Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

João Pedro Fernandes Bastos

Aplicação de técnicas de machine learning à análise de padrões de aprendizagem em gaming

Tese de Mestrado

Sistemas e Tecnologias de Informação para as Organizações

Professor Doutor Rui Pedro Duarte

Professor Doutor Carlos Augusto da Silva Cunha

Financiamentos:

N/A

Dezembro 2019

Dedicado à minha Mãe.

RESUMO

A motivação para a produção deste trabalho está essencialmente ligada com o interesse particular na área de *machine learning* e no potencial da sua aplicação no quotidiano. Trata-se de uma área extremamente complexa, com desafios interessantes tanto a nível de conceitos teóricos como nível de tecnologias e metodologias de implementação. Esta área mostra um potencial de evolução enorme para os próximos anos, tendo uma capacidade poderosa de processamento e suporte à tomada de decisão inigualável na humanidade, sendo um conceito aplicável transversalmente a qualquer indústria e que, se corretamente aplicada, poderá trazer benefícios até agora inalcançáveis para a qualidade de vida do nosso dia-a-dia.

Este trabalho pretende explorar a aplicação de técnicas de *machine learning* à análise de padrões de aprendizagem em jogos (*gaming*), com o intuito principal de identificar e otimizar as capacidades de algoritmos atualmente existentes e ainda com vista a investigar a relevância da aplicação do conceito de curiosidade nesses algoritmos. Esta exploração passará pela aplicação dessas técnicas a vários jogos, através da implementação de algoritmos de *machine learning* que interajam diretamente com os ambientes dos jogos e que aprendam a conhecer todas as características inerentes, com vista a reforçar o conhecimento e daí obterem a capacidade de concluir os jogos com sucesso. Desta forma, torna-se possível identificar padrões específicos a cada jogo.

Globalmente, através da aplicação de técnicas de *machine learning*, nomeadamente *Q-learning*, uma técnica de treino de modelos de aprendizagem com o intuito de ser agnóstica perante o ambiente onde opera e dotada da capacidade de aprender a conhecer um ambiente e os seus obstáculos por forma a superá-los, pretende-se obter resultados de *benchmarking* para a comparação de padrões de aprendizagem no que se refere à performance, facilidade de implementação e aplicabilidade em cenários reais. Estes resultados servirão ainda para retirar conclusões da mesma natureza sobre a aplicação do conceito de curiosidade num algoritmo de aprendizagem inteligente em *gaming*, nomeadamente nos jogos Snake e Tetris, que serão abordados no contexto deste documento.

ABSTRACT

The motivation to produce this project is intrinsically associated with the particular author's interest in the area of machine learning and its potential applications. Machine learning is an extremely complex area, with a wide variety of applications and exciting challenges, both in theoretical concepts as well as methodologies of implementation. This area shows great potential in the upcoming years since it has an excellent capacity for data processing and supports decision-making unmatched on human beings. It is a field with a wide variety of applications on all industries, and may, therefore, bring benefits until now unreachable to the quality of our day-to-day lives.

This project intends to explore the application of machine learning techniques to the analysis of learning patterns in the practical context of gaming, with the primary purpose to identify and optimize the capacity of existing algorithms and to investigate the relevance of applying the concept of curiosity to these types of algorithms. The tasks to explore this concept will primarily be using the techniques to multiple games, through implementing machine learning algorithms that interact directly with the gaming environments. They can learn and understand the games' characteristics, and to use this information to reinforce the knowledge and being capable of beating these games successfully. This way, it shall be possible to identify learning patterns for each game. Globally, through the application of machine learning techniques, namely Q-learning, a technique intended to train learning models agnostically in terms of what environment it operates on and with the capacity to learn from an environment what are the obstacles and how to surpass them, it is expected to obtain results of benchmarking and for the comparison of learning patterns in the aspects of performance, ease of implementation, and application in real-life scenarios. These results will provide conclusions of the same nature in the form of a concept of curiosity on an intelligent agent of Artificial Intelligence interacting with gaming, namely the games Snake and Tetris, which will be explored in this document.

PALAVRAS CHAVE

Machine Learning
Reinforcement Learning
Q-Learning
Redes Neurais
Gaming

KEYWORDS

Machine Learning
Reinforcement Learning
Q-Learning
Neural Networks
Gaming

AGRADECIMENTOS

Agradeço aqui todas a contribuições feitas de teor pessoal, académico e profissional relativas à realização deste projeto.

Começo por agradecer em especial à minha Mãe, minha família e também a amigos próximos, com especial relevo para Júlio André Rodrigues, João Paulo Cachinho e Nuno Martins dos Santos, que frequentaram comigo o MSTIO. Foram incondicionais no apoio e suporte das decisões que tenho tomado academicamente e que me permitiram chegar à conclusão deste documento.

Gostaria de agradecer com especial relevância aos professores Rui Pedro Duarte e Carlos Augusto da Silva Cunha, orientador e coorientador respetivamente, pela forma incansável e meticulosa como me ajudaram a escrever este documento e em geral no meu percurso académico de mestrado.

Por fim, gostaria de deixar um agradecimento especial a duas pessoas fundamentais na conclusão deste projeto, sendo elas Pedro André Beltrão e João Oliveira Fernandes, que reviram e partilharam notas tanto sobre o documento como sobre o projeto e me suportaram através da partilha de ideias e *brainstorming* sobre as demais tarefas associadas à conclusão do mesmo.

ÍNDICE GERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ÍNDICE DE FÓRMULAS

ABREVIATURAS E SIGLAS

IPV	Instituto Politécnico de Viseu
ESTGV	Escola Superior de Tecnologia e Gestão de Viseu
MSTIO	Mestrado em Sistemas e Tecnologias de Informação para as Organizações
ML	*Machine Learning*
DL	*Deep Learning*
RL	*Reinforcement Learning*
IA	Inteligência Artificial
GPU	*Graphical Processing Unit*
CPU	*Computer Processing Unit*
DQN	*Deep Q-learning Network*
MLP	*Multilayer Perceptron*
CNN	*Convolutional Neural Network*
RNN	*Recurrent Neural Network*
LSTM	*Long-Short Term Memory*
SARSA	*State–Action–Reward–State–Action*
CBR	*Case Based Reasoning*

1. Introdução

O projeto apresentado neste documento demonstra a aplicação de técnicas de Inteligência Artificial (IA) - dentro da área de *machine learning (ML)* - com vista a explorar e analisar padrões que permitam obter métricas e conclusões sobre vários aspetos associados a este tipo de algoritmos (Hsiang, 2018). Neste trabalho serão analisados os seus níveis de performance, adaptabilidade, complexidade de implementação e principais desafios a ultrapassar, permitindo assim demonstrar resultados positivos na aplicação de inteligência e automatização em *gaming*.

Neste documento, é feita uma análise ao Estado da Arte na área do ML aplicado ao *gaming* e consequentemente uma descrição dos conceitos e tecnologias subjacentes a este tema. Esta descrição será, numa primeira fase, constituída pelo estudo das arquiteturas, tecnologias e técnicas disponíveis para a implementação de algoritmos de aprendizagem. Numa segunda fase, serão testados vários padrões de aprendizagem com recurso à adaptação e aplicação de algoritmos, que permitirão obter resultados e dessa forma perceber a relevância que os parâmetros destes algoritmos de aprendizagem têm para a sua otimização. Adicionalmente, será analisada a aplicação do conceito de curiosidade a algoritmos de aprendizagem (Vincent, 2018), por forma a permitir que sejam mais dinâmicos e tenham uma aprendizagem mais ágil, em contraste com a metodologia tipicamente utilizada de recompensa por cumprimento de objetivos específicos (Burda, 2018).

Uma das dificuldades mais expressivas na análise de padrões e resultados na aplicação de algoritmos de ML é que estes algoritmos estão condicionados por vários fatores que limitam o seu desempenho: volume e variedade de dados, tempo útil de treino/aprendizagem, adaptabilidade e possibilidade de customização de ambientes e uma capacidade colaborativa e que forneça feedback neutro (Marr, 2018). As plataformas de *gaming* atuais podem ser trabalhadas para aproveitar todos estes pontos, considerando a diversidade de jogos existentes, mecanismos implementados para a captação de interesse e consequente investimento de tempo. Os ambientes de *gaming* são ainda

altamente customizáveis, sendo possível implementar alterações aos cenários de forma rápida, e contam também com uma enorme base de fãs que experimentam e usam extensivamente estas plataformas no dia-a-dia. Todos estes fatores tornam as plataformas de *gaming* apelativas para serem suportes de informação para algoritmos de aprendizagem (Epand, 2019).

Este trabalho visa o estudo e exploração dos conceitos de aprendizagem automatizada para conferir o seu potencial junto dos desafios atualmente existentes no treino de agentes de Inteligência Artificial (IA) e na possibilidade de estes agentes se tornarem mais autónomos (Sukhadeve, 2016).

Este documento está organizado da seguinte forma: começa com o capítulo da Introdução, onde é feita uma introdução à temática abordada e ao trabalho realizado para este projeto. De seguida, na Contextualização são discriminadas as motivações para a elaboração deste projeto, os objetivos e abordagens definidas e as principais contribuições. No capítulo do Estado da Arte é feita uma análise aos conceitos, tecnologias e métodos utilizados na implementação de ML, em especial *Q-learning*, aplicado em *gaming*. No capítulo seguinte de *Q-learning* Aplicado em *Gaming* é demonstrado o desenho do trabalho realizado, passando pela arquitetura funcional da implementação, a parametrização dos algoritmos implementados, as métricas e contextualização da sua aplicação em *gaming*, os tipos de testes que podem ser executados e os padrões de aprendizagem identificados provenientes dos resultados dos testes. Por fim, capítulo das Conclusões, consiste no enquadramento com objetivos pessoais e profissionais no desenvolvimento do projeto, a visão global do cumprimento dos objetivos identificados neste trabalho, lições retidas e ainda perspetivas e melhorias futuras.

2. Contextualização

Este capítulo descreve as motivações para a realização deste projeto, os principais objetivos a cumprir e abordagens utilizadas, e ainda as principais contribuições sobre a temática principal do projeto.

2.1 Motivação

A principal motivação para a elaboração deste projeto surge enquadrada com o interesse pessoal do autor sobre a temática de ML. Esta é uma área que tem evoluído bastante nos últimos anos e tem provado ser extremamente eficiente em resolver problemas no quotidiano, podendo ser aplicada a várias indústrias distintas (Sukhadeve, 2016).

A temática de ML é agnóstica de indústria ou contexto, tendo como principal objetivo o de tomar partido de tecnologia e do largo volume de dados disponível e utilizar essa combinação para automatizar a construção de modelos de aprendizagem que possibilitem superar obstáculos e resolver problemas no dia-a-dia das organizações e pessoas em geral. É assim uma área extremamente complexa e desafiante para qualquer profissional na área de sistemas de informação e ciências de dados.

A motivação para realizar o projeto e o documento que o descreve prende-se assim com o desafio e o entusiasmo em termos de potencial que a temática de ML apresenta para qualquer pessoa que pretenda explorar cientificamente os conceitos e experimentar novas metodologias e abordagens com esta temática.

2.2 Objetivos e Abordagens

De seguida é descrita uma visão global sobre os objetivos deste projeto:

- Implementar um algoritmo de aprendizagem com recurso a técnicas de ML que será aplicado a ambientes de *gaming*, por forma a demonstrar a capacidade deste tipo de algoritmos em superar os obstáculos inerentes à capacidade de o algoritmo aprender a jogar um determinado ambiente;

- Determinar o impacto dos hiperparâmetros, que sucintamente são as variáveis que definem a estrutura de uma rede neural (camadas de entrada, camadas escondidas, camadas de saída, nós das unidades) e as variáveis que determinam como uma rede neural é treinada (taxa de aprendizagem, taxa de exploração, número de episódios), na performance de algoritmos de aprendizagem em *gaming*. Para o efeito, os modelos de aprendizagem serão testados dinamicamente através da sua aplicação em diferentes ambientes e com variadas configurações de parâmetros. Desta forma, é possível definir diferentes paradigmas que permitam que os algoritmos sejam ajustáveis aos vários tipos de desafios presentes neste tipo de aprendizagem;

- Explorar a relevância da curiosidade na performance de algoritmos de aprendizagem. Para o efeito, será explorada uma abordagem de ML que consiste essencialmente na execução de várias sessões de treino de modelos em que o agente de IA é dotado com a capacidade de explorar livremente as diferentes ações que pode tomar dentro de um determinado ambiente, permitindo posteriormente uma análise aos resultados obtidos e assim averiguar a relevância da aplicação do conceito de curiosidade. O principal objetivo será estudar a criação de um incentivo intrínseco na exploração de novas soluções que possam levar à aquisição de novo conhecimento pelos modelos, em contraste com a recompensa pela utilização do conhecimento adquirido (Burda, 2018).

- Identificar padrões resultantes da aplicação de arquiteturas de redes neurais a *gaming* seguindo uma abordagem de *reinforcement learning (RL)*. Serão identificados e analisados vários padrões na aplicação de algoritmos de RL, para determinar os critérios de sucesso e insucesso na otimização destes algoritmos no contexto da resolução de problemas e desafios associados ao *gaming* (Rana, 2018).

A visão detalhada dos objetivos permite fazer o desdobramento dos objetivos globais do projeto, onde são detalhados os requisitos da implementação alinhados com estes objetivos, as métricas relevantes a observar para retirar as conclusões necessárias e as técnicas aplicadas por forma a alcançar os objetivos definidos.

2.2.1 Implementação de algoritmo de aprendizagem em ambientes gaming

O primeiro objetivo deste projeto consiste na implementação de um algoritmo de aprendizagem dotado da capacidade de interagir com ambientes de *gaming* e aprender a superar os obstáculos destes ambientes. De forma a atingir este objetivo, será implementado um algoritmo com recurso a técnicas de ML, que permitirá a um agente de IA interagir com um jogo que lhe é, até então, desconhecido. O agente aprende quais os obstáculos presentes no jogo e qual a forma mais eficiente de os superar.
Finalmente, o algoritmo de aprendizagem implementado será aplicado a dois jogos com características distintas (Snake e Tetris), tanto ao nível de funcionamento como a critérios de sucesso.

2.2.2 Impacto de hiperparâmetros na performance de algoritmos de aprendizagem

O segundo objetivo deste projeto passa por determinar o impacto dos hiperparâmetros de algoritmos de aprendizagem na performance obtida aquando da sua aplicação em ambientes de *gaming*.
Para medir o impacto dos hiperparâmetros na performance deste tipo de algoritmos, serão executados vários testes com configurações distintas nos valores dos hiperparâmetros. A variedade de testes realizados permitirá averiguar o impacto que estes hiperparâmetros têm na performance, tanto a nível individual (impacto que cada parâmetro tem) como a nível coletivo (impacto que um conjunto de parâmetros tem).
Na medição do impacto dos hiperparâmetros, será especificamente analisado o impacto dos seguintes parâmetros:

- Taxa de aprendizagem;
- Taxa de exploração;
- Número de episódios;
- Número de camadas implementadas na rede neural (camadas escondidas)
- Número de nós de processamento dentro de cada camada da rede neural

Adicionalmente, as métricas a adotar para a avaliação dos resultados dos algoritmos e de diferentes configurações de hiperparâmetros serão as seguintes:

- Tempo de treino
- Pontuação máxima
- Pontuação média

2.2.3 Explorar a relevância da curiosidade em modelos de aprendizagem

O terceiro objetivo deste projeto foca-se na relevância que a aplicação do conceito de curiosidade pode ter em modelos de aprendizagem. Para este efeito, serão executados múltiplos testes com configurações de níveis de curiosidade diferentes, na forma de taxas de exploração diferentes e que potenciam que o agente procure novas ações em prol de aplicar ações com pesos de relevância

previamente identificados. Assim, estes testes têm o propósito de medir o impacto da aplicação deste conceito de curiosidade em modelos de aprendizagem já treinados para a conclusão de um determinado ambiente de *gaming*.
Para completar este objetivo, será explorada a aplicação deste conceito e especificamente se é vantajoso, em termos das métricas de avaliação detalhadas acima (tempo de treino, pontuação máxima, pontuação média), encorajar os agentes a explorar novas opções mais frequentemente ou optar por executar ações aprendidas numa primeira fase do treino.

2.2.4 Identificar padrões em algoritmos de aprendizagem em gaming

O quarto objetivo deste projeto é composto pela identificação de padrões resultantes da aplicação de algoritmos de aprendizagem em *gaming*.
Por forma a concluir este objetivo, será feita uma análise específica aos critérios de sucesso e insucesso associados aos problemas e desafios inerentes ao *gaming*, na forma de desafios na implementação de algoritmos de aprendizagem, vantagens e desvantagens da utilização deste tipo de algoritmos e na identificação de padrões de aplicação para concluir potenciais abordagens como solução para a resolução destes problemas.

2.3 Contribuições Principais

As principais contribuições científicas com a realização deste trabalho são *benchmarking* através de testes na área de RL aplicado a *gaming*, exploração da aplicação do tema de curiosidade em algoritmos de aprendizagem e investigação do impacto que os vários hiperparâmetros de uma rede neural podem ter no treino de modelos de aprendizagem.
O treino do tipo de algoritmos e modelos de aprendizagem abordados neste relatório é essencialmente um exercício de tentativa-erro (Porr, 2008), na medida em que é necessária a exploração de diferentes abordagens na forma como os modelos são treinados para encontrar uma solução ótima. Neste sentido, o conjunto de testes a efetuar no contexto deste projeto contribuem diretamente para o *benchmark* dos diferentes parâmetros explorados dentro da área de ML.

Outra contribuição académica relevante deste projeto é a da exploração do conceito de curiosidade no treino dos modelos de aprendizagem. Tipicamente, estes modelos são treinados considerando exclusivamente as consequências de ações anteriores que formam a base para suportar a decisão de quais as próximas ações a tomar. Este projeto explora uma abordagem diferente, ambicionando medir o impacto que a aplicação de um conceito de curiosidade, onde o algoritmo é implementado de forma a explorar novas opções em prol de executar apenas ações baseado em informação previamente conhecida, possa afetar a performance no treino dos modelos de aprendizagem (Vincent, 2018).

3. Estado da Arte

Neste capítulo é feita uma análise ao Estado da Arte de ML dentro daqueles que são os conceitos tecnológicos associados e cenários de aplicação específica em *gaming*.

3.1 Machine Learning

Conceptualmente, ML é um ramo da IA que atesta a ideia de que sistemas inteligentes podem aprender a partir de dados e com essa aprendizagem identificar padrões, fazer predições e tomar decisões assertivas com intervenção humana mínima (El Naqa & Murphy, 2015). Tecnicamente, é um método de análise de dados que automatiza a criação de modelos de dados e permite uma análise precisa e alimentada por (virtualmente) infinitas fontes de dados que, tendo em conta a variedade e volume de dados existente, enriquece esta aprendizagem e permite perceber novos paradigmas de análise.

