Tecnicas de Aprendizagem por Reforço na resolução do Mundo de Wumpus

Rodrigo Moraes Rodrigues[1], **Otávio Noura Teixeira**[2], **Natália Freitas Araújo**[3]

[1]Faculdade de Engenharia de Computação – Universidade Federal do Pará (UFPA)

{igo.moraes07,onoura,taiaraujo20}@gmail.com

Abstract. *This work aims to analyze the performance of an agent based on Reinforcement Learning. Your learning engine is based on three algorithms: Q-learning (QL), Deep Q-Network (DQN) and Double Deep Q-Network (DDQN). To validate the agent and its methods, it was defined as environment the World of Wumpus, which was modeled according to the environment standards adopted by DeepMind Lab. From the experiments performed and their respective configurations, it was observed that the agents managed to reach the main objective only in two configurations of environments. In the 4x4 environment the winning percentage of the QL, DQN algorithms and DDQN were 0.005, 22.96, 18.73% respectively, which drastically reduced specifically for the 10x10 scenario and failing to meet the objective for the other environments.*

Resumo. *Este trabalho tem por objetivo analisar o desempenho de um agente baseado em Aprendizagem por Reforço. O seu mecanismo de aprendizagem está baseado em três algoritmos: Q-learning (QL), Deep Q-Network (DQN) e Double Deep Q-Network (DDQN). Para validação do agente e seus métodos, foi definido como ambiente o Mundo de Wumpus, o qual foi modelado segundo os padrões de ambientes adotados pela DeepMind Lab. A partir dos experimentos realizados e suas respectivas configurações, foi observado que os agentes conseguiram alcançar o objetivo principal somente em duas configurações de ambientes. No ambiente 4x4 a porcentagem de vitória dos algoritmos QL, DQN e DDQN foram 0.005, 22.96, 18.73 % respectivamente, o que reduziu drasticamente para o cenário 10x10 e não conseguindo cumprir o objetivo para os demais ambientes.*

Palavras-chave: *Q-Learning, Deep Q-Network, Double Deep Q-Network, mundo de wumpus*

1. Introdução

A inteligência computacional é o estudo de projeto de agentes inteligentes, onde seu desempenho é medido por sua capacidade de tomar decisões apropriadas para suas circunstâncias e seus objetivos [Poole et al. 1998]. Partindo desse pressuposto, [Russel and Norvig 2010] definem que um agente é capaz de analisar seu ambiente através de seus sensores e de agir de acordo com sua percepção através de seus atuadores.

Nos últimos anos devido aos avanços tecnológicos se tornou cada vez mais frequente a utilização de agentes inteligentes para execução de tarefas que antes eram realizadas por seres humanos [Damião et al. 2014]. Um dos principais mecanismos de aprendizagem utilizados por esses agentes são os métodos do ramo da inteligência artificial

(IA) conhecida como Aprendizagem de Máquina (do Inglês Machine Learning - ML), subdividida em: Aprendizagem Supervisionada, Aprendizagem Não Supervisionada e Aprendizagem por Reforço [Russel and Norvig 2010].

Na Aprendizagem Supervisionada o agente ajusta suas ações através dos rótulos recebidos que o direcionado o quão próximo ou distante está da decisão adequada para o problema enfrentado. Por outro lado, na Aprendizagem Não Supervisionada o agente aprende os padrões baseados nas entradas sem que haja nenhum feedback [Russel and Norvig 2010]. A Aprendizagem por Reforço (do Inglês Reinforcement Learning – RL) é um método computacional que tem como base treinar agentes capazes de adquirir conhecimento através da interação com o seu ambiente, com o objetivo de descobrir quais ações produzem a maior recompensa cumulativa [Sutton and Barto 2020].

Na última década as pesquisas em métodos de RL vem ganhando notoriedade no meio científico, devido ao desenvolvimento da Aprendizagem Profunda (do Inglês Deep Learning - DL) e sua poderosa função em aproximar e representar as propriedades de aprendizado de redes neurais [Dias 2019], impulsionando o desenvolvimento de novos algoritmos como: Deep Q-Network [Mnih et al. 2015] e Double Deep Q-Network [Van Hasselt et al. 2016].

O Deep Q-Network (DQN) surgiu como uma nova abordagem do popular método tabular Q-Learning para lidar com problemas em que demandam uma grande quantidade de estados, limitação encontrada pelo seu antecessor [Mnih et al. 2015]. O algoritmo Double Deep Q-Network (DDQN) foi desenvolvido como uma melhoria do DQN, com o objetivo de minimizar a superestimação do valor Q [Van Hasselt et al. 2016].

A Aprendizagem por Reforço tem aplicabilidade em diversas áreas, como: jogos, educação, visão computacional, processamento de linguagem natural e entre outras. Os jogos são uma importante ferramenta de estudos para RL/IA [Li 2017], propiciando o desenvolvimento de diversos trabalhos como: [Mnih et al. 2015] faz uso de jogos Atari como ambiente para o DQN e suas extensões, Backgammon [Tesauro 1994], AlphaGo [Silver et al. 2017], Tower Defense [Dias 2019] e entre outros.

Os jogos de tabuleiros são um dos principais bancos de ensaio de RL, por apresentarem interessante espaço de busca e os jogadores revelarem informações perfeitas sobre o ambiente [Li 2017]. O Mundo de Wumpus apresentado por [Russel and Norvig 2010] é um excelente ambiente para análise de desempenho de agentes inteligentes. Apesar de sua simplicidade este clássico jogo de tabuleiro apresenta pontos importantes sobre inteligência [Russel and Norvig 2010].

No que diz respeito a jogos eletrônicos, grande parte dos trabalhos desenvolvidos utilizam pixels como entrada de dados para os agentes, o que demanda uma tarefa de pré-processamento e maior custo computacional [Dias 2019]. Neste trabalho foi abordado a utilização de metadados como dados de entrada para os agentes, simplificando o modelo e reduzindo o tempo de processamento.

