TRABALHO DE GRADUAÇÃO

Aprendizagem por reforço utilizando Q-Learning e redes neurais artificiais em jogos eletrônicos

Ícaro da Costa Mota

Brasília, Julho de 2018

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Aprendizagem por reforço utilizando Q-Learning e redes neurais artificiais em jogos eletrônicos

Ícaro da Costa Mota

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Dr. Marcus Vinicius Lamar, CIC/UnB _____
Orientador

Profa. Dra. Carla Denise Castanho, CIC/UnB _____
Examinador interno

Prof. Dr. Guilherme Novaes Ramos, CIC/UnB _____
Examinador interno

Brasília, Julho de 2018

FICHA CATALOGRÁFICA

ÍCARO, DA COSTA MOTA
Aprendizagem por reforço utilizando Q-Learning e redes neurais artificiais em jogos eletrônicos

[Distrito Federal] 2018.

vi, 50p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2018). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Inteligência Artificial 2. Aprendizado por reforço 3. Aprendizagem Profunda
4. Redes Neurais 5. OpenAI Gym 6. ROS

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

MOTA, ICARO DA COSTA, (2018). Aprendizagem por reforço utilizando Q-Learning e redes neurais artificiais em jogos eletrônicos. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°4$, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 50p.

CESSÃO DE DIREITOS

AUTOR: Ícaro da Costa Mota

TÍTULO DO TRABALHO DE GRADUAÇÃO: Aprendizagem por reforço utilizando Q-Learning e redes neurais artificiais em jogos eletrônicos.

GRAU: Engenheiro ANO: 2018

Ícaro da Costa Mota

SQS 209 Bloco B, Asa Sul.

70272-520 Brasília – DF – Brasil.

Dedicatória

Dedico esse trabalho aos meus pais, por todo o suporte e apoio que me deram durante a minha graduação e toda minha vida

Ícaro da Costa Mota

Agradecimentos

A cada dia aprendo mais a valorizar as pessoas queridas a mim. Só tenho a agradecer aos meus pais por todo o apoio e incentivo que me deram ao longo destes duros anos, e por toda a confiança que sempre depositaram em mim.

Agradeço ao meu orientador, Prof. Dr. Marcus Vinicius Lamar, pela excelente orientação e pela paciência de orientar um aluno que mudou de estado duas vezes ao longo do trabalho.

Ao meu grande companheiro Naves, que me fez companhia durante toda esta jornada, aprendendo e crescendo junto comigo. Obrigado por todas as tardes, noites e madrugadas de estudos, jogatinas e séries. Não teria sido o mesmo sem você.

Muito obrigado aos meus companheiros de turma e de bar, Bruno, Moisés, Brandon, Zago, Saulo, Guilherme e Lucas, que alegraram muitas das minhas noites quando estive precisando, assim como meus amigos de jogatina, Matheus, Rafael e Yuji, que tornaram esta jornada muito mais divertida.

Agradeço o meu querido clã: Ivy, Tcholas, Arthur, Matheus, Jorge, Anderson e Alison. Mais que amigos, irmãos. Obrigado também a Dalila e Paula por ouvir todos meus desabafos e sempre me apoiarem.

A UnBall: Sousa, Vieira, Vitor, Samuel, Sofia, Beduin, Fabrícia, Kelson e Anderson. Guardo com muito carinho todas as noites mal dormidas fazendo robôs com vocês.

Agradeço também a Iza por ouvir todos os meus desabafos e ter se tornado uma grande amiga numa fase nova da minha vida, longe de casa. Também agradeço ao Guiga, amigo e companheiro, que teve muita paciência em nossos projetos nesta reta final.

Muito obrigado aos meus grandes mentores escoteiros, que me ajudaram a ser quem sou hoje: Márcio, André, Reys, Glen, De Paula, Felipe, Guilherme e Rosana. Obrigado por toda a dedicação, vocês sempre foram uma fonte de inspiração.

Obrigado aos queridos amigos que fiz em outro país, Arthur, Sady, Zé, Rodrigo, Letícia e Mariana, por todos os momentos de descontração. A todos do mexe-mexe, amigos tão queridos que hoje consigo apenas descrevê-los como família. Obrigado Ana, que acompanhou uma boa parte da minha jornada.

Agradeço também os meus amigos que moram comigo, Bacic, Alessandra e Lucas, que pacientemente me aguardaram para jogar tabuleiro enquanto escrevia este trabalho.

Obrigado, principalmente, por acreditarem em mim.

Ícaro da Costa Mota

RESUMO

Em aprendizagem por reforço, um agente deve aprender com suas experiências ao interagir com o ambiente no qual se encontra. Este trabalho propõe um sistema de aprendizagem profunda com o algoritmo *Deep Q-Learning* para ensinar um agente genérico a jogar jogos eletrônicos distintos, utilizando redes neurais artificiais para estimar o valor de executar-se uma ação no estado no qual o agente se encontra. O trabalho foi desenvolvido utilizando a ferramenta ROS para gerenciar a comunicação entre os sistemas. Aplicou–se as técnicas desenvolvidos nos jogos *Enduro*, *Ms. Pacman*, *Breakout* e *Pong*, emulados pela ferramenta OpenAI Gym, desenvolvida especificamente para auxiliar em trabalhos de aprendizagem por reforço. O agente demonstrou aprender no jogo *Ms. Pacman*, porém a modelagem do estado foi insuficiente nos jogos *Breakout* e *Pong*, resultando na inabilidade do agente em selecionar a melhor ação no estado em que se encontrava. No jogo *Enduro*, o agente não conseguiu interagir o suficiente com o ambiente para obter recompensas e aprender a maximiza-las.

Palavras Chave: Inteligência Artificial, aprendizado por reforço, aprendizagem profunda, redes neurais, OpenAI Gym, ROS

ABSTRACT

In reinforcement learning, an agent must learn from past experiences by interacting with its environment. This work proposes a deep learning system with the Deep Q-Learning algorithm to teach a generic agent to play distinct electronic games, by using artificial neural networks to estimate the value of executing an action in the state the agent finds itself. The work was developed by using the ROS resource to manage the communication between systems. The developed techniques were applied to the games Enduro, Ms. Pacman, Breakout, and Pong, emulated by the OpenAI Gym toolkit, developed specifically to aid in reinforced learning projects. The agent has shown to learn in the Ms. Pacman environment, but the state representation was insufficient in the games Breakout and Pong, resulting in the agent's inability to select the best action in its current state. In the game Enduro, the agent did not interact enough with its environment to obtain rewards and learn to maximize them.

Keywords: Artificial Intelligence, reinforced learning, deep learning, neural networks, OpenAI Gym, ROS

SUMÁRIO

1 Introdução 1

1.1 Inteligência Artificial 1
1.2 Aprendizagem 2
1.2.1 Aprendizagem em jogos 2
1.3 Agente 3
1.4 Objetivo Geral 3
1.5 Objetivos Específicos 4
1.6 Hipótese 4
1.7 Apresentação do manuscrito 4

2 Referencial Teórico 5

2.1 Aprendizagem por Reforço 5
2.1.1 Estados 6
2.1.2 Recompensa 7
2.1.3 Política 8
2.1.4 Função de Valor 9
2.1.5 Processo de Decisão de Markov 11
2.1.6 Aprendizado com diferença temporal 15
2.2 Q-Learning 15
2.3 Redes Neurais 17
2.4 Aprendizagem Profunda 18
2.5 Deep Q-Learning 19

3 Sistema Proposto 21

3.1 OpenAI 21
3.2 ROS 22
3.2.1 Mensagens utilizadas 23
3.3 Arquitetura 23
3.4 Estado do agente 25
3.5 Função Q 26
3.6 Ambientes 26

4 Experimentos 29

4.1 Ambientes de Testes 29
4.2 Roteiro dos testes 30
4.3 Resultados 31
4.3.1 Enduro 32
4.3.2 Ms. Pacman 34
4.3.3 Breakout 36
4.3.4 Pong 38
4.4 Análise Geral 40
4.4.1 Comparação dos resultados 41

5 Conclusões 43
5.1 Trabalhos Futuros 44

REFERÊNCIAS BIBLIOGRÁFICAS 45

Anexos 48

I Descrição do conteúdo do CD 49

II Programas utilizados 50

LISTA DE FIGURAS

2.1 Loop de interação do agente e ambiente (Fonte: [1]) 6
2.2 Exemplo de uma cadeia de Markov 11
2.3 Exemplo de uma cadeia de Markov com recompensas 12
2.4 Processo de Decisão de Markov 14
2.5 Exemplo de um neurônio 17
2.6 Rede Neural Artificial com 1 camada intermediária e múltiplas saídas (Fonte: [2]) ... 18
2.7 Rede Neural Artificial Profunda (Fonte: [2]) 19
2.8 Possíveis abordagens de RNAs em Deep Q-Learning (Fonte: [3]) 20

3.1 Exemplo de nós, tópicos e mensagens no ROS 22
3.2 Arquitetura do Agente 24
3.3 Arquitetura do sistema de aprendizagem 25
3.4 RNA utilizada com entradas S_t^a e a_t e saída Q(s,a) 27
3.5 Exemplos de ambientes do OpenAI Gym 27

4.1 Capturas de tela do jogo Enduro 32
4.2 Evolução do agente no jogo Enduro no Experimento 1 33
4.3 Capturas de tela do jogo Ms. Pacman 35
4.4 Experimentos do Ms. Pacman 36
4.5 Capturas de tela do jogo Breakout 37
4.6 Experimentos do Breakout 37
4.7 Capturas de tela do jogo Pong 39
4.8 Experimentos do Pong 40

LISTA DE TABELAS

4.1 Parâmetros dos experimentos 30
4.2 Resultado dos experimentos 41

LISTA DE SÍMBOLOS

Símbolos Latinos

A	Ação
E	Expectativa
G	Máxima recompensa esperada
H	História
O	Observação
Q	Função de valor do par estado ação
R	Recompensa
S	Estado
t	Instante de tempo
V	Função de valor do estado

Símbolos Gregos

α	Fator de aprendizagem
γ	Fator de desconto
σ	Função Sigmoide
π	Política
Δ	Variação
ϵ	Valor de probabilidade

Siglas

IA	Inteligência Artificial
MDP	*Markov Decision Process*
RL	*Reinforcement Learning*
RNA	Rede Neural Artificial
TD	*Temporal Difference*

Capítulo 1

Introdução

Desde o início da história, os seres humanos desenvolvem máquinas e ferramentas para auxiliá-los. Para isso, sempre desejou-se construir máquinas que atuassem independente da ação humana. Na mitologia Grega, acreditava-se que o deus Hefesto tivesse construído criados feitos de ouro para auxiliá-lo em seu trabalho [4].

A teoria de controle é o estudo matemático das propriedades de sistemas automatizados [5]. Surgiu como um ramo da engenharia mecânica, e era utilizada para controlar sistemas hidráulicos, térmicos, moinhos e eventualmente, máquinas a vapor.

Durante a segunda guerra mundial, Norbert Wiener desenvolveu um novo campo de estudo a partir da teoria de controle e estudos de biologia e neurociência, a cibernética [6]. A ideia é que, com o estudo da cibernética, máquinas pudessem se inspirar no comportamento de seres vivos para reproduzir comportamentos conforme àqueles encontrados na natureza. A cibernética combinava duas áreas até então inexistentes, a robótica e a inteligência artificial (IA).

1.1 Inteligência Artificial

A inteligência artificial surgiu oficialmente em 1956, em uma conferência realizada na Universidade de Dartmouth, nos Estados Unidos [5]. Hoje, a IA é reconhecida como o ramo da Ciência da Computação que se propõe a elaborar técnicas que permitam com que máquinas realizem tarefas de alta complexidade, simulando a capacidade humana de raciocinar.

O conceito de inteligência é amplo, e ainda não há um consenso do significado do termo entre os pesquisadores. De toda forma, a inteligência artificial é a inteligência apresentada por máquinas e computadores, em contrapartida à inteligência natural, apresentada por seres humanos e outros animais. Coloquialmente, o termo inteligência artificial é utilizado para caracterizar uma máquina de imita funções cognitivas associadas com o pensamento humano, tais como aprendizagem ou soluções de problemas [7].

A inteligência artificial é amplamente utilizada nos dias de hoje, até mesmo no cotidiano das pessoas. Assistentes pessoais digitais, como a Alexa da Amazon ou o novo assistente digital da

Google, são capazes de auxiliar no dia-a-dia, conversando com os usuários utilizando tecnologias de reconhecimento de fala, ligando eletrodomésticos e até agendando compromissos para o usuário por meio de ligações telefônicas.

Diversas áreas da indústria são diretamente influenciadas pelo desenvolvimento de inteligências artificias. Robôs autônomos utilizam tecnologias de inteligência artificial para aprender a locomover-se e a resolver tarefas. Um exemplo disso são carros autônomos, capazes de perceber o trânsito ao redor e se dirigirem sozinhos [8]. Mídias sociais utilizam inteligência artificial para identificar os gostos dos usuários e oferecer propagandas. A inteligência artificial também tem tido um impacto no campo da medicina, auxiliando em cirurgias e até mesmo a encontrar tratamentos corretos para câncer.

