Técnicas de Aprendizado por Reforço Aplicadas em Jogos Eletrônicos na Plataforma Geral do *Unity*

Gabriel Prudencio Haddad[1], **Rita Maria Silva Julia**[1], **Matheus Prado Prandini Faria**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Uberlândia (UFU)
Caixa Postal 38408-100 – Uberlândia – MG – Brazil

gabriel.haddad@ufu.br, {ritasilvajulia, matheusprandini.96}@gmail.com

Abstract. *The aim of this work is to investigate the performance of player agents based on reinforcement learning - more specifically, on Q-Learning and Deep Q-Networks (DQN) algorithms - through the Unity platform. For that, the authors implement in it Basic and GridWorld player agents that are trained according to such algorithms. In order to evaluate the agents' performance, a comparative analysis is carried out between them and the best player agents of these games available on the Unity platform, which are based on the Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) learning techniques.*

Resumo. *O objetivo do presente trabalho é investigar o desempenho de agentes jogadores baseados em aprendizado por reforço - mais especificamente, nos algoritmos Q-Learning e Deep Q-Networks (DQN) - por meio da plataforma Unity. Para tanto, os autores implementam nela agentes jogadores de Basic e GridWorld que são treinados segundo tais algoritmos. A fim de avaliar o desempenho desses agentes, foi efetuada uma análise comparativa entre eles e os melhores agentes jogadores desses jogos disponibilizados na plataforma Unity, os quais são baseados nas técnicas de aprendizagem Proximal Policy Optimization (PPO) e Soft Actor-Critic (SAC).*

1. Introdução

Com o rápido avanço dos computadores, da Internet e da Inteligência Artificial (IA) nesta última década, tornou-se cada vez mais fácil para os seres humanos realizarem, em poucos segundos, tarefas que antes levavam anos para serem cumpridas [Stats 2021]. A evolução das técnicas de Aprendizado de Máquina (AM) proporcionou a abertura de um leque imenso para experimentação, simulação de tarefas e resolução de problemas nos mais variados cenários [Juliani et al. 2018] como, por exemplo, carros autônomos e jogos digitais. Neste contexto, diversas pesquisas envolvendo a criação de agentes jogadores vêm servido como laboratório de estudo para impulsionar as técnicas de AM, principalmente de Aprendizagem Profunda (AP).

Dentre os vários aspectos que diferenciam os métodos de aprendizagem de agentes, destaca-se a forma com que eles são treinados durante o processo de aprendizagem. Considerando a abordagem de Aprendizado por Reforço (AR), os agentes são treinados de forma a ajustarem seu processo de tomada de decisão em função de reforços recebidos (recompensa ou punição) pelas ações que escolheram ao longo de sua atuação [Russell and Norvig 2013]. Por outro lado, no Aprendizado Supervisionado (AS)

os agentes ajustam sua maneira de agir por meio de dados rotulados que os norteiam indicando o quão eles se afastaram da decisão adequada em cada situação que enfrentaram [Russell and Norvig 2013].

No contexto da construção de sistemas de AM, tanto AS quanto AR, projetar um extrator de características (*features*) que transformasse os dados brutos (como os valores dos *pixels* de uma imagem) em uma representação interna adequada a partir do qual o subsistema de aprendizagem pudesse detectar ou classificar padrões na entrada, demandava um complexo trabalho de engenharia e um profundo conhecimento do domínio [LeCun et al. 2015]. Tal limitação persistiu até o surgimento do Aprendizado Profundo (AP) - ou do inglês *Deep Learning* - que permitiu a criação de modelos computacionais compostos de várias camadas de processamento, conhecidos como Redes Neurais Profundas (RNPs), que são capazes de aprender representações de dados com múltiplos níveis de abstração. O AP permite que uma máquina seja alimentada com dados brutos de imagem e automaticamente descubra as representações necessárias para facilitar as tarefas de detecção ou classificação [LeCun et al. 2015]. Ainda assim, pré-processamento é necessário, no sentido de passar os dados para um formato adequado para leitura pela rede neural.

O ramo do AM que combina o AR com o AP é denominado Aprendizado por Reforço Profundo (ARP) [Mousavi et al. 2018], o qual tem permitido a construção de agentes que superam o nível de jogo humano em diversas tarefas dentro do domínio de jogos. Tal ramo é comumente composto por RNPs, principalmente as Redes Neurais Convolucionais (RNCs), que têm apresentado um excelente nível de desempenho em tarefas envolvendo dados visuais (baseados em imagens) para representar os estados do ambiente. Dentre os principais algoritmos do ARP, destacam-se o Deep Q-Networks (DQN) [Mnih et al. 2015], o Soft Actor Critic (SAC) [Haarnoja et al. 2018] e o Proximal Policy Optimization (PPO) [Schulman et al. 2017].