Baseado nas premissas apresentadas, o presente trabalho tem como principal objetivo explorar técnicas de RL, com foco nos métodos Q-Learning, DQN e DDQN, utilizando o mundo de Wumpus como ambiente de validação. Os objetivos específicos deste artigo são: 1) Modelagem do mundo de Wumpus baseado nos padrões estabelecidos pela plataforma Gym da renomada DeepMind Lab [Beattie et al. 2016]. A plataforma Gym

concentra uma variedade de ambientes para experimentação dos algoritmos de Aprendizagem por Reforço. 2) Implementar agentes baseados nos algoritmos QL, DQN e DDQN. 3) Análise comparativa de desempenho dos agentes na resolução do mundo de Wumpus em suas variadas configurações. Os parametros avaliativos utilizados foram porcentagem em que matou o wumpus, porcentagem em que alcançou o ouro, quantidade de vezes em que pegou o ouro e saiu da caverna.

A organização do artigo dar-se da seguinte forma: A seção 2 apresenta os principais conceitos para a compreensão e desenvolvimento deste trabalho. A seção 3 apresenta alguns trabalhos correlatos. A seção 4 descreve os materiais e métodos utilizados. A seção 5 apresenta os resultados e discussões. E por fim, a seção 6 apresenta as conclusões e trabalhos futuros.

2. Referencial Teorico

2.1. Aprendizagem por Reforço

A Aprendizagem por Reforço é um campo de estudos de ML onde agentes aprendem a executar ações de acordo com o estado do ambiente explorado, ou seja, através das diversas interações e recompensas recebidas. O agente mapeia quais ações resultarão na maior recompensa cumulativa, sem a necessidade de especificar qual tarefa deve ser executada [Morales 2020].

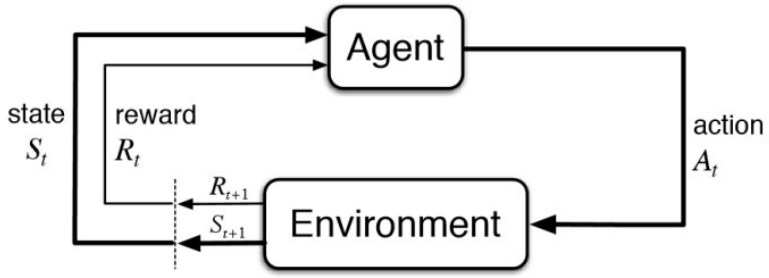

Figura 1. Diagrama de Aprendizagem Por Reforço
Fonte: [Sutton and Barto 2020].

Agentes baseados em RL observam o ambiente, executam uma ação, transacionam para um novo estado e recebem uma recompensa nesta transição [Sewak 2019], conforme ilustrados na Figura 1. Em maioria, os problemas de RL as noções de estados do ambiente, recompensas e ações do agente são descritas segundo o formalismo matemático do Processo de Decisão de Markov, discutido na seção a seguir.

2.2. Processo de Decisão de Markov

Em problemas de Aprendizagem por Reforço o agente executa suas ações baseado apenas no estado atual do ambiente. Um estado é dito markoviano ou que possui as propriedade de Markov, se a partir do estado atual é possível predizer a recompensa esperada para cada provável ação executada. Um jogo ou qualquer outro problema que exiba essas propriedades é considerado um processo de decisão de Markov (MDP - Markov Decision Process) [Brown and Zai 2020].

Um processo de decisão de Markov é modelado formalmente pelo conjunto (S, A, P, R), no qual: S corresponde ao conjunto de estados do problema enfrentado, A ao espaço de ações do agente, P é a função de probabilidade, também conhecida como função de transição e por fim, R é a recompensa que aponta o reforço recebido ao executar a ação $a \in A$ no estado $s \in S$ [Sutton and Barto 2020].

2.3. Q-Learning

Um dos primeiros avanços em RL foi o surgimento do algoritmo Q-Learning desenvolvido por [Walkins 1989]. O Q-Learning é classificado como um método de diferença temporal off-policy, uma vez que a função de ação-valor aprendida Q se aproxima diretamente da função ação-valor ótima Q*, através da atualização dos pares de estado-ação, à medida que os mesmo são visitados e independente da política que está sendo seguida [Sutton and Barto 2020], isto significa que a política que está sendo adotada é diferente do comportamento executado pelo agente. O Q-Learning fundamenta-se na equação de Bellman como função de atualização dos pares de estado-ação e é denotada por:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (1)$$

A função ação-valor deste algoritmo é representada por uma matriz Q, com dimensões NxM, onde N é o número de estados do ambiente e M é a quantidade de ações possíveis do problema enfrentado. No início do treinamento a matriz Q é inicializada com valores aleatórios e através das recompensas ou punições recebidas ao longo de sucessivas experimentações em que o agente varias suas ações, os valores da referida tabela são ajustados com objetivo de aumentar gradativamente sua avaliação dos pares (s, a). Os valores correspondentes aos pares (s, a) na matriz Q são denominados valor Q.

2.4. Redes Neurais

As Redes Neurais Artificiais (RN) são um modelo computacional inspirado no comportamento biológico do cérebro humano, com intuito de traduzir os dados de entrada para a saída desejada [Sewak 2019]. A estrutura de uma rede neural consiste em camadas constituídas de neurônios, pesos, função de ativação e bias. Uma rede neural básica é constituída de três camada: camada de entrada (input layer), camada oculta (hidden layer) e camada de saída (output layer). A camada de entrada é responsável por receber os dados de entrada, processá-los e encaminhá-los como entrada para a camada seguinte. A camada oculta recebe os dados da camada de entrada, aplica os pesos e através de uma função de ativação gera uma saída para os neurônios das camadas ocultas posteriores ou para a camada de saída [Sutton and Barto 2020].

A Figura 2 demonstra de forma visual a estrutura de uma RN, onde X são os dados de entrada, a_i são os neurônios de cada camada, as arestas são os pesos e a função $h_\theta(x)$ é a saída resultante da operação matricial dos dados recebidos da camada anterior com os pesos da camada de saída, aplicados a uma função de ativação $g(x)$, matematicamente expressos por $h_\theta(x) = g(\theta^T x)$.