1.2 Aprendizagem

Dentro do campo de estudo da inteligência artificial, existe o campo da aprendizagem de máquina (do inglês, *Machine Learning*), que vem sido extensivamente estudado nos últimos anos. A aprendizagem de máquina surgiu do campo de reconhecimento de padrões e da teoria do aprendizado computacional. É definida como o campo de estudos que dá aos computadores a capacidade de aprender, sem serem explicitamente programados [9]. Um programa de computador pode aprender a tomar decisões, como veremos posteriormente neste trabalho, ou a reconhecer padrões e fazer previsões. Formalmente, um programa de computador é dito para aprender com a experiência E com a relação a alguma classe de tarefas T e medida de desempenho P, se o seu desempenho em tarefas em T, medida pelo P, melhora com a experiência E [10]. Em outros termos, entende-se aprendizagem como a capacidade do agente de melhorar seu desempenho utilizando experiências adquiridas ao longo do tempo [11].

Existem três grandes classificações de aprendizagem de máquina: aprendizagem supervisionada, aprendizagem não-supervisionada e aprendizagem por reforço [12]. Em aprendizagem supervisionada, apresenta-se ao programa desenvolvido um conjunto de entradas e saídas desejadas a partir desta entrada, para que este aprenda um padrão entre os dados fornecidos e seja capaz de obter respostas corretas para novos dados [13]. A aprendizagem não-supervisionada não fornece nenhum tipo de rótulo aos dados fornecidos ao programa, que deve encontrar sozinho alguma estrutura nos dados fornecidos para conseguir agrupá-los ou determinar a distribuição de probabilidade que gerou os dados [14]. A aprendizagem por reforço consiste em um agente interagindo com um ambiente dinâmico a fim de resolver um objetivo. Diferente dos casos de aprendizagem supervisionada, o programa não é fornecido com uma resposta correta, mas apenas com um sinal de realimentação, servindo de premiação ou punição pelas ações executadas [12].

1.2.1 Aprendizagem em jogos

Para desenvolver e avaliar algoritmos de aprendizagem de máquina, frequentemente se utiliza jogos eletrônicos como ferramenta de estudo, pois o ambiente eletrônico permite a validação do

cumprimento de regras sem a assistência de um agente humano, e o resultado de um agente pode ser medido através da pontuação e resultados obtidos no jogo. Além disso, a utilização de ambientes simulados tornam os experimentos mais baratos [15] do que utilizar robôs e máquinas no mundo físico.

Em 1950, Alan Turing escreveu o primeiro programa capazes de jogar Xadrez [16]. O problema do jogo de Xadrez vem desde então sendo abordado na área da computação, ao ponto de que os sistemas atuais podem desafiar campeões do esporte. Até poucos anos atrás, não havia uma inteligência artificial capaz de derrotar um jogador profissional no jogo de Go, devido à imensa árvore de possibilidades do jogo. Contudo, em 2016, a IA Alpha Go derrotou o campeão Lee Sedol, detentor de 18 títulos internacionais, 4 vezes em uma sequência de 5 jogos [17].

Mais recentemente, a equipe de pesquisa da plataforma OpenAI desenvolveu um time de jogadores do jogo Dota2 para desafiar jogadores profissionais do esporte virtual [18]. O jogo envolve a cooperação entre as cinco IAs que compõe o time, que através de um espaço gigantesco de estados do jogo e seleção de ações em um espaço contínuo de ações, deve destruir a base da equipe adversária. A ferramenta já mostrou ser capaz de vencer equipes amadoras.

Diversas atividades possuem tarefas comparáveis àquelas de jogos [19]. Através da utilização de jogos eletrônicos como plataforma de aprendizagem, espera-se expandir o campo da inteligência artificial de forma a desenvolver algoritmos que possam ser utilizados em tarefas mais complexas, até mesmo em sistemas que atuem no mundo físico.

1.3 Agente

Neste trabalho, implementou-se um agente computacional para aprender a jogar jogos eletrônicos distintos. Considera-se um agente algo que seja capaz de perceber o seu ambiente e agir sobre ele [11]. O agente também deve possuir um objetivo ou objetivos relacionados ao estado do ambiente. Um agente robótico percebe o ambiente através de sensores e atua utilizando efetuadores, enquanto um agente computacional existe apenas no mundo da simulação.

Ao desenvolver um agente, faz-se necessário definir a métrica a qual o agente é avaliado. Em ambientes de jogos digitais, é comum que se utilize a pontuação do jogo para avaliar o desempenho do agente.

1.4 Objetivo Geral

Criar um agente genérico capaz de, sem intervenção humana, aprender a jogar jogos eletrônicos distintos a partir de informações visuais da tela do jogo após ser treinado no ambiente do jogo selecionado. Para isso, considera-se que deve ser implementado especificamente para cada ambiente uma forma de aquisição da pontuação do jogo e uma forma de reiniciá-lo para que o treinamento seja executado de forma automática.

1.5 Objetivos Específicos

1. Escolher os ambientes
2. Realizar a comunicação assíncrona entre ambiente e o agente
3. Pré-processar as imagens obtidas de forma a diminuir o custo computacional do algoritmo de aprendizagem
4. Definir e treinar uma estrutura de rede neural para estimar o valor das ações possíveis
5. Implementar um algoritmo de aprendizagem por reforço que use a rede neural para estimar o valor das ações
6. Avaliar a eficácia das técnicas propostas nos jogos *Enduro*, *Ms. Pacman*, *Breakout* e *Pong*.

1.6 Hipótese

É possível atender ao objetivo geral deste trabalho através da implementação do algoritmo de aprendizagem por reforço *Deep Q-Learning*, treinando uma rede neural artificial para, a partir da imagem da tela do jogo, estimar corretamente a recompensa esperada por tomar-se ações em ambientes de jogos eletrônicos em diferentes estados e comparativamente selecionar a melhor, nos jogos *Enduro*, *Ms. Pacman*, *Breakout* e *Pong*, no ambiente de simulação OpenAI Gym.

1.7 Apresentação do manuscrito

Esta monografia está organizada em cinco capítulos. No Capítulo 2, revisa-se os conceitos computacionais e matemáticos necessários para compreensão e desenvolvimento do trabalho. No Capítulo 3 apresenta-se a solução proposta para o problema descrito nesta introdução, mostrando as ferramentas utilizadas e justificando as decisões tomadas. No Capítulo 4, expõe-se e analisa-se os resultados obtidos com a solução desenvolvida. Por fim, no Capítulo 5, apresenta-se as conclusões sobre o trabalho e as propostas para trabalhos subsequentes.

Capítulo 2

Referencial Teórico

Neste capítulo, apresenta-se os conceitos computacionais e matemáticos de aprendizagem por reforço e aprendizagem profunda utilizados para o desenvolvimento deste trabalho, assim como o algoritmo utilizado.

2.1 Aprendizagem por Reforço

Um agente precisa constantemente selecionar ações para executar no ambiente no qual se encontra [20]. Como o ambiente pode mudar a cada instante, o agente deve selecionar uma ação para executar em cada estado a qual se deparar. Desta necessidade, naturalmente surge a pergunta de qual é a melhor ação a ser escolhida pelo agente para o estado no qual ele se encontra. Diversas técnicas de aprendizagem de máquina utilizam métodos diferentes para abordar este problema.

Aprendizagem por reforço (do inglês, *RL*) é um método no qual o agente, sem o conhecimento prévio de quais ações são melhores de serem executadas, deve experimentar o ambiente e aprender conforme interage com ele [12]. A expectativa é que, conforme o agente interage com o ambiente, seja capaz de melhorar seu desempenho. É uma área da aprendizagem de máquina inspirada pela psicologia comportamental, sendo comumente chamada de método da tentativa e erro, simulando a forma de aprendizagem humana. Em problemas de aprendizagem por reforço, o ambiente e as regras que o regem são desconhecidos pelo agente.

Nesta forma de aprendizagem, o agente deve aprender a partir de suas próprias experiências. Não há uma classificação específica de ações corretas ou erradas, apenas recompensas que indicam se o agente está mais próximo ou longe do seu objetivo [12].

A Figura 2.1 ilustra a configuração clássica de um problema de aprendizado por reforço, em que um agente observa o ambiente para então poder atuar sobre este ambiente e receber uma recompensa sobre a ação selecionada.

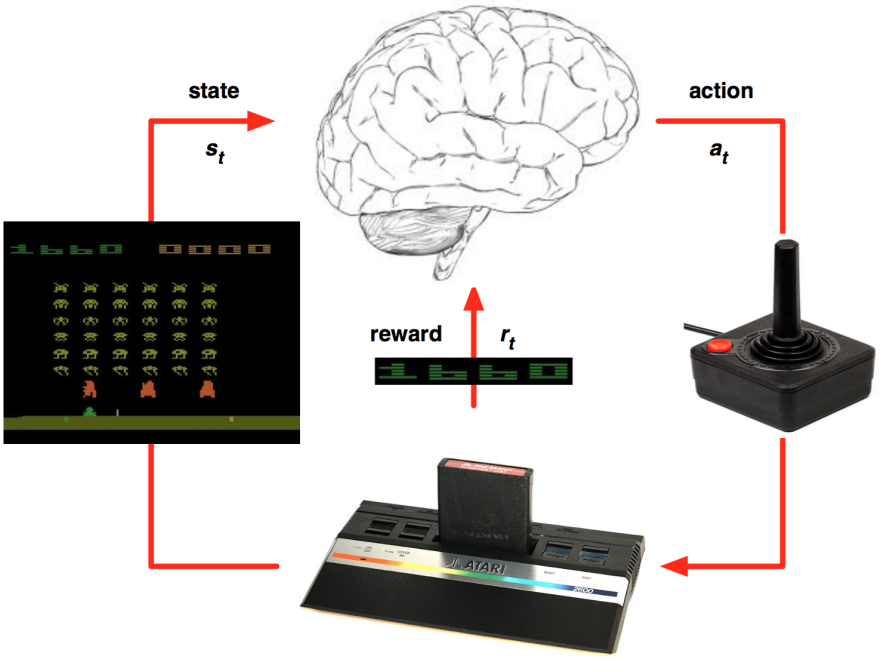


Figura 2.1: Loop de interação do agente e ambiente (Fonte: [1])

2.1.1 Estados

A cada interação com o ambiente, o agente executa uma ação a_t no instante de tempo t, realiza uma observação sobre o ambiente O_t e adquire uma recompensa r_{t-1}, referente a ultima ação a_{t-1} executada, no instante de tempo $t-1$. A sequência de todos os conjuntos $[a_t,O_t,r_{t-1}]$ é chamada de história [21]. A história nada mais é do que o conjunto de todas as variáveis observáveis pelo agente até o instante t, definida por

$$H_t = [a_1, O_1, r_1], ..., [a_t, O_t, r_t] \tag{2.1}$$

O agente possui apenas a informação disponível na história realizar suas ações. Contudo, a história é composta de muita informação, que, geralmente, não é de toda útil para o agente. Por isso, para tomar suas ações, o agente utiliza do que chamamos de estado do agente S_t^a, que nada mais é do que uma parcela de toda a história, definido por

$$S_t^a = f(H_t) \tag{2.2}$$

O estado do agente pode ser qualquer função da história, incluindo apenas a última observação recebida pelo agente ou até mesmo a história completa. Esta é a informação que o agente utiliza para selecionar a próxima ação.

O estado do agente é uma representação interna do agente sobre o estado do ambiente S_t^e[21]. O estado do ambiente é a uma representação interna do ambiente que é utilizada para determinar o que acontece no próximo instante de tempo $t+1$. Naturalmente, o estado do ambiente não é sempre visível para o agente, que apenas o observa.

Em alguns casos, o agente tem acesso e observa diretamente o estado do ambiente, tendo assim observabilidade completa sobre o ambiente $O_t = S_t^e$. Contudo, geralmente lidamos com casos de observabilidade parcial [22], em que o agente não tem acesso à todas as informações do ambiente e precisa observar o ambiente de forma indireta. Nestes casos, precisamos representar o estado do agente através da observação de alguma forma.

Existem diversas formas de tentar estimar o estado. Pode-se utilizar a história completa $S_t^a = H_t$, utilizando toda a sequência de informações obtidas até o instante t. Pode-se também tentar estimar o estado atual dentro da sua representação através de probabilidades [23]. Outra forma possível para solucionar este problema é a utilização de redes neurais para estimar o estado do agente atual [24].

2.1.2 Recompensa

Em aprendizagem por reforço, não há avaliação direta se as ações tomadas pelo agente foram boas ou ruins. Após executar uma ação a_t, o agente coleta a observação do ambiente no próximo instante O_{t+1} e um sinal de recompensa r_t. Esta recompensa é um indicador de quão bom é para o agente ter executado a ação escolhida no estado em que se encontrava no instante de tempo t, e serve para avaliar a eficiência do agente em atingir seus objetivos [12]. O agente visa executar ações que maximizem as recompensas recebidas.

Esta recompensa é chamada de recompensa imediata, pois é recebida a partir do estado atual em que o agente se encontra. Em contrapartida, a recompensa cumulativa leva em consideração todo o histórico de recompensas do início ao fim [12]. O objetivo do agente é maximizar a recompensa cumulativa recebida [21]. Ou seja, a cada passo o trabalho do agente é executar uma ação que maximize a recompensa que ele espera receber.