Os jogos digitais possuem uma grande relevância em pesquisas de AM devido aos seus impactos econômicos e culturais no mundo. No ano de 2020, o mercado global de jogos foi estimado no valor de quase 180 bilhões de dólares e com uma previsão de crescimento para os próximos anos [Witkowski 2020]. Além disso, os jogos envolvem dificuldades técnicas muito semelhantes às enfrentadas pelos agentes que lidam com problemas da vida real, porém em um ambiente mais controlado, o que elimina os riscos envolvidos em se fazer avaliações de desempenho por meio de aplicações diretas ao mundo real [Dosovitskiy et al. 2017].

Neste sentido, diversas plataformas simuladoras de jogos vêm sendo propostas na literatura para permitir que sejam testados algoritmos - especialmente aqueles baseados em IA - em jogos com uma maior facilidade e flexibilidade. As precursoras reuniam conjuntos de ambientes simples e normalmente semelhantes entre si para testar a capacidade de generalização de agentes, destacando-se o *Arcade Learning Environment* (ALE) [Bellemare et al. 2013], o qual permitiu o desenvolvimento bem-sucedido do DQN, um dos algoritmos mais conhecidos quando se trata de ARP, no vídeo game ATARI 2600. Outras plataformas mais recentes começaram a estudar e criar simuladores que permitiriam a execução de tarefas mais complexas, como locomoção ou navegação em primeira pessoa, destacando-se o renomado DeepMind Lab [Beattie et al. 2016].

Em 2018, foi desenvolvida a plataforma inovadora *Unity & ML-Agents* [Juliani et al. 2018], que provê uma grande flexibilidade na criação de ambientes com variadas características físicas e visuais, onde há muita liberdade de definição dos limites de complexidade das tarefas executada pelos elementos que compõem esses ambientes, bem como do nível de interação entre eles. Tal flexibilidade se deve à combinação do motor gráfico de jogos atuais chamado *Unity* com os recursos de aprendizagem propiciados pela plataforma de pesquisas em IA *ML-Agents*. Essa combinação propiciou um rico ambiente de validação de novos algoritmos de aprendizagem que se queira propor.

Motivado pelos argumentos apresentados, o presente trabalho tem como objetivo principal investigar técnicas de AR, com foco principal nos métodos Q-learning e DQN, utilizando como estudo de caso o flexível ambiente de jogo da plataforma *Unity & ML-Agents*. As principais contribuições desse artigo são: 1) Implementação de um agente jogador para o problema simples de navegação denominado *Basic*. Como a sua simplicidade dispensa técnicas de AP, o agente construído foi baseado no Q-Learning (o qual já alcança um ótimo desempenho); 2) Implementação de agentes jogadores para o problema chamado *GridWorld*. Destaca-se que tal problema é muito utilizado na literatura por permitir várias configurações de jogo que ajudam na experimentação e teste de vários algoritmos (ou seja, é bastante flexível para criar cenários com diferentes complexidades). Neste contexto, foram avaliados métodos que envolvem AP - DQN, PPO e SAC, sendo que esses dois últimos integram o grupo de algoritmos já implementados na plataforma - e que não envolvem AP - Q-Learning. Cada um dos quatro agentes foram baseados em um dos referidos métodos; 3) Análise comparativa de desempenho entre os agentes projetados para cada problema aqui abordado (*Basic* e *GridWorld*). Os parâmetros avaliativos utilizados foram o tamanho total de passos e a taxa de vitórias dos agentes treinados.

Este trabalho está organizado da seguinte maneira: a Seção 2 apresenta os fundamentos teóricos; a Seção 3 apresenta os trabalhos relacionados; a Seção 4 descreve o desenvolvimento dos agentes aqui implementados; a Seção 5 analisa os experimentos e resultados; e, finalmente, a Seção 6 apresenta as conclusões e os trabalhos futuros.

2. Fundamentos Técnicos e Teóricos

Esta seção apresenta conceitos técnicos e teóricos importantes para a compreensão deste trabalho.