Em *gaming*, podem ser aplicadas técnicas de ML para preparar um agente de IA a estar apto a resolver os passos que levam à conclusão de um determinado jogo. A implementação e aplicação destas técnicas permite uma análise às tendências que suportarão a definição de padrões eficientes na resolução deste problema. Exemplos da aplicação de ML em *gaming* podem ser encontrados em (Joshi, 2002), (Fried, 2019) e (Li, 2018).

Para a definição dos modelos de análise a dados, o ML toma partido de hiperparâmetros (Bengio, 2000). Estes são parâmetros cujos valores são definidos numa fase anterior à do início da aprendizagem de um algoritmo. Os algoritmos de aprendizagem utilizam estes hiperparâmetros para definirem que parâmetros são relevantes de extrair da informação analisada para otimizar a aprendizagem (Snoek, Larochelle & Adams, 2012).

Nas secções seguintes, serão detalhados os elementos das técnicas a serem aplicadas na elaboração deste projeto: *deep learning (DL)* e *RL*.

3.2 Deep Learning

Deep learning (DL) é uma técnica de ML que foi descoberta nos anos 80, (Alom et al., 2018) e que permite essencialmente ensinar a agentes de IA a forma como devem executar uma determinada tarefa, seguindo o mesmo princípio básico de ensinamento de seres humanos: aprender através de exemplificação (LeCun, Bengio & Hinton, 2015).

Os modelos de DL aprendem a classificar um determinado objeto representado por imagens, texto ou som. Estes modelos são tipicamente treinados através da utilização de largos volumes de dados (de qualquer um ou a combinação dos vários tipos assinalados acima), e a utilização de arquiteturas de redes neurais (LeCun et al., 2015). O DL tem ganho notoriedade junto do mundo da tecnologia, tendo aplicações vantajosas em processos onde é necessário um alto nível de precisão de reconhecimento de padrões e tem, recentemente, obtido resultados extremamente positivos, dos quais se destacam alguns superiores até à própria capacidade humana (LeCun et al., 2015).

O DL tornou-se útil apenas recentemente, principalmente devido a ser necessário:

- Poder computacional substancial. O treino de modelos de DL requer grande capacidade de processamento, já que tem de analisar e simular largos volumes de dados em tempo reduzido para produzir efeitos vantajosos na aprendizagem dos agentes de IA. A combinação de arquiteturas recentes de *Graphical Processing Units (GPU's)* (que impulsionaram a capacidade de processamento) com o crescimento de tecnologias de computação na nuvem permitem que o treino destes modelos seja reduzido em termos de tempo e que a complexidade dos dados utilizados na aprendizagem possa ser maior e por consequência mais relevante. (Zhai et al., 2018);
- Qualidade e quantidade de informação disponível para ser analisada. É extremamente mais rápido e útil treinar modelos para terem níveis aceitáveis de performance numa determinada tarefa se existirem dados consistentes e em grande volume, que suportem a aprendizagem de um dado modelo. Exemplos de artigos que exploram este tema são (Grimmer, 2015) e (Frey & Larch, 2017).

A eficácia da abordagem de DL encontra-se condicionada pelos parâmetros (Young, Rose, Karnowski, Lim, & Patton, 2015). As secções seguintes apresentam o funcionamento desta técnica, as arquiteturas associadas ao DL e ainda alguns exemplos de aplicação no quotidiano.

3.2.1 Arquiteturas de Redes Neurais

Esta secção apresenta as várias arquiteturas de redes neurais atualmente exploradas na área de ML.

As redes ***Multilayer Perceptron (MLP)*** são o tipo de redes neurais mais clássico. São frequentemente usadas para conjuntos de dados em formato tabular (Ramchoun, Idrissi, Ghanou & Ettaouil, 2016), aplicado a classificação de modelos de previsão e a problemas de previsão de regressão. Em termos gerais, os pixéis de uma imagem podem ser reduzidos a uma linha de dados e podem ser usados como entrada num MLP. Também, as palavras de um documento podem ser passadas como uma linha de dados de entrada numa MLP. São também adequadas para tratamento de dados de imagens, dados de texto, dados de séries temporais. Alguns exemplos de aplicação deste tipo de rede neural em (Isa & Mamat, 2011; Panchal, Ganatra, Kosta & Panchal, 2011; Lee & Choeh, 2014).

As ***Convolutional Neural Networks (CNN)*** são um tipo de rede neural desenhado para mapear dados de imagens numa variável de saída. Assim, são extremamente adequadas para desenvolver uma representação interna de uma imagem 2D, classificação de modelos de previsão e a problemas de previsão de regressão. Genericamente são bastante adequadas para dados que possuem uma relação espacial (Gehring, Auli, Grangier, Yarats & Dauphin, 2017).

As ***Recurrent Neural Networks (RNN)*** foram desenhadas para trabalhar numa sequência de modelos de previsão (Sak, Senior & Beaufays, 2014). São exemplos destes modelos:

- Um-para-vários em que uma observação de entrada é mapeada numa sequência com múltiplos passos como saída;
- Vários-para-um em que uma sequência de múltiplos passos de entrada é mapeada numa classe ou numa quantidade;
- Vários-para-vários: Uma sequência de múltiplos passos de entrada é mapeada numa sequência com múltiplos passos de saída.

Tradicionalmente, as RNNs são muito difíceis de treinar. O *Long Short-Term Memory* (LSTM), ou rede LSTM é a RNN mais popular uma vez que resolve os problemas de treinar uma RNN e tem sido adotada por muitas aplicações, nomeadamente em *gaming* para aprendizagem de experiência de jogo (Mnih et al., 2015; Patel, Carver & Rahimi, 2011; Tastan & Sukthankar, 2011). Esta será a arquitetura de redes neurais explorada no desenvolvimento deste trabalho, estando diretamente associada ao conceito de RL (Mnih et al., 2016) abordado nas secções seguintes.

3.2.2 Elementos Parâmetros

Os principais parâmetros de DL são a taxa de aprendizagem, a taxa de exploração e o número de episódios. Estes parametrizáveis na configuração de uma rede neural que suporta o DL e são detalhados nas secções seguintes.

A **taxa de aprendizagem** é um dos principais parâmetros de um modelo de aprendizagem e controla o nível de alteração que é aplicado aos pesos de relevância de ações por cada iteração de

aprendizagem feito nesse mesmo modelo (Eskandar, & Williams, 2010; Hinton, Srivastava & Swersky, 2012). Tipicamente, taxas de aprendizagem mais baixas vão reduzir a perda de informação obtida durante a aprendizagem e são menos propensas a serem afetadas pelo ruído gerado por uma aprendizagem não estruturada, mas vão aumentar o tempo que um modelo leva a chegar a um resultado ótimo. Nestes casos, é comum encontrar-se um problema de *overfitting*. Para taxas de aprendizagem maiores, o pressuposto é de que os algoritmos vão demorar menos tempo a chegar a um resultado sub-ótimo, correndo no entanto o risco de demorarem drasticamente mais tempo a chegar a um resultado ótimo, com o risco potencial de nunca conseguirem chegar a esse resultado devido a priorizar sempre explorar novas opções em prol de aplicar o conhecimento já adquirido.

A **taxa de exploração** é outro parâmetro crucial na configuração de um modelo de aprendizagem já que define quão rapidamente esse modelo abandona aquilo que aprendeu para ir à procura de uma nova solução (Chao, Tao, Yang, Li & Wen, 2015). Este valor varia entre 0 e 1, sendo valores mais próximos do 0 propensos a explorarem menos as ações disponíveis e executando apenas ações exclusivamente baseadas no conhecimento previamente adquirido, enquanto que com valores mais próximos do 1 o algoritmo procurará aplicar aleatoriedade na tomada de decisão, permitindo assim explorar opções fora da área de conforto e aprender sobre combinações de ações-recompensas/consequências ainda não exploradas.

O **número de episódios** representa o número de vezes que um agente é treinado num modelo de aprendizagem, executando ações num determinado ambiente e sendo avaliado nessas ações com um índice de peso sobre a ação e estado em que este agente se encontra. O número de episódios necessários para obter resultados ótimos numa análise dependerá sempre dos fatores inerentes à complexidade e aos critérios de sucesso do ambiente onde estes episódios são aplicados (Duan et al., 2016).

3.2.3 Elementos Arquiteturais

Os principais elementos arquiteturais de DL são os nós de camadas, as camadas de entrada, as camadas escondidas, as camadas de saída e as funções de ativação (Schmidhuber, 2015). Estes elementos são descritos nas secções que se sucedem.

Um(a) **nó/unidade de camada** é a unidade básica de computação de uma determinada camada de uma rede neural. Cada nó numa rede tem um peso (relevância) de uma terminada ação associado a si. Este nó é responsável por receber dados de outro nó da rede (*inputs*) e calcular a soma de índice de relevância (o peso) baseado nos *inputs* anteriores, através da aplicação de uma função de ativação.

Uma **camada de entrada** representa uma camada de entrada numa determinada rede neural. Nesta camada, é submetido um determinado valor para análise que será posteriormente processado pelas

camadas subsequentes, através de funções de ativação (secção 3.2.1.7) responsáveis por processar a informação dentro de uma (ou mais) camadas escondidas.

As **camadas escondidas** são o conjunto de camadas dentro de uma rede neural que se encontram entre as camadas de entrada e saída, tendo a responsabilidade de medir os parâmetros de entrada, atribuir pesos (*Q-values*) de relevância e passar os novos valores para a camada de saída ("Hidden Layer", n.d).

A **camada de saída** representa a última camada numa rede neural, responsável pela produção do resultado final desta rede neural. O resultado é processado através da intervenção das camadas escondidas presentes na rede neural e dos *inputs* recebidos nas camadas de entrada.

Uma **função de ativação** é um mecanismo responsável por definir qual o *output* de uma determinada camada escondida, sendo que este *output* é sempre baseado nos *inputs* submetidos à camada numa determinada iteração. Existem dois grandes grupos de funções de ativação: lineares e não lineares. As mais comuns nos dias de hoje são as não lineares e tipicamente devolvem valores entre 0 e 1, ou -1 e 1, consoante a função de ativação escolhida (Gupta, 2017).

3.2.4 Problemas Comuns

Existem vários problemas na implementação de algoritmos de DL (Schmidhuber, 2015) tais como o *overfitting*, qualidade e consistência de dados disponíveis para alimentar os modelos e necessidade de altas capacidades de processamento.

O principal problema de ***overfitting*** acontece normalmente quando as taxas de aprendizagem testadas são elevadas, podendo causar um cenário onde o algoritmo aprende depressa demais e assume rapidamente ter encontrado o resultado ótimo, em prol de explorar continuamente a melhor ação a executar num determinado estado e comprometendo assim a capacidade do modelo se generalizar. Existem várias abordagens para mitigar este fenómeno, nomeadamente quer testando diferentes taxas de aprendizagem, quer através de técnicas de penalização, que na prática são a aplicação de penalizações aos agentes de IA na forma de pontos descontados na sua performance geral e que assim desencorajam a repetição consecutiva dos mesmos erros no processo de aprendizagem (Domingos, 2012).

Outro dos principais desafios na implementação de DL continua a ser a **generalização dos modelos de aprendizagem**, ou seja, a complexidade de garantir que os modelos de aprendizagem tenham a capacidade de se generalizar. Especificamente, estes modelos conseguirem adaptar-se a ambientes onde são aplicados sem treino prévio nesse ambiente em particular, tendo apenas acesso ao conhecimento obtido em sessões de treino anteriores, não relacionadas especificamente com o problema atual (Tassa et al., 2012).

A **ineficiência e a escassez de amostras de dados** são dois problemas presentes na implementação de algoritmos de DL.

Em aplicações de DL em *gaming*, a ineficiência nas amostras de dados é retratada em estudos recentes relativos ao *benchmark* da performance de algoritmos de DL em *gaming* (Hessel et al., 2018) detalhando a necessidade de altas capacidades de processamento e volumes de dados para conseguir competir com a performance que um ser humano consegue adquirir em minutos em algumas das aplicações observadas neste estudo.

A escassez de amostras de dados para outros problemas não relacionados com mundos virtuais (como é o caso do *gaming*) é ainda mais facilmente observável, tendo em consideração a facilidade inerente à configuração de ambientes virtuais e de executar simulações de experiência nestes ambientes, quando comparando o mesmo cenário num problema do mundo quotidiano.

3.2.5 Exemplo de Estrutura de Rede Neural

A estrutura de uma rede neural é constituída por uma ou mais camadas de entrada, uma ou mais camadas de processamento (camadas escondidas), uma ou mais funções de processamento (funções de ativação) e uma ou mais camadas de saída (Nguyen & Widrow, 1990), tal como apresentado na Figura 3-1.

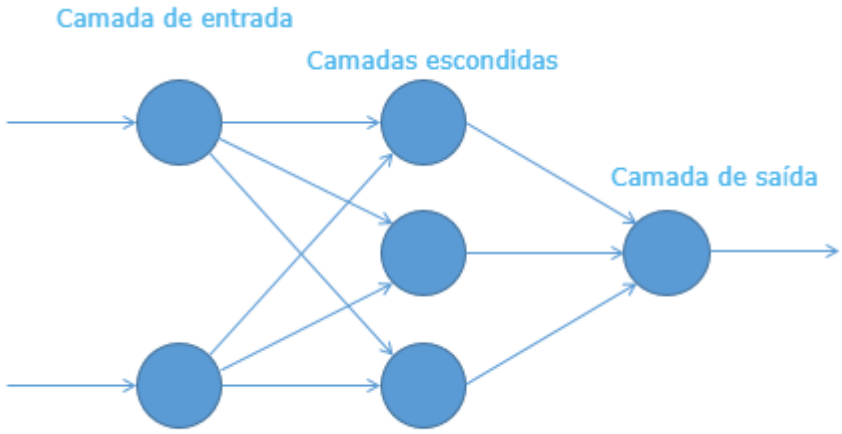

Figura 3-1: Exemplo de Estrutura de uma Rede Neural

3.2.6 Áreas de Aplicação

O DL é aplicado em várias áreas, tendo sido especialmente relevante no desenvolvimento de novas funcionalidades nas seguintes áreas: reconhecimento automático de discurso (Deng & Li, 2013),

reconhecimento automático de imagens (Cevikalp & Triggs, 2010), processamento de linguagem natural (Collobert et.al., 2011) sistemas de deteção de fraude fiscal (Basta et al., 2009) e piloto automático para veículos de transporte sem condutor (Gupta & Merchant, 2016).

Na Figura 3-2, são ainda retratadas outras áreas de aplicação para DL, nomeadamente relativas a investigação e estudos publicados nas áreas descritas (Choudhary, 2018).

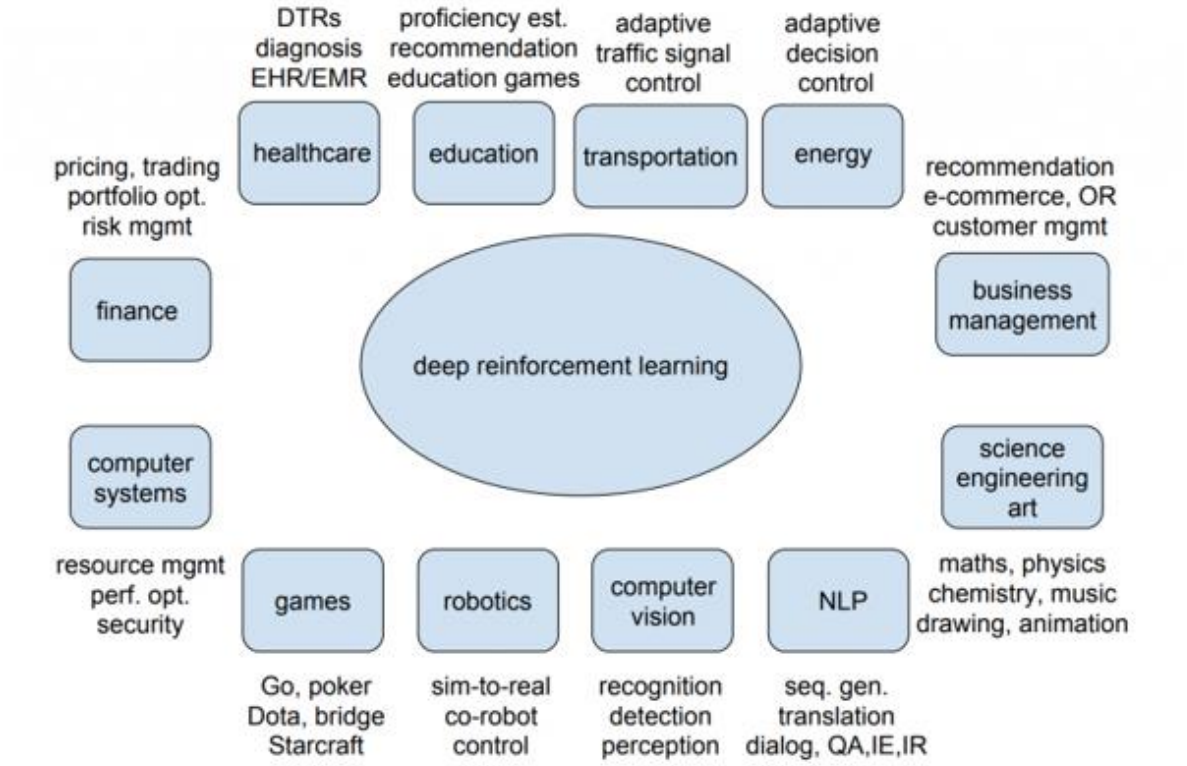

Figura 3-2: Exemplos de Aplicação de Deep Learning (Choudhary, 2018)

Assim, as várias indústrias e áreas onde DL é aplicado demonstram a necessidade de generalização dos modelos trabalhados com esta abordagem e também o potencial de evolução que a área de DL tem no mundo tecnológico.

3.3 Reinforcement Learning

O *reinforcement learning (RL)* é uma área de ML que consiste num conjunto de técnicas e soluções para dar a agentes de IA a capacidade de interagir num determinado ambiente, tendo como principal missão a de capacitar estes agentes com inteligência e que lhes permita tomar ações que resultem na maximização de uma recompensa cumulativa (Nicholson, n.d).

Tendo como principal objetivo a escolha da melhor ação possível para cada estado (aquela que dá ao agente o valor de recompensa mais alta), o RL tem a capacidade de classificar em termos de valor as ações e respetivas recompensas. Como estas ações estão sempre dependentes de um determinado estado, um dos principais fundamentos deste conceito é a criação de pares de ação-estado para que os algoritmos possam utilizar esta informação e consigam definir quais as próximas ações a tomar.

Outro fator fundamental na implementação de RL é a escolha do ambiente onde o algoritmo executa as ações. O sucesso da aprendizagem destes algoritmos está diretamente ligado com a qualidade do ambiente, no que toca ao que é possível observar, dentro daquilo que são as definições de características, variáveis e limitações de cada ambiente. Isto torna a área de *gaming* ideal para a implementação e treino de algoritmos de RL, sendo estes ambientes completamente virtuais e totalmente parametrizáveis.