Um agente pode vir a se deparar com situações em que suas ações possuam benefícios distintos. Um helicóptero autônomo pode se encontrar em uma situação em que deva escolher entre chegar mais rápido ao objetivo ou com mais combustível. Mesmo que ambos resultados sejam satisfatórias, o agente consegue selecionar apenas uma única ação. Para isso, a recompensa pode ser um valor escalar, positivo ou negativo, de forma que a recompensa esperada por duas ações possam ser comparáveis, e portanto as ações em si também possam ser comparadas [21].

Parte do trabalho quando se lida com aprendizagem por reforço é modelar corretamente a forma que o agente irá obter as recompensas a partir do ambiente para que sempre seja possível comparar a recompensa esperada por ações diferentes. A partir disto, constrói-se a hipótese de que todo e qualquer objetivo de um agente pode ser descrito como a maximização da recompensa cumulativa esperada [12].

O agente é capaz de alterar o ambiente ao interagir com o mesmo, e por isso espera-se que suas ações impactem nas observações e recompensas que passa a receber. Frequentemente, as ações do agente afetam não apenas o estado subsequente, mas continuam a ter impacto no futuro [12]. Com isso, surge o problema de como aprender que uma ação contribuiu para aquisição de uma recompensa se a ação foi executada antes da recompensa ser obtida. Para lidar com o problema

de recompensas atrasadas, é possível utilizar o método de aprendizado por diferença temporal [12], que será explicado mais para frente. Desta forma, uma ação que impacte em um resultado positivo no futuro ainda é recompensada, mesmo que a recompensa do instante seguinte a sua execução não seja positiva.

O objetivo do agente é maximizar a recompensa esperada G_t calculada a partir de um estado s conforme

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{n} \gamma^k R_{t+k+1} \tag{2.3}$$

onde $t+k$ representa o instante de tempo em que se adquire uma recompensa R_{t+k} e γ é um fator de desconto.

O fator de desconto γ é um valor no intervalo $[0, 1)$ que representa o quanto consideramos recompensas futuras ao selecionar ações [12]. Quanto maior o valor de γ, mais considera-se recompensas que recebidas no futuro. Quanto menor o valor, maior a prioridade de recompensas imediatas. Um valor de $\gamma = 0$ indica que o agente considera apenas recompensas imediatas no algoritmo. Não recomenda-se a escolha de $\gamma = 1$, pois isso pode resultar em um somatório infinito ao calcular a recompensa total esperada em estados cíclicos que continuem se visitando [21]. A seleção de $\gamma = 1$ só é possível caso consiga-se garantir que não existam estados cíclicos no sistema, garantindo assim que a sequência de estados visitados seja finita, de forma que o agente compare recompensas esperadas finitas e consiga tomar decisões.

Assim, o agente é recompensado por recompensas futuras, embora recompensas imediatas, mesmo que menores, ainda sejam preferidas pelo agente, por não serem descontadas. Para o agente, receber uma recompensa $r_{t+k} = R$ no instante de tempo $t + k$ equivale a receber uma recompensa imediata $r_t = \gamma^k R$, devido ao fator de desconto.

Outro motivo para utilizar um fator de desconto é a incerteza da recompensa futura esperada [21]. Recompensas imediatas são preferíveis à recompensas futuras porque seu valor é conhecido pelo agente. Recompensas futuras precisam ser estimadas pelo agente, e por isso estão propensas a erros, já que seus valores não são, de fato, conhecidos. Quanto maior a confiança do agente nas previsões de recompensas futuras, maior pode ser o valor de γ. Caso o estado do ambiente fosse completamente conhecido pelo agente, poderia-se definir $\gamma \to 1$, pois não haveriam incertezas na previsão de recompensas futuras. O fator de desconto mostra mais uma vez como aprendizagem por reforço é espelhado no comportamento de seres humanos e animais, pois na natureza observa-se que animais e seres humanos mostram preferência por recompensas imediatas do que a recompensas futuras.

2.1.3 Política

Um agente de RL deve constantemente responder a pergunta: Qual a melhor ação a ser tomada no estado atual? A cada instante de tempo t o agente deve executar uma ação a_t, em resposta a uma observação O_t. A maneira que o agente utiliza para escolher suas ações é chamada de política

(π), e pode ser compreendida como o comportamento do agente [21]. A política define qual será a ação que o agente tomará em cada estado em que se encontrar. A política π independe do instante de tempo em que o agente se encontra, levando em consideração apenas o estado s do mesmo.

Uma política pode ser determinística ou estocástica. Uma política determinística é essencialmente um mapeamento direto de um estado para uma ação,

$$a = \pi(s), \tag{2.4}$$

determinando a ação a que deve ser tomada por encontrar-se em um estado qualquer s. Já uma política estocástica determina a probabilidade de um agente escolher uma ação a dado que o agente encontra-se no estado s,

$$a = \pi(a|s) = P(A = a|S = s) \tag{2.5}$$

Um algoritmo de aprendizagem por reforço tem como objetivo melhorar sua política conforme interage com o ambiente, de forma a obter recompensas melhores com o passar do tempo. Frequentemente, inicializa-se o agente com uma política aleatória e deixa-se que o agente aprenda do zero com suas próprias experiências [3].

O objetivo do agente é maximizar a recompensa cumulativa esperada, e para isso o agente deve selecionar as ações que experimentou no passado que aumentaram a recompensa recebida. Contudo, é possível que existam soluções melhores que não foram encontradas por falta de interações. Deseja-se que o agente, enquanto aprende ao interagir com o ambiente, não siga sempre o caminho que considera o melhor possível. Para isso, o agente deve explorar caminhos e tentar ações que não havia tentado previamente, na esperança de encontrar soluções melhores inesperadas. Por isso, implementa-se no algoritmo de aprendizagem um comportamento de exploração [12]. O agente deve, a cada interação, optar por selecionar a melhor ação conhecida, dada por π, ou explorar uma ação inusitada.

Deve tomar-se cuidado ao implementar a exploração no agente, pois isto faz com que o agente gaste tempo explorando caminhos que ele já aprendeu não serem tão vantajosos. O agente deve ponderar entre o comportamento de exploração e usufruir de ações já aprendeu serem vantajosas. Nas primeiras interações, deseja-se que o agente possua um comportamento mais exploratório, pois ele ainda interagiu pouco com o ambiente. Ao passar do tempo, diminui-se o fator de exploração, pois espera-se que o agente já tenha um maior conhecimento sobre as recompensas provenientes das ações que deve executa, podendo assim selecionar as ações que o deixem mais próximo de cumprir seu objetivo.

2.1.4 Função de Valor

A função de valor do estado $V(s)$ é uma função que prediz o valor da recompensa futura esperada. Sabendo a recompensa esperada por estar em um estado, o agente pode avaliar o quão

bom é encontrar-se naquele estado, e pode comparar o quão vantajoso é encontrar-se em diferentes estados, e utilizar estes valores como base para tomar suas decisões [12].

A função de valor tem um conceito semelhante ao sinal de recompensa, por ser utilizada para medir o desempenho do agente. Diferentemente do sinal de recompensa, a função de valor avalia o desempenho do agente a longo prazo, enquanto o sinal de recompensa é uma medida imediata [12]. Naturalmente, utilizamos a função de valor para selecionar as ações do agente, pois nosso objetivo é maximizar a recompensa total acumulada pelo agente. Para isso, deve-se sacrificar recompensas imediatas em prol de ganhos maiores no futuro [12].

Ao elaborar-se a função de valor V_π, trabalha-se com a expectativa da recompensa E, ou valor esperado de recompensa, pois desconhecemos o ambiente, e por isso não somos capazes de prever com completa precisão as recompensas futuras. Em outras palavras, podemos considerar o ambiente como sendo estocástico.

$$V_\pi(s) = E_\pi\{R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ...|S_t^a = s\} \tag{2.6}$$

Observa-se que V_π depende da política utilizada, pois a forma que o agente seleciona suas ações impacta as recompensas que ele recebe. A função de valor para uma política específica V_π é igual ao valor esperado por encontrar-se em um estado s da recompensa imediata, somada as recompensas futuras esperadas, descontadas por um fator de desconto γ.

Além da função de valor do estado, pode-se implementar uma função de valor de ação

$$q_\pi(s, a) = E_\pi\{R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ...|S_t = s, A_t = a\} \tag{2.7}$$

que, semelhante à função de valor do estado, determina o valor de recompensa esperada ao tomar-se uma ação a estando no estado s, caso siga-se uma política π [21]. Esta função é mais útil para um agente de RL, visto que o agente não pode controlar diretamente o estado que deseja atingir, e sim as ações que pode executar.

A Equação de Bellman [12]

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... = R_{t+1} + \gamma V_\pi(s_{t+1}) \tag{2.8}$$

nos diz que a ação ótima para todos os passos do problema pode ser resumida como a ação ótima do passo atual mais a ação ótima do próximo passo [21]. Isso significa que as funções valores dados nas Eqs. 2.6 e 2.7 podem ser simplificadas nas equações

$$V_\pi(s) = E_\pi\{G_t|S_t^a = s\} \tag{2.9}$$

e

$$q_\pi(s, a) = E_\pi\{G_t|S_t = s, A_t = a\}, \tag{2.10}$$

respectivamente. É preferível trabalhar com as equações simplificadas, pois não precisa-se saber todos os valores das recompensas esperadas nos próximos passos, apenas a recompensa imediata por estar-se em um estado e a recompensa esperada no estado imediatamente seguinte.

2.1.5 Processo de Decisão de Markov

Uma cadeia de Markov é um modelo matemático que descreve a sequência de possíveis eventos em um problema. Modelam-se os estados, que descrevem as possíveis situações que um agente pode se deparar. É uma máquina de estados, onde os estados da cadeia possuem a propriedade Markoviana, que diz que a probabilidade de alcançar um estado S_{t+1} baseado no estado atual S_t é igual a probabilidade de alcançar este mesmo estado baseado em todos os estados que já passaram [21]. Ou seja, a probabilidade de atingir-se um próximo estado S_{t+1} depende apenas do estado atual, e não da história, deste modo

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, ..., S_t). \tag{2.11}$$

Em uma cadeia de Markov, o agente encontra-se sempre em algum estado s e pode mover-se para os estados adjacentes com uma certa probabilidade p. Na Figura 2.2, há a representação de uma cadeia de Markov em forma de grafo. Os estados são representados pelos nós e as probabilidades de mudança de estados estão indicadas nas arestas. Uma cadeia de Markov deve possuir ao menos um estado terminal [21], representado por S_5 na Figura 2.2. O estado terminal nada mais é do que um estado com laço para si mesmo, de forma que a sequência possa se encerrar.

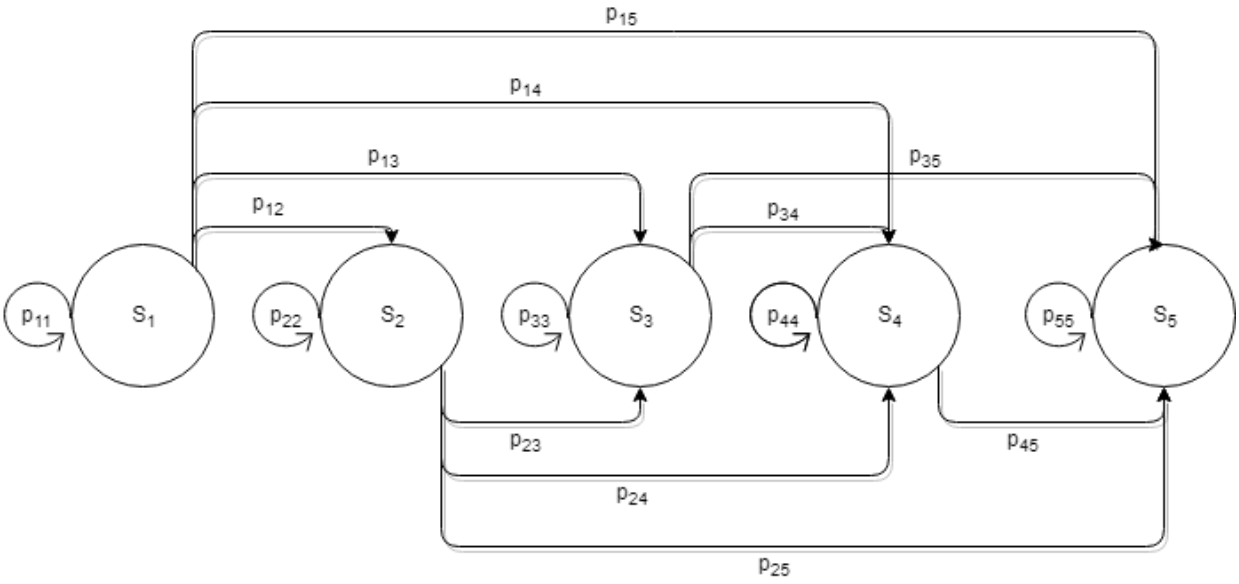


Figura 2.2: Exemplo de uma cadeia de Markov

Outra maneira de representar uma cadeia de Markov é com uma matriz de transições. Os elementos das matrizes representam a probabilidade de transição entre estados, que por sua vez são indicados pelos índices e colunas da matriz. Para saber qual a probabilidade da transição de

um estado S_i para um estado S_j, basta procurar na matriz pelo elemento p_{ij} na linha i e coluna j [25]