2.1. Redes Neurais Convolucionais (RNC)

As RNCs são arquiteturas de aprendizado profundo, como as (RNPs), que se inspiram no modo como o córtex visual humano interpreta as imagens captadas pelos olhos, processando-as de modo a tentar reconhecer padrões delas [Sewak 2019]. Elas são compostas principalmente por 5 tipos de camadas: camada de entrada, camada de convolução, camada de *pooling*, camada totalmente conectada (fully-connected layer) e camada de saída. Uma arquitetura padrão consiste na repetição de pares de camadas de convolução e de *pooling* intercaladas seguidos por uma ou mais camadas totalmente conectadas, ligando-se a última delas à camada de saída.

2.2. Q-Learning

Q-Learning é um algoritmo de AR no qual a política de decisão do agente se baseia em dois critérios: 1) Uma *função valor*, que produz uma avaliação de cada ação *a* que pode

ser executada a partir de cada estado *s*, ou seja, uma avaliação de cada par *(s,a)* do espaço de estados do problema [Russell and Norvig 2013]; e 2) Um critério de aleatoriedade.

A função valor consiste em uma Tabela Q representada por meio de uma matriz NxM, onde N e M correspondem, respectivamente, aos números de estados e de ações possíveis no problema estudado. No início do treinamento do agente, tal tabela é inicializada com valores aleatórios. Ao longo do treinamento, então, por meio de reforços obtidos em sucessivas experimentações em que o agente varia sua escolha dentre as ações possíveis para um mesmo estado [Watkins 1989], os valores da referida tabela são ajustados, de forma a aumentar gradativamente a acuidade da avaliação de cada par *(s,a)*.

Tal estratégia garante a seguinte vantagem ao Q-Learning operando em problemas com espaço de estados de dimensão compatível (conforme será visto na seção 5, a sua eficiência é muito comprometida quando usado em problemas com elevado espaço de estados): a fácil adaptabilidade ao lidar com problemas com transições e recompensas estocásticas. O uso de tal algoritmo será visto em maiores detalhes na seção 4.2.1.

2.3. Deep Q-Networks (DQN)

O algoritmo DQN é uma combinação do Q-learning com o conceito de RNC. Tal combinação tem como propósito tornar o algoritmo uma alternativa viável para a construção de agentes aptos a lidar com problemas em que a elevada dimensão do espaço de estados se torna um grande desafio [Mnih et al. 2015] (situação que inviabiliza o uso do Q-Learning). Em sua essência, o DQN difere do Q-Learning por substituir a tabela Q que armazena as avaliações de todos os pares *(s,a)* do espaço de estados do problema (ou seja, a função ação-valor) por uma RNC. No caso, a entrada desta rede é uma imagem que representa o estado corrente s do ambiente no qual o agente opera. Cada neurônio da camada de saída representa uma ação a_i possível de ser executada a partir de s, sendo que o valor produzido por ele representa a avaliação do par *(s, a_i)*. Maiores detalhes sobre o uso do DQN como uma abordagem de aprendizado serão apresentados na seção 4.3.1.

3. Trabalhos Relacionados

Dentre os trabalhos correlatos, é fundamental citar [Mnih et al. 2015], proposto pela equipe *Google Deepmind*, onde o DQN foi originalmente apresentado e confirmado como uma notável ferramenta de aprendizagem no cenário de jogos digitais. Em tal trabalho, os agentes são treinados sem supervisão por técnicas de ARP a partir das entradas representadas pelas imagens cruas (*pixels* brutos) de jogo. Tal independência de aprendizagem em relação ao conhecimento humano permitiu que os agentes implementados de acordo com o DQN ultrapassassem a expertise humana em vários jogos do clássico vídeo game *Atari* 2600.

Além desse, é importante ressaltar [Min et al. 2019], um trabalho correlato que demonstra como a flexibilidade do ambiente do *Unity* com o *ML-Agents* permite o enfrentamento de problemas complexos, bem como serve de base para investigar o desempenho de algoritmos de AM. O objetivo do artigo é treinar um agente inteligente para operar como supervisor de sistemas assistentes de direção já disponíveis no mercado, de modo a produzir um sistema de direção autônoma de carros confiável. Ele utiliza vários algoritmos conhecidos da literatura como DQN, Double Deep Q-Network (DDQN), Dueling Deep Q-Network (Dueling DQN) e Quantile Regression Deep Q-Network (QR-

DQN), junto à um ambiente estocástico de uma rodovia. Este estudo confirma a possibilidade de usar o algoritmo de reforço distributivo profundo para treinar um supervisor para o controle de um veículo autônomo e sugere uma nova direção de estudo na área de ARP.