3.3.1 Elementos

Os principais elementos de RL são o agente, a ação, o ambiente, o estado, a recompensa e a política. Estes elementos são retratados de seguida (Burda, 2018).

O **agente** interage e executa ações num determinado **ambiente**. No caso de RL, o **agente** é o algoritmo que interage com um determinado **ambiente**.

Uma **ação** representa todas as ações que um agente pode executar num determinado estado. Estas ações são variáveis consoante o ambiente onde o agente interage. Exemplos dentro do contexto de *gaming*: Caminhar para a esquerda/direita, saltar, parar.

O **ambiente** representa o mundo pelo qual o agente navega. O **ambiente** tem em consideração o estado do **agente** e a sua próxima **ação** como *inputs*, e devolve ao **agente** a **recompensa** pela **ação** tomada e o seu próximo **estado**.

O elemento **estado** representa o estado/situação atual em que o **agente** se encontra. Pode tomar a forma de um local, um momento ou um acontecimento.
O elemento de **recompensa** é essencialmente uma recompensa que provém do *feedback*, determinando o sucesso ou insucesso do **agente** relativamente a uma dada **ação** executada.
Uma **política** representa a estratégia que o **agente** aplica para definir a sua próxima **ação**, baseada no **estado** atual. Tipicamente, a **política** é definida ao mapear **estados** e **ações** às **recompensas** associadas a cada par de **ação**/**estado**, de forma a identificar eficientemente a próxima **ação**.

3.3.2 Q-learning

O *Q-learning* é uma técnica de RL que introduz uma nova abordagem ao RL por forma a conseguir medir a ação ideal para cada par de ação-estado, tendo em conta não só o estado atual, mas a

cumulatividade de todas as ações-estado medidas até um determinado momento. (Watkins & Dayan, 1992).

A técnica de *Q-learning* aplica a Equação de Bellman (3-1) para determinar as recompensas a longo-prazo. Esta equação atribui fundamentalmente a recompensa a longo prazo para uma determinada ação igual à recompensa imediata da ação no estado corrente, combinada com a melhor recompensa possível expectável da ação a ser executada no estado imediatamente seguinte (Barron & Ishii, 1989).

$$Q(s.a) = r + \gamma(max(Q(s', a')) \quad (3\text{-}1)$$

Na sua forma mais simplista, o *Q-learning* é implementado para armazenar esta informação em tabelas (Tabela 3-1: Exemplo de *Q-table*). Cada linha nestas tabelas armazena o estado e uma ação possível, tal como a sua recompensa.

Tabela 3-1: Exemplo de *Q-table*

	Ação	
Estado	**Direita**	**Esquerda**
0	0	0
1	-50	65.71
2	55.049	73.7
3	72.9	91
4	63	99
5	81	100

Para implementações mais avançadas, onde tipicamente existe maior volume de dados relativamente a pares de ação-estado e respetivas recompensas, é possível combinar esta técnica com RNNs e implementar d*eep Q-learning*, que essencialmente permite que sejam calculadas infinitas (ou pelo menos, até que o treino seja cancelado) possibilidades e respetivos resultados em termos de recompensa a longo termo. Exemplos disto são os trabalhos de (Mnih et al., 2013; Hester et al., 2018; James & Johns, 2016).

A técnica de *Q-learning* conta com duas propriedades, que determinam a forma como os pesos (relevância) de cada ação num determinado estado são avaliadas, sendo estas o *Q-value* e o Fator de Desconto, que são detalhados de seguida:

- *Q-value*: É o valor de retorno esperado a longo prazo para um determinado conjunto de pares de ação-estado, condicionado pelo Fator de Desconto. Este valor mapeia posteriormente os pares de ação-estado com as recompensas possíveis.
- Fator de Desconto: É essencialmente um valor entre 0 e 1 que determina o fator de desconto para futuras recompensas, em contraste com o valor que as recompensas imediatas têm para o agente. O valor das recompensas futuras é reduzido consoante o quão no futuro estas serão recebidas. Para exemplificar, um Fator de Desconto de 1 significa que as

recompensas futuras têm tanta relevância como as recompensas imediatas. Quanto mais baixo for o valor, mais as recompensas futuras são "descontadas" perante as recompensas imediatas.

3.3.3 Cenário Ilustrativo

Nesta secção é ilustrada a aplicação de um algoritmo de RL, com recurso à técnica de *Q-learning* e à utilização de redes neurais, ao jogo FrozenLake (Hu & Hu, 2019), apresentado na Figura 3-3.

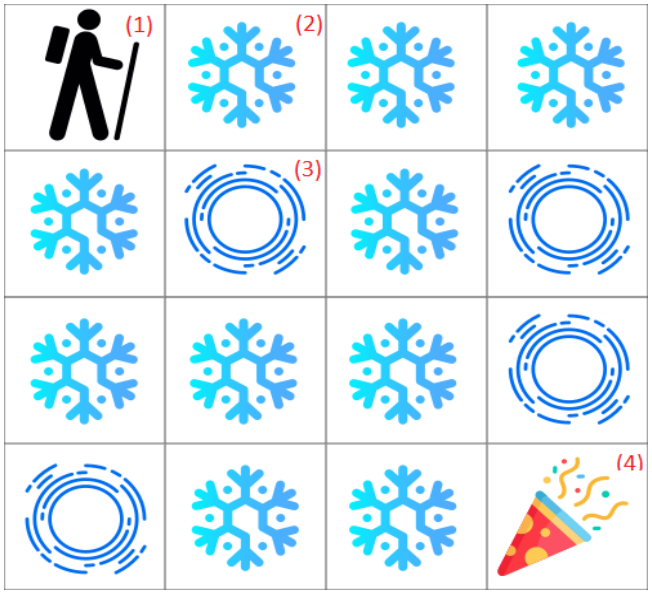

Figura 3-3: Exemplo de cenário de FrozenLake

O jogo *FrozenLake* consiste essencialmente numa tabela 4x4 que contém exclusivamente um dos seguintes 4 objetos: bloco de entrada **(1)**, bloco de saída **(4)**, bloco seguro **(2)** ou bloco com buracos **(3)**. O objetivo do jogo é deslocar-se desde o bloco de entrada até ao bloco de saída, sem nunca atingir um bloco com buracos no gelo. Em qualquer momento do jogo, o agente pode tomar uma das seguintes ações de movimento: esquerda, direita, cima e baixo. A recompensa para qualquer movimento é sempre 0, com a exceção do movimento que leva o agente a atingir o bloco de saída, em que recebe uma recompensa 1. Assim, um algoritmo que consiga determinar recompensas no longo-termo é o ideal para resolver este jogo.
Relativamente à estrutura do jogo, existem 16 estados (um por cada bloco) e 4 ações possíveis (os movimentos), podendo então transformar este cenário numa matriz 16x4, com os respetivos pares de ação-estado e as recompensas para cada um. Assim, para resolver o jogo com sucesso o modelo será numa primeira fase inicializado com os valores de recompensa a 0 para qualquer ação, sendo estes atualizados à medida que a aprendizagem do agente progride no ambiente.

Utilizando um paradigma de RL, o processo de aprendizagem é inicializado com o agente a iterar sobre os vários estados onde se encontra e executando ações aleatoriamente, sendo cada uma destas ações avaliadas na forma de uma recompensa. A recompensa é calculada através da utilização da rede neural definida, que é responsável por receber na sua camada de entrada o estado atual e,

mediante conhecimento prévio, atribuir através das camadas escondidas um *Q-value* a cada par de estado-ação disponíveis. A camada de saída da rede neural é responsável por produzir os *outputs* desta avaliação e suportadas a decisão da próxima ação do agente. Este processo iterativo é continuo até que aconteça uma de duas condições: o jogo termina porque o agente atingiu o objetivo final (chegar ao bloco de saída) ou porque o número máximo de episódios (iterações) foi atingido.

Neste cenário, a ideia fundamental aplicada via RL é que, dentro das parametrizações definidas, o agente seja incentivado, através da recompensa definida, executar o maior número de combinações de ações possíveis para tentar obter essa recompensa e assim concluir o jogo com sucesso.

3.4 Tecnologias e Bibliotecas

Nesta secção é feito um breve resumo daquelas que foram as principais tecnologias utilizadas neste projeto: Python, TensorFlow, Keras e OpenAI Gym.

3.4.1 Python

Python é uma linguagem de programação de interpretação de alto nível e orientada a objetos que conta com estruturas de dados de alto nível e dinamismo na escrita e processamento de métodos (Millman & Aivazis, 2011). Python tem uma sintaxe propositadamente perto da linguagem humana, realçando a leitura de código e assim reduzindo o esforço de manutenção de código. Tendo como principal foco a produtividade, suportando o desenvolvimento e aprovisionamento rápido de aplicações, é uma linguagem de programação extremamente fácil de aprender e normalmente enaltecida pelos programadores (Millman & Aivazis, 2011).

É precisamente pelos fatores descritos acima que Python é uma das linguagens de programação mais utilizadas no mundo na área de ML (Pedregosa et.al, 2011; Raschka & Mirjalili, 2017).

3.4.2 Keras

Keras é uma API escrita em Python, que trabalha em alto nível com redes neurais e que é suportada pela plataforma TensorFlow.

A API do Keras foi implementada com o principal objetivo de disponibilizar uma plataforma de desenvolvimento com redes neurais (Gulli & Pal, 2017) que permita experimentação rápida e eficaz. É ideal para a montagem de protótipos com redes neurais e suporta redes neurais dos tipos MLP, CNN e RNN.

A API do Keras conta com quatro características principais, nomeadamente:

- Amigável para o utilizador: Minimiza a consistência e complexidade da invocação dos métodos presentes na API com métodos muito simples de entender e implementar. Utiliza linguagem muito próxima da humana para relatar erros ocorridos na plataforma;
- Modular: Contém vários tipos de modelos e torna a configuração destes modelos associados a ML muito simples. Permite a combinação de modelos de vários tipos;
- Extensível: Permite estender modelos existentes e também adicionar novos modelos. A possibilidade de extensão destes modelos torna o Keras ideal para investigação de nível médio/avançado nesta área;
- Integrável com Python: Os modelos utilizados nesta API podem ser escritos em Python, tornando a sua montagem e todo o processo inerente a testar e estender estes modelos mais simples.

3.4.3 TensorFlow

TensorFlow é uma plataforma de código aberto desenvolvida e suportada pela Google, que permite a implementação de modelos de ML, através da criação de *dataflow graphs* que são suportados por algoritmos de computação numérica para aplicar a lógica criada nesses *graphs* (Abadi et.al., 2016).

Os principais objetivos da plataforma *TensorFlow* são facilitar o desenvolvimento de mecanismos que permitam adquirir dados, treinar modelos de aprendizagem e utilizar esta informação para conseguir produzir predições e refinar estes resultados. A *TensorFlow* suporta este aspeto ao expor as suas bibliotecas de código-aberto, permitindo que sejam criados modelos de ML e estruturadas redes neurais que tenham a responsabilidade de processar a informação contida nesses modelos. Assim, a *TensorFlow* poderá ser vista como o motor responsável por processar de forma eficiente todos os dados envolvidos num algoritmo de ML.

A plataforma *TensorFlow* pode ser consumida através de aplicações escritas em Python, disponibilizando uma *Application Programming Interface* (API) para esse efeito. Conta ainda, no *backend*, com um motor de processamento escrito em C++, que trata da computação dos algoritmos implementados.

3.4.4 OpenAI Gym

A OpenAI é uma organização sem fins lucrativos que tem como principal missão a de democratizar e garantir que a IA beneficia toda a humanidade de igual forma.

A OpenAI disponibiliza publicamente um conjunto de ferramentas (Gym) de investigação na área de RL com o principal intuito de permitir desenvolver e comparar algoritmos de nessa área (Brockman et al. 2016).

O *Gym* da OpenAI é constituído por duas componentes principais (OpenAI, 2016): uma coleção de *benchmarks* a problemas expostos numa interface comum, que detalha problemas comuns e modelos de resolução na área de RL, e um website que permite que os seus utilizadores partilhem a resolução desses problemas os resultados obtidos na comparação dos algoritmos. Para este efeito, a OpenAI disponibiliza acesso a vários ambientes de jogos, permitindo que sejam escritos e testados quaisquer algoritmos de RL para solucionar as coleções de testes expostas nesta plataforma e ainda que seja medida a sua performance.

3.5 Diagrama de Tecnologias e Bibliotecas

A forma como as tecnologias e bibliotecas abordadas neste projeto interagem entre si é demonstrada na Figura 3-4.

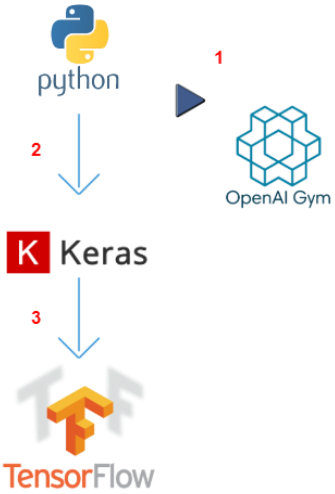

Figura 3-4: Diagrama de Tecnologias e Bibliotecas

Neste diagrama, a aplicação é escrita em **Python (1)**, que integra com a plataforma **OpenAI Gym** para instanciar os ambientes de jogos que serão utilizados nos modelos de aprendizagem. A aplicação escrita em **Python (1)** integra ainda com a API do **Keras (2)**, que expõe as funcionalidades necessárias à interação com redes neurais. O processo necessário para a criação, modelação e alimentação das redes neurais é tratado entre a API do **Keras (2)** e a plataforma do **TensorFlow (3)**.

3.6 Exemplos de Reinforcement Learning aplicado em Gaming

Para os jogos abordados neste projeto, já existem alguns artigos que descrevem a aplicação de algoritmos de RL e que descrevem essencialmente a técnica aplicada e alguns resultados.

Exemplos destes artigos (Ma, Tang & Zhang, 2016; Carr, 2005; Romdhane & Lamontagne, 2008) demonstram a aplicação de técnicas de *Reinforcement Learning* aos jogos Snake e Tetris. No exemplo abordado no artigo do Snake, são aplicadas as técnicas de *Q-learning* e SARSA (*State–Action–Reward–State–Action*) com recurso ao suporte de redes neurais do tipo CNN, e é feita uma análise de comparação de resultados obtidos após aplicar estas duas técnicas. No artigo que aborda o jogo Tetris, é aplicada a técnica de *Q-learning* com recurso a uma metodologia de *Case Based Reasoning* (CBR), em que são definidos vários casos, cada um representando um padrão, que descrevem a altura relativa de cada conjunto de peças do Tetris, por forma a calcular onde a próxima peça deverá ser calculada. Em termos de arquitetura de implementação, este artigo não menciona a aplicação de qualquer tipo de rede neural. Em termos de resultados, são demonstrados resultados provenientes de abordagens sem treino, com treino sem exploração, e com treino incluindo uma taxa de exploração.

4. Q-learning Aplicado em Gaming

Neste capítulo é feita uma análise ao trabalho produzido durante a implementação e exploração dos algoritmos para a aplicação da técnica de *Q-learning* em *gaming*. Esta análise passa pela visão global sobre o funcionamento, regras e considerações dos jogos escolhidos, a exploração dos ambientes de testes de jogos e a parametrização dos algoritmos, e dos seus hiperparâmetros, por forma a ser possível analisar com detalhe os resultados experimentais obtidos na aplicação da técnica de *Q-learning* aos diferentes paradigmas definidos para o âmbito deste projeto.

Para explorar a técnica de *Q-learning* em *gaming*, foram selecionados 2 jogos com características distintas, por forma a demonstrar a aplicabilidade desta técnica independentemente das regras de funcionamento do ambiente onde a técnica é aplicada. Os jogos selecionados são o *Snake* e o *Tetris*.

Para demonstrar as capacidades da técnica de *Q-learning* exploradas neste projeto, nas próximas secções será detalhado o desenho do algoritmo implementado, a aplicação deste algoritmo nos ambientes de jogo e os resultados obtidos na exploração da aplicação da técnica em cada um destes ambientes.

4.1 Desenho do Algoritmo

Nesta secção, é demonstrada a visão global relativa ao desenho do algoritmo, passando pela descrição da aplicação do *Q-learning* aos jogos, a sua utilização com redes neurais, as técnicas utilizadas para mitigar os principais problemas anteriormente descritos relativos à implementação de RL e exemplos de código relevantes na elaboração deste projeto.

O algoritmo explorado neste projeto recorre à aplicação do *Q-learning* para treinar o modelo de aprendizagem, através da observação de comportamento do agente e medição de uma recompensa atribuída a cada ação tomada pelo mesmo. Existe ainda uma componente de escolha de uma ação aleatória (em prol de medida) por forma a garantir que o algoritmo seja dotado da capacidade de exploração de novas opções, ao invés de seguir sempre por um caminho onde escolhe uma ação previamente aprendida, garantindo assim a exploração de opções adicionais e uma maximização dos resultados obtidos. Numa visão de alto nível, o algoritmo executa a seguinte lógica:

- O agente inicia o jogo e os pesos (*Q-values*) de todos os movimentos possíveis inicializados aleatoriamente;
- O algoritmo recebe o estado inicial do agente;
- O agente executa uma ação relativa ao estado atual onde se encontra. Esta ação é escolhida em função de um de dois fatores: o conhecimento prévio que obtém dos *Q-values*, escolhendo aquela ação que tem maior peso atribuído, ou escolhendo uma ação aleatória;
- Quando o agente executa uma ação, o valor da recompensa desse par de ação/estado é guardado e, em função de todo o conhecimento adquirido até então, são calculados, em jeito de predição, os valores de todas as ações disponíveis para a iteração imediatamente seguinte;
- Na próxima iteração, os valores da potencial recompensa de cada par de ação/estado disponíveis são considerados na decisão do algoritmo sobre qual a próxima ação a executar;
- Todos os passos descritos nos pontos anteriores (à exceção do primeiro) são repetidos até que o jogo termine ou que seja atingido o número máximo de episódios parametrizado.

A rede neural é uma peça instrumental no aperfeiçoamento da aplicação do *Q-learning* nos jogos abordados neste projeto, já que permite que o algoritmo processe não só o estado atual e possível recompensa de uma determinada ação, como também todos os estados e recompensas de todos os movimentos possíveis num determinado momento. Desta forma, permite prever qual a melhor ação a executar nesse mesmo momento. Assim, a rede neural é estruturada com a camada de entrada, onde recebe o estado do agente, as camadas de processamento, onde são aplicadas as funções de ativação que permitem obter estas predições (na forma de *Q-values*), e a camada de saída, onde são expostos os *Q-values* de cada ação para suportar a próxima ação do agente.

Na avaliação do peso (*Q-value*) de cada ação, a rede neural é responsável por avaliar cada estado e ação e, especificamente a cada jogada, quais são os benefícios e as potenciais consequências negativas de executar cada uma destas ações dentro de um dado estado.