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} & p_{15} \\ 0 & p_{22} & p_{23} & p_{24} & p_{25} \\ 0 & 0 & p_{33} & p_{34} & p_{35} \\ 0 & 0 & 0 & p_{44} & p_{45} \\ 0 & 0 & 0 & 0 & p_{55} \end{bmatrix} \tag{2.12}$$

Conhecendo a matriz de transição de um sistema, pode-se utilizá-la para calcular a probabilidade de atingir um estado s' a partir de um estado s:

$$P_{ss'} = P(S_{t+1} = s' | S_t = s) \tag{2.13}$$

A cadeia de Markov serve para determinar a probabilidade de atingir-se um estado em um processo estocástico. Contudo, é possível utilizar uma cadeia de Markov para modelar problemas que, assim como em RL, o ambiente forneça uma recompensa $R_t(s)$ a um agente por encontrar-se em um estado s. Como o estado do ambiente é desconhecido ao agente, utiliza-se uma função de recompensa para estimar este valor de recompensa.

$$R_t(s) = E\{R_{t+1} | S_t = s\} \tag{2.14}$$

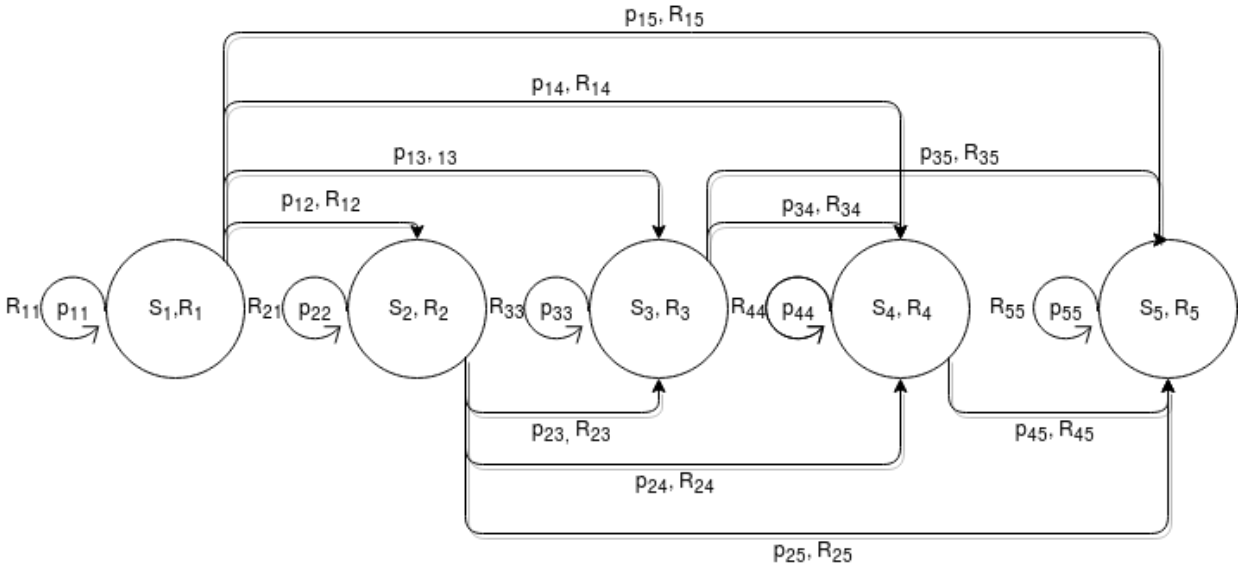


Figura 2.3: Exemplo de uma cadeia de Markov com recompensas

Tendo conhecimento de $R_t(s)$ de cada estado de uma cadeia de Markov, torna-se possível determinar a recompensa esperada por encontrar-se em cada um dos estados de uma cadeia de Markov utilizando a Equação 2.8. Tendo construído-se uma cadeia com tais características, tal ilustrado na Figura 2.3, basta observar-se o estado de maior valor de recompensa esperada para

saber qual o melhor estado. Esta conclusão depende do valor escolhido para γ, pois valores menores tenderão a preferir estados de maior recompensa imediata. No extremo, quando $\gamma = 0$, teremos $V(t) = R_t(s) \forall s$.

Além da representação por um grafo com nós e arestas como ilustrado na Figura 2.3, também possível representar uma cadeia de Markov utilizando uma representação matricial da equação de Bellman [21]

$$v = R + \gamma P v, \tag{2.15}$$

onde

$$v = \begin{bmatrix} v(1) \\ v(2) \\ ... \\ v(n) \end{bmatrix}, R = \begin{bmatrix} R_1 \\ R_2 \\ ... \\ R_n \end{bmatrix}, P = \gamma \begin{bmatrix} P_{11} P_{12} ... P_{1n} \\ P_{21} P_{22} ... P_{2n} \\ ... \\ P_{n1} P_{n2} ... P_{nn} \end{bmatrix} \tag{2.16}$$

Com auxílio da Eq. 2.15, pode-se calcular linearmente o valor de $v(s)$ para qualquer estado s, através de

$$v = \frac{R}{1 - \gamma P}. \tag{2.17}$$

Contudo, por envolver inversões matriciais, o cálculo direto de $v(s)$ é uma operação custosa computacionalmente, tendo complexidade $O(n^3)$ [21]. Por isso, é comum utilizar de algoritmos interativos para encontrar a solução de cadeias de Markov grandes demais para serem solucionadas analiticamente em um intervalo de tempo viável.

Pode-se utilizar a equação de Bellman para avaliar a função valor do estado de diferentes políticas, bastando que

$$v_\pi = (1 - \gamma P^\pi)^{-1} R^\pi \tag{2.18}$$

seja aplicada a todas as políticas as quais deseja-se comparar. A seguir, torna-se fácil a seleção da melhor política, pois esta será a que possuir os maiores valores de v_π, conforme

$$v_*(s) = max(v_\pi(s)) \tag{2.19}$$

Até então analisamos a probabilidade de um agente encontrar-se em cada um dos estados de uma cadeia de Markov e pudemos determinar quais os melhores estados para este agente. Contudo, sabemos que um agente não apenas existe em um ambiente estocástico, mas também executa ações sobre o mesmo. Por isso, um Processo de Decisão de Markov (do inglês, *Markovian Decision Proccess*, MDP)[26] inclui tomada de ações na modelagem de cadeias de Markov. Processos de

decisão de Markov são utilizadas para solucionar problemas de decisão sequencial com incerteza, e podem ser vistos simplesmente como cadeias de Markov controladas [20].

As arestas do MDP na Figura 2.4 representam a ação selecionada, que levam o agente de um estado a outro, ao invés de uma probabilidade. Modelos mais completos de um MDP podem não ser determinísticos, incluindo a probabilidade de se chegar a um estado S_{t+1} a partir de um estado S_t ao tomar a ação a. Desta forma, um agente pode utilizar um MDP para selecionar ações que maximizem a recompensa adquirida, utilizando a informação da recompensa esperada associada a cada estado. A Figura 2.4 ilustra um MDP.

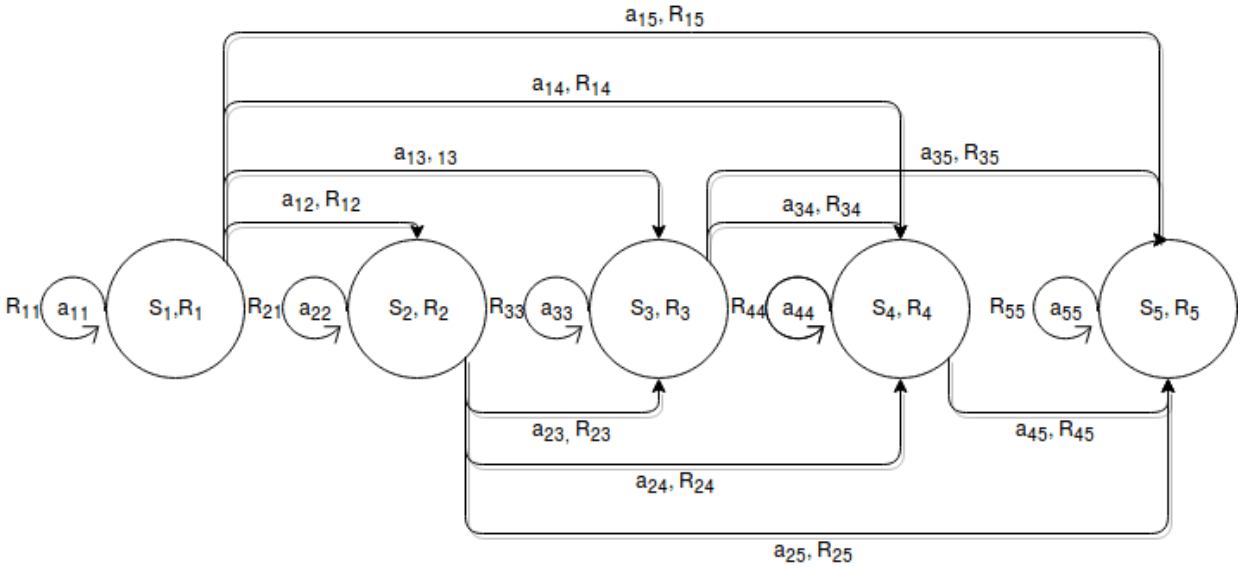


Figura 2.4: Processo de Decisão de Markov

Analogamente à função de valor do estado, é possível calcular a melhor função de valor de ação q^*,

$$q^*(s,a) = max(q_\pi(s,a)), \tag{2.20}$$

simplesmente comparando todas as funções q_π e selecionando a que retorna o maior valor de recompensa esperada para o par de entrada s, a. O objetivo do agente é encontrar q^*, pois uma vez o feito, o agente sempre saberá a melhor ação a tomar, comportando-se de forma ótima [12].

Quase todos os problemas de RL podem ser formalizados como um MDP [21], visto que os algoritmos de aprendizagem por reforço costumam utilizar técnicas iterativas. Algoritmos de RL não precisam conhecer o MDP e se destinam a MDPs grandes onde os métodos de solução determinísticos se tornam computacionalmente inviáveis.

Em aprendizagem por reforço, o estado do ambiente S_t^e é uma propriedade *markoviana* por definição, pois é utilizando a informação do estado do ambiente que o ambiente define o próximo estado [21]. Consequentemente, a H_t também é markoviana, já que o estado é uma função da história. Por isso, sabemos que sempre existe um estado de Markov, mesmo que não seja a representação mais útil ou prática. Ao modelar-se um agente de RL, deseja-se que o estado do

agente S_t^a seja markoviano, para que o agente possua toda a informação necessária para poder tomar as decisões sobre o problema.

2.1.6 Aprendizado com diferença temporal

Existem problemas que não podem ser resolvidos de forma analítica, muitas vezes pelo alto custo computacional associado à solução. Para estes casos, tais como MDPs grandes, utiliza-se técnicas iterativas para encontrar a resposta. Um método iterativo possível de ser utilizado em problemas de aprendizagem por reforço é o aprendizado com diferença temporal (do inglês, *Temporal Difference, TD* [12]. Em TD, o aprendizado baseia-se na diferença entre predições suscetivas no tempo. Desta forma, a previsão do agente é corrigida a cada passo para melhor corresponder aos valores recebidos na próxima vez [27].

Utilizando TD, pode-se estimar V_π ao executar ações e receber a recompensa pelas ações selecionadas, por

$$V(s_{t-1}) = V(s_{t-1}) + \alpha[r_t + \gamma V(s_t) - V(s_{t-1})], \tag{2.21}$$

onde s_t é o estado do agente após executar uma ação no instante s_{t-1}, r_t é a recompensa associada a esta ação, α é o fator de aprendizagem e γ o fator de desconto [12]. A correção de V_{st-1} é feita imediatamente ao agente receber a recompensa, diferente de outros algoritmos que realizam o aprendizado apenas após a conclusão do episódio.

O fator de aprendizagem α é utilizado para limitar o passo de aprendizagem, reduzindo a variação de $V(s_{t-1})$ a cada iteração [12]. Diferente de γ, o fator de aprendizagem α não pode ser 0 para que exista aprendizagem, devendo ter seus valores limitados na faixa de valores $(0, 1)$ [12].

2.2 Q-Learning

Q-Learning é uma técnica interativa de aprendizagem por reforço sem política e sem modelo utilizada para controlar agentes [12], podendo ser utilizado para encontrar v_π^* em um MDP. O Q-Learning é dito um algoritmo sem política pois utiliza apenas a função de valor para selecionar as ações do agente. Ou então, a política (ou comportamento) do agente pode ser descrita como uma simples seleção da maior função de valor: $\pi(s_t) = max(v(s_t))$

Por ser uma técnica sem modelo, algoritmos de Q-Learning não precisam conhecer o ambiente e modelar a cadeia de Markov para resolvê-la.

A cada instante, o Q-Learning atualiza a função de valor de ação $Q(s, a)$ com base nas experiências passadas e na recompensa recebida utilizando a equação de Bellman

$$Q(s_t^a|a) \leftarrow (1 - \alpha)Q(s_t^a|a) + \alpha(r_t + \gamma Max(Q(s_{t+1}^a, a_i))), \tag{2.22}$$

levando em consideração o fator de aprendizagem α e fator de desconto γ. Para identificar o

valor associado ao próximo estado s^a_{t+1}, utiliza-se a expressão $Max(Q(s^a_{t+1}, a_i))$ para comparar as recompensas associadas a cada ação a_i possível de ser executada no estado s^a_{t+1}.