2.5. Deep Q-Network

Na Seção 2.3 foi apresentado os conceitos sobre o algoritmo Q-Learning e seu método de aprendizagem através da atualização dos pares (s, a) da matriz Q. Por ser um método

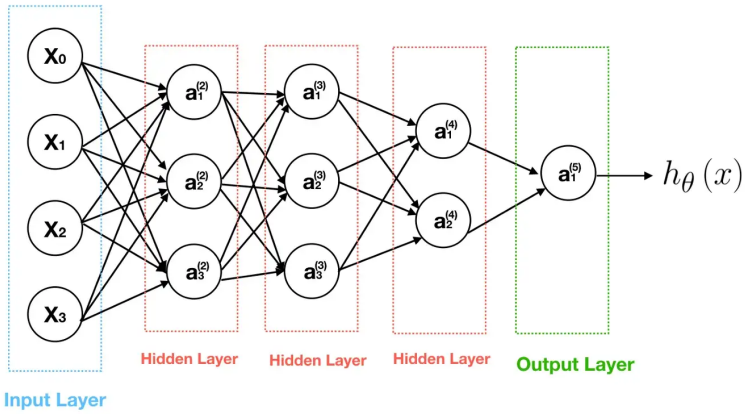

Figura 2. Redes Neurais Artificiais
Fonte: [Amber 2019].

tabular, onde há a necessidade de representar através de matriz a quantidade de estados do problema enfrentado, este popular método de RL torna-se computacionalmente inviável para resolução de problemas reais. Em que sua grande maioria apresenta uma alta quantidade de estados, por exemplo o jogo backgammon que apresenta 10^{20} estados.

A partir dessa problemática, [Mnih et al. 2015] apresentaram uma solução factível para o problema de elevadas dimensões de estados, combinando os conceitos do algoritmo Q-Learning e as propriedades de redes neurais profundas, diante dessas definições é apresentado o algoritmo Deep Q-Network. Diferente do seu antecessor Q-Learning que utiliza uma tabela para guardar informações dos pares de estado-ação (s, a) do problema enfrentado, ou seja, a função ação-valor, este referido algoritmo faz uso de uma rede neural profunda para estimar a ação de acordo com o estado corrente. A Figura 3 ilustra a comparação entre os métodos de aprendizagem utilizados por esses algoritmos.

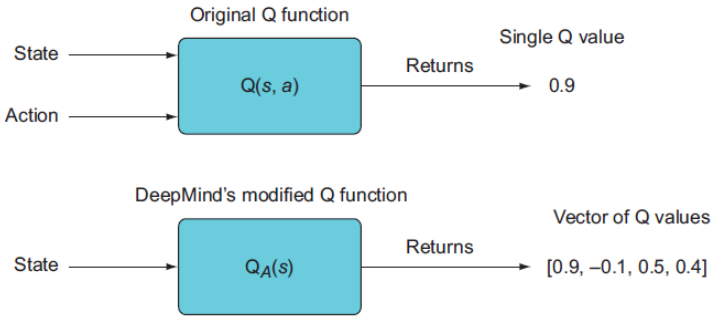

Figura 3. Comparação entre função ação-valor do Q-Learning x função de aproximação via redes neurais
Fonte: [Brown and Zai 2020].

Na abordagem apresentada por [Mnih et al. 2015] o ambiente explorado são os jogos Atari, onde os pixels que representam o estado corrente s são passados como dados de entrada para rede neural. Cada neurônio da camada de saída corresponde a uma provável

ação a_i a ser executada de acordo com o estado corrente s, sendo a avaliação produzida em cada neurônio representa o valor Q, ou seja, a avaliação dos pares de estado-ação (s, a).

$$Y_t^{DQN} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t) \tag{2}$$

O algoritmo DQN atualiza os valores Q da rede neural através da Equação (2), onde R_{t+1} é a recompensa recebida no instante de tempo $t + 1$ e γ é o valor de desconto. O termo $\max_a Q(S_{t+1}, a; \theta_t)$ representa a estimativa do maior valor Q, executando a ação a no estado S_{t+1}, o parâmetro θ representa os pesos da rede neural no referido instante de tempo. No DQN tanto a estimativa quanto a avaliação da ação para determinado estado é feita pela mesma rede neural.

2.6. Double Deep Q-Network

Na função alvo utilizada por [Mnih et al. 2015] a estimativa do valor Q e a escolha da ação são feitas pelo mesmo conjunto de parâmetros θ, que segundo [Van Hasselt et al. 2016] essa abordagem leva à superestimação dos valores ótimos Q. Como forma de minimizar esse problema [Van Hasselt et al. 2016] propôs desacoplar essas etapas, conforme mostrado na Equação (3)

$$Y_t^{DoubleDQN} \equiv R_{t+1} + \gamma Q(S_{t+1}, argmaxQ(S_{t+1}, a, \theta_t); \theta_t^-) \tag{3}$$

Nesta nova abordagem utiliza-se duas redes com as mesmas configurações, onde a primeira representada por θ_t (denominada on-line) é utilizada para estimar a ação a no estado S_{t+1}. A segunda rede θ_t^- (rede alvo) é utilizada para avaliar a ação predita no estado S_{t+1}, ou seja, a estimativa do valor Q. A cada instante de tempo τ os parâmetros θ_t^- são atualizados com pesos de θ_t, onde a letra grega (τ) representa o intervalo de ações para atualização desses parâmetros, ou seja, a cada τ ações executadas no ambiente a rede alvo é atualizada com os pesos da rede on-line. Os detalhes do uso do DQN e DDQN como metodos de aprendizagem serão abordados na Seção 4.

3. Trabalhos Relacionados

Os jogos eletrônicos são uma importante ferramenta para análise de desempenho dos algoritmos de RL, dentre os trabalhos desenvolvidos nesse cenário é fundamental citar [Mnih et al. 2015], onde o algoritmo DQN foi apresentado como uma notável ferramenta de aprendizagem para essa classe de ambientes. No referido trabalho, as imagens dos jogos do clássico videogame Atari 2600 são passadas como dados de entrada para o agente. O desempenho alcançado por esses agentes superaram a performance de jogadores humanos em diversos jogos.