Para mitigar o problema de *overfitting* já descrito neste relatório, foram aplicadas diferentes taxas de aprendizagem e aplicadas penalizações nas recompensas definidas para cada ação. A especificação das recompensas e penalizações definidas para cada jogada é detalhada nas secções 4.2 e 4.3.

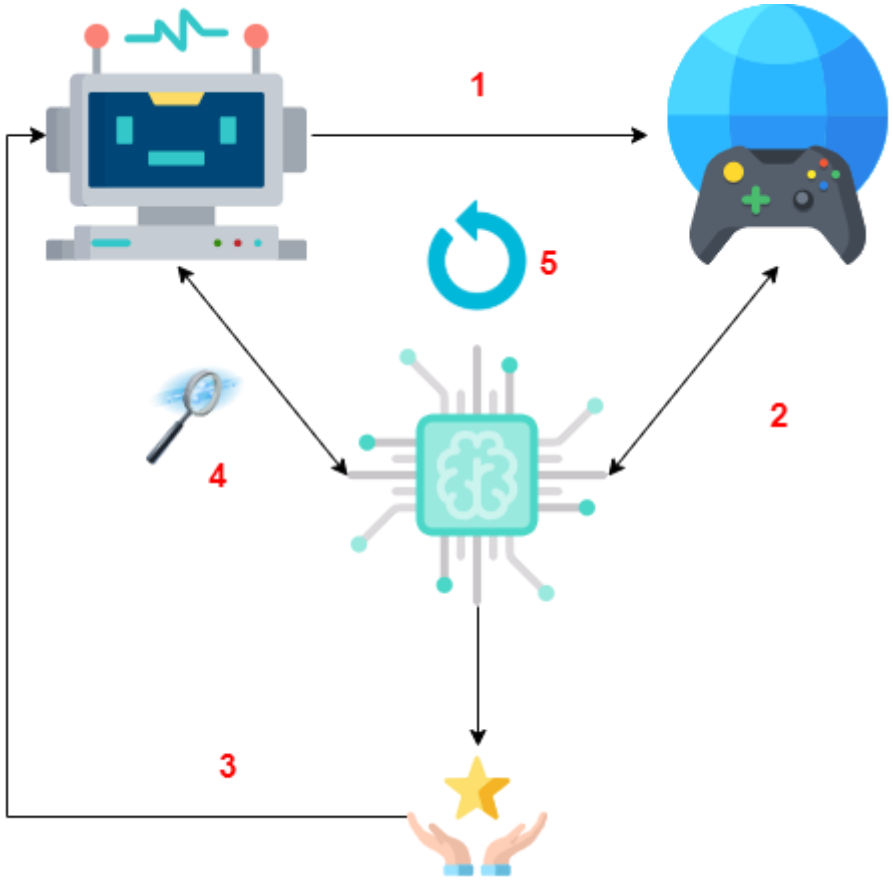

Figura 4-1: Diagrama Funcional do Algoritmo

Nesta secção, é descrita uma visão global sobre a estrutura funcional do algoritmo, as parametrizações exploradas, as métricas e o contexto para a aplicação, os requisitos técnicos e os pressupostos a ter em conta na análise ao trabalho produzido neste projeto.

4.1.1 Diagrama Funcional

O diagrama na Figura 4-1 descreve o funcionamento do algoritmo explorado neste projeto. O funcionamento do algoritmo pode ser interpretado na forma de um agente **(1)** que interage iterativamente **(5)** com um determinado ambiente de jogo, procurando com suporte de uma rede neural **(4)** qual a próxima ação que deverá executar e que lhe tratará uma recompensa associada **(3)**. Este processo iterativo **(5)** permite que a rede neural calcule e armazene a informação associada ao treino **(2)** do agente por forma a suportar uma decisão eficiente por parte do mesmo.

4.1.2 Parametrização

Na fase de desenho do algoritmo, foi prevista a definição dos hiperparâmetros e respetivos valores do modelo de aprendizagem. O próximo passo é detalhar os parâmetros e o conjunto das configurações exploradas neste projeto. Os hiperparâmetros configuráveis no modelo de aprendizagem são:

- Número de camadas escondidas (número inteiro);

- Taxa de aprendizagem (percentagem);
- Taxa de exploração (percentagem);
- Número de unidades dentro das camadas escondidas (número inteiro);
- Número de episódios (número inteiro).

4.1.3 Métricas e Contexto de Análise

As métricas de análise de performance do algoritmo revistas durante a aplicação do algoritmo de *Q-learning* foram as seguintes:

- Número de episódios
- Tempo de treino
- Pontuação máxima obtida
- Pontuação média obtida

O número de episódios permite que sejam simulados vários testes com número de iterações diferentes, enriquecendo assim as amostras de dados na análise dos resultados obtidos. O tempo de treino representa o tempo necessário para atingir um objetivo de performance. A pontuação máxima e a pontuação média permitem analisar resultados de *benchmarking* dentro daquilo que são os máximos de resultados obtidos e a performance média ao longo das iterações testadas dentro do número de episódios analisado.

Considerando os parâmetros descritos acima, na exploração deste projeto foram definidos os conjuntos de configurações detalhados na Tabela 4-1.

Tabela 4-1: Conjuntos de configurações de hiperparâmetros

Estrutura Rede Neural	**Parâmetros Testados**
Rede neural com 1 e 2 camadas escondidas	Taxas de aprendizagem de (%): 0.05%, 0.1%, 1%; Taxas de exploração de (%): 40%, 70%; Número de unidades por cada camada: 60, 120; Número de episódios testados: 50, 100, 200, 500.

Todos os testes executados na elaboração deste projeto foram ainda conduzidos dentro de um contexto onde o agente não tem qualquer conhecimento prévio ou aprendizagem num dado ambiente.

4.1.4 Requisitos Técnicos

Os componentes técnicos necessários à instalação e execução do trabalho produzido neste projeto são:

- Plataforma Python (versão igual ou superior a 3.5), que suporta a linguagem de programação e o processamento do código desenvolvido;
- O TensorFlow (versão igual ou superior a 1.7.2), que representa a camada responsável por aprovisionar as redes neurais abordadas;
- A API do Keras (versão igual ou superior a 2.0.0) que serve de camada de comunicação entre a aplicação (Python) e as redes neurais (TensorFlow);
- O Numpy (versão igual ou superior a 1.0.0), uma biblioteca que permite trabalhar, em Python, com matrizes e *arrays* de largas dimensões e que expõe os métodos matemáticos necessários para este efeito;
- O PyGame, uma biblioteca de Python que permite customizar diferentes parâmetros de um determinado jogo;
- OpenAI Gym, plataforma já detalhada neste relatório.

4.1.5 Ambiente de Execução do Trabalho Experimental

A performance de algoritmos de aprendizagem varia consoante as características do ambiente de execução do trabalho experimental. Este ambiente, que suporta todas as atividades de testes e processamento dos modelos de aprendizagem, conta com uma configuração técnica de uma CPU (*Computer Processing Unit*) e suportados pela mesma máquina, com a as seguintes especificações técnicas:

- **Sistema Operativo** Microsoft Windows 10 Enterprise;
- **CPU** Intel Xeon E-2176M CPU @ 2.70Ghz, 2712Mhz, 6 Cores;
- **RAM** 64GB a 1600Mhz;
- **Armazenamento** Toshiba SSD de 1TB.

4.2 Snake

O Snake é um jogo de labirinto, onde o jogador controla uma linha, que representa uma cobra e cresce em comprimento à medida que vai obtendo mais pontos, representado na forma de comida, num ambiente com obstáculos na forma de paredes. Este jogo é baseado num outro jogo com objetivos semelhantes, o Blockade, que surgiu em 1976. O jogo Snake ganhou bastante notoriedade nos anos 90 devido à sua inclusão nos telemóveis da marca Nokia (“Top 100 Games of All Time”, 2019).

Nas secções seguintes, são descritos os componentes essenciais à compreensão da implementação do algoritmo de *Q-learning* aplicado a este jogo: A estrutura do jogo, o seu funcionamento em termos de regras e critérios de sucesso, as recompensas específicas atribuídas ao jogo mediante a sua natureza e, por fim, uma análise aos resultados obtidos durante a execução dos testes do agente neste jogo.

4.2.1 Estrutura

A estrutura do jogo é composta por três componentes principais: a cobra, a maça e o cenário de jogo.

As dimensões da estrutura do jogo implementado no contexto deste projeto são representadas na Tabela 4-2.

Tabela 4-2: Estrutura do cenário de Snake

Componente	**Altura**	**Largura**
Retângulo (cenário)	400	400
Linha (cobra)	20	20
Quadrado (maça)	20	20

A Figura 4-2 mostra um exemplo do cenário do jogo explorado no contexto deste projeto.

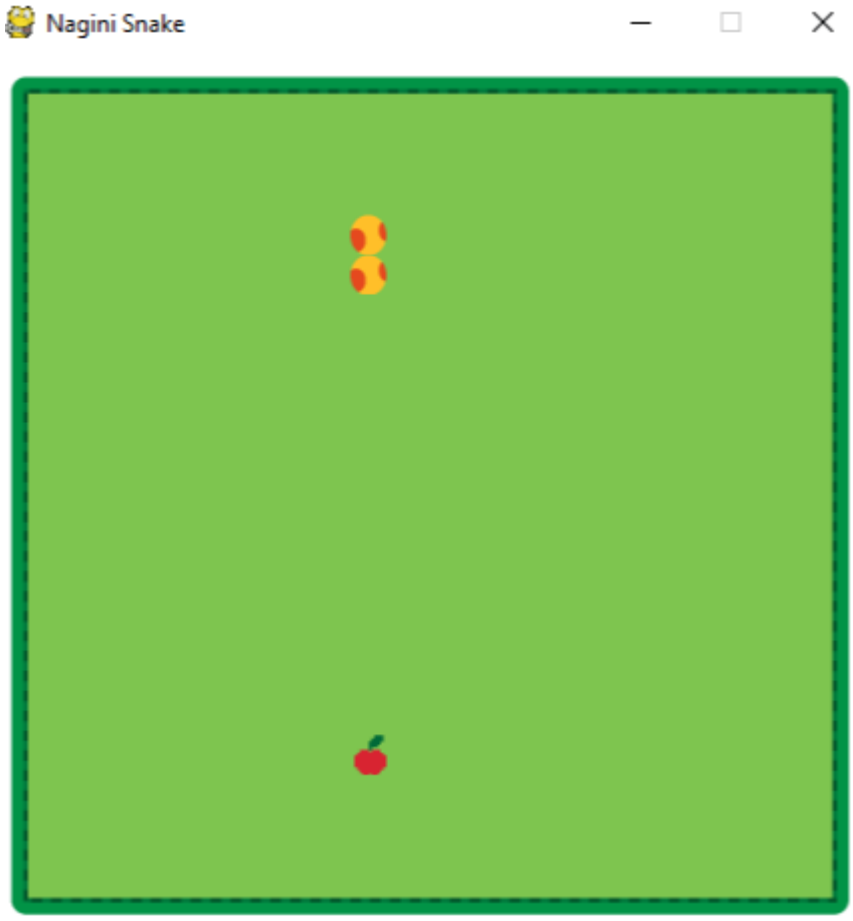

Figura 4-2: Exemplo de cenário do Snake

4.2.2 Funcionamento, Regras e Critérios de Sucesso

O Snake funciona na forma de um objeto que é controlado pelo jogador e que tem a possibilidade de fazer um de três movimentos no tabuleiro: esquerda, direita ou continuar sem alterar direção. Na sua versão mais simplista, é apenas limitado ao longo das extremidades pelas paredes. No

tabuleiro, para além do objeto controlado pelo jogador (i.e., cobra), estará também sempre presente, numa posição aleatória, a maça que representa o objetivo principal do jogo.

O Snake é um jogo de sobrevivência. O seu sucesso é medido pela capacidade de o jogador sobreviver a maior quantidade de tempo possível e conseguir atingir a maior quantidade de pontos. Sendo assim, o objetivo principal do jogo garantir o controlo do movimento da cobra, até que seja possível apanhar a maça presente no tabuleiro. Ao apanhar a maça, o jogador recebe 1 ponto e por consequência fará aumentar o tamanho da cobra – até que o jogo termine. O jogo pode terminar num dos seguintes cenários:

- A cobra embate contra a parede, que é toda a extremidade do tabuleiro;
- A cobra embate contra si própria, proveniente de um movimento que faça com que, à medida que a cobra vai crescendo de tamanho, embate sobre si própria e tem assim o mesmo efeito de embater contra uma parede.

4.2.3 Recompensas e Penalizações Definidas

De acordo com o funcionamento, regras e critérios de sucesso para o jogo já detalhadas anteriormente, foram definidas recompensas a atribuir ao agente na forma da Tabela 4-3

Tabela 4-3: Recompensas e Penalizações Snake

Descrição	**Valor Recompensa**
Comer maçã	10
Mover sem colidir num obstáculo	1
Colidir com obstáculo	-10

4.3 Tetris

O Tetris é um jogo de quebra-cabeças, onde o jogador controla diferentes peças geométricas por forma a tentar criar o maior número de linhas horizontais possível. O Tetris conta atualmente com várias versões, suportadas em múltiplas plataformas e sistemas operativos. A primeira versão pública deste jogo surgiu no ano de 1984, mas foi apenas em 1989, com a inclusão deste jogo no Game Boy, que estabeleceu a sua posição como um dos melhores jogos de sempre (“Top 100 Games of All Time “, 2019).

Nas secções seguintes, são descritos os componentes essenciais à compreensão da implementação do algoritmo de *Q-learning* aplicado a este jogo: A estrutura do jogo, o seu funcionamento em termos de regras e critérios de sucesso, as recompensas específicas atribuídas ao jogo mediante a sua natureza e, por fim, uma análise aos resultados obtidos durante a execução dos testes do agente neste jogo.

4.3.1 Estrutura

Estruturalmente, o jogo é composto pelos seguintes componentes: Um campo de jogo, a peça que o jogador controla num determinado momento, e todas as peças que já foram previamente posicionadas no campo. A Figura 4-3 representa um exemplo de cenário de um jogo de Tetris.

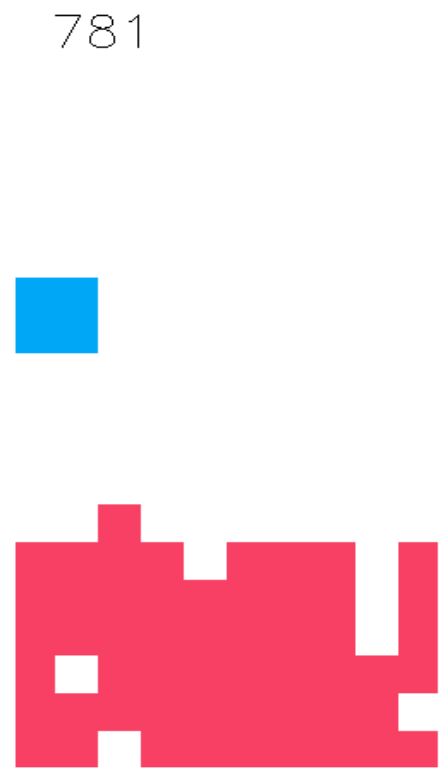

Figura 4-3: Exemplo de cenário de Tetris

No Tetris, existem sete versões de peças geométricas diferentes. A sua estrutura geométrica é demonstrada na Figura 4-4.

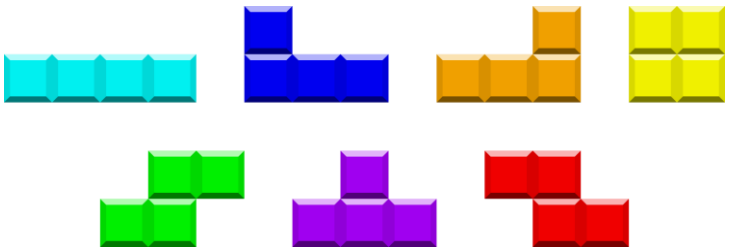

Figura 4-4: Peças do Tetris

4.3.2 Funcionamento, Regras e Critérios de Sucesso

O Tetris é um jogo de quebra-cabeças, tendo como principal objetivo o de completar, dentro do campo, o maior número de linhas horizontais com peças geométricas. O jogador é desafiado a controlar os vários tipos de peças existentes no jogo, que vão descendo verticalmente ao longo do

campo, por forma a conseguir posicioná-las de forma eficiente, ambicionando completar linhas horizontais e sendo assim premiado com pontos por cada nova linha que completar.

No Tetris, existem três componentes variáveis que condicionam a dificuldade do jogo:

- O formato da peça geométrica, que pode tomar a forma de um dos sete tipos visíveis na Figura 4-4. Peças do Tetris e que, considerando as diferentes formas, pode tornar mais ou menos complexa a conclusão das linhas horizontais;
- A forma como as peças são aleatoriamente geradas numa determinada instância do jogo. O jogador controla uma peça de cada vez, sendo a próxima peça gerada assim que a última peça tenha sido posicionada no campo;
- A velocidade a que a peça vai descendo verticalmente. Esta velocidade vai aumentando gradualmente à medida que o jogo avança.

Em termos de movimentos possíveis, o jogador controla num dado momento uma peça geométrica e tem a possibilidade de:

- Movimentar literalmente a peça;
- Rodar a peça;
- Acelerar a descida a peça.

4.3.3 Recompensas e Penalizações Definidas

De acordo com o funcionamento, regras e critérios de sucesso para o jogo já detalhadas anteriormente, foram definidas recompensas a atribuir ao agente na forma Tabela 4-4:

Tabela 4-4: Recompensas e Penalizações Tetris

Descrição	**Valor Recompensa**	**Notas**
Posicionar uma peça	1	
Completar uma linha (sem buracos)	10	
Completar várias linhas simultaneamente (n)	10*n+n	n representa o número de linhas completadas simultaneamente num determinado momento
Posicionar uma peça que cause o fim do jogo	-10	Penalização por perder o jogo

4.4 Análise aos Resultados Experimentais

A análise dos resultados experimentais permite determinar o impacto das diferentes combinações de configurações com hiperparâmetros num dado ambiente e a sensibilidade da performance do algoritmo a variações dos parâmetros especificados para cada configuração. De seguida, são detalhados os resultados obtidos nos testes efetuados dentro dos dois ambientes abordados,

considerando as métricas que foram definidas para o âmbito de testes a executar neste projeto. As combinações específicas de configuração são igualmente detalhadas.

Numa visão geral, é possível perceber que a aplicação da técnica de *Q-learning* nos ambientes de jogo tem a capacidade de resolver o jogo com sucesso. O algoritmo adapta-se, através da aprendizagem, aos obstáculos inerentes ao jogo e supera estes obstáculos progressivamente. No entanto, existe uma clara disparidade entre os resultados obtidos em termos de *benchmarking* da performance do algoritmo, que é causada pelo impacto que os parâmetros configurados têm na forma como o modelo de aprendizagem é treinado.