Na equação (2.22), atualiza-se a função de valor $Q(s^a_t|a)$ por primeiro descontar-se parte do conhecimento já aprendido, utilizando o fator de aprendizagem em $(1 - \alpha)$. Quanto maior o valor de α, mais rápido o agente aprende com experiências novas em detrimento de experiências antigas. A nova experiência é calculada como a recompensa imediata recebida, somada com a máxima experiência que obtém-se no estado seguinte, considerando o maior valor dentre todas as ações possíveis $Max(Q(s^a_{t+1}, a_i))$ naquele estado, descontado por γ.

Inicialmente o Q-Learning não possui informações suficientes sobre os estados e as recompensas. Porém, ao frequentar o mesmo estado várias vezes, pode experimentar ações diferentes e aprender com as mesmas. Após um número suficiente de interações, que pode variar drasticamente conforme o problema, a função Q passa a convergir à política ótima [12].

O algoritmo do *Q-Learning* consiste em realizar a observação do ambiente no instante t e adquirir a recompensa referente a ação tomada previamente, no instante de tempo $t-1$. O agente então calcula o estado atual utilizando alguma função f da observação O_t, e em seguida, atualiza sua função de valor Q utilizando a Equação (2.22) calcula a próxima ação a ser executada. O algoritmo está ilustrado em pseudocódigo abaixo.

Algoritmo 1 Q-Learning

Inicializar **Q** arbitrariamente
Enquanto episódio $\neq$ fim
Receber O_t e r_{t-1}
$S^a_t = f(O_t)$
$Q(s^a_{t-1}, a) \leftarrow (1 - \alpha)Q(s^a_{t-1}, a) + \alpha(r_{t-1} + \gamma MaxQ(s^a_t, a_i))$
$a = Max_a Q(s_t, a_i)$
Fim enquanto

É importante ressaltar que a recompensa r_t referente a uma ação a_t só é recebida pelo agente no instante $t + 1$, junto com a observação O_{t+1}. Por isso, adaptou-se a equação (2.22) para implementar o algoritmo. O agente deve memorizar qual estado esteve no passo de tempo $t - 1$ e a ação tomada, S_{t-1} e a_{t-1}, para quando receber a recompensa r_{t-1} no instante t, poder atualizar a função Q.

Infelizmente o Q-Learning conforme descrito não é capaz de resolver problemas em ambientes contínuos [28] em decorrência do número infinito de estados possíveis, não podendo assim realizar o mapeamento direto de um estado para uma ação. Contudo, existem formas de lidar com muitos estados, tais como utilizar redes neurais artificiais para implementar-se um algoritmo de Deep Q-Learning.

2.3 Redes Neurais

Redes Neurais Artificiais (RNAs) são modelos computacionais inspirados na estrutura do cérebro, e são excepcionalmente eficientes em encontrar padrões em dados altamente estruturados [3]. Em uma RNA, várias estruturas denominadas neurônios se conectam entre si. As conexões entre os neurônios são chamadas de pesos ω, valores numéricos que multiplicam o valor de entrada [2]. A entrada de um neurônio x é alguma combinação de todos os valores que entram no neurônio multiplicadas por seus respectivos pesos. Um exemplo de um neurônio individual pode ser visto na Figura 2.5.

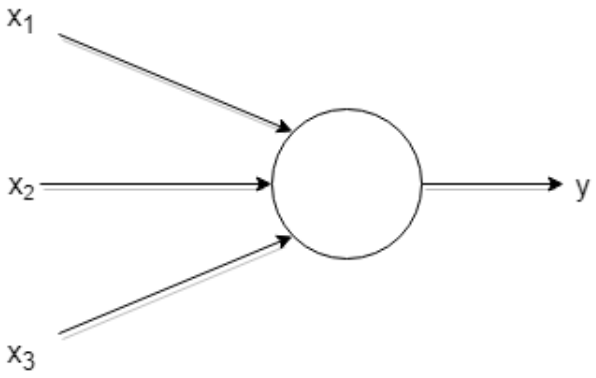


Figura 2.5: Exemplo de um neurônio

Um neurônio possui uma função de ativação, e sua saída depende da função utilizada. Um modelo simples de neurônios são perceptrons, que utilizam uma função degrau para realizar a sua ativação, fazendo com que a saída seja igual a 1 caso o somatório ponderado das entradas seja maior que um valor de ativação b (do inglês, *bias*), e 0 caso contrário [29], conforme

$$y = \begin{cases} 0, & \text{se } \sum_{i=1}^{n} \omega_i x_i \leq b \\ 1, & \text{se } \sum_{i=1}^{n} \omega_i x_i > b \end{cases} \tag{2.23}$$

Neste caso, a aprendizagem ocorre ao alterar-se o valor dos pesos entre neurônios para atender o conjunto de entradas e saídas utilizados para treinar a rede. Ao longo de várias iterações, a rede observa o valor atual correspondente a uma entrada e corrige seus pesos de uma forma a obter um valor mais próximo ao esperado [29].

Em perceptrons, uma mudança pequena em um peso pode resultar na mudança do estado de um neurônio de inativo para ativo ou vice-versa, causando em uma mudança drástica na saída do mesmo [2]. Contudo, às vezes deseja-se que o aprendizado seja gradual, e que pequenas mudanças nos pesos causem pequenas variações na saída da rede [2]. Para solucionar este problema, é comum utilizar-se funções *sigmoides* σ como função de ativação, devido à ativação mais suave e variações menos bruscas conforme a variação da entrada, conforme

$$\Delta y = \sum_{i=1}^{n} \frac{\partial y}{\partial w_j} \Delta w_j + \frac{\partial y}{\partial ativação} \Delta ativação \tag{2.24}$$

onde Δy é a variação da saída , ω os pesos e os b *bias*, permitindo que mudanças pequenas nos pesos e nos *bias* causem mudanças igualmente pequenas na saída do neurônio [2]. Mesmo com essa diferença, neurônios implementados com funções de ativação sigmoides possuem o mesmo comportamento qualitativo de *perceptrons* [2].

Geralmente uma RNA é constituída em camadas de neurônios, possuindo obrigatoriamente a camada de neurônios de entrada e a camada de saída, podendo conter um número qualquer de camadas intermediárias [29].

As redes neurais podem ser recorrentes ou sem realimentação. As redes neurais sem realimentação não permitem laços nos grafos que as representam, e podem ser organizadas em camadas [30], conforme mostrado na Figura 2.6.

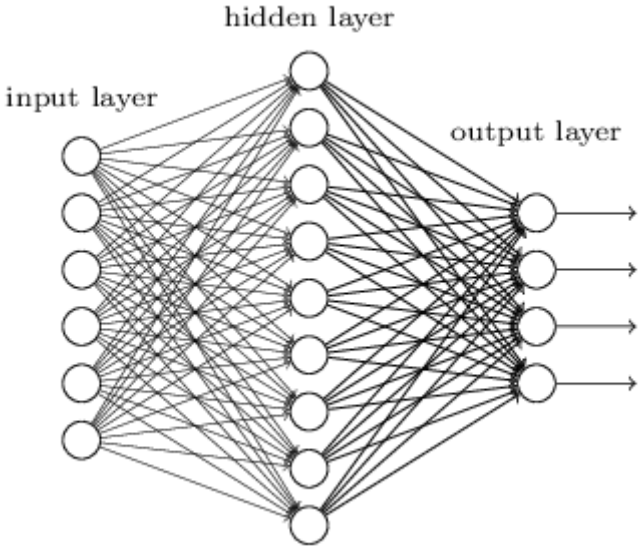


Figura 2.6: Rede Neural Artificial com 1 camada intermediária e múltiplas saídas (Fonte: [2])

Os neurônios se conectam com cada neurônio da camada seguinte, e essas conexões possuem pesos que modificam o sinal de entrada e que são ajustados conforme treina-se a rede neural [29]. Para treinar uma RNA pelo método supervisionado fornece-se à rede um conjunto de dados de entrada e as saídas esperadas para aquelas entradas. A rede atualiza os pesos entre os neurônios para tentar fazer com que uma entrada igual à fornecida resulte na saída desejada. Conforme treina-se a rede, os pesos vão se adaptando de forma a tentar satisfazer as regras ou padrões, até então desconhecidos, que regem o conjunto de dados fornecidos [29]. Ao treinar-se a rede com um conjunto suficientemente grande e diverso de dados, a rede torna-se capaz de encontrar saídas corretas para entradas que se enquadrem nos padrões treinados [29].

2.4 Aprendizagem Profunda

Aprendizagem profunda (do inglês, *Deep Learning*) baseia-se em tentar modelar abstrações de alto níveis utilizando redes neurais. Se uma rede neural possui muitas camadas, a definimos como uma rede profunda. Esta definição é ampla, e não existe uma separação clara entre redes profundas

e rasas. Conforme [2], uma rede com duas ou mais camadas é considerada como profunda, e redes com uma única camada são consideradas rasas. Esta definição é mutável, e a tendência é que conforme o avanço do campo e redes maiores passam a ser desenvolvidas, uma rede neural passa a precisar de mais camadas para realmente ser considerada profunda. Uma rede neural profunda está ilustrada na Figura 2.7.

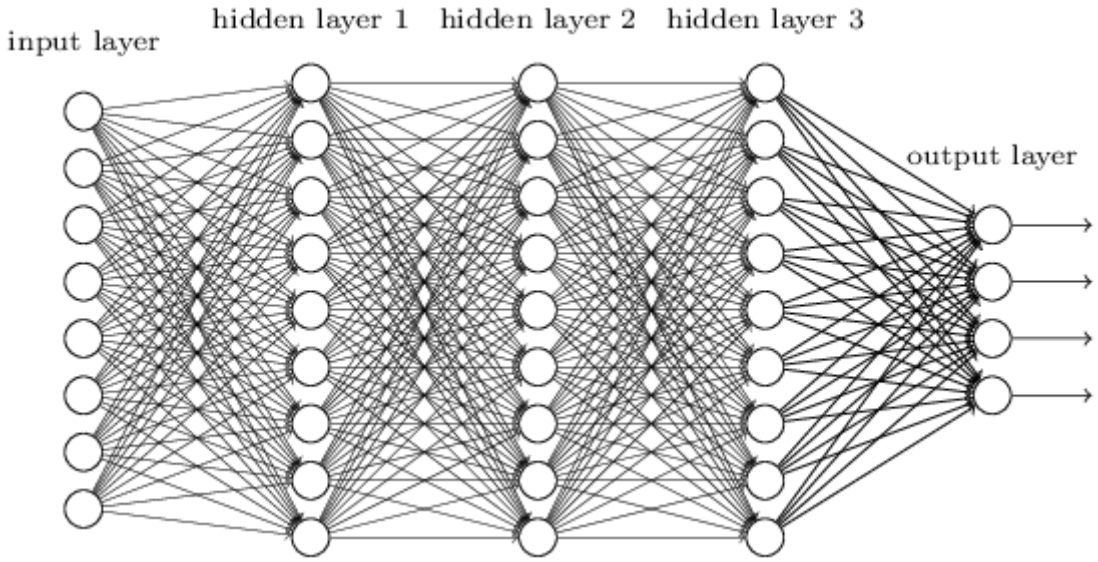


Figura 2.7: Rede Neural Artificial Profunda (Fonte: [2])

As camadas em uma rede neural profunda correspondem a níveis de abstração ou composição. O número de camadas e o tamanho de cada uma impacta diretamente no nível de abstrações que a rede é capaz de fornecer [31]. As camadas de nível superior aprendem com as camadas de nível inferior, trabalhando em uma abstração acima da camada prévia. Desta forma, uma rede treinada com um espaço amostral grande e diverso o suficiente consegue fornecer uma saída correta à uma entrada até então nunca vista, por ter aprendido a reconhecer padrões graças aos dados de treinamento.

Técnicas de *Deep Learning* permitem que se lide com uma quantidade alta de dados e com dados de abstrações mais complexas. Por isso, é comum que várias arquiteturas de *Deep Learning* sejam utilizadas nos campos da visão computacional, do reconhecimento de fala e do processamento de linguagem natural. Naturalmente, considera-se a utilização de RNAs profundas em problemas de RL nos quais o ambiente fornece imagens como observações.

2.5 Deep Q-Learning

Deep Q-Learning é uma adaptação do algoritmo de Q-Learning tradicional para lidar com um número muito alto de estados. Quando se lida com imagens, cada conjunto distinto de pixels é um estado diferente, o que significa que a cadeia de Markov torna-se extremamente grande. Para fins computacionais, a quantidade de espaços é virtualmente infinita. É inviável que o agente visite cada um destes estados repetidas vezes para realizar o treinamento.

Para resolver este problema, é possível treinar uma RNA para estimar S_t^a a partir de O_t ou até mesmo para estimar a própria função valor $Q(s, a)$ [3]. Redes neurais evitam este problema porque os pares de estado-ação não precisam ser armazenados na memória e nem precisam ser especificamente revisitados. A rede apenas precisa ajustar os valores dos pesos dos neurônios para adaptar-se as novas leituras feitas.

Redes neurais artificiais são tradicionalmente treinadas utilizando um conjunto de entradas e saídas esperadas. Contudo, também são frequentemente utilizadas em problemas de aprendizagem por reforço. Ao invés de treinar-se a rede com um conjunto de dados de uma única vez, a rede é treinada conforme o agente interage com o ambiente.