A partir do trabalho de [Mnih et al. 2015], diversos outros trabalhos foram desenvolvidos. Na abordagem proposta por [Haddad et al. 2021] os algoritmos QL e DQN são utilizados na resolução dos jogos Basic e GridWorld através da plataforma Unity. [Montalvao 2021] propõe a combinação do DQN e do método Deep Deterministic Policy Gradient para resolução de uma variação do jogo da velha, onde as ações dos agentes são divididas em partes numéricas e categóricas.

A utilização do Mundo de Wumpus como ambiente de validação de algoritmos é abordado em diversos trabalhos. [Karanik and Gramajo 2010] fazem uso do mundo

de wumpus para análise de desempenho do algoritmo Sarsa, tendo como entrada ações combinadas. [Araújo et al. 2019] aplica algoritmo genético como mecanismo de aprendizagem na resolução do mundo de wumpus.

4. Materiais e Métodos

Nesta seção serão discutidos os detalhes da modelagem do mundo de wumpus, segundos os padrões de ambientes da plataforma Gym. Da mesma maneira, será detalhado as ferramentas e métodos utilizados para desenvolvimento dos agentes baseados nos algoritmos QL, DQN e DDQN.

4.1. Mundo de Wumpus

O mundo de Wumpus consiste em uma caverna constituída por várias salas, onde esses compartimentos são interligados através de passagens. Em algum lugar neste cenário encontra-se o terrível monstro Wumpus que é capaz de devorar qualquer guerreiro que adentrar na sala em que está localizado. Esse monstro pode ser atingido pelo agente através da única flecha que ele carrega. Dentro desta caverna algumas salas possuem poços sem fundo que emitem brisa para as salas vizinhas. A única motivação para explorar esta caverna é a possibilidade de encontrar um monte de ouro [Russel and Norvig 2010].

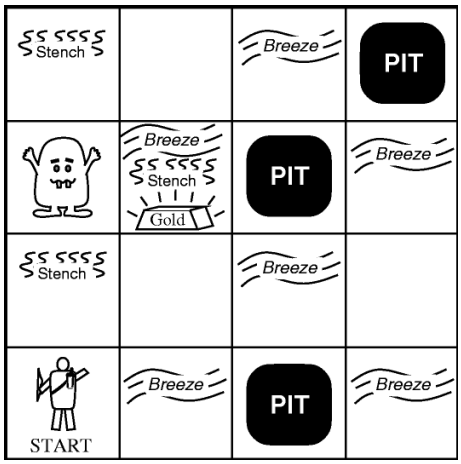

Figura 4. Mundo de Wumpus
Fonte: [Russel and Norvig 2010].

- **Estados:** Os estados desse ambiente são representados através de um vetor contendo a posição do agente no tabuleiro, direção em que está olhando (no início do treinamento está olhando para a direita), sensação de brisa, sensação de odor, sensação de brilho, quantidade de flecha e indicação binária se o wumpus está morto ou vivo.
- **Ações:** Este ambiente contém um espaço de ações discretas, sendo 5 as possibilidades: Ir para frente, Virar 90º à direita, Virar 90º à esquerda, Agarrar e Atirar.
- **Recompensas:** Durante a exploração deste cenário o agente recebe algumas punições e recompensas, sendo elas: -1 para cada ação executada, -100 caso seja morto pelo Wumpus ou se cair em algum dos poços, 100 por matar o monstro, 500 por pegar o ouro e 1000 caso saia da caverna de posse do ouro.Vale ressaltar que no início do treinamento é atribuído ao agente o valor de recompensa igual a zero.

Além das dimensões padrões desse ambiente (originalmente uma matriz com dimensões 4x4), este trabalho propõe outros 3 cenários, sendo eles 8x8, 10x10 e 15x15. Em cada um dos referidos cenários a quantidade de poços presente no cenário é dada por N - 1, onde N é o número de linhas do ambiente.

4.2. Q-Learning

Nesta seção serão abordados os detalhes de implementação e os métodos utilizados nesse algoritmo.

4.2.1. Pseudocódigo

O Pseudocódigo 1 ilustra o processo de aprendizagem dos agentes baseados no algoritmo Q-Learning. As principais etapas no processo de interação desse algoritmo concentram-se nos passos: Escolha da ação, execução da ação e atualização da tabela Q.

Algorithm 1 Algoritmo Q-Learning

Inicializar a tabela Q arbitariamente.
for cada episodio **do**
    Inicializar o estado inicial s
    for cada passo **do**
        Escolher uma ação a usando a politica derivada de Q (e.g., ϵ-greedy)
        Executar a ação a, observar r, $s^{'}$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
        $s \leftarrow s^{'}$
        if estado s é terminal **then**
            break
        end if
    end for
end for

4.2.2. Escolha da Ação

Nessa etapa o agente escolhe uma ação baseada na política epsilon-greedy [Morales 2020]. Essa política consiste em escolher uma ação de acordo com o valor de epsilon, onde no início do treinamento é inicializado com o valor 1 e que ao longo dos episódios esse valor é decrementado segundo o método adotado (neste trabalho foi utilizado a técnica de decaimento exponencial). Durante a escolha da ação um valor aleatório no intervalo $\{0; 1\}$ é gerado, caso esse valor seja maior que epsilon, a ação é escolhida seguindo os valores presentes na tabela Q, caso contrário o agente toma uma ação aleatória. Esse processo ocorre até que o epsilon seja igual ao valor mínimo estabelecido. Neste artigo foi adotado o valor mínimo igual a 0.01, ou seja, o agente passará a utilizar os valores da tabela Q com 99% de chance e apenas 1% os valores aleatórios. Essa abordagem é adotada para que o agente tenha a possibilidade de explorar algumas ações que possam resultar em maior recompensa. O Pseudocódigo 2 ilustra o funcionamento da política epsilon-greedy.