4. Desenvolvimento

Esta seção apresenta os agentes Basic e GridWorld implementados no ambiente Unity & ML-Agents. A Figura 1 ilustra a arquitetura geral desses agentes. Ela é composta por três módulos principais: *Unity Engine*, que gera a representação de estados; Método de Tomada de Decisão, que realiza as tomadas de decisões; e o método de AR, que corresponde ao algoritmo de aprendizagem utilizado.

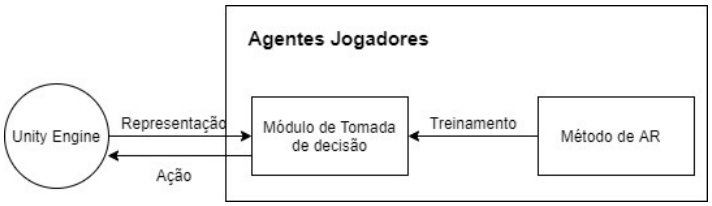

Figura 1. Arquitetura dos Agentes Jogadores para Aprendizado por Reforço

4.1. Ambientes de Jogo

A fim de cumprir com o objetivo de avaliar os algoritmos Q-Learning e DQN no contexto de jogos, os agentes jogadores implementados neste trabalho são apresentados abaixo:

- **Agente Basic**: como tal jogo apresenta um espaço de estados bastante modesto, que dispensa o uso de técnicas de AP, será implementado um único agente Basic baseado no Q-Learning;
- **Agentes GridWorld**: considerando a significativa dimensão do espaço de estados do GridWorld, serão implementados quatro agentes para lidar com tal jogo, cada um deles baseado em uma das seguintes abordagens de aprendizado: Q-Learning, DQN, SAC e PPO. No caso, o objetivo é comprovar, por meio de experimentos, que: 1) diferentemente dos algoritmos de AP representados pelo DQN, SAC e PPO, o Q-Learning não é adequado para lidar com problemas com grande complexidade em termos de espaço de estados; 2) Dentre os algoritmos de AP investigados, o DQN é mais apropriado do que o SAC e o PPO para lidar com problemas com espaço de estados discreto como o GridWorld;

Assim sendo, o módulo *métodos de AR* apresentado na Figura 1 refere-se a um desses quatro algoritmos.

As subseções 4.1.1 e 4.1.2 descrevem os ambientes de jogo, respectivamente, dos agentes Basic e GridWorld implementados.

4.1.1. Ambiente Basic

O ambiente do Basic pode ser resumido da seguinte forma:

- **Estados**: A representação discreta de estados, produzida a partir da vista superior 2D do cenário real de jogo[1], corresponde a um vetor de 10 posições, cada uma representando uma casa do tabuleiro.
- **Ações**: É um ambiente com espaço de ações discreto. Existem 3 possíveis ações, que correspondem a Parado, Esquerda e Direita.
- **Recompensas**: O cenário é composto por dois objetivos, um menor e um maior, que, quando atingidos, geram ao agente a recompensa de +0.1 e +1.0, respectivamente, sendo o ambiente reiniciado sempre que o agente encosta em um deles. Além disso, para cada passo do agente que não leve a uma posição terminal, ele é recompensado com -0.01, de forma a incentivar o agente a dar o menor número de passos possível para atingir a posição desejada (não existe um número de passos limite que ele pode efetuar).

4.1.2. Ambiente GridWorld

Esta subseção resume o ambiente de jogo usado na implementação de todos os agentes GridWorld aqui implementados.

- **Estados**: É um ambiente representado por uma visão de cima do cenário de jogo[2], com uma matriz de *pixels* de tamanho 64x84.
- **Ações**: É um ambiente com espaço de ações discreto, sendo 5 as possíveis ações: Parado, Baixo, Cima, Esquerda e Direita.
- **Recompensas**: O cenário é composto por um objetivo e um obstáculo que quando atingidos, geram ao agente as recompensa de +1.0 e -1.0, respectivamente, sendo o ambiente reiniciado com os objetos em posições aleatórias do tabuleiro. Além disso, a cada passo do agente que não seja para uma posição de objetivo ou obstáculo, ele é punido com -0.01, de forma a incentivá-lo a dar o menor número de passos possível para atingir o objetivo. No entanto, se o agente realizar 100 passos no ambiente em um episódio, o ambiente é reiniciado e o agente recebe a recompensa de -0.01.

4.2. Agentes Baseados no Q-learning

Apresentam-se aqui os agentes Basic e Grid-World implementados com base no Q-Learning. A subseção 4.2.1 descreve o algoritmo de treinamento usado, ao passo que as subseções subsequentes introduzem tais agentes.

4.2.1. Pseudocódigo do Q-learning

O Algoritmo da Figura 1 resume a dinâmica do aprendizado por Q-learning.

Os principais processamentos do algoritmo ocorrem entre as linhas 5 à 9, na qual o agente escolhe uma ação dada por meio da política *decaying epsilon-greedy* [Sutton 1988], para o atual estado de jogo s (Linha 5). Executando essa ação ele recebe uma recompensa e um novo estado s' (Linha 6) e, então, re-calcula o valor na Tabela

[1]https://github.com/Unity-Technologies/ml-agents/blob/main/docs/images/basic.png
[2]https://github.com/Unity-Technologies/ml-agents/blob/main/docs/images/gridworld.png

Algoritmo 1 Q-Learning

Saída: Função Ação-Valor Q

1: Inicializar aleatoriamente a tabela Q.
2: **for** *cada episódio* **do**
3: Inicializar o estado inicial s
4: **for** *cada passo do episódio* **do**
5: $a \leftarrow$ ação dada pela política π para o estado s
6: executar ação a, observar novo estado s' e recompensa r
7: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_a Q(s',a) - Q(s,a)]$
8: $s \leftarrow s'$
9: **if** estado s é terminal **then**
10: **break**
11: **end if**
12: **end for**
13: **end for**

Q para aquele par estado-ação (Linha 7). Finalmente, ele coloca o novo estado s' como o atual (Linha 8) e verifica se o estado atual é o de final de jogo (Linha 9). Se for, ele finaliza o episódio. Caso contrário, ele continua o processamento.

4.2.2. Agente Basic (Q-learning)

O ambiente é representado por um vetor de tamanho 1 (numero de linhas) x 10 (numero de colunas). O agente começa na posição 5 do vetor e deve se movimentar até o objetivo. Os objetivos correspondem aos extremos do vetor, sendo o menor objetivo a primeira posição da lista e, o maior, a última posição. A tabela Q tem dimensão de 10 (tamanho do vetor de representação) x 3 (número de ações possíveis).

4.2.3. Agente GridWorld (Q-learning)

O método de AR utilizado foi o Q-learning e o Método de Tomada de Decisão foi uma Tabela Q, que representa a função ação-valor para ação de cada estado possível. Para o problema do *GridWorld*, o agente possui uma Tabela Q de tamanho 13800x3, sendo que o valor 13.800 indica o tamanho do vetor de representação e, o valor 3, o número de ações possíveis. O número de 13.800 representa a quantidade de estados do jogo que deverá ser representada na tabela Q. Logo, para o problema usando um tabuleiro de 5 linhas e 5 colunas (5x5), tendo 1 objetivo, 1 obstáculo e 1 agente, são necessárias pelo menos 13.800 posições para representar todos os estados possíveis.

4.3. Agentes Baseados no GridWorld

Apresentam-se aqui os agentes GridWorld implementados com base no DQN. A subseção 4.3.1 descreve o algoritmo de treinamento usado, ao passo que as subseções subsequentes introduzem tais agentes.

4.3.1. Pseudocódigo do DQN

Conforme mencionado na subseção 2.3, a função ação-valor do DQN corresponde a uma RNC. Assim sendo, no presente trabalho a entrada desta rede é uma imagem que representa o estado s correspondente ao cenário corrente do jogo GridWorld. Cada neurônio

da camada de saída representa uma ação a_i do jogo, sendo que o valor produzido por ele representa a avaliação do par *(s, a_i)*.

Logo, o algoritmo de aprendizagem do DQN é usado para treinar a RNC que compõe a arquitetura do agente, de forma que ela, ao longo desse processo de treinamento, progressivamente aumente a acuidade das avaliações que produz para os pares *(s,a)*.