Existe mais de uma maneira possível de realizar o mapeamento de $Q(s, a)$ à recompensa R utilizando uma RNA, conforme mostra a Figura 2.8. Alguns trabalhos mapeiam o estado s_t para várias saídas, cada uma representando a recompensa esperada R_{ai} por executar uma ação a_i [3]. Outra maneira é utilizar o par s, a como entrada da RNA, e observar uma única saída de recompensa esperada R_a por ter-se executado a ação.

Independente do método específico escolhido, o fundamental é que a rede deve ser treinada utilizando informações do estado do agente S_t^a, ação executada a_t e recompensa r_t associada ao executar a em S_t^a, e que as recompensas esperadas de todas as ações sejam comparadas para que possa selecionar-se a melhor.

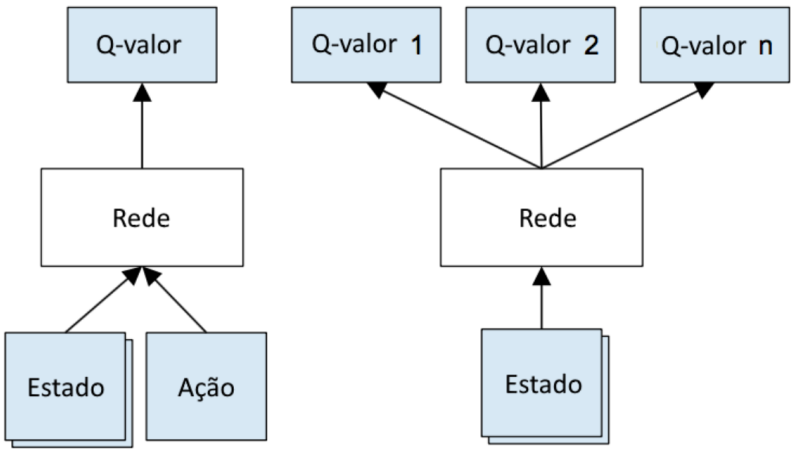


Figura 2.8: Possíveis abordagens de RNAs em Deep Q-Learning (Fonte: [3])

No próximo capítulo, apresenta-se o modelo proposto para implementar a técnica de aprendizagem profunda por reforço descrita neste capítulo.

Capítulo 3

Sistema Proposto

Neste capítulo explica-se as decisões tomadas e questões técnicas envolvidas no desenvolvimento do agente computacional que aprende a jogar jogos eletrônicos distintos. Apresenta-se as ferramentas utilizadas, escolha de parâmetros e simplificações utilizadas no processo. Todo o trabalho foi realizado utilizando ferramentas *open-source*.

3.1 OpenAI

Para simular ambientes distintos, utilizou-se da ferramenta OpenAI Gym Toolkit, uma interface *open source* de ambientes de aprendizagem por reforço [32]. Esta ferramenta permite flexibilidade, visto que não faz suposições sobre a estrutura do agente. Os ambientes providos pela ferramenta retornam os valores de observação O_t e recompensa R_t ao enviar-se a ação selecionada. Além disso, o ambiente também informa sobre o término do episódio, seja porque o agente cumpriu seu objetivo ou está incapacitado de cumpri-lo, tal como quando morre em um jogo eletrônico. O agente desenvolvido utiliza a interface comum a todos os ambientes, independentemente de qual esteja sendo executado no momento, possibilitando a produção e teste de algoritmos genéricos.

Normalmente, é difícil a comparação entre trabalhos de RL, visto que diferenças sutis na definição do problema, tais como a implementação da função de recompensa $R(S_t)$ ou no conjunto de ações utilizados alteram drasticamente a natureza do problema, dificultando a reprodução e comparação de trabalhos publicados. A utilização de uma interface como a provida pelo Open AI permite uma padronização nas métricas utilizadas em algoritmos de RL.

A ferramenta OpenAI utiliza a estrutura clássica de aprendizagem por reforço, onde executa-se um laço em que o ambiente recebe uma ação a_t e retorna ao agente uma observação e uma recompensa. Além disso, a ferramenta fornece a informação de encerramento do episódio, caso o agente tenha vencido ou perdido o jogo. O ambiente também fornece informações de diagnóstico que podem ser utilizadas para depuração. Contudo, como essas informações não são utilizadas para realizar as avaliações do agente [32], foram desconsideradas neste trabalho.

3.2 ROS

Para simular uma situação mais próxima da estrutura proposta em RL, não desejou-se desacoplar o ambiente da implementação do agente. Neste trabalho, resolveu-se utilizar da tecnologia ROS (*Robotics Operating System*). ROS é uma ferramenta *open-source* descrita como um meta sistema operacional para robótica [33]. Optou-se por utilizar ROS neste trabalho pelo serviço de gerenciamento de mensagens assíncrono entre processos, permitindo implementar-se uma interface para os ambientes e o agente de forma independente um do outro, respeitando apenas o convencionamento de mensagens selecionado.

O ROS implementa o sistema de envio de mensagens através de nós. As mensagens possuem tipos definidos, e são enviadas através de tópicos. Os nós podem implementar *publishers*, que enviam mensagens ao sistema através dos tópicos, e *subscribers*, que observam as mensagens que são escritas nos tópicos. A Figura 3.1 ilustra um processo de comunicação utilizando o ROS.

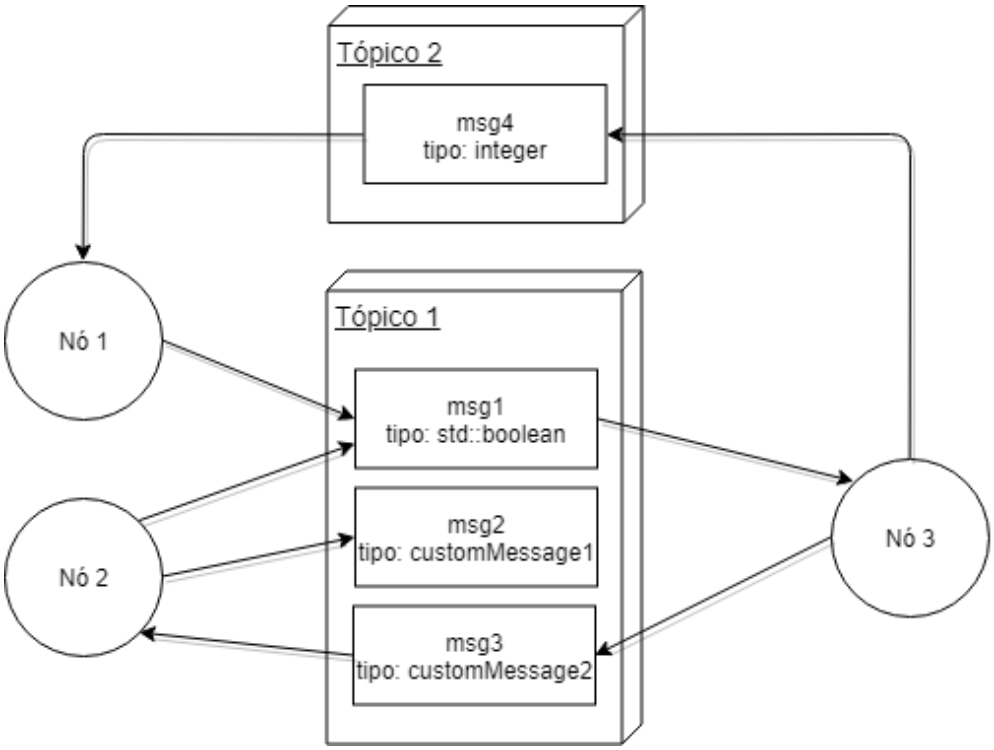


Figura 3.1: Exemplo de nós, tópicos e mensagens no ROS

Um *publisher* precisa especificar qual o tópico no qual deseja publicar, e qual o tipo da mensagem que está publicando. De forma semelhante, um *subscriber* precisa especificar qual o tópico que deseja observar e qual o tipo de mensagem que deseja ler dentro daquele tópico. Ao receber uma mensagem do tipo correto no tópico especificado, o *subscriber* executa uma função de *callback* implementada para lidar com o dado recém recebido.

O sistema de distribuição de mensagens embutido do ROS não apenas poupa o tempo de desenvolvimento como também aumenta a robustez do projeto, visto que o agente e ambiente podem ser desacoplados, podendo-se facilmente trocar o jogo a qual o agente está submetido.

Além disso, o sistema de mensagens força o desenvolvimento de interfaces claras entre os nós do sistema, melhorando a reusabilidade e escalabilidade do código. [33]

O ROS já possui um conjunto de mensagens padrão que podem ser utilizadas em qualquer tópico por qualquer nó. Alguns dos tipos de mensagens padrão são: *Bool, Byte, Float32, Float64, Int16, Int32* e *Int64*. Embora o ROS possua uma alta variedade de mensagens padrão, muitas vezes não é o suficiente para a aplicação, especialmente quando deseja-se enviar grupos de dados. Para solucionar este problema, é possível implementar mensagens personalizadas, que nada mais são do que agrupamentos de variáveis e vetores para serem utilizados como uma única mensagem em um tópico. A seção seguinte discute as mensagens personalizadas utilizadas neste trabalho.

3.2.1 Mensagens utilizadas

Neste trabalho, implementou-se dois nós: o nó do ambiente e do agente. O nó do ambiente publica mensagens de observações em um tópico de observações. As mensagens de observações são constituídas de uma observação O_t e uma recompensa r_{t-1}, sendo O_t correspondente ao estado atual do jogo e r_{t-1} a recompensa correspondente a última ação executada pelo agente.

O agente observa as mensagens de observação emitidas pelo nó do ambiente e, por sua vez, publica mensagens de ação no tópico de ações. A mensagem de ação é um valor inteiro que corresponde a a ação a qual deseja-se executar, a qual tanto o agente quanto o ambiente tem acesso por estarem listadas em um arquivo de configurações.

Como utilizamos um ambiente simulado, o nó do ambiente precisa também receber as ações selecionadas pelo agente para inseri-la na simulação. Por isso, o nó do ambiente observa as mensagens de ação escritas no tópico de ações.

3.3 Arquitetura

O ambiente do jogo é executado independente do agente. É possível executar qualquer jogo sem sequer a presença de um agente. Neste caso, o jogador não irá executar ações, ou que o jogador irá executar a ação *nenhuma* a cada passo de tempo Δt. No OpenAI Gym, como as ações são representadas como números, isso significa que $a_t = 0 \forall t$. A cada passo de tempo Δt o ambiente irá atualizar seu estado e publicar as mensagens de observação e recompensa, independente de haver recebido ou não uma nova ação. Ao receber uma ação do agente, o ambiente apenas armazena a ação selecionada para utilizá-la no próximo passo de tempo da simulação. Caso o ambiente não receba uma nova ação, considera-se que a última ação recebida continua sendo executada, pois até mesmo não fazer nada é considerada uma ação para o sistema.

Embora o agente seja tecnicamente independente do ambiente, ele aguarda até receber a mensagem de observação para executar a sua parte, pois precisa da informação da observação O_t e da recompensa r_{t-1} para realizar o aprendizado e selecionar a próxima ação. Ao receber uma mensagem de observação, o agente utiliza as informações adquiridas para atualizar sua função de valor e selecionar a próxima ação a ser executada.

Para atualizar a função de valor da ação, o agente calcula o valor Q atual utilizando a Equação (2.22) para, então, treinar a RNA. O agente utiliza da recompensa r_{t-1} recebida no instante de tempo t por ter-se executado uma ação a_{t-1} no estado s_{t-1}. Por isso, o agente deve armazenar as informações do estado do ambiente e ação executada até o próximo passo t, no qual receberá a recompensa associada à ação.

Além de atualizar a função de valor da ação, o agente precisa calcular a ação imediata a ser tomada a_t. Para isso, o agente precisa extrair o estado do agente S_t^a a partir da observação recebida O_t. Então, o agente escolhe entre utilizar a função de valor $Q(s, a)$ para selecionar sua próxima ação, ou realizar uma ação aleatória com probabilidade ϵ, decrescente ao longo dos episódios, para explorar o ambiente. A estrutura interna do agente para tomada de decisões e aprendizagem pode ser vista na Figura 3.2.