Algorithm 2 Epsilon-greedy

Seleciona-Ação (Q, s, ϵ) {
n $\leftarrow$ valor aleatório entre 0 e 1
if $n < \epsilon$ **then**
    $a \leftarrow$ ação aleatória do espaço de ações
else
    $a \leftarrow \max Q(s, .)$
end if
}

4.2.3. Execução da ação e Atualização da tabela Q

Ao executar a ação escolhida na etapa anterior, o agente recebe uma recompensa r e um novo estado $s^{'}$. Para treinamento dos agentes baseados no algoritmo Q-Learning os estados do ambiente são representados por um valor inteiro, calculado pela Equação (4) [Gym 2022].

$$estado\_atual \leftarrow linha\_atual * N + coluna\_atual \tag{4}$$

A partir do estado do ambiente os valores presente na matriz Q são atualizados através da equação de Bellman. Por fim, o estado s é atualizado com $s^{'}$ e caso esse estado seja terminal o episódio é finalizado, caso contrário as etapas descritas anteriormente são repetidas.

Tabela 1. Tabela de Hiperparametros Q-Learning

Hiperparametro	**Valor**
Fator de desconto (γ)	0.95
Taxa de Aprendizagem (α)	0.83
Valor inicial de epsilon (ϵ)	1.0
Valor minimo de epsilon (ϵ_{min})	0.01

Fonte: Autoria própria.

A Tabela 1 apresenta os hiperparâmetros utilizados para treinamento do Q-Learning. Os parâmetros fator de desconto (γ), taxa de aprendizagem (α) e valor mínimo de epsilon (ϵ_{min}) foram definidos de forma empírica. O valor inicial de epsilon é sempre inicializado com o valor 1, pois está relacionado a probabilidade de exploração do ambiente, onde inicialmente o agente possui total incentivo a exploração e no decorrer dos episódios esse número é reduzido até o valor mínimo [Morales 2020].

4.3. Deep Q-Network

Nesta seção será abordado o processo de aprendizagem do agente DQN e os mecanismos utilizados para seu desenvolvimento.

Conforme discutido na Seção 2.5, a implementação da função ação-valor deste algoritmo é dada através de uma rede neural profunda. Desta forma, no presente trabalho

os dados de entrada dessa rede são os metadados descritos no tópico **estados** da Seção 4.1. Cada neurônio da camada de saída corresponde a uma possível ação do agente. Portanto, o algoritmo DQN é utilizado para treinar a rede neural que compõem a estrutura do agente, de modo que ao longo do processo de treinamento aumente progressivamente a precisão da avaliação das ações de acordo com o estado recebido.

Além da rede neural, o conceito denominado **Experience Replay** é de suma importância no processo de aprendizagem desse algoritmo. Este mecanismo consiste em armazenar as experiências do agente ao longo do treinamento. A cada passo dado no ambiente a tupla (s_t, a_t, r_t, s_{t+1}) é armazenada em um lote de tamanho D, de forma que ao alcançar uma quantidade mínima de experiências, são usadas a fim de ajustar os pesos da rede neural e como consequência ajustar as avaliações dos estados do ambiente.

Figura 5. Representação do estado do ambiente

- **Posição do Agente:** Definida através da equação (4).
- **Direção:** Direção em que o agente está olhando.
- **Brisa, Fedor, Brilho, Quantidade de flechas e Wumpus:** Indicação binária, sendo 0 para ausência e 1 para presença.

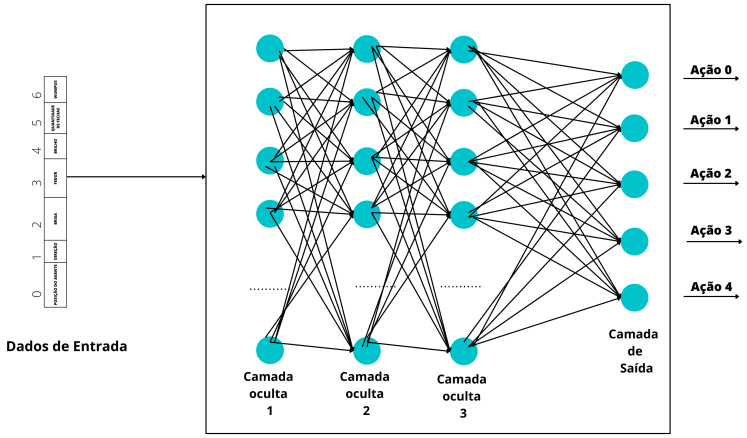

Figura 6. Estrutura da rede neural do agente DQN
Fonte: Autoria própria.

4.3.1. Pseudocódigo

O Pseudocódigo 3 ilustra a dinâmica do algoritmo DQN. A cada passo de tempo t, o agente escolhe uma ação a através da política epsilon-greedy, executa esta ação no ambiente e recebe como resposta uma recompensa r e um novo estado s_{t+1}. As experiências obtidas nesse instante de tempo são armazenadas em um lote D, seguido pela atualização da rede neural através da técnica Experience Replay. É importante ressaltar que o termo $\max Q(s_{t+1}, a^{'}; \theta)$ refere-se ao maior valor de saída gerado pela rede, sendo assim, a avaliação da melhor ação disponível para o referido estado do mini-lote escolhido.

Algorithm 3 Algoritmo Deep Q-Network

Inicializar replay memory D com capacidade N
Rede neural Q inicializada com parametros θ aleatórios
for cada episodio **do**
    Inicializar o estado inicial s_t
    for $t = 1, T$ **do**
        Escolher uma ação a_t usando a politica derivada de Q (e.g., ϵ-greedy)
        Executar a ação a_t, observar r_t, s_{t+1}
        Armazenar a tupla de transições (s_t, a_t, r_t, s_{t+1}) na memoria D
        Selecionar aleatóriamente um mini-lote de D contendo K experiências (s_t, a_t, r_t, s_{t+1}). (Experience Replay)
        for cada experiência no mini-lote **do**
            if estado s_t é terminal **then**
                y $\leftarrow$ r
            else
                $y \leftarrow r + \gamma \max Q(s_{t+1}, a^{'}; \theta)$
            end if
            Atualizar os parametros do modelo com $(y - Q(s, a))^2$
        end for
        $s_t \leftarrow s_{t+1}$
    end for
end for

4.3.2. Agente DQN

A rede neural utilizada pelo agente DQN é constituída de 5 camadas, sendo 1 camada de entrada, 3 camadas ocultas e uma camada de saída. A Tabela 2 resume os parâmetros utilizados na construção da rede neural. A Tabela 3 apresenta alguns hiperparâmetros utilizados pelo algoritmo.

Tabela 2. Tabela de parâmetros da rede neural do agente DQN

Camada	Dimensão	Função de ativação
Entrada	7x256	Tanh
Oculta 1	256x256	Tanh
Oculta 2	256x256	Tanh
Oculta 3	256x256	Tanh
Saída	256x5	Tanh

Fonte: Autoria própria.

Tabela 3. Tabela de Hiperparametros DQN

Hiperparametro	Valor
Tamanho da replay memory D	50.000
Tamanho do mini-lote N	64
Fator de desconto (γ)	0.95
Taxa de Aprendizagem (α)	0.83
Valor inicial de epsilon (ϵ)	1.0
Valor minimo de epsilon (ϵ_{min})	0.01

Fonte: Autoria própria.

4.4. Double Deep Q-Network

O algoritmo Double Deep Q-Network tem seu mecanismo de aprendizagem similar ao DQN, sendo diferenciado apenas por utilizar duas redes neurais denominadas online e target network. A cada K interações os pesos da rede online são copiados para target com intuito de manter a instabilidade no processo de aprendizagem do agente. O algoritmo 4 ilustra a dinâmica de aprendizagem dos agentes baseados no DDQN. A Tabela 4 contém os hiperparâmetros utilizados para treinamento do agente.

Tabela 4. Tabela de Hiperparametros DDQN

Hiperparametro	Valor
Tamanho da replay memory D	50.000
Tamanho do mini-lote N	64
Fator de desconto (γ)	0.95
Taxa de Aprendizagem (α)	0.83
Valor inicial de epsilon (ϵ)	1.0
Valor minimo de epsilon (ϵ_{min})	0.01
Atualização da target network	25

Fonte: Autoria própria.

Os parâmetros que compõem a rede neural dos agentes DQN e DDQN foram definidos de forma empírica, assim como os hiperparâmetros fator de desconto (γ), taxa de aprendizagem (α), atualização da target network e valor mínimo de epsilon (ϵ_{min}). Os hiperparâmetros tamanho da replay memory D e tamanho do mini-lote N foram definidos conforme a abordagem de [Morales 2020].

Algorithm 4 Algoritmo Double Deep Q-Network

Inicializar replay memory D com capacidade N
Rede neural Q inicializada com parametros θ aleatórios
Rede neural target $\hat{Q}$ inicializada com parametros $\theta^- = \theta$
for cada episodio **do**
    Inicializar o estado inicial s_t
    for $t = 1, T$ **do**
        Escolher uma ação a_t usando a politica derivada de Q (e.g., ϵ-greedy)
        Executar a ação a_t, observar r_t, s_{t+1}
        Armazenar a tupla de transições (s_t, a_t, r_t, s_{t+1}) na memoria D
        Selecionar aleatóriamente um mini-lote de D contendo K experiências (s_t, a_t, r_t, s_{t+1}). (Experience Replay)
        for cada experiência no mini-lote **do**
            if estado s_t é terminal **then**
                y $\leftarrow$ r
            else
                $y \leftarrow r + \gamma \max \hat{Q}(s_{t+1}, a^{'}; \theta^-)$
            end if
            Atualizar os parametros do modelo com $(y - Q(s, a; \theta))^2$
        end for
        A cada K interações $Q(s, a; \theta)$ = $\hat{Q}(s, a; \theta^-)$
        $s_t \leftarrow s_{t+1}$
    end for
end for

Para desenvolvimento dos agentes e do ambiente mundo de Wumpus foi utilizado a linguagem de programação Python, juntamente com as bibliotecas numpy, keras, tensorflow e gym. A avaliação dos agentes se deu através da execução de 20.000 episódios em cada um dos respectivos cenários. As métricas obtidas durante o treinamento foram armazenadas em um arquivo csv que contém: episódio, recompensa, uma flag para o ouro, quantidade de passos, flag indicando se o agente matou o wumpus e uma flag indicando se o agente saiu da caverna com o ouro. Para treinamento dos agentes foi realizada uma única execução em cada uma das respectivas configurações de ambientes, sendo essas realizadas em uma máquina com processador i5-8265 U, 8GB de memória RAM e placa gráfica dedicada Nvidia Geforce mx110 2GB.

5. Resultados e Discussões

A análise de desempenho dos agentes foi dada através da execução de 20.000 episódios nas configurações 4x4, 8x8, 10x10 e 15x15. Em cada cenário a quantidade de ações no ambiente foram limitadas respectivamente por: 100, 150, 200 e 250. As figuras 7 e 8 ilustram a complexidade de cada um dos referidos cenários. Para finalidade de entendimento, as configurações de ambientes 4x4, 8x8, 10x10 e 15x15 serão citados nesta seção como ambiente 1, ambiente 2, ambiente 3 e ambiente 4.

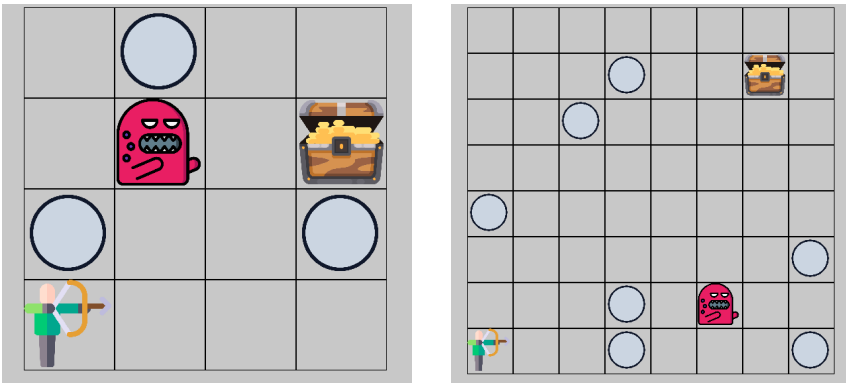

Figura 7. Ambientes 4x4 e 8x8
Fonte: Autoria própria.

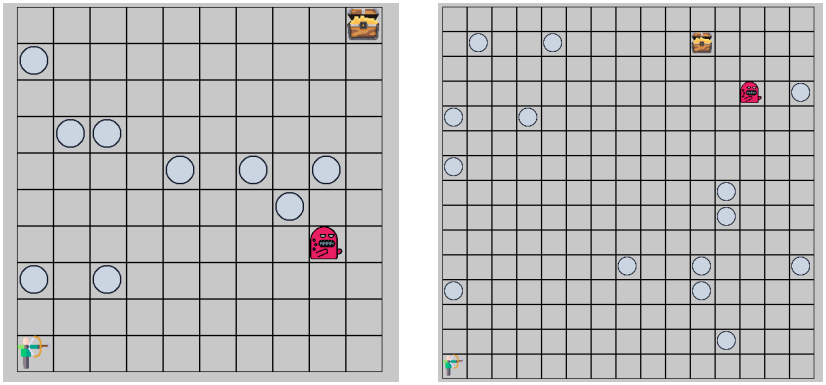

Figura 8. Ambientes 10x10 e 15x15
Fonte: Autoria própria.

A Tabela 5 demonstra as recompensas obtidas pelos agentes em cada configuração de ambiente. As linhas apresentam os algoritmos e seus resultados, as colunas apresentam as variáveis analisadas. Em maioria dos casos o **valor mínimo** apresentou-se similar para todos os algoritmos, exceto para o ambiente 4. Para a variável **valor máximo** o DDQN obteve maior valor nos ambiente 2 e ambiente 3, no ambiente 1 os algoritmos DDQN e DQN obtiveram os mesmos resultados. Por fim, o algoritmo QL obteve a maior recompensa no ambiente 4. O algoritmo DQN apresentou melhor desempenho para a variável **valores médio** no ambiente 1 e ambiente 4. Os agentes QL e DDQN apresentaram melhor valor médio nos ambientes 2 e 3 respectivamente. O jogo mundo de Wumpus tem como objetivo principal encontrar o ouro e sair da caverna sem cair nos poços ou ser morto pelo monstro. Esse ambiente apresenta dois objetivos secundários: somente pegar o ouro e matar o wumpus. O DQN obteve a maior taxa de vitória para o objetivo principal no ambiente 1, conseguindo cumprir esse objetivo 22,96%. No ambiente 3 o agente QL obteve a maior porcentagem de vitória, alcançando 0.175%, ressalta-se que nos ambientes 2 e 4 nenhum dos agentes conseguiu atingir o objetivo principal. A Tabela 6 demonstra o objetivo principal e secundário alcançados pelos agentes.

Tabela 5. Valores minimo, maximo, media e desvio padrão de recompensas

Ambiente	Agente	Valor Minimo	Valor Maximo	Media	Desvio Padrão
4x4	QL	302	1407	362.4384	72.5799
4x4	DQN	302	1585	955.9765	571.4532
4x4	DDQN	302	1585	861.6854	557.6125
8x8	QL	252	351	330.1772	35.1199
8x8	DQN	252	380	345.5036	18.9572
8x8	DDQN	252	381	346.6264	17.0118
10x10	QL	202	1394	334.3647	199.4354
10x10	DQN	202	1406	301.1405	30.9973
10x10	DDQN	202	1421	301.2272	38.1289
15x15	QL	161	303	244.8243	26.2779
15x15	DQN	165	251	245.0	20.2978
15x15	DDQN	152	281	240.4737	31.0526

Fonte: Autoria própria.

Tabela 6. Analise de desempenho dos agentes

Ambiente	Agente	%Matou o Wumpus	%Pegou o ouro	%Pegou o ouro e saiu da caverna
4x4	QL	0.645	1.38	0.005
4x4	DQN	19.93	45.925	22.96
4x4	DDQN	19.325	45.35	18.73
8x8	QL	0.0	0.395	0.0
8x8	DQN	0.03	34.81	0.0
8x8	DDQN	0.035	37.215	0.0
10x10	QL	0.145	4.675	0.175
10x10	DQN	0.385	26.08	0.02
10x10	DDQN	0.035	28.67	0.035
15x15	QL	0.0	0.37	0.0
15x15	DQN	0.0	0.105	0.0
15x15	DDQN	0.0	0.095	0.0

Fonte: Autoria própria.

As Figuras 9 e 10 ilustram a média móvel de recompensa dos agentes ao longo dos episódios, onde cada amostra de dados foi calculada a cada 100 episódios. Observa-se que os algoritmos DQN e DDQN mostraram desempenho próximos ao longo das execuções, tendo destaque para o DDQN que apresenta mais estabilidade, desta maneira, confirmando de forma prática o propósito de seu desenvolvimento. Os agentes baseados em DQN e DDQN mostraram maior valor de recompensa ao longo dos episódios para o ambiente 1, 2 e 3. Apenas no ambiente 4 que todos agentes apresentaram comportamento similar ao longo dos episódios.