Para uma visão mais detalhada sobre os resultados experimentais, são descritos de seguida os resultados contextualizados com cada jogo abordado neste documento.

4.4.1 Snake

Ao observar o *benchmarking* de performance e impacto dos hiperparâmetros (Figura 4-5, Figura 4-6, Figura 4-7, Figura 4-8), os melhores resultados em termos de **pontuação média** (22.768) foram obtidos com a menor **taxa de aprendizagem** configurada **(0.05%)**, **1 camada escondida** com **120 unidades** e num cenário de **500 episódios**, enquanto que os piores (0.278) foram obtidos com a maior **taxa de aprendizagem** configurada **(1%)**, **1 camada escondida** e **60 unidades** num cenário de **500 episódios**.

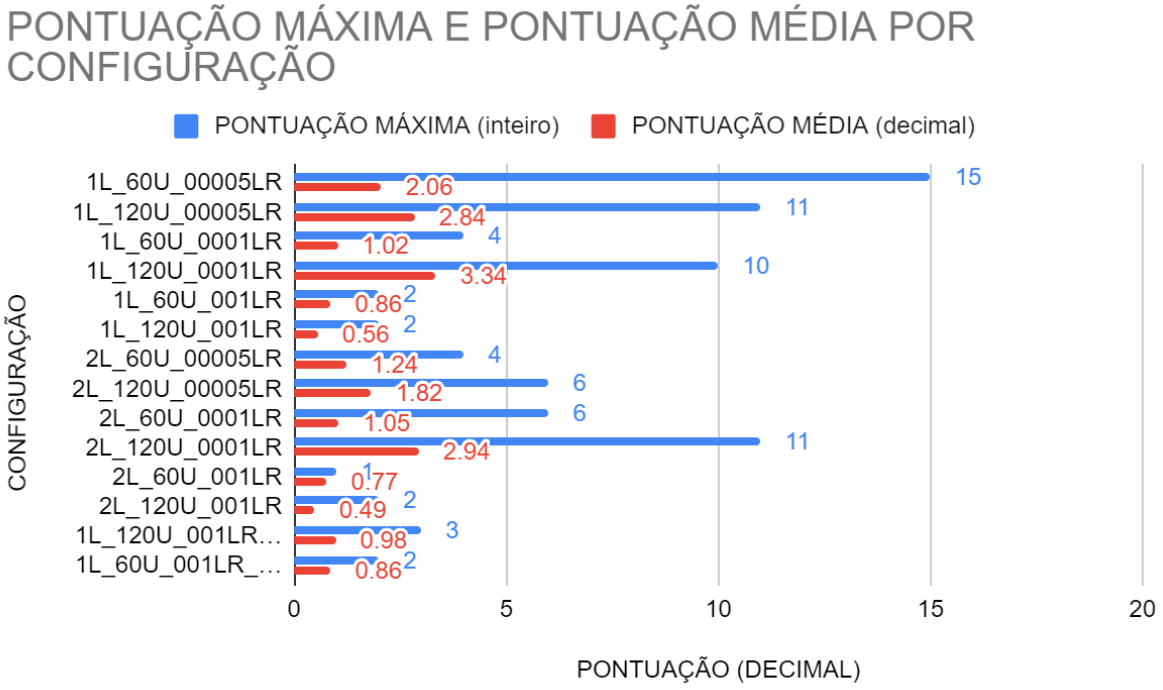

Figura 4-5: *Benchmarking* (pontuação) em 50 Episódios (Snake)

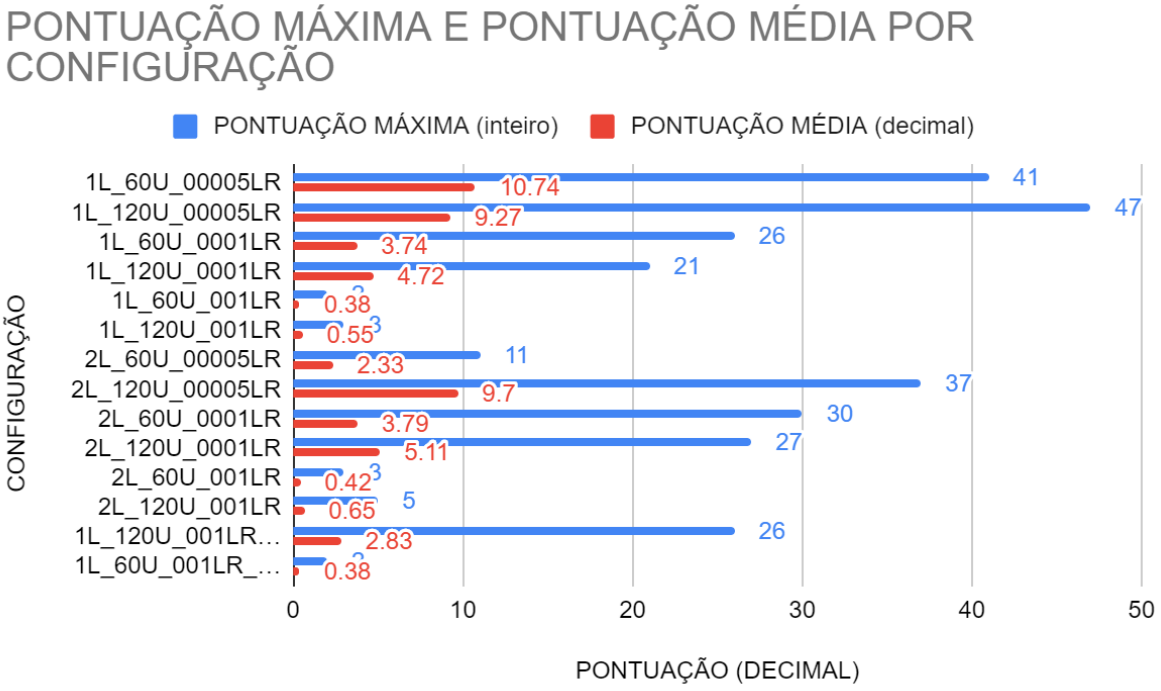

Figura 4-6: *Benchmarking* (pontuação) em 100 Episódios (Snake)

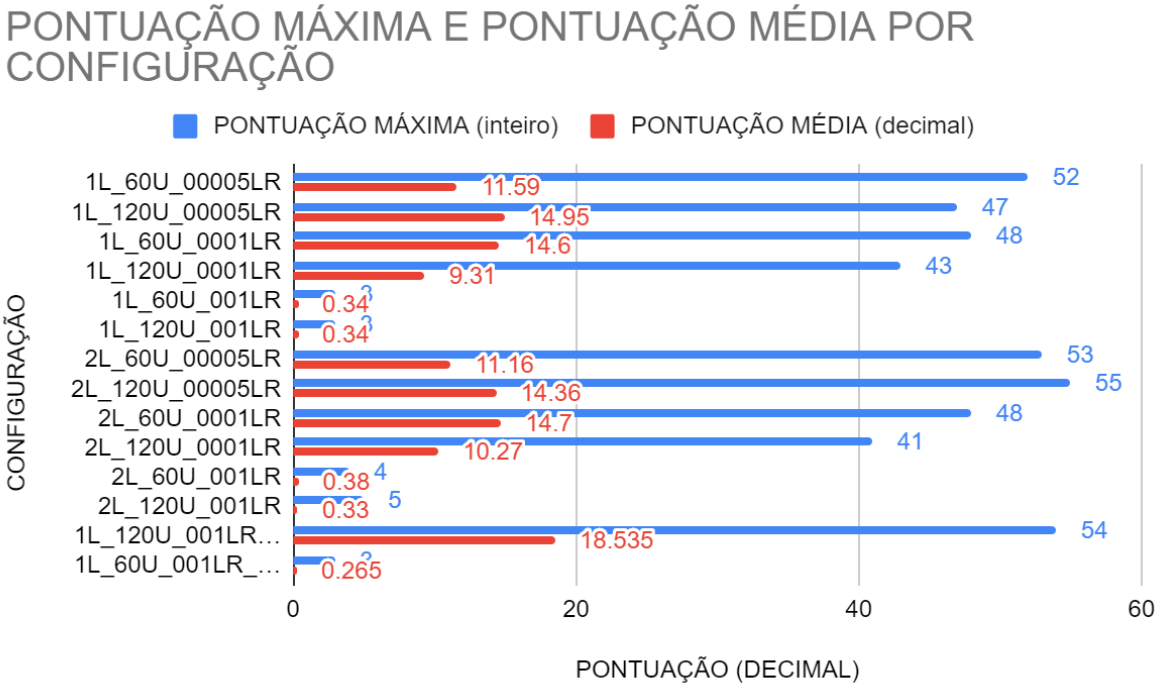

Figura 4-7: *Benchmarking* (pontuação) em 200 Episódios (Snake)

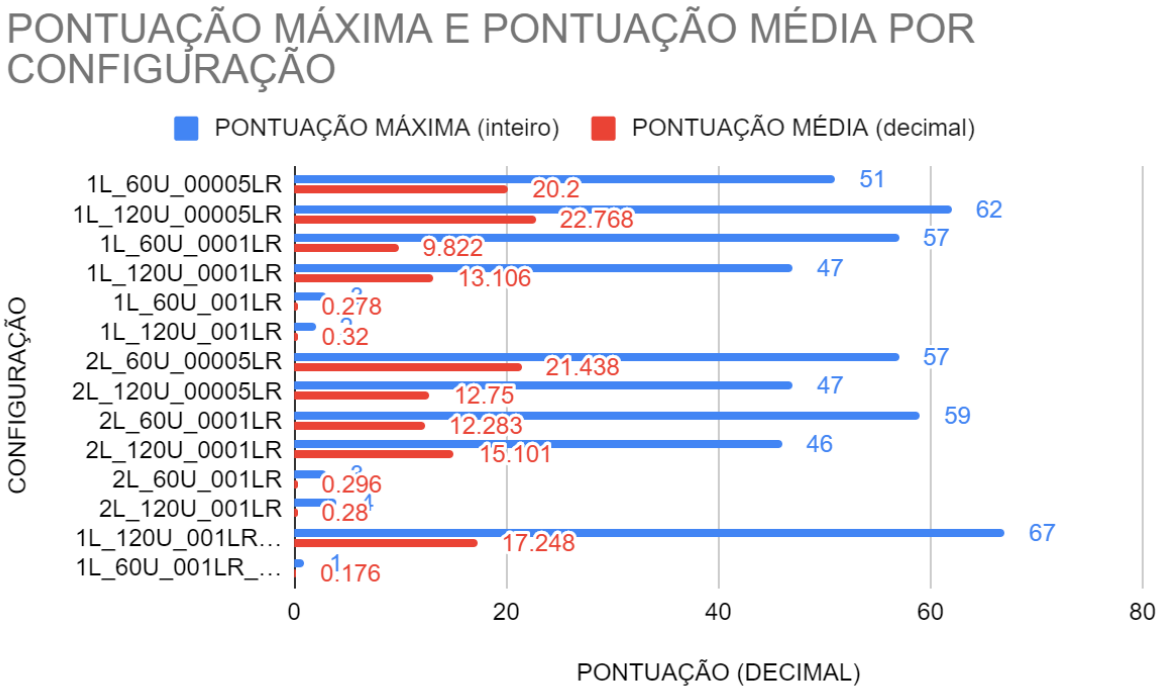

Figura 4-8: *Benchmarking* (pontuação) em 500 Episódios (Snake)

Ao observar o *benchmarking* de performance e impacto dos hiperparâmetros (Figura 4-9, Figura 4-10, Figura 4-11, Figura 4-12), os maiores resultados em termos de **tempo de treino** foram obtidos com a menor **taxa de aprendizagem** configurada **(0.05%)**, **1 camada escondida** com **120 unidades** e num cenário de **500 episódios**, enquanto que os menores foram obtidos com a maior **taxa de aprendizagem** configurada **(1%)**, **2 camadas escondidas** com **120 unidades** num cenário de **50 episódios**.

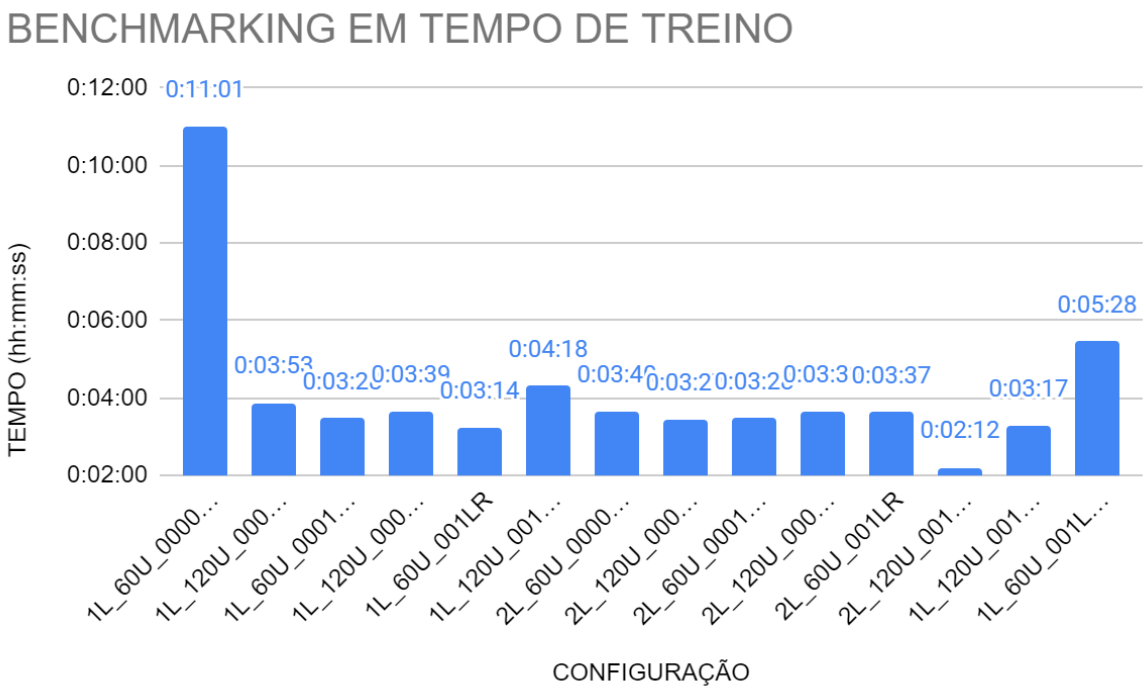

Figura 4-9: *Benchmarking* (tempo) em 50 Episódios (Snake)

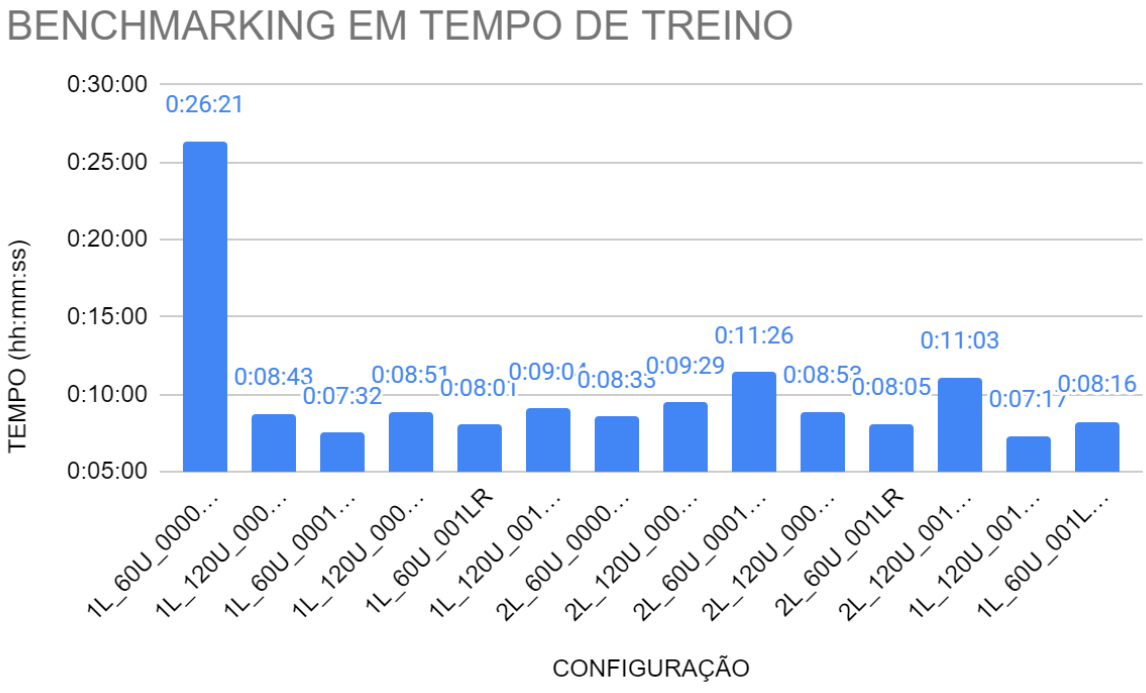

Figura 4-10: *Benchmarking* (tempo) em 100 Episódios (Snake)

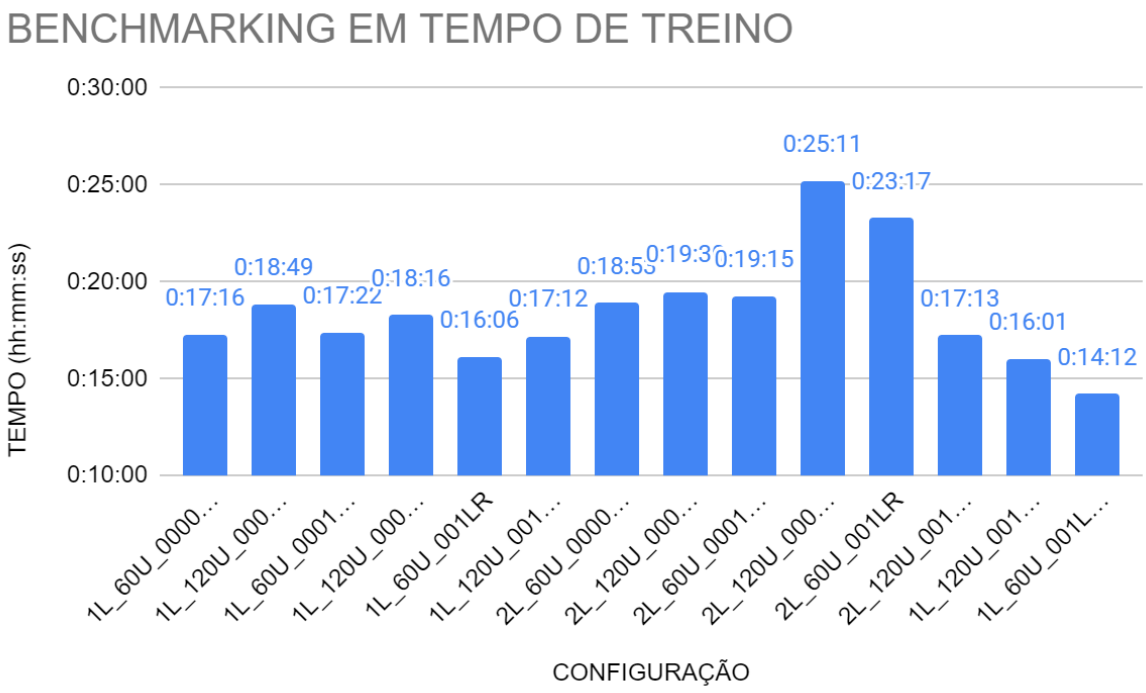

Figura 4-11: *Benchmarking* (tempo) em 200 Episódios (Snake)