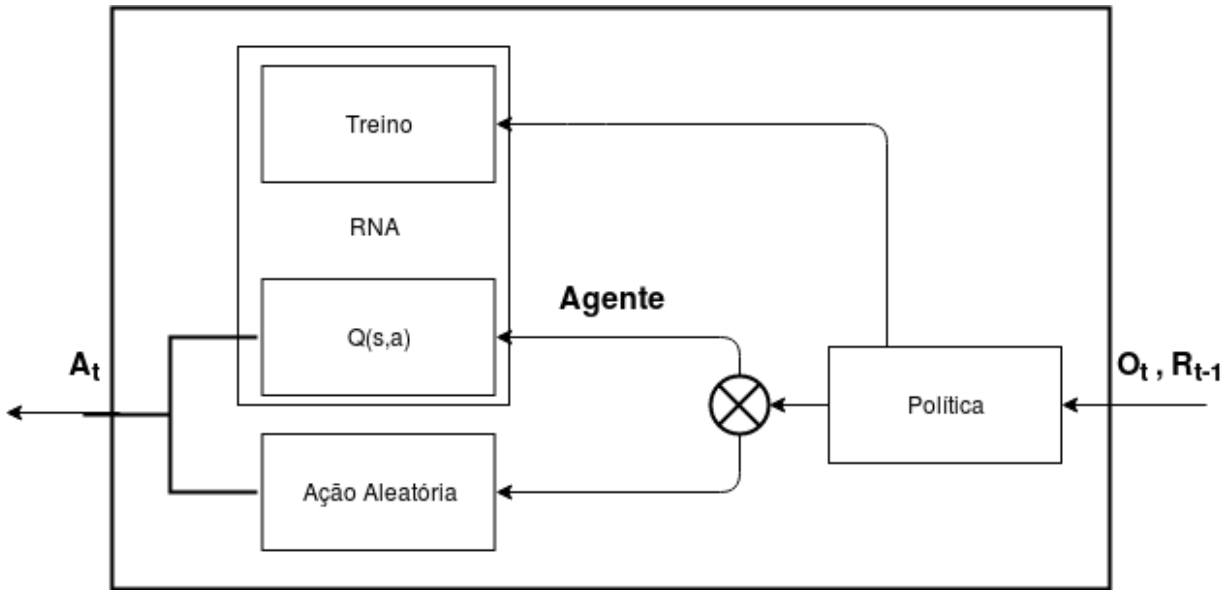


Figura 3.2: Arquitetura do Agente

Graças à utilização do OpenAI Gym, o cálculo de término do episódio não precisa ser executado pelo agente, pois a interface já fornece um aviso de encerramento. Seria possível fazer com que o agente realizasse um cálculo próprio de encerramento e solicitar que o ambiente seja reiniciado, porém optou-se por utilizar a padronização fornecida pela interface para evitar ter-se que obter esta informação a partir da imagem do jogo, visto que decidiu-se por não utilizar as informações de depuração fornecidas pela ferramenta OpenAI Gym neste trabalho. Ao receber-se a notificação de que o episódio se encerrou, faz-se com que o ambiente reinicie o jogo.

A Figura 3.3 ilustra a arquitetura completa do sistema desenvolvido, incluindo a comunicação entre o ambiente e o agente conforme a estrutura clássica de aprendizagem por reforço, as configurações utilizado por ambos para determinar o jogo executado, e a estrutura interna do agente.

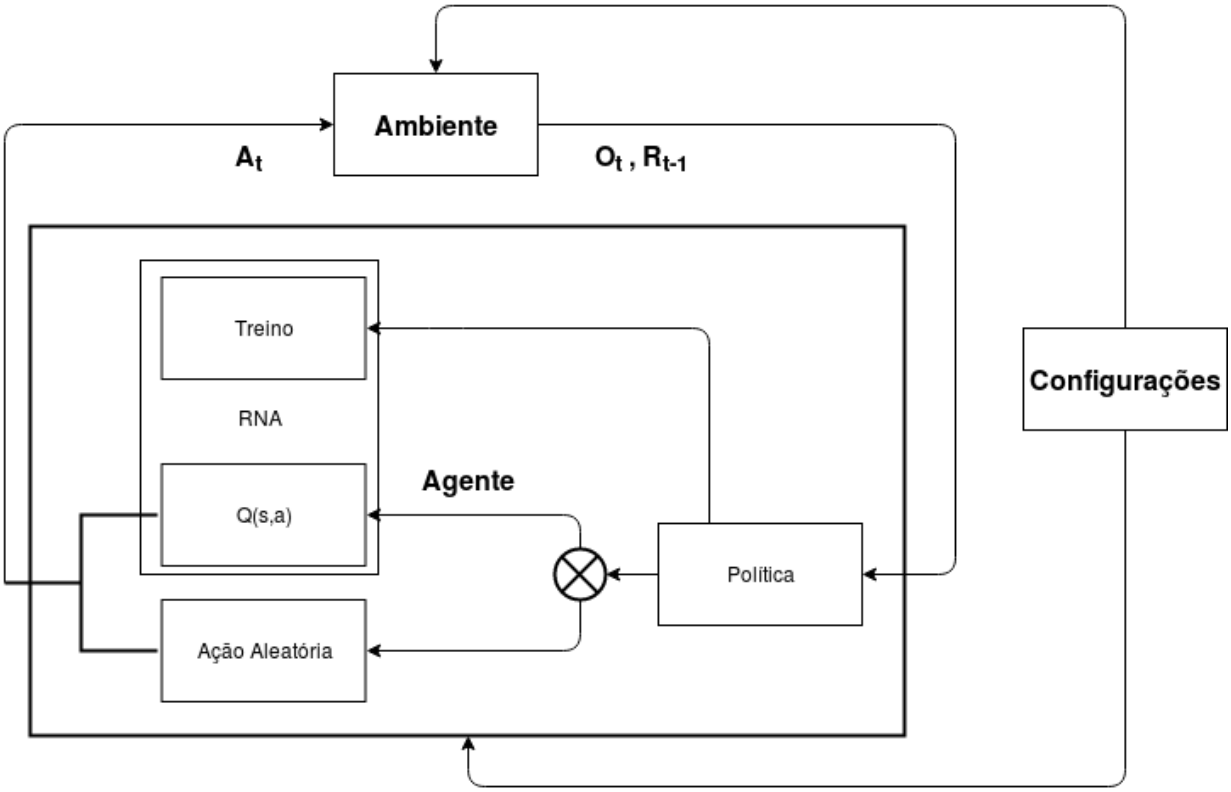


Figura 3.3: Arquitetura do sistema de aprendizagem

3.4 Estado do agente

A interface da ferramenta do OpenAI Gym fornece a observação do ambiente O_t. O agente deve estimar seu estado S_t^a utilizando os dados obtidos nestas leituras, em alguma função da história H_t, sendo $S_t^a = F(H_t)$.

Nos ambientes selecionados, os jogos *Enduro*, *Ms. Pacman*, *Breakout* e *Pong*, as observações recebidas do ambiente tratam-se de imagens RGB da tela dos jogos. As imagens possuem altura e largura de 210 e 160 pixeis. Uma imagem em RGB possui 210×160×3 valores,totalizando uma matriz de 100.800 elementos. Como cada elemento da matriz possui um valor entre 0 a 255 representando a intensidade da cor, a quantidade de estados possíveis é de 256^{100800}. Embora não pretenda-se visitar todos os estados possíveis para treinar-se o agente graças ao uso da RNA, 100800 elementos ainda é um número muito alto de informações para ser processado em tempo viável, com os recursos computacionais disponíveis. Devido ao grande volume de informação, tratou-se as imagens recebidas para diminuir a quantidade de informação e acelerar o treino da rede neural utilizada.

Primeiramente, transformou-se as imagens de RGB para escala de cinza, considerando a hipótese de que não haveria perda significante de informação útil para o agente [3]. Desta forma, reduziu-se a quantidade de pixels em um terço, pois os três canais de cores se tornaram um único canal em escala de cinza. A quantidade de pixels processados diminuiu de de 210×160×3 para 210×160, fazendo com que o tamanho da imagem caísse de 100.800 para 33.600.

As imagens da tela do jogo possuem informações de pontuação, número de vidas e interface que não afetam diretamente o andamento dos jogos. Por isso, cortou-se as extremidades da imagem adquirida, conforme o jogo sendo executado.

Para reduzir de forma ainda mais significante o número de dados de entrada, compactou-se a imagem do jogo utilizando um filtro *anti-aliasing* reduzindo cada um dos eixos da imagem em um terço do seu tamanho original, fazendo uma redução de 9:1 da imagem de entrada. Com isso, o estado S_t^a possui dimensões distintas conforme o jogo executado. Em todos os jogos utilizados, após o tratamento da imagem, o estado do agente é pelo menos 27 vezes menor que a imagem original, o que corresponde a 3733 elementos.

3.5 Função Q

A função $Q(s)$ mapeia o estado do agente à uma ação. Embora seja teoricamente possível fazer um mapeamento linear do estado à ação para recompensas esperadas em uma tabela, optou-se por não fazê-lo neste trabalho, visto que seria inviável mapear todas as combinações de estados possíveis, mesmo após o tratamento de imagem. De fato, por transformar-se a imagem para escala de cinza, cada pixel possui um valor real entre 0 e 1, inviabilizando o mapeamento direto dos estados em uma tabela. Um agente utilizando um mapeamento direto entre o par estado-ação e valores Q dificilmente convergiria nestas condições.

Para contornar o grande número de estados possíveis, utilizou-se uma rede neural artificial para fazer o mapeamento estado-ação. A rede neural foi utilizada como a função valor $v(s, a)$ para prever a recompensa imediata esperada ao executar uma ação a_t em um estado s.

A rede neural implementada tem como entrada o estado do agente S_t^a e a ação executada pelo agente naquele estado a_t. A saída da rede é um valor único dado pela atualização do valor da função $Q(s, a)$, através da Equação (2.22).

A função Q implementada realiza a comparação de todas as ações possíveis no espaço amostral de ações do jogo para o estado atual do agente, selecionando a ação com o maior valor de recompensa esperada.

Tanto o estado do agente quanto a ação ilustrados na Figura 3.4 foram implementados como vetores. O estado é um vetor contendo a imagem em escala de cinza e compactada do jogo. Já a ação é um vetor de valores binários de tamanho igual ao espaço de ações do jogo atual sendo executado. Cada ação corresponde a um elemento do vetor e a ação selecionada será a única que possuirá valor 1, indicando sua seleção. Desta forma, cada ação possui pesos diferentes e independentes conectando-os à primeira camada da RNA.

3.6 Ambientes

O OpenAI Gym possui uma gama diversa de ambientes, que variam desde problemas de controle simples, como o pêndulo invertido, até problemas de robótica complexos, tais como

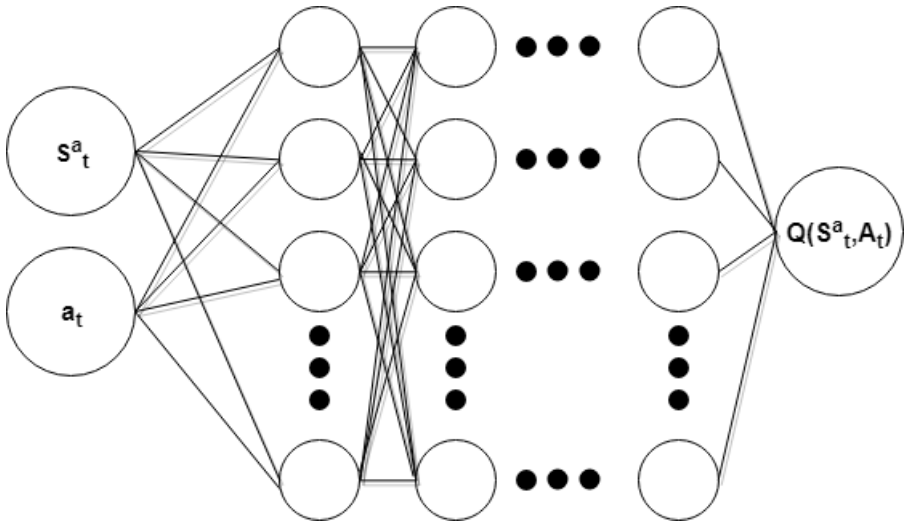


Figura 3.4: RNA utilizada com entradas S_t^a e a_t e saída Q(s,a)

caminhada e levantar de robôs humanoides, trajetória de robôs manipuladores e manipulação de objetos em mãos robóticas, conforme mostra a Figura 3.5. Neste trabalho, optou-se por utilizar ambientes de jogos eletrônicos da coleção Atari, composta por jogos clássicos de da plataforma Atari, devido à simplicidade dos jogos.

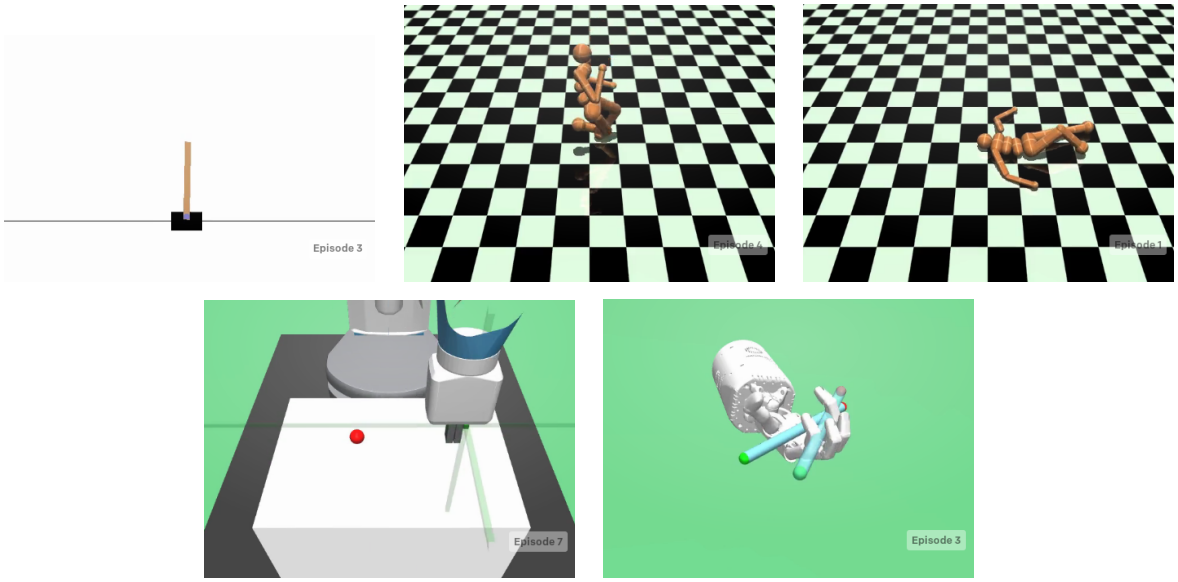


Figura 3.5: Exemplos de ambientes do OpenAI Gym

Implementou-se um arquivo de configurações para realizar a troca entre ambientes de jogo. Neste arquivo, armazena-se todas as informações que variam entre os jogos. Para trocar o ambiente de testes, basta trocar o nome do jogo selecionado dentro do arquivo de configuração, e tanto o ambiente quanto o agente estarão prontos para executar os testes.