Os seguintes conceitos integram o processo de aprendizagem por DQN:

- **Experience Replay**: Uma das técnicas propostas para melhorar a execução do DQN é a implementação do *Experience Replay* (Repetição de Experiencias). Uma "experiência" é representada por uma tupla (s, a, r, s'). A técnica do *Experience Replay* é usada de modo a atualizar os pesos da rede convolucional e, consequentemente, ajustar a função ação-valor. A implementação da técnica de *Experience Replay* é muito útil na construção de agentes jogadores que percebem o ambiente por meio de imagens. Isso se deve ao seguinte fato: nessas situações o agente recebe vários quadros em rápida sucessão. Levando em consideração que os quadros (*frames*) disponibilizados para a RNC representam cenários subsequentes de jogo, eles tendem a ser muito semelhantes entre si. Assim sendo, um treinamento com reajustes feitos a partir de quadros consecutivos de jogos tenderia a ser bastante ineficiente, pois desperdiçaria tempo efetuando vários reajustes com pouca relevância em situações em que houve pouca variação no cenário de jogo.
- **Target Network Adicional**: *Target* é o nome dado para a parte da função de atualização que soma a Recompensa Imediata recebida à melhor avaliação que uma RNC Q', que opera como função ação-valor, retorna em sua saída para um dado estado s' resultante da execução de uma ação anterior (sendo tal avaliação ponderada por uma constante γ, descrita na Tabela 1). Logo, a expressão correspondente ao *target* é a seguinte: $r + \gamma max_a Q'(s', a)$. No artigo original do DQN, outro método criado para melhorar ainda mais a estabilidade do algoritmo junto à rede neural é usar uma outra RNC Q' (além da RNC original Q usada como função ação-valor) para gerar os *targets* na hora da atualização dos pesos. A cada C atualizações, sendo C um inteiro, que representa um intervalo de episódios, a rede neural Q é clonada para obter a rede *target* Q', sendo essa última usada nas próximas C atualizações dos parâmetros da rede Q.

O Algoritmo 2 resume a dinâmica de treinamento do DQN. A Tabela 1 apresenta alguns dos hiperparâmetros usados pelo algoritmo.

A execução começa quando o algoritmo recebe uma imagem que representa o estado corrente s_c e baseado nele escolhe uma ação definida pela política *decaying epsilon-greedy* [Sutton 1988] (Linha 7), então executa a ação e recebe um novo estado s'_c e uma recompensa r. Ele guarda a experiência obtida em um vetor D e seleciona um mini-lote aleatório desse vetor para atualizar a rede neural. Nas linhas 12 à 19 é feito essa atualização da rede, onde é importante ressaltar a Linha 16, no qual o termo $max_{a'} Q'(s', a')$ se refere ao maior valor de saída da rede, representando, assim, a avaliação da melhor ação disponível a' partir de s' (de acordo com que o agente aprendeu até então).

Algoritmo 2 Deep Q-learning com replay de experiências

Inicializar o vetor de memórias D com a capacidade N.
Inicializar a função estado-valor Q com pesos aleatórios θ.
Inicializar a função estado-valor alvo Q' com pesos aleatórios $\theta' = \theta$. (***Target Network Adicional***)
for *cada episódio* **do**
    Pré-processamento que gera a imagem correspondente ao estado corrente s_c.
    for *cada passo do episódio* **do**
        $a_c \leftarrow$ ação dada pela política π para o estado s_c
        executar ação a_c, armazenar o novo estado corrente s'_c e recompensa r_c
        Pré-processamento que gera a imagem correspondente ao novo estado s'_c.
        Guardar experiência (s_c, a_c, r_c, s'_c) em D
        Selecione aleatoriamente em D um mini-lote contendo K experiências (s, a, r, s'). (***Experience Replay***)
        for *cada experiência do mini-lote* **do**
            if estado s é terminal **then**
                $y \leftarrow r$
            else
                $y \leftarrow r + \gamma max_{a'} Q'(s', a')$.
            end if
            Atualize os parâmetros do modelo com $(y - \mathbf{Q(s, a)})^2$
        end for
        $s_c \leftarrow s'_c$
    end for
    A cada C episódios faça $Q' \leftarrow Q$. (***Target Network Adicional***)
end for

Tabela 1. Hiperparâmetros associados ao DQN

Hiperparâmetro	**Significado**
Tamanho do Mini-Lote (K)	Número de transições utilizadas para efetuar a atualização da *Q-Network*
Tamanho do *buffer* de experiências (N)	Número de transições que podem ser armazenadas no *buffer*
Atualização da *Target Network* (*C*)	Quantidade de episódios necessários para efetuar a atualização da *Target Network* pela *Q-Network* atual
Fator de desconto (γ)	Indica o quão importante o futuro é levado em consideração na atualização da *Target Network*
Taxa de aprendizagem	Representa o quanto será aprendido pelo agente com relação ao ajuste calculado
Valor inicial de ϵ	Representa o valor de exploração que o método adota no início do treinamento
Valor final de ϵ	Representa o valor mínimo de exploração que o método ao longo do treinamento

4.3.2. Agente GridWorld (DQN)

A RNC usada no agente GridWorld tem 3 camadas de convolução e duas camadas completamente conecta das de classificação. A Tabela 2 resume os parâmetros usados no treinamento dessa rede.

Tabela 2. Configuração do RNC usado no GridWorld

Camada	**Entrada**	**Tamanho do Filtro**	***Stride***	**Num Filtros**	**Ativação**	**Saida**
conv1	64x84	8x8	4	32	ReLU	20x20
conv2	20x20	4x4	2	64	ReLU	9x9
conv3	9x9	3x3	1	64	ReLU	7x7
fc4	7x7	-	-	-	ReLU	512
fc5	512	-	-	-	Linear	5

5. Experimentos e Resultados

Esta seção apresenta os experimentos executados com a finalidade de avaliar os agentes jogadores de *Basic* e *GridWorld* implementados neste trabalho.

Conforme mencionado anteriormente, diferentemente do Q-Learning e do DQN - que são os métodos de aprendizagem que o presente trabalho se propõe a investigar - o SAC e o PPO são métodos de AR que já estão implementados no ambiente *Unity & ML-*

Agents. Em virtude disso, eles são introduzidos nos experimentos aqui propostos com propósito meramente comparativo, usando as seguintes métricas:

- **Média da recompensa cumulativa**: é a média da soma de todas as recompensas obtidas ao longo de um episódio.
- **Taxa de vitória global**: é a porcentagem do número de vitórias dividido pela quantidade de episódios totais.

5.1. Experimento 1 - Avaliação do Agente *Basic* baseado no Q-Learning

Apesar de o *Basic* apresentar características simples, ele é apropriado pra demonstrar que o Q-learning consegue, com poucos passos (nos experimentos foram empiricamente limitados a 100.000), atingir o objetivo obtendo uma recompensa elevada. O tamanho do ambiente aqui utilizado tem dimensão 1x10. Assim sendo, na Tabela 3 é possível perceber que, no final do treinamento, o agente alcançou uma taxa de vitórias de 90%. Este fato, combinado com as informações de recompensa cumulativa, mostram que o agente teve sucesso na aprendizagem e conseguiu se direcionar rumo ao objetivo de forma rápida.

Tabela 3. Resultados agente Basic baseado no Q-learning

Métrica	Quantidade Passos	Valor
Média da Recompensa Cumulativa	100.000	0.7
Taxa de Vitória Global	100.000	90%

5.2. Experimento 2 - Avaliação dos Agentes *GridWorld* baseados no Q-Learning, DQN, SAC e PPO

Nessa seção, é realizado um paralelo entre os algoritmos PPO e SAC, já implementados no *Unity* pelo *framework ML-Agents*, com o DQN e o Q-learning no problema do *GridWorld*. PPO e SAC são algoritmos implementados e testados para espaços de ações contínuos. No entanto, neste trabalho foram aplicados ao problema discreto do *GridWorld* na intenção de observar seus comportamentos em tal cenário. Por outro lado, é interessante salientar que o contrário não pode ser feito no que diz respeito ao DQN e o Q-learning (projetados para espaços discretos), uma vez que nenhum deles pode ser usado em espaços contínuos pelo seguinte motivo: ambos definem suas políticas para pares discretos de (ação, estado), o que não é possível de ser replicado em espaços contínuos, pois achar uma política *greedy* requer que o algoritmo ache a melhor ação a para o estado s a cada passo no tempo [Lillicrap et al. 2016].

Conforme Tabela 4, o DQN apresenta a convergência mais rápida (obtendo recompensa cumulativa média de 0.6 após 60.000 passos de treinamento), enquanto que o SAC e o PPO requerem, respectivamente, 150.000 e 300.000 passos para alcançar esse mesmo valor de recompensa. O Q-learning, mesmo após 600.000 passos, não consegue chegar ao mesmo valor que os outros algoritmos.

Saliente-se que esses resultados apresentados pelo DQN e PPO são coerentes com aqueles obtidos em [Karttunen et al. 2020], onde o autor avalia o desempenho de tais algoritmos considerando um problema contínuo P_c e produzindo sua versão discreta P_d. Na sequência, ele avalia o desempenho do DQN no P_d e o do PPO no P_c. A partir desses resultados é possível perceber que o PPO não é tão *sample efficient* quanto o DQN.

Tabela 4. Resultados agentes GridWorld

Algoritmo	Total De Passos	Média da Recompensa Cumulativa	Taxa de Vitória Global
Q-learning	600.000	0.2	70%
DQN	60.000	0.5	90%
SAC	130.000	0.5	-*
PPO	250.000	0.5	-*

* Os valors desses parametros não são disponibilizados pelo Unity.

Sample Efficiency ou eficiência da amostra, denota a quantidade de experiência que um agente precisa gerar em um ambiente (por exemplo, o número de ações que executa e o número de estados + recompensas que observa) durante o treinamento para atingir um determinado nível de desempenho. Além disso, o resultado obtido pelo SAC também é coerente, uma vez que [Haarnoja et al. 2018] mostra que ele é mais *sample efficient* que o PPO.

Por fim, a taxa de vitória global obtida pelo DQN e pelo Q-learning evidencia a vantagem da utilização de uma RNC no processo de treinamento, uma vez que o DQN alcança uma taxa de 90% em 60.000 passos, enquanto que o Q-learning alcança uma taxa de apenas 70% em 600.000 passos.

6. Conclusão

Este trabalho efetuou uma investigação comparativa sobre a performance de agentes jogadores baseados nos algoritmos de AR Q-Learning e DQN nos ambientes de jogo Basic e GridWorld da plataforma Unity. Tal investigação levou em consideração tanto a taxa de vitórias quanto o tempo de convergência dos agentes. Os resultados obtidos nos experimentos mostraram que o agente *Basic* baseado no Q-Learning alcançou um ótimo desempenho em ambos os parâmetros avaliados. Com relação ao agente *GridWorld* baseado no Q-Learning, ele não obteve resultados satisfatórios. Tal agente alcançou uma taxa de vitórias de aproximadamente 70% (bem menor comparado aos outros agentes) e com um tempo de convergência mais lento. Por outro lado, o agente *GridWorld* baseado no DQN apresentou o melhor desempenho dentre todos aqueles aqui estudados, alcançando uma taxa de vitórias de aproximadamente 90%, sendo 10 vezes mais rápido (em termos do número de passos) que o agente baseado no Q-Learning. Tal resultado já era esperado, uma vez que o DQN conta com uma RNC preparada para o processamento e classificação de imagens com um poder maior de generalização do que a Tabela de Valores Q utilizada pelo Q-Learning. Em comparação aos agentes PPO e SAC, o DQN mostrou-se muito mais eficiente em termos de convergência. Por fim, os autores pretendem estudar mais a fundo os diferentes métodos de aprendizagem profunda, como o PPO e SAC, em ambientes com ações e estados contínuos e comparando-os a outros algoritmos do Estado da Arte.

References

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. (2016). Deepmind lab. *arXiv preprint arXiv:1612.03801*.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Juliani, A., Berges, V.-P., Vckay, E., Gao, Y., Henry, H., Mattar, M., and Lange, D. (2018). Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

Karttunen, J., Kanervisto, A., Kyrki, V., and Hautamäki, V. (2020). From video game to real robot: The transfer between action spaces. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3567–3571.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521:436–444.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR) 2016*.

Min, K., Kim, H., and Huh, K. (2019). Deep distributional reinforcement learning based high-level driving policy determination. *IEEE Transactions on Intelligent Vehicles*, 4(3):416–424.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518:529–533.

Mousavi, S. S., Schukat, M., and Howley, E. (2018). Deep reinforcement learning: An overview. In Bi, Y., Kapoor, S., and Bhatia, R., editors, *Proceedings of SAI Intelligent Systems Conference (IntelliSys)*, pages 426–440.

Russell, S. and Norvig, P. (2013). *Artificial Intelligence - 3ª Ed*. Elsevier.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sewak, M. (2019). *Deep reinforcement learning*. Springer.

Stats, I. W. (2021). Internet growth statistics.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

Watkins, C. J. C. H. (1989). Learning from delayed rewards.

Witkowski, W. (2020). Videogames are a bigger industry than movies and north american sports combined, thanks to the pandemic.