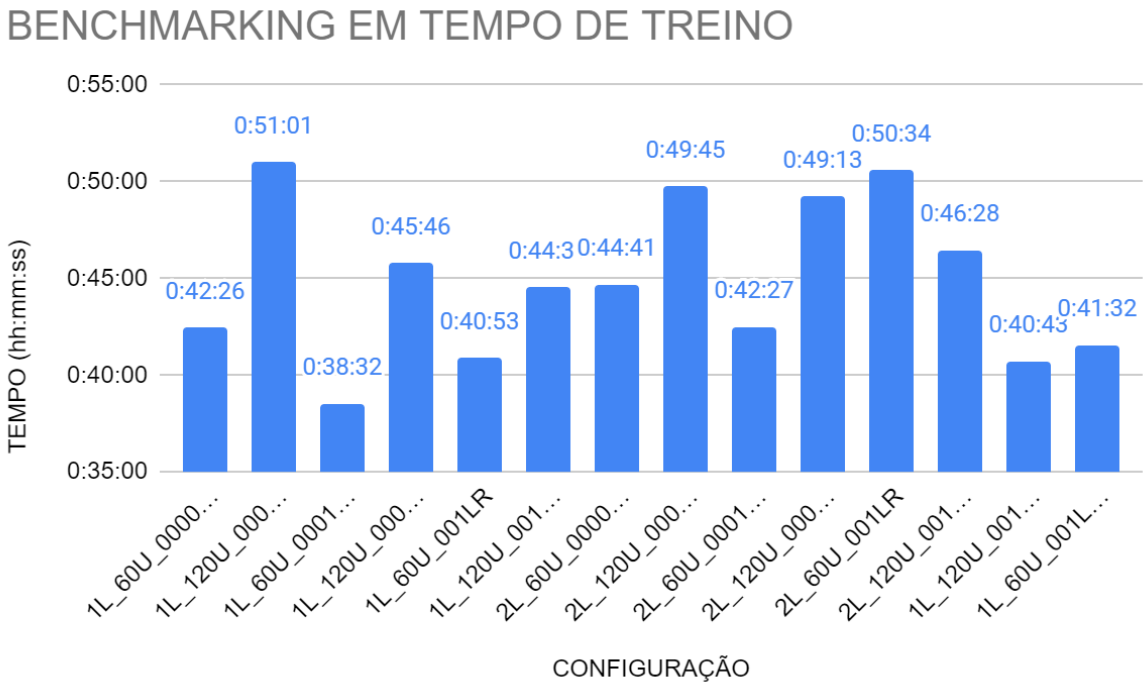

Figura 4-12: *Benchmarking* (tempo) em 500 Episódios (Snake)

Assim, através dos resultados demonstrados, é possível perceber que o hiperparâmetro com mais impacto na performance é a **taxa de aprendizagem**, com resultados tendencialmente maiores consoante menor é definida esta taxa. Para taxas demasiado altas (1%), a ineficiência da técnica é visível nos resultados.

O **número de episódios** afeta claramente o tempo de treino necessário para cada modelo, como já seria de esperar, mas demonstra também que, em praticamente todos os cenários, existe uma tendência de o algoritmo conseguir melhores pontuações. que representam o atingimento de marcos importantes na performance, e também altas médias de pontuação quando o número de episódios é maior, o que indica uma performance consistente e progressivamente melhor à medida que o modelo vai sendo treinado.

O **número de unidades** aparenta ter um impacto ligeiro no tempo de treino, onde este cresce conforme o **número de unidades** testado é maior. No entanto, com o acréscimo de tempo de treino verifica-se também uma média superior de pontos.

O **número de camadas** testadas (1 e 2) não parece ter um impacto relevante na performance do algoritmo, tendo ligeiros desvios em termos de pontuações máxima e pontuações média, quando testado em conjunto com as restantes configurações descritas acima.

Por fim, denotar ainda que existem alguns resultados com comportamentos inconsistentes, como são o caso de um cenário de **100 episódios** demorar mais a treinar do que um cenário com **200 episódios**, exatamente com as mesmas configurações. Este caso pode dever-se a uma inconsistência momentânea no processamento da máquina utilizada ou simplesmente ao facto de o tempo de cada iteração tenha sido maior devido ao algoritmo explorar opções em vez de atingir o objetivo ou algum obstáculo.

Relativamente à aplicação do conceito de curiosidade, foram testadas as **taxas de exploração** a **40%** e a **70%** , para dois cenários: o cenário onde foi obtida uma melhor performance (tanto de pontuação máxima como média) e uma pior performance, por forma a ser possível observar o impacto que incentivar a curiosidade possa ter na performance desta técnica tanto para superar os melhores resultados já obtidos, tal como para validar a eficiência de aumentar os piores resultados obtidos na amostra.

Assim, nos resultados analisados, é possível verificar que, comparativamente com o melhor cenário obtido com uma **taxa de exploração de 40%**, ao aumentar para uma **taxa de exploração de 70%** verifica-se uma melhoria dos resultados obtidos quando o número de episódios é maior (200 e 500) mas não para números de episódios pequenos (50 e 100). Em contraste, no pior cenário não existe qualquer melhoria relevante ao aumentar a taxa de exploração.

As tabelas abaixo descrevem em detalhe todos os resultados obtidos neste contexto e devem ser usadas como suporte à análise feita neste documento.

Tabela 4-5: Resultados Snake em 50 Episódios

ID / CONFIGURAÇÃO	TEMPO (hh:mm:ss)	PONTUAÇÃO MÁXIMA (inteiro)	PONTUAÇÃO MÉDIA (decimal)
1L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:11:01	15	2.06
1L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:03:53	11	2.84
1L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:03:28	4	1.02
1L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:03:39	10	3.34
1L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:03:14	2	0.86
1L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:04:18	2	0.56
2L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:03:40	4	1.24
2L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:03:27	6	1.82
2L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:03:28	6	1.05
2L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de	0:03:39	11	2.94

exploração 0.4 (40%), 2 camadas escondidas com 120 unidades			
2L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:03:37	1	0.77
2L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:02:12	2	0.49
1L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.7 (70%), 1 camada escondida com 60 unidades	0:03:17	3	0.98
2L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.7 (70%), 2 camadas escondidas com 120 unidades	0:05:28	2	0.86

Tabela 4-6: Resultados Snake em 100 Episódios

ID / CONFIGURAÇÃO	**TEMPO (hh:mm:ss)**	**PONTUAÇÃO MÁXIMA (inteiro)**	**PONTUAÇÃO MÉDIA (decimal)**
1L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:26:21	41	10.74
1L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:08:43	47	9.27
1L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:07:32	26	3.74
1L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:08:51	21	4.72
1L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:08:01	2	0.38
1L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:09:04	3	0.55

2L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:08:33	11	2.33
2L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:09:29	37	9.7
2L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:11:26	30	3.79
2L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:08:53	27	5.11
2L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:08:05	3	0.42
2L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:11:03	5	0.65
1L_120U_001LR_07RR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.7 (70%), 1 camada escondida com 120 unidades	0:07:17	26	2.83
1L_60U_001LR_07RR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.7 (70%), 1 camada escondida com 60 unidades	0:08:16	2	0.38

Tabela 4-7: Resultados Snake em 200 Episódios

ID / CONFIGURAÇÃO	**TEMPO (hh:mm:ss)**	**PONTUAÇÃO MÁXIMA (inteiro)**	**PONTUAÇÃO MÉDIA (decimal)**
1L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:17:16	52	11.59
1L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:18:49	47	14.95
1L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de	0:17:22	48	14.6

exploração 0.4 (40%), 1 camada escondida com 60 unidades			
1L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:18:16	43	9.31
1L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:16:06	3	0.34
1L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:17:12	3	0.34
2L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:18:53	53	11.16
2L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:19:30	55	14.36
2L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:19:15	48	14.7
2L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:25:11	41	10.27
2L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:23:17	4	0.38
2L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:17:13	5	0.33
2L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.7 (70%), 2 camadas escondidas com 120 unidades	0:16:01	54	18.535
1L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.7 (70%), 1 camada escondida com 60 unidades	0:14:12	3	0.265

Tabela 4-8: Resultados Snake em 500 Episódios

ID / CONFIGURAÇÃO	TEMPO (hh:mm:ss)	PONTUAÇÃO MÁXIMA (inteiro)	PONTUAÇÃO MÉDIA (decimal)
1L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:42:26	51	20.2
1L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:51:01	62	22.768
1L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:38:32	57	9.822
1L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:45:46	47	13.106
1L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:40:53	3	0.278
1L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:44:31	2	0.32
2L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:44:41	57	21.438
2L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:49:45	47	12.75
2L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:42:27	59	12.283
2L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:49:13	46	15.101

2L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:50:34	3	0.296
2L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:46:28	4	0.28
1L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.7 (70%), 1 camada escondida com 120 unidades	0:40:43	67	17.248
1L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.7 (70%), 1 camada escondida com 120 unidades	0:41:32	1	0.176

4.4.2 Tetris

Ao observar o *benchmarking* de performance e impacto dos hiperparâmetros (Figura 4-13, Figura 4-14, Figura 4-15, Figura 4-16), o melhor resultado em termos de **pontuação média** (21342) foi obtido com a maior **taxa de aprendizagem** configurada **(1%)**, **2 camadas escondidas** com **120 unidades** e num cenário de **500 episódios**, enquanto que o melhor resultado em termos de **pontuação média** (i.e., 449.89) foi obtido com a maior **taxa de aprendizagem (1%), 2 camadas escondidas** com **60 unidades**. Os piores resultados em termos de pontuação máxima e pontuação média (20 e 13.33) foram obtidos com a menor **taxa de aprendizagem** configurada **(0.05%)**, **2 camadas escondidas** e **60 unidades** num cenário de **100 episódios**.

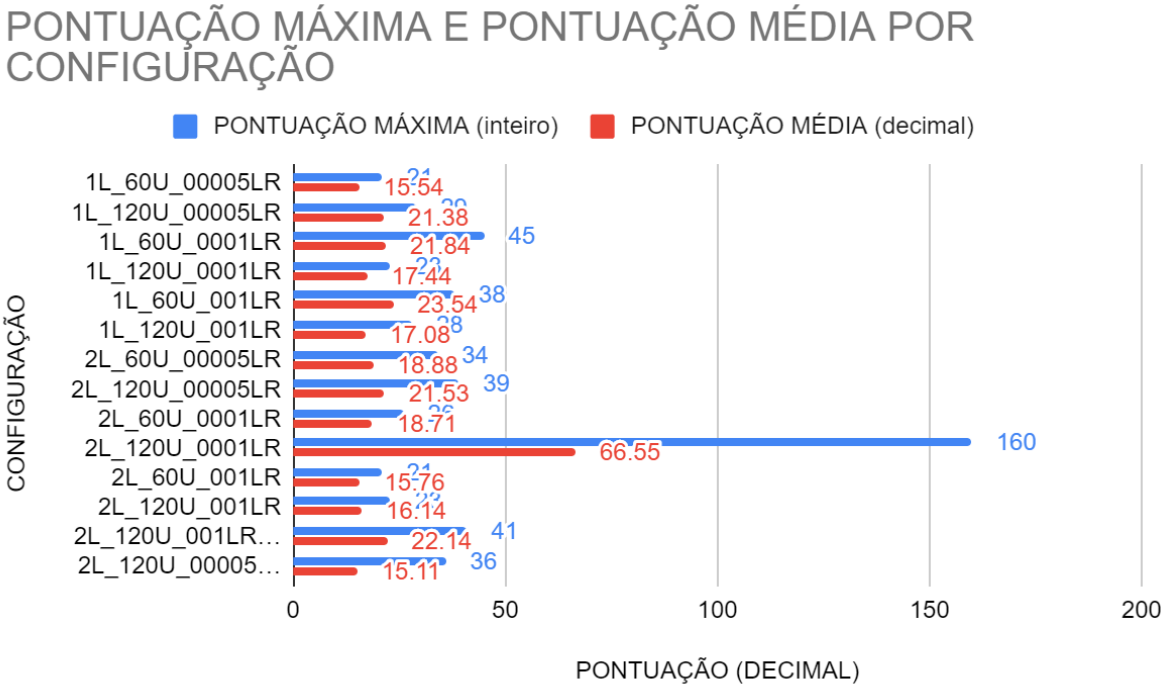

Figura 4-13: *Benchmarking* (pontuação) em 50 Episódios (Tetris)

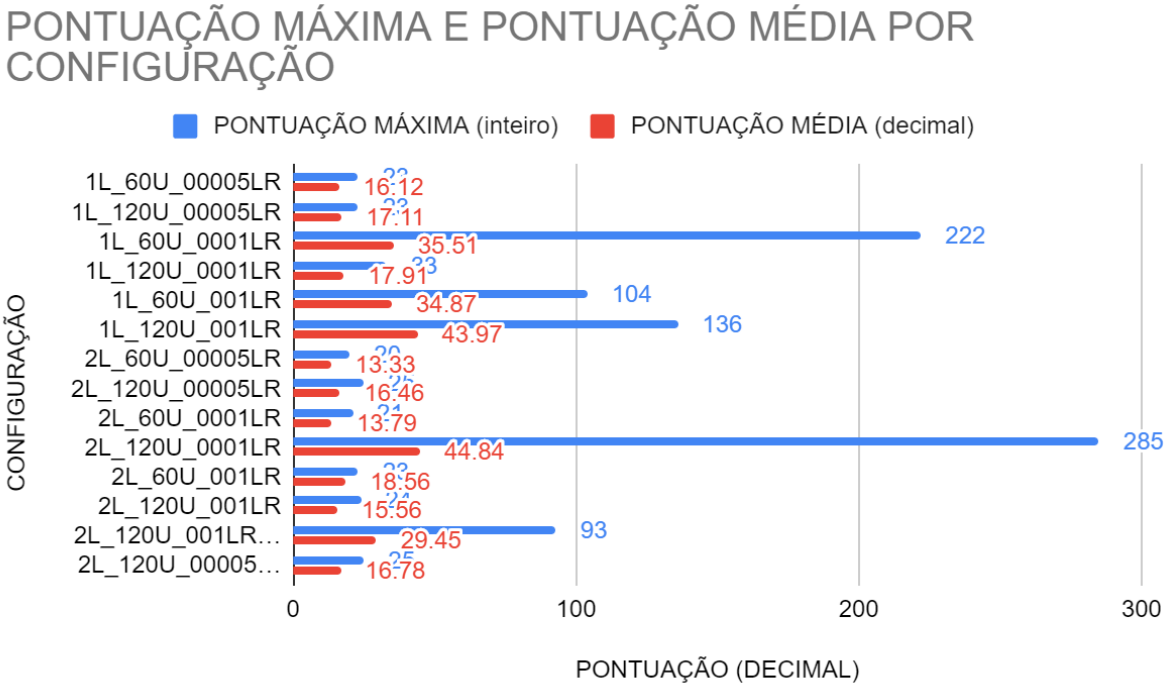

Figura 4-14: *Benchmarking* (pontuação) em 100 Episódios (Tetris)

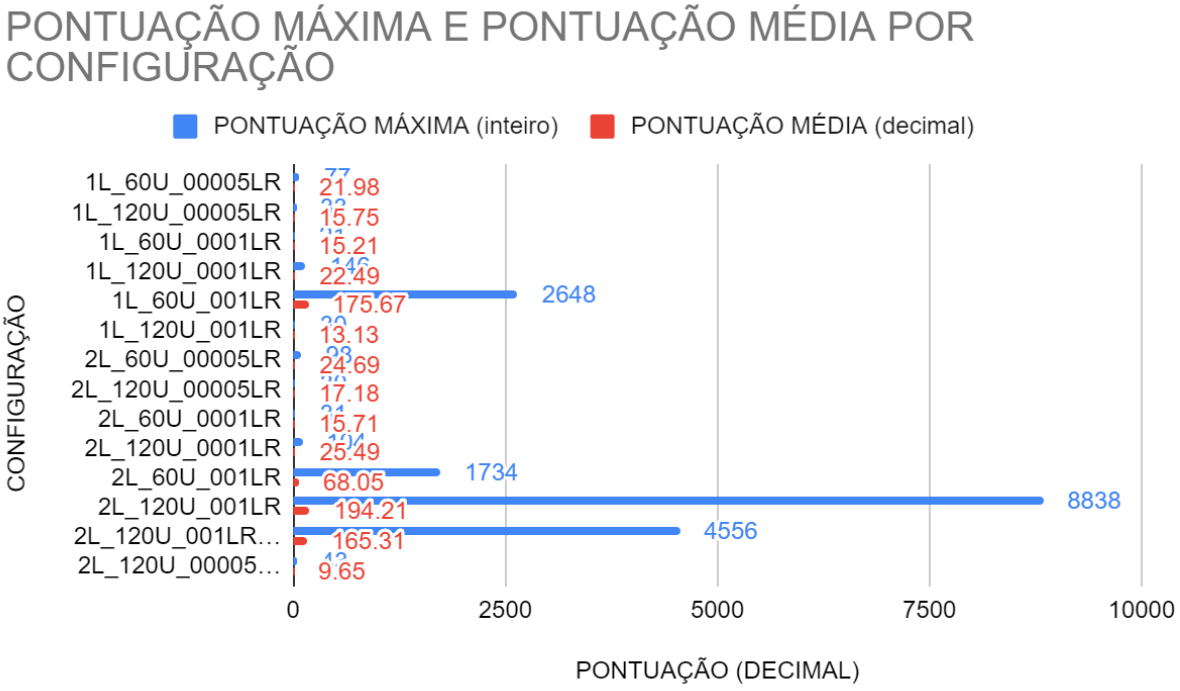

Figura 4-15: *Benchmarking* (pontuação) em 200 Episódios (Tetris)

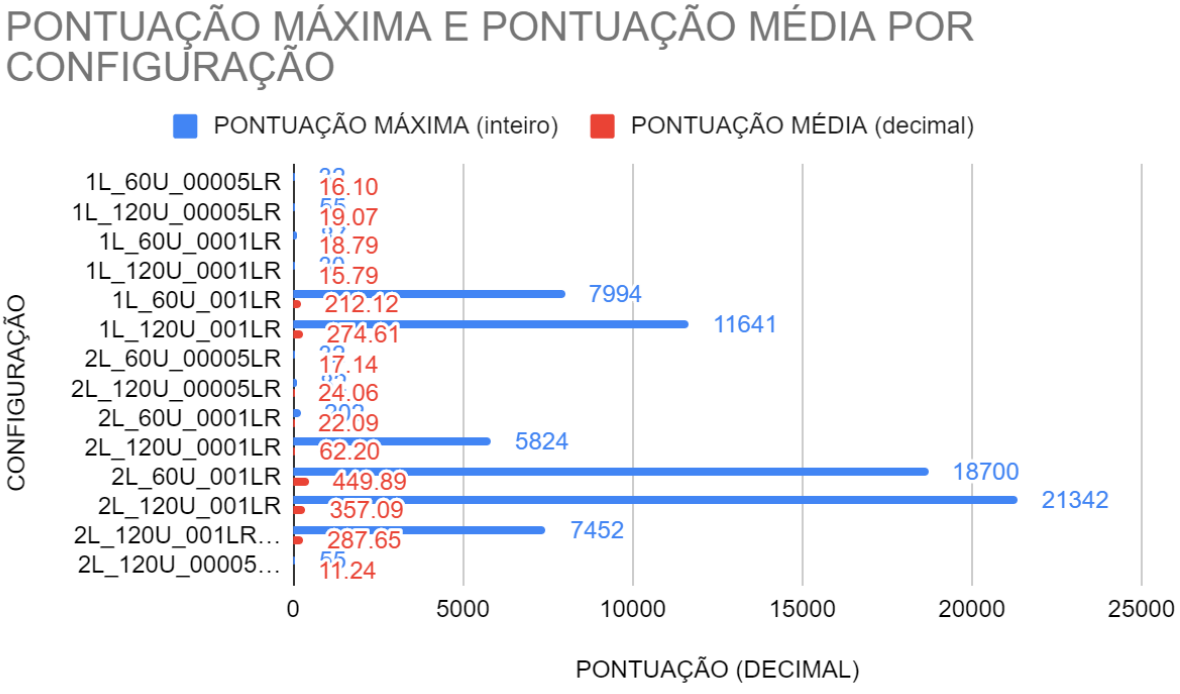

Figura 4-16: *Benchmarking* (pontuação) em 500 Episódios (Tetris)

Ao observar o *benchmarking* de performance e impacto dos hiperparâmetros (Figura 4-17, Figura 4-18, Figura 4-19, Figura 4-20), os maiores resultados em termos de **tempo de treino** foram obtidos com a maior **taxa de aprendizagem** configurada **(1%)**, **2 camadas escondidas** com **60 unidades** e num cenário de **500 episódios**, enquanto que os menores foram obtidos com a maior **taxa de aprendizagem** configurada **(1%)**, **2 camadas escondidas** e **120 unidades** num cenário de **50 episódios**.