Os ambientes fornecidos pelo OpenAI Gym se diferenciam em dois aspectos relevantes para o agente: o tamanho do espaço de ações que o agente pode executar e o tamanho da imagem do jogo. Ambas informações estão armazenadas no arquivo de configurações para conhecimento do

agente. Estas duas informações não alteram o desenvolvimento da interface do ambiente. Esta interface utiliza o arquivo apenas para definir o jogo selecionado.

Para testar o sistema proposto, selecionou-se quatro ambientes da plataforma Atari disponíveis pela ferramente OpenAI Gym. Optou-se por selecionar jogos de Atari devido à simplicidade dos jogos dessa plataforma. Os jogos selecionados para testar e o algoritmo desenvolvido neste trabalho foram o Enduro, MsPacman, Breakout e Pong. Dentro dos jogos de Atari disponíveis na plataforma OpenAI Gym, estes jogos foram escolhidos arbitrariamente, apenas para que houvessem jogos de diferentes estilos para treinar o agente. O capítulo seguinte mostra as considerações feitas para testar-se o agente em cada um dos jogos, assim como os experimentos realizados e resultados obtidos.

Capítulo 4

Experimentos

Neste capítulo, descrevem-se os experimentos realizados para validar o modelo proposto no capítulo anterior. Apresenta-se os jogos digitais aos quais o agente foi submetido a jogar. Para cada um dos jogos, apresenta-se e analisa-se os resultados obtidos.

Os experimentos realizados tem como objetivo responder as seguintes perguntas:

- O sistema implementado faz com que o agente aprenda a selecionar ações que maximizem sua recompensa?
- O sistema implementado é genérico o suficiente para funcionar em ambientes distintos sem a alteração no código do agente?

4.1 Ambientes de Testes

A validação da metodologia apresentada neste trabalho foi realizada através de simulações computacionais em quatro ambientes de testes distintos, os jogos *Enduro*, *Ms. Pacman*, *Breakout* e *Pong*. Ao trocar-se o jogo, reinicia-se o agente para que ele não utilize a mesma função $Q_a(s)$ aprendida no jogo anterior. Para que o sistema execute um jogo diferente, basta trocar a seleção do jogo no arquivo de configuração implementado. O arquivo de configuração possui as seguintes informações referentes ao jogo:

- Nome do ambiente no OpenAI: Informação utilizada para selecionar o jogo através da interface do OpenAI Gym. Utilizado apenas pelo ambiente.
- Tamanho da imagem original: Dimensões da imagem, em pixels, antes do pré-processamento. Não é utilizado pelo agente nem pelo ambiente.
- Tamanho de S_t^a: Dimensões da observação do ambiente após ser tratada pelo agente. Utilizado tanto pelo agente como pelo ambiente.
- Espaço de ações: Um índices de ações que o agente pode tomar naquele jogo específico. Utilizado apenas pelo agente.

As informações de tamanho da imagem e espaço de ações podem ser adquiridas através da interface implementada pelo OpenAI Gym. Contudo, como o agente e ambiente são executados paralelamente, e porque não deseja-se implementar comunicação entre os sistemas que não possa ser reproduzida por um sensor ou um atuador, optou-se por utilizar estar informações nas configurações do jogo.

4.2 Roteiro dos testes

O agente foi treinado em quatro ambientes de teste distintos, os jogos *Enduro*, *Ms. Pacman*, *Breakout* e *Pong*.

Para implementar-se o Q-Learning, necessita-se definir os parâmetros fator de aprendizagem γ_{agente} e fator de desconto α_{agente} do agente. Além disso, é necessário configurar os parâmetros da rede neural, fator de desconto γ_{rede}, o número de camadas ocultas na rede e a quantidade de neurônios N_i por camada oculta i. Não é necessário especificar-se a camada de entrada e de saída, pois estas sempre terão as dimensões da entrada S_t^a e da saída R_t, respectivamente.

Para cada jogo, realizou-se testes com parâmetros distintos, a fim de ver quais parâmetros resultam nos melhores resultados. Cada conjunto de parâmetros foi categorizado como um experimento distinto a ser realizado com cada um dos jogos. Os parâmetros dos experimentos foram escolhidos arbitrariamente, e estão descritos na Tabela 4.1.

Tabela 4.1: Parâmetros dos experimentos

Experimento	γ_{agente}	α_{agente}	γ_{rede}	N1	N2	Episódios
1	0.9	0.65	0.5	10	5	1000
2	0.65	0.9	0.5	10	5	1000

Em cada experimento, realizam-se episódios, onde o agente joga uma partida do início ao fim. O episódio se encerra quando o agente ganha ou perde o jogo. Em cada jogo, realizou-se o treino ao agente ao longo de 1000 episódios. Tendo em vista que trabalhos semelhantes obtiveram resultados conclusivos com apenas 500 episódios [34], optou-se por utilizar 1000 episódios para garantir que seria suficiente para observar se houve ou não aprendizagem no experimento. O valor exato de 1000 episódios foi escolhido arbitrariamente. Inicialmente, utilizou-se um fator de aleatoriedade de 100% na escolha das ações, reduzindo este valor em 0.105% a cada episódio, para que os últimos 50 jogos possuíssem aleatoriedade 0, atuando apenas com o conhecimento adquirido pelo agente ao longo do treinamento. Realizou-se uma média da pontuação destes 50 jogos, para que pudessem servir como fonte de avaliação do aprendizado do agente. Utilizou-se uma média de um número relativamente alto de jogos para filtrar casos atípicos em que o agente, por sorte, adquira muitos pontos, ou por azar, faça poucos pontos.

Para cada jogo, dispõe-se de um gráfico distinto da evolução da pontuação do agente para cada um dos experimentos realizados. Os resultados obtidos são dispostos nos gráficos, e fez-se

uma regressão dos dados para descobrir se houve ou não aprendizado do agente. Caso a curvatura da regressão seja positiva, o agente melhorou o seu desempenho com suas experiências, indicando que houve aprendizado. Caso contrário, tem-se a indicação de que não houve aprendizado.

Para servir de comparativo, realizou-se 50 episódios com uma política completamente aleatória, e utilizou-se uma média simples dos resultados obtidos como referência no gráfico gerado. Esta média pode ser observada como uma linha horizontal no gráfico. Desta forma, é possível observar se o agente aprendeu com suas experiências e consegue obter uma pontuação maior do que comportando-se aleatoriamente.

Como os últimos 50 episódios possuem aleatoriedade 0 na tomada de ações, utiliza-se uma média destes resultados para avaliar o aprendizado do agente. Compara-se os valores obtidos com cada experimento, incluindo o experimento aleatório, com valores obtidos em trabalhos semelhantes. Expõe-se estas comparações em uma tabela distinta para cada jogo utilizado.

4.3 Resultados

Os resultados estão divididos conforme o jogo executado. Apresenta-se a descrição do funcionamento do jogo, regras e forma de pontuação e de derrota. Em seguida, mostra-se a imagem do jogo fornecida pela interface do OpenAI Gym, e ao lado a imagem após o pré-processamento, em escala de cinza e cortada para ser utilizada pelo agente.

Enumera-se as características específicas de cada jogo que necessitaram ser configuradas para conhecimento do agente ou do ambiente. Isto inclui o nome do ambiente a ser carregado na interface do OpenAI Gym. Embora o tamanho da imagem antes do pré-processamento não seja utilizada no programa, incluiu-se esta informação para fins de comparação com o tamanho da imagem após o tratamento.

Apresenta-se, também, o espaço de ações do agente no jogo. O agente não precisa saber o significado de cada uma das ações, visto que deve aprender através da experimentação. Contudo, o agente precisa saber a quantidade de ações possíveis a serem executadas. As ações são enumeradas, e esta enumeração deve ser padronizada entre o agente e o ambiente. Por isso, mostra-se o espaço de ações possíveis a serem tomadas pelo agente, enumeradas conforme apresentadas pela interface do OpenAI Gym.

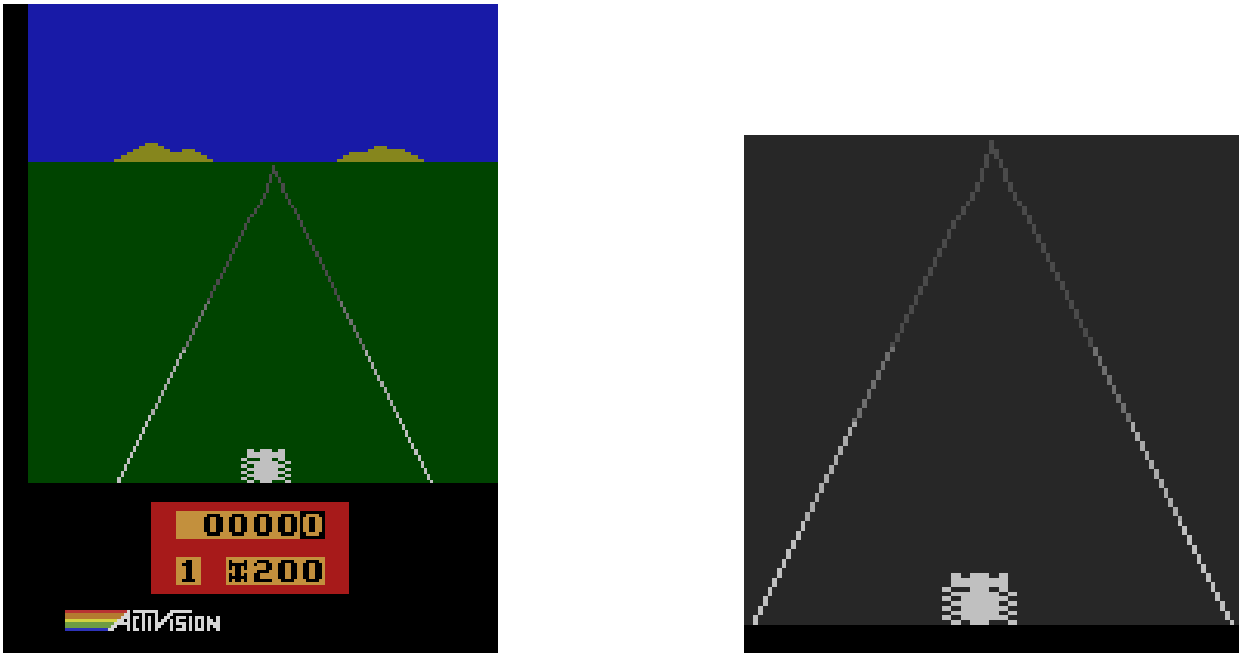
Para cada jogo, explica-se como são as recompensas fornecidas pela interface do OpenAI Gym e as adaptações feitas em cima das recompensas obtidas, justificando-as. No final de cada sessão, apresenta-se o gráfico mostrando a evolução do desempenho do agente ao longo dos episódios. Há uma curva para cada experimento feito no ambiente, cortados por uma reta horizontal representando o desempenho médio do algoritmo aleatório. Inicialmente, o comportamento do agente é predominantemente aleatório. Para visualizar de forma mais clara a evolução do desempenho do agente, realizou-se uma regressão linear dos dados obtidos.

4.3.1 Enduro

Enduro é um jogo eletrônico que simula uma corrida de carros em 2 dimensões, com uma perspectiva de falsa profundidade vista de trás do carro, conforme mostrado na Figura 4.1a. O jogador joga sozinho, contra outros carros controlados pelo próprio sistema do jogo. O jogador pode movimentar o carro horizontalmente e aumentar sua aceleração, manobrando o carro em um percurso desconhecido. Para passar de nível, o jogador deve ultrapassar uma quantidade arbitrária de carros de acordo com o nível atual, antes do final do tempo do nível. Caso o jogador não consiga ultrapassar o número de carros determinados antes do término da corrida, o jogador perde o jogo.

- Nome do ambiente no OpenAI: Enduro-v0
- Tamanho da imagem original: 210×160×3
- Tamanho de S_t^a: 37×35
- Espaço de ações: [0,1,2,3,4,5,6,7,8]

A Figura 4.1a mostra o jogo assim como é percebido, ou seja, mostra a imagem fornecida pelo ambiente, a interface do OpenAI Gym. Já a Figura 4.1b mostra a imagem do jogo após o pré-processamento, a imagem que o agente realmente utiliza para tomar suas decisões.



(a) Enduro simulado no OpenAI

(b) Enduro após o tratamento de imagem

Figura 4.1: Capturas de tela do jogo Enduro

No Enduro, a interface do OpenAI fornