nas páginas 13, 17, 25 e 37.

HARGREAVES, S. *Allegro*. 1990. Disponível em: <https://liballeg.org/index.html>. Acesso em: 8 out 2019. Citado 3 vezes nas páginas 17, 23 e 28.

HASSELT, H. van; GUEZ, A.; SILVER, D. *Deep Reinforcement Learning with Double Q-learning*. 2015. Citado na página 40.

HSU, F.-H. *Behind Deep Blue: Building the Computer That Defeated the World Chess Champion*. Princeton, NJ, USA: Princeton University Press, 2002. ISBN 0691090653. Citado na página 15.

Karavolos, D.; Liapis, A.; Yannakakis, G. N. Using a surrogate model of gameplay for automated level design. In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. [S.l.: s.n.], 2018. p. 1–8. Citado na página 18.

LIN, L. Reinforcement learning for robots using neural networks. In: . [S.l.: s.n.], 1992. Citado na página 37.

MARR, B. *Is Artificial Intelligence Dangerous? 6 AI Risks Everyone Should Know About*. 2018. Disponível em: <https://www.forbes.com/sites/bernardmarr/2018/11/19/is-artificial-intelligence-dangerous-6-ai-risks-everyone-should-know-about/#256480952404>. Acesso em: 11 nov 2019. Citado na página 21.

MENABREA, L. et al. *Sketch of the Analytical Engine invented by Charles Babbage ... with notes by the translator. Extracted from the 'Scientific Memoirs,' etc. [The translator's notes signed: A.L.L. ie. Augusta Ada King, Countess Lovelace.]*. R. & J. E. Taylor, 1843. Disponível em: <https://books.google.com.br/books?id=hPRmnQEACAAJ>. Citado na página 13.

MILLINGTON, I.; FUNGE, J. *Artificial Intelligence for Games, Second Edition*. 2nd. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009. ISBN 0123747317, 9780123747310. Citado na página 18.

MNIH, V. et al. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. Disponível em: <http://arxiv.org/abs/1312.5602>. Citado 2 vezes nas páginas 33 e 35.

MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. SN -, v. 518, p. 529 EP –, 02 2015. Disponível em: <https://doi.org/10.1038/nature14236>. Acesso em: 2 ago 2019. Citado na página 18.

Nassif, A. B. et al. Speech recognition using deep neural networks: A systematic review. *IEEE Access*, v. 7, p. 19143–19165, 2019. Citado 2 vezes nas páginas 17 e 21.

NICHOLSON, C. *A Beginner's Guide to Deep Reinforcement Learning*. 2016. Disponível em: <https://skymind.ai/wiki/deep-reinforcement-learning>. Acesso em: 8 out 2019. Citado na página 27.

Piergigli, D. et al. Deep reinforcement learning to train agents in a multiplayer first person shooter: some preliminary results. In: *2019 IEEE Conference on Games (CoG)*. [S.l.: s.n.], 2019. p. 1–8. Citado na página 18.

Ripamonti, L. A. et al. Believable group behaviours for npcs in fps games. In: *2017 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.: s.n.], 2017. p. 12–17. Citado na página 18.

RIPAMONTI, L. A. et al. Procedural content generation for platformers: designing and testing fun pledge. *Multimedia Tools and Applications*, v. 76, n. 4, p. 5001–5050, Feb 2017. ISSN 1573-7721. Disponível em: <https://doi.org/10.1007/s11042-016-3636-3>. Citado na página 18.

ROBU, V. et al. Consider ethical and social challenges in smart grid research. *Nature Machine Intelligence*, 2019. Disponível em: <https://doi.org/10.1038/s42256-019-0120-6>. Citado na página 13.

RODRIGUES, J. *O que é o Processamento de Linguagem Natural?* 2017. Disponível em: <https://medium.com/botsbrasil/o-que-é-o-processamento-de-linguagem-natural-49ece9371cff>. Acesso em: 2 set 2019. Citado na página 13.

SILVA, A. P. Ambiente para desenvolvimento de inteligência artificial em jogos allegro. Departmento de Ciência da Computação (DCC) da Universidade Federal de Minas Gerais (UFMG), Belo horizonte, Brasil, 2019. Disponível em: <https://github.com/artphil/allegro_game_ai>. Acesso em: 8 out 2019. Citado 3 vezes nas páginas 29, 34 e 51.

SUTTON, R.; BARTO, A. *Reinforcement Learning: An Introduction*. [S.l.]: MIT Press, 1998. Citado na página 36.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. 2. ed. [S.l.]: MIT Press, 2018. <http://incompleteideas.net/book/the-book.html>. Citado 3 vezes nas páginas 13, 25 e 26.

Tang, J. et al. Enabling deep learning on iot devices. *Computer*, v. 50, n. 10, p. 92–96, 2017. Citado na página 21.

TAO, Y. et al. A deep neural network modeling framework to reduce bias in satellite precipitation products. *Journal of Hydrometeorology*, v. 17, n. 3, p. 931–945, 2016. Disponível em: <https://doi.org/10.1175/JHM-D-15-0075.1>. Citado na página 21.

THOMAS, A. 2020. Disponível em: <https://adventuresinmachinelearning.com/double-q-reinforcement-learning-in-tensorflow-2/>. Acesso em: 17 out 2020. Citado na página 40.

VINYALS, O. et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, v. 575, n. 7782, p. 350–354, 2019. Disponível em: <https://doi.org/10.1038/s41586-019-1724-z>. Citado na página 19.

WATKINS, C.; DAYAN, P. Technical note: Q-learning. *Machine Learning*, v. 8, p. 279–292, 05 1992. Citado na página 36.

YANNAKAKIS, G. N. Game ai revisited. In: *Proceedings of the 9th Conference on Computing Frontiers*. New York, NY, USA: ACM, 2012. (CF '12), p. 285–292. ISBN 978-1-4503-1215-8. Disponível em: <http://doi.acm.org/10.1145/2212908.2212954>. Citado na página 18.

YEUNG, S. et al. A computer vision system for deep learning-based detection of patient mobilization activities in the icu. *npj Digital Medicine*, v. 2, n. 1, p. 11, 2019. Disponível em: <https://doi.org/10.1038/s41746-019-0087-z>. Citado na página 18.

YOUNG, T. et al. Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, Institute of Electrical and Electronics Engineers (IEEE), v. 13, n. 3, p. 55–75, Aug 2018. ISSN 1556-6048. Disponível em: <http://dx.doi.org/10.1109/mci.2018.2840738>. Citado na página 18.