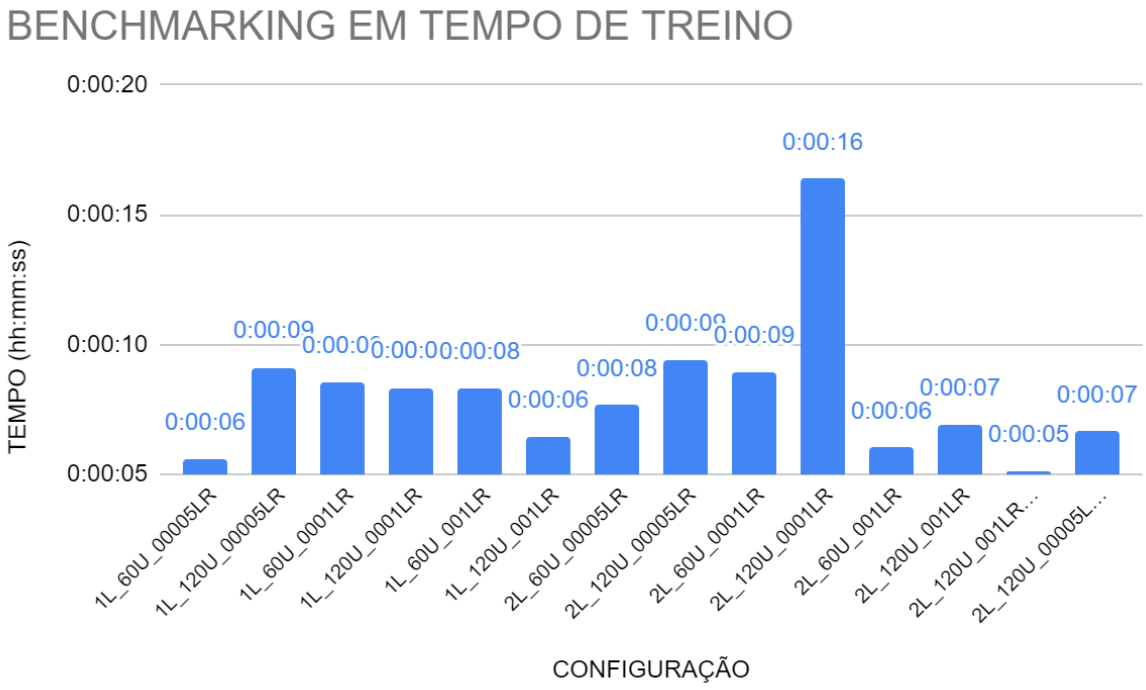

Figura 4-17: *Benchmarking* (tempo) em 50 Episódios (Tetris)

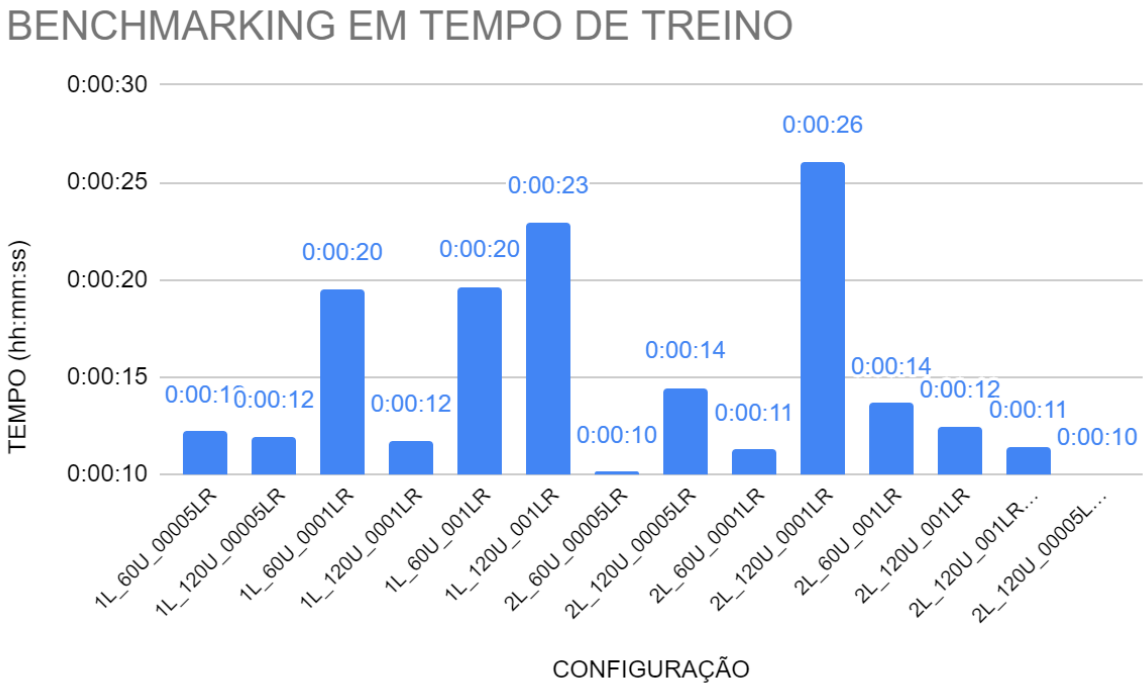

Figura 4-18: *Benchmarking* (tempo) em 100 Episódios (Tetris)

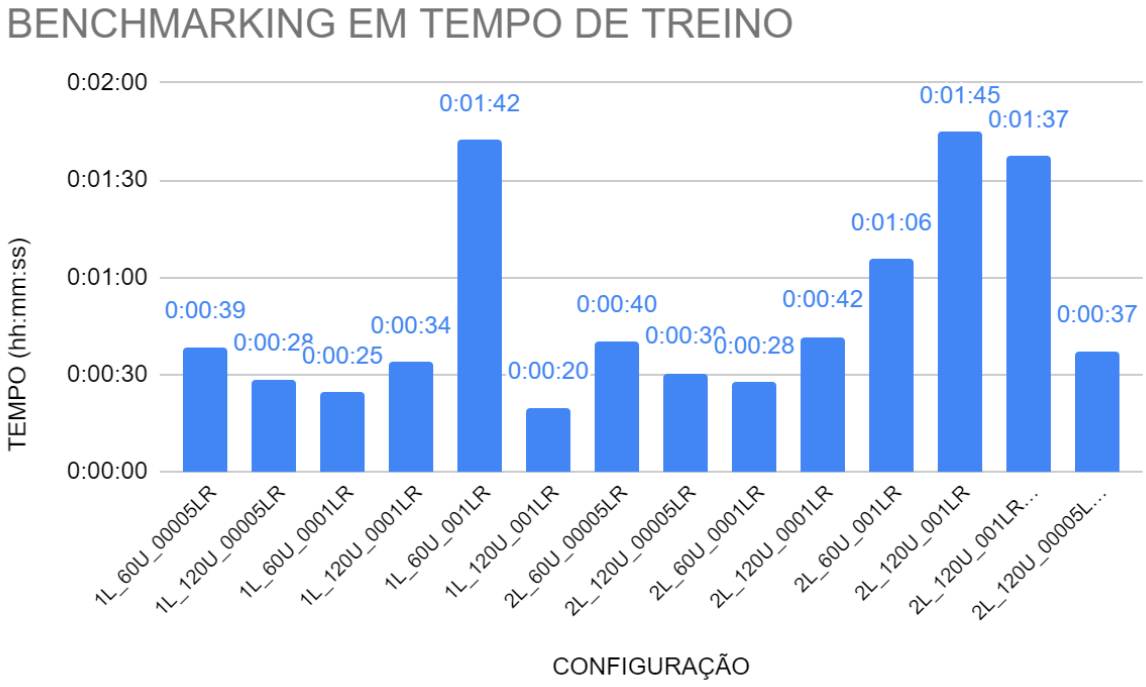

Figura 4-19: *Benchmarking* (tempo) em 200 Episódios (Tetris)

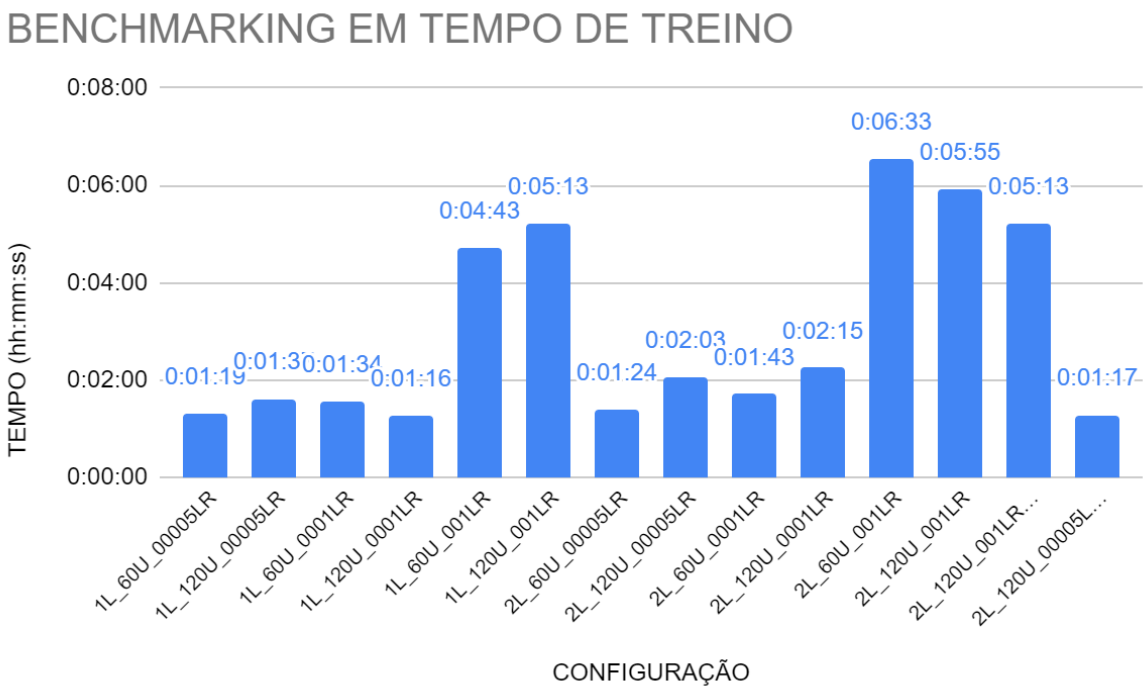

Figura 4-20: *Benchmarking* (tempo) em 500 Episódios (Tetris)

Através dos resultados demonstrados acima, é possível perceber que o hiperparâmetro com mais impacto na performance é a **taxa de aprendizagem**, com resultados tendencialmente maiores consoante maior é definida esta taxa. Para taxas demasiado baixas (1%), a ineficiência da técnica é visível nos resultados.

O **número de episódios** afeta claramente o tempo de treino necessário para cada modelo, como já seria de esperar, mas demonstra também que, em praticamente todos os cenários, existe uma tendência de o algoritmo conseguir melhores pontuações, que representam o atingimento de marcos importantes na performance, e também altas médias de pontuação quando o número de episódios é maior, o que indica uma performance consistente e progressivamente melhor à medida que o modelo vai sendo treinado.

O **número de unidades** aparenta ter um impacto ligeiro no tempo de treino, onde este cresce conforme o **número de unidades** testado é maior. No entanto, com o acréscimo de tempo de treino verifica-se também uma média superior de pontos.

O **número de camadas** testadas (1 e 2) não parece ter um impacto relevante na performance do algoritmo, tendo ligeiros desvios em termos de pontuações máxima e pontuações média, quando testado em conjunto com as restantes configurações descritas acima.

Relativamente à aplicação do conceito de curiosidade, foram testadas as **taxas de exploração** a **40%** e a **70%** , para dois cenários: o cenário onde foi obtida uma melhor performance (tanto de pontuação máxima como média) e uma pior performance, por forma a ser possível observar o impacto que incentivar a curiosidade possa ter na performance desta técnica tanto para superar os melhores resultados já obtidos, tal como para validar a eficiência de aumentar os piores resultados obtidos na amostra.

Nos resultados acima, é possível verificar que, comparativamente com o melhor cenário obtido com uma **taxa de exploração de 40%**, ao aumentar para uma **taxa de exploração de 70%** verifica-se uma perda de performance nos resultados obtidos quando o número de episódios é maior (200 e 500) mas não para números de episódios pequenos (50 e 100). Em contraste, no pior cenário, não existe qualquer melhoria relevante ao aumentar a taxa de exploração, com a exceção da simulação com **200 episódios**, onde uma **taxa de exploração de 70%** obteve resultados consideravelmente superiores.

As tabelas abaixo (Tabela 4-9, Tabela 4-10, Tabela 4-11, Tabela 4-12) descrevem em detalhe todos os resultados obtidos neste contexto e devem ser usadas como suporte à análise feita neste documento.

Tabela 4-9: Resultados Tetris em 50 Episódios

ID / CONFIGURAÇÃO	TEMPO (hh:mm:ss)	PONTUAÇÃO MÁXIMA (inteiro)	PONTUAÇÃO MÉDIA (decimal)
1L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:00:06	21	15.54
1L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:00:09	29	21.38
1L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:00:09	45	21.84
1L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:00:08	23	17.44
1L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:00:08	38	23.54

1L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:00:06	28	17.08
2L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:00:08	34	18.88
2L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:00:09	39	21.53
2L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:00:09	26	18.71
2L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:00:16	160	66.55
2L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:00:06	21	15.76
2L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:00:07	23	16.14
2L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.7 (70%), 2 camadas escondidas com 120 unidades	0:00:05	41	22.14
1L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.7 (70%), 1 camada escondida com 60 unidades	0:00:07	36	15.11

Tabela 4-10: Resultados Tetris em 100 Episódios

ID / CONFIGURAÇÃO	**TEMPO (hh:mm:ss)**	**PONTUAÇÃO MÁXIMA (inteiro)**	**PONTUAÇÃO MÉDIA (decimal)**
1L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:00:12	23	16.12

1L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:00:12	23	17.11
1L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:00:20	222	35.51
1L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:00:12	33	17.91
1L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:00:20	104	34.87
1L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:00:23	136	43.97
2L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:00:10	20	13.33
2L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:00:14	25	16.46
2L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:00:11	21	13.79
2L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:00:26	285	44.84
2L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:00:14	23	18.56
2L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:00:12	24	15.56
2L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de	0:00:11	93	29.45

exploração 0.7 (70%), 2 camadas escondidas com 120 unidades			
2L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.7 (70%), 2 camadas escondidas com 60 unidades	0:00:10	25	16.78

Tabela 4-11: Resultados Tetris em 200 Episódios

ID / CONFIGURAÇÃO	**TEMPO (hh:mm:ss)**	**PONTUAÇÃO MÁXIMA (inteiro)**	**PONTUAÇÃO MÉDIA (decimal)**
1L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:00:39	77	21.98
1L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:00:28	33	15.75
1L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:00:25	21	15.21
1L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:00:34	146	22.49
1L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:01:42	2648	175.67
1L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:00:20	30	13.43
2L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:00:40	98	24.69
2L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:00:30	30	17.18
2L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de	0:00:28	31	15.71

exploração 0.4 (40%), 2 camadas escondidas com 60 unidades			
2L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:00:42	104	25.49
2L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:01:06	1734	68.05
2L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:01:45	8838	194.21
2L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.7 (70%), 2 camadas escondidas com 120 unidades	0:01:37	4556	165.31
1L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.7 (70%), 1 camada escondida com 60 unidades	0:00:37	433	19.65

Tabela 4-12: Resultados Tetris em 500 Episódios

ID / CONFIGURAÇÃO	**TEMPO (hh:mm:ss)**	**PONTUAÇÃO MÁXIMA (inteiro)**	**PONTUAÇÃO MÉDIA (decimal)**
1L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:01:19	32	16.10
1L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:01:37	55	19.07
1L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 60 unidades	0:01:34	87	18.79
1L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:01:16	30	15.79
1L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4	0:04:43	7994	212.12

(40%), 1 camada escondida com 60 unidades			
1L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 1 camada escondida com 120 unidades	0:05:13	11641	274.61
2L_60U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:01:24	32	17.14
2L_120U_00005LR / taxa de aprendizagem 0.0005 (0.05%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:02:03	82	24.06
2L_60U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:01:43	202	22.09
2L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:02:15	5824	62.20
2L_60U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 60 unidades	0:06:33	18700	449.89
2L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.4 (40%), 2 camadas escondidas com 120 unidades	0:05:55	21342	357.09
2L_120U_001LR / taxa de aprendizagem 0.01 (1%), taxa de exploração 0.7 (70%), 2 camadas escondidas com 120 unidades	0:05:13	7452	287.65
1L_120U_0001LR / taxa de aprendizagem 0.001 (0.1%), taxa de exploração 0.7 (70%), 1 camada escondida com 120 unidades	0:01:17	55	14.24

5. Conclusões

5.1.1 Desenvolvimento Pessoal e Profissional

A realização deste trabalho potenciou o desenvolvimento pessoal e profissional do autor, já que implicou um estudo aprofundado sobre a temática de ML e consequente desenvolvimento de capacidades académicas para a investigação, estudo e aplicação dos conceitos inerentes ao tópico discutido neste documento.