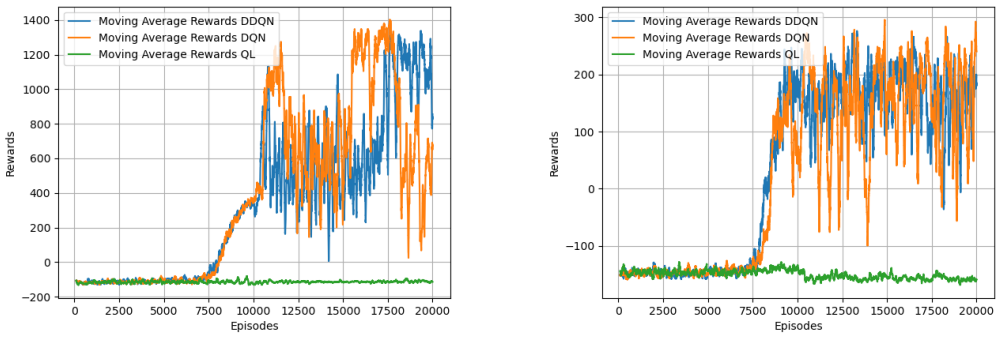

Figura 9. Média Movel de recompensa ao longo dos episodios para o ambiente 1 e 2

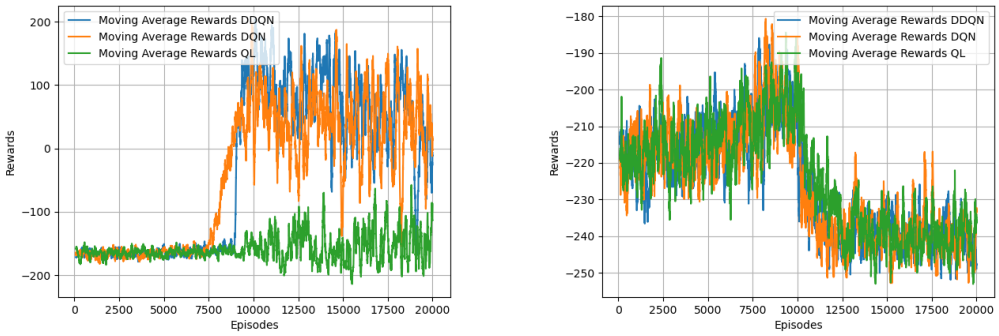

Figura 10. Média Movel de recompensa ao longo dos episodios para o ambiente 3 e 4

6. Conclusão e Trabalhos Futuros

Este artigo propôs implementar agentes baseados em Aprendizagem por Reforço com o objetivo de solucionar o clássico jogo mundo de Wumpus, tendo foco no algoritmo Q-Learning e suas versões baseadas em redes neurais. Através dos experimentos realizados foi constatado que os agentes que utilizam redes neurais em seu mecanismo de aprendizagem apresentam melhor desempenho em maioria dos cenários. Também foi observado que para cenários onde o objetivo está distante dos obstáculos, os agentes DDQN e DQN apresentaram resultados inferiores ao método tabular Q-Learning, pois para esse tipo de problema a rede neural gera valores similares de ações para esses estados. O que não ocorre para o método tabular, pois possui valores mapeados para cada estado do ambiente, o que levou a melhor desempenho nesta configuração.

O mundo de Wumpus apresentou-se um importante cenário para validação de agentes baseados em Aprendizagem por Reforço. Desta maneira, como trabalhos futuros alguns hiperparâmetros podem ser modificados com a finalidade de melhorar o desempenho dos agentes na resolução desse jogo, como a utilização do método proposto por [He et al. 2015] para inicialização dos pesos da rede neural, assim como a experimentação de diferentes valores máximo e mínimo de lote no método experience replay. Com a finalidade de melhorar o desempenho do agente DDQN neste ambiente, objetiva-se variar a

quantidade de passos mínimos para atualização da rede target com os pesos da rede online. Através da observação desses valores espera-se que seja estabelecida uma relação entre o tamanho do ambiente e quantidade mínima de passos para atualização desses parâmetros.

O presente artigo realizou experimentos utilizando uma quantidade fixa de episódios e estabeleceu uma quantidade máxima de passos em cada configuração de ambiente. Como proposta de melhoria é sugerido a experimentação de diferentes quantidades de ações e episódios em cada um dos cenários propostos. Desta maneira, teremos a quantidade mínima de ações e episódios para solucionar de maneira eficiente o problema apresentado.

7. Agradecimentos

Agradeço por meio desta seção aos meus orientadores e amigos Otávio Noura e Natalia Araújo por todo apoio e conhecimento compartilhado para a pesquisa e desenvolvimento do presente trabalho, assim como o laboratório de pesquisa MEC^2A e seus integrantes. Minha imensa gratidão a SAEST-UFPA por todo apoio financeiro recebido através dos programas permanência e moradia, os quais foram fundamentais para prosseguimento e dedicação integral a minha formação.

Referências

Amber (2019). Coursera's machine learning notes – week4, non-linear hypothesis and neural network (nn). urlhttps://medium.com/@qempsil0914/courseras-machine-learning-notes-week4-non-linear-hypothesis-and-artificial-neural-network-a3c4bbdecb41.

Araújo, N. F., Teixeira, O. N., and dos Santos, A. A. A. (2019). *Algoritmo Genético como Mecanismo de Aprendizagem do Agente na Resolução do Mundo de Wumpus*, volume 1. UFPA.

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*.

Brown, B. and Zai, A. (2020). *Deep Reinforcement Learning in Action*. Manning Publications Co, 1th edition.

Damião, M. A., Caçador, R. M. C., and Lima, S. M. B. (2014). *PRINCÍPIOS E ASPECTOS SOBRE AGENTES INTELIGENTES*, volume 1. Revista Eletrônica da Faculdade Metodista Granbery.

Dias, A. V. F. (2019). *Estudo de Aprendizagem por Reforço em Jogo Tower Defense*. PhD thesis, Instituto Politecnico de Braganca (Portugal).

Gym (2022). frozen$_l ake.urlhttps://github.com/openai/gym/blob/master/gym/envs/toy_text/froze$

Haddad, G. P., Julia, R. M. S., and Faria, M. P. P. (2021). Técnicas de aprendizado por reforço aplicadas em jogos eletrônicos na plataforma geral do unity. In *Anais do XVIII Encontro Nacional de Inteligência Artificial e Computacional*, pages 571–582. SBC.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Karanik, M. J. and Gramajo, S. D. (2010). Using combination of actions in reinforcement learning. *Journal of Computer Science & Technology*, 10.

Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

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(7540):529–533.

Montalvao, T. C. G. (2021). *APLICANDO MODELOS DE APRENDIZADO POR REFORÇO PROFUNDO EM UM JOGO ADVERSARIO*, volume 1. UFRJ.

Morales, M. (2020). *grokking Deep Reinforcement Learning*. Manning Publications Co, 1th edition.

Poole, D. I., Goebel, R. G., and Mackworth, A. K. (1998). *Computational intelligence*, volume 1. Oxford University Press New York.

Russel, S. and Norvig, P. (2010). *Inteligência artificial*. Prentice Hall, 3th edition.

Sewak, M. (2019). *Deep Reinforcement Learning Frontiers of Artificial Intelligence 123*. Springer Nature Singapore Pte Ltd, 1th edition.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Sutton, R. S. and Barto, A. G. (2020). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning, 2th edition.

Tesauro, G. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Walkins, C. (1989). *Leaming from Delayed Rewards*. PhD thesis, PhD thesis. King's College, University of Cambridge. UK.