5.1.2 Cumprimento de Objetivos

No âmbito definido de objetivos para este projeto consta a implementação de um algoritmo de aprendizagem para ambientes *gaming*, a medição de sensibilidade do impacto dos hiperparâmetros na performance deste algoritmo, a exploração do conceito de curiosidade na aplicação deste algoritmo e por fim a identificação de padrões em algoritmos de aprendizagem.
A implementação do algoritmo aplicando a técnica de *Q-learning* a dois ambientes distintos (Snake e Tetris) permitiu que fosse comprovada a sua eficácia na resolução dos obstáculos presentes em cada um dos ambientes, possibilitando assim criar modelos de aprendizagem onde um agente aprende por si, sem supervisão humana, a concluir os jogos com sucesso.
A exploração de várias combinações de configurações, adaptando os valores dos diferentes hiperparâmetros abordados na execução do trabalho experimental, permitiu retirar conclusões junto de cada ambiente e considerando as métricas definidas que, conjuntamente, demonstram os resultados de *benchmarking* de performance deste algoritmo em ambientes *gaming*. Adicionalmente, a aplicação de um conceito de curiosidade, incentivando a exploração em prol da aplicação de conhecimento prévio, permitiu a comparação entre estes dois cenários a nível da performance do mesmo algoritmo.

Uma análise dos resultados obtidos em todos os testes executados no contexto da elaboração deste projeto permitiu que fossem retiradas ilações sobre o desenho de padrões em algoritmos de aprendizagem, nomeadamente sobre as vantagens e desvantagens numa abordagem de RL com recurso a *Q-learning* e a sua aplicação a ambientes de *gaming* e ainda na identificação de potenciais abordagens numa perspetiva de evolução e trabalho futuro a realizar nesta área.

5.1.3 Considerações sobre o Trabalho Desenvolvido

Para a elaboração deste projeto, foi necessário um estudo extensivo sobre conceitos imediatamente adjacentes à temática principal do projeto, o ML. Este estudo acabou por enriquecer consideravelmente o conhecimento na medida em que aborda um largo número de conceitos teóricos e que para explorar esses conceitos, requereu também a exploração de várias abordagens técnicas por forma a inteirar da área no que toca às suas principais capacidades e limitações.

No conjunto de trabalho produzido na elaboração deste projeto, constam algumas áreas relevantes onde foi necessário aprimorar o nível de conhecimento e que permitiram obter melhorias visíveis a nível pessoal e profissional. Numa visão global, estas foram as áreas abordadas: Python, *machine learning*, *reinforcement learning*, *Q-learning*, funcionamento e implementação de arquiteturas com redes neurais.

Em termos de dificuldades, um obstáculo encontrado foi a enorme quantidade de variáveis que a configuração de redes neurais que trabalham com algoritmos de ML suportam e o tempo necessário para conduzir testes e analisar os resultados obtidos através desses testes. Neste projeto são explorados quatro dessas variáveis (os hiperparâmetros), mas existem outras variáveis que podem ser configuradas dentro da estrutura de uma rede neural, e que têm impacto na forma como processam e trabalham a informação de aprendizagem, tornando exponencialmente maior o tempo necessário à conclusão de testes associados a este tipo de exercícios.

Outra dificuldade encontrada foi a definição de recompensas. Os algoritmos de *reinforcement learning* desmontram bom funcionamento quando as recompensas são claras e lineares, ou seja, quando há um objetivo claro no jogo (e.g., obter o maior número de pontuação possível) mas têm de ser incentivados a explorar as opções de jogo (e.g., recompensar por movimentos que não causem o fim do jogo) por forma a serem eficientes na resolução de jogos. No entanto, esta abordagem incorre no risco de fazer com que os algoritmos vagueiem infinitamente no ambiente, devido à recompensa mínima (mas constante) de movimentação e não dando portanto tanta relevância à conclusão do jogo em si.

5.1.4 Potencial de Evolução e Trabalho Futuro

Numa perspetiva de potencial de exploração e trabalho futuro sobre o a realização deste projeto, existe espaço para explorar abordagens adicionais na procura da melhoria dos resultados dos algoritmos aqui observados.

Uma primeira abordagem pode ser executar testes com maior número de episódios e com diferentes configurações de hiperparâmetros das abordadas neste documento, possibilitando assim ter uma maior diversidade na amostra de dados e resultantes desta, uma visão mais global sobre o impacto dos parâmetros sobre a performance dos algoritmos.
Outra abordagem, dentro dos hiperparâmetros revistos, pode ser a aplicação de um conceito variação nos parâmetros que demonstraram ter mais impacto sobre a performance (e.g., taxa de aprendizagem). Dentro de uma simulação onde existem múltiplas iterações, pode ser testado um cenário onde este valor varie (crescendo ou diminuindo) para medir o impacto e tentar estabilizar os resultados da aprendizagem.
Existem ainda outras abordagens de RL, não exploradas no decorrer deste projeto, que podem ser interessantes a nível de resultados de performance quando aplicadas nos mesmos ambientes observados neste documento. Exemplos de algumas destas abordagens são *Inverse Reinforcement Learning* (ou *Imitation Reinforcement Learning*), essencialmente um conceito onde o algoritmo aprende através da observação do comportamento de uma entidade que já tenha conhecimento sobre como resolver um dado problema, ou *Multi-Agent Reinforcement Learning*, onde o algoritmo é desenhado para interações com ambientes onde mais do que 1 agente atuam em simultâneo, promovendo a competitividade entre agentes o que poderá acelerar tempos de treino necessários a atingir determinadas metas de performance.

Por fim, outra abordagem interessante a explorar será a da observação da técnica de *Q-learning* abordada neste relatório comparativamente a outra técnica de *machine learning* que não esteja relacionada com *reinforcement learning*. Um exemplo será comparar um algoritmo de *Q-learning* com um outro de *Supervised Learning*, onde este último algoritmo é alimentado com informação relevante à resolução de um dado problema.

REFERÊNCIAS

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Gulli, A., & Pal, S. (2017). *Deep Learning with Keras*. Packt Publishing Ltd.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016). Tensorflow: A system for large-scale machine learning. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16) (pp. 265-283).

Hu, W., & Hu, J. (2019). Q Learning with Quantum Neural Networks. Natural Science, 11(01), 31.

Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.

Barron, E. N., & Ishii, H. (1989). The Bellman equation for minimizing the maximum cost. Nonlinear Analysis: Theory, Methods & Applications, 13(9), 1067-1090.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural networks, 61, 85-117.

Hinton, G., Srivastava, N., & Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Cited on, 14, 8.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., & Abbeel, P. (2016). Rl $^ 2$: Fast reinforcement learning via slow reinforcement learning. arXiv preprint arXiv:1611.02779.

Nguyen, D. H., & Widrow, B. (1990). Neural networks for self-learning control systems. IEEE Control systems magazine, 10(3), 18-23.

Young, S. R., Rose, D. C., Karnowski, T. P., Lim, S. H., & Patton, R. M. (2015, Novembro). Optimizing deep learning hyper-parameters through an evolutionary algorithm. In Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments (p. 4). ACM.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., ... & Asari, V. K. (2018). The history began from alexnet: A comprehensive survey on deep learning approaches. arXiv preprint arXiv:1803.01164.

El Naqa, I., & Murphy, M. J. (2015). What is machine learning?. In Machine Learning in Radiation Oncology (pp. 3-11). Springer, Cham.

Bengio, Y. (2000). Gradient-based optimization of hyperparameters. Neural computation, 12(8), 1889-1900.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In Advances in neural information processing systems (pp. 2951-2959).

Joshi, S. P. (2002). U.S. Patent No. 6,485,367. Washington, DC: U.S. Patent and Trademark Office.

Fried, L. (2019). Making machine learning arduino compatible: A gaming handheld that runs neural networks-[Resources_Hands On]. IEEE Spectrum, 56(8), 14-15.

Li, L. (2018). Machine Learning Prediction System Based on Tensor-Flow Deep Neural Network and its Application to Advertising in Mobile Gaming.

Zhai, Y., Danandeh, N., Tan, Z., Gao, S., Paesani, F., & Götz, A. W. (2018). Parallel implementation of machine learning-based many-body potentials on CPU and GPU.

Grimmer, J. (2015). We are all social scientists now: How big data, machine learning, and causal inference work together. PS: Political Science & Politics, 48(1), 80-83.

Frey, M., & Larch, M. (2017). Statistical Learning in the Age of “Big Data” and Machine Learning.

Ramchoun, H., Idrissi, M. A. J., Ghanou, Y., & Ettaouil, M. (2016). Multilayer Perceptron: Architecture Optimization and Training. IJIMAI, 4(1), 26-30.

Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017, Agosto). Convolutional sequence to sequence learning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 1243-1252). JMLR. org.

Sak, H., Senior, A., & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In Fifteenth annual conference of the international speech communication association.

Panchal, G., Ganatra, A., Kosta, Y. P., & Panchal, D. (2011). Behaviour analysis of multilayer perceptrons with multiple hidden neurons and hidden layers. International Journal of Computer Theory and Engineering, 3(2), 332-337.

Lee, S., & Choeh, J. Y. (2014). Predicting the helpfulness of online reviews using multilayer perceptron neural networks. Expert Systems with Applications, 41(6), 3041-3046.

Isa, N. A. M., & Mamat, W. M. F. W. (2011). Clustered-hybrid multilayer perceptron network for pattern recognition application. Applied Soft Computing, 11(1), 1457-1466.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, Junho). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529.

Patel, P. G., Carver, N., & Rahimi, S. (2011, Setembro). Tuning computer gaming agents using q-learning. In 2011 Federated Conference on Computer Science and Information Systems (FedCSIS) (pp. 581-588). IEEE.

Tastan, B., & Sukthankar, G. R. (2011, Outubro). Learning policies for first person shooter games using inverse reinforcement learning. In Seventh Artificial Intelligence and Interactive Digital Entertainment Conference.

Domingos, P. M. (2012). A few useful things to know about machine learning. Commun. acm, 55(10), 78-87.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., ... & Silver, D. (2018, Abril). Rainbow: Combining improvements in deep reinforcement learning. In Thirty-Second AAAI Conference on Artificial Intelligence.

Tassa, Y., Erez, T., & Todorov, E. (2012, Outubro). Synthesis and stabilization of complex behaviors through online trajectory optimization. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 4906-4913). IEEE.

Gupta, A., & Merchant, P. S. (2016). Automated lane detection by k-means clustering: a machine learning approach. Electronic Imaging, 2016(14), 1-6.

Deng, L., & Li, X. (2013). Machine learning paradigms for speech recognition: An overview. IEEE Transactions on Audio, Speech, and Language Processing, 21(5), 1060-1089.
Cevikalp, H., & Triggs, B. (2010, Junho). Face recognition based on image sets. In 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (pp. 2567-2573). IEEE.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. Journal of machine learning research, 12(Aug), 2493-2537.

Basta, S., Fassetti, F., Guarascio, M., Manco, G., Giannotti, F., Pedreschi, D., ... & Pisani, S. (2009, Dezembro). High quality true-positive prediction for fiscal fraud detection. In 2009 IEEE International Conference on Data Mining Workshops (pp. 7-12). IEEE.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., ... & Dulac-Arnold, G. (2018, Abril). Deep q-learning from demonstrations. In Thirty-Second AAAI Conference on Artificial Intelligence.

James, S., & Johns, E. (2016). 3d simulation for robot arm control with deep q-learning. arXiv preprint arXiv:1609.03759.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

Raschka, S., & Mirjalili, V. (2017). Python machine learning. Packt Publishing Ltd.

Millman, K. J., & Aivazis, M. (2011). Python for scientists and engineers. Computing in Science & Engineering, 13(2), 9-12.

Eskandar, E., & Williams, Z. (2010). U.S. Patent No. 7,725,192. Washington, DC: U.S. Patent and Trademark Office.

Chao, L., Tao, J., Yang, M., Li, Y., & Wen, Z. (2015, Outubro). Long short term memory recurrent neural network based multimodal dimensional emotion recognition. In Proceedings of the 5th International Workshop on Audio/Visual Emotion Challenge (pp. 65-72). ACM.

Ma, B., Tang, M., & Zhang, J. (2016). Exploration of Reinforcement Learning to SNAKE.

Carr, D. (2005). Applying reinforcement learning to Tetris. Department of Computer Science Rhodes University.

Romdhane, H., & Lamontagne, L. (2008, Maio). Reinforcement of Local Pattern Cases for Playing Tetris. In FLAIRS Conference (pp. 263-268).

Yura Burda (Outubro 2018). Reinforcement Learning with Prediction-Based Rewards. Consultado em Janeiro 3, 2019, de https://blog.openai.com/reinforcement-learning-with-prediction-based-rewards/

Victor Epand (Março 2019). The Benefits Of Artificial Intelligence In Computer Games. Consultado em Janeiro 3, 2019, de https://www.streetdirectory.com/travel_guide/141271/gaming/the_benefits_of_artificial_intelligence_in_computer_games.html

Bernard Marr (2018). Artificial Intelligence: The Clever Ways Video Games Are Used. Consultado em Janeiro 3, 2019, de https://www.forbes.com/sites/bernardmarr/2018/06/13/artificial-intelligence-the-clever-ways-video-games-are-used-to-train-ais/

Serdar Yegulalp (Junho 2019). What is TensorFlow? The *machine learning* library explained. Consultado em Janeiro 3, 2019, de https://www.infoworld.com/article/3278008/tensorflow/what-is-tensorflow-the-machine-learning-library-explained.html

Python Developers (n.d). What is Python? Executive Summary. Consultado em Janeiro 3, 2019, de https://www.python.org/doc/essays/blurb/

OpenAI (2016). OpenAI Gym. Consultado em Janeiro 3, 2019, de https://github.com/openai/gym

Christina Voskoglou (Maio 2017). What is the best programming language for Machine Learning?. Consultado em Janeiro 4, 2019, de https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7

Ashish Rana (Setembro 2018). Reinforcement Learning with OpenAI Gym. Consultado em Janeiro 4, 2019, de https://towardsdatascience.com/reinforcement-learning-with-openai-d445c2c687d2

Ashish Sukhadeve (Novembro 2016). Machine Learning: what it means and why it is the future of growth Consultado em Janeiro 4, 2019, de https://www.datasciencecentral.com/profiles/blogs/machine-learning-what-it-means-and-why-it-is-the-future-of-growth

Kung-Hsiang (Janeiro 2018). Introduction to Various Reinforcement Learning Algorithms. Part I. Consultado em Janeiro 4, 2019, de https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287

"Multilayer Perceptron", (Junho 2018). Multilayer Perceptron. Consultado em Janeiro 4, 2019, de http://deeplearning.net/tutorial/mlp.html

"Convolutional Neural Network", (n.d). Convolutional Neural Network - Stanford Deep Learning. Consultado em Janeiro 4, 2019, de http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

Survro Banerjee (Maio 2018). An Introduction to Recurrent Neural Networks – Explore Artificial. Consultado em Janeiro 4, 2019, de https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912

"Machine Learning - What it is and why it matters" (n.d). Machine Learning: What it is and why it. Consultado em Janeiro 4, 2019, de https://www.sas.com/en_us/insights/analytics/machine-learning.html

Sagar Sharma (Setembro 2017). Epoch vs Batch Size vs Iterations. Consultado em Janeiro 5, 2019, de https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

Google Developers (Outubro 2018). Reducing Loss: Learning Rate | Machine Learning Crash Course. Consultado em Janeiro 5, 2019, de https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate

"Hidden Layer" (n.d). What is a Hidden Layer? - Definition de Techopedia. Consultado em Janeiro 5, 2019, de https://www.techopedia.com/definition/33264/hidden-layer-neural-networks

Disha Gupta (Outubro 2017). Fundamentals of Deep Learning - Activation Functions and their use. Consultado em Janeiro 5, 2019, de https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

Chris Nicholson (n.d). A Beginner's Guide to Deep Reinforcement Learning. Consultado em Janeiro 7, 2019, de https://skymind.ai/wiki/deep-reinforcement-learning

Josh Greaves (n.d). Understanding RL: The Bellman Equations. Consultado em Janeiro 7, 2019, de https://joshgreaves.com/reinforcement-learning/understanding-rl-the-bellman-equations/

“What is Deep Learning?” (n.d). What is Deep Learning?. Consultado em Setembro 7, 2019, de https://www.mathworks.com/discovery/deep-learning.html

Keith Foote (Fevereiro 2017). A Brief History of Deep Learning. Consultado em Setembro 7, 2019, de https://www.dataversity.net/brief-history-deep-learning/

Ankit Choudhary (Abril 2018). A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python. Consultado em Outubro 18, 2019, de https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

Ahmed Gad (Junho 2018). “How Many Hidden Layers/Neurons to Use in Artificial Neural Networks?”. Consultado em Novembro 11, 2019, de https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e

James Vincent (Novembro 2018). “HOW TEACHING AI TO BE CURIOUS HELPS MACHINES LEARN FOR THEMSELVES”. Consultado em Novembro 15, 2019, de https://www.theverge.com/2018/11/1/18051196/ai-artificial-intelligence-curiosity-openai-montezumas-revenge-noisy-tv-problem

Michael Kana, Ph.D (Novembro 2019). “This Is How Reinforcement Learning Works”. Consultado em 15 de Março, 2019, de https://towardsdatascience.com/this-is-how-reinforcement-learning-works-5080b3a335d6

Alex Irpan (Fevereiro 2018). “Deep Reinforcement Learning Doesn't Work Yet”. Consultado em Novembro 19, 2019, de https://www.alexirpan.com/2018/02/14/rl-hard.html

Dr. Bernd Porr (2008). “Reinforcement learning”. Consultado em 27 de Novembro de 2019, de http://www.scholarpedia.org/article/Reinforcement_learning

Top 100 Games of All Time (2019). “Top 100 Video Games of All time”. Consultado em Novembro 22, 2019, de https://www.ign.com/lists/top-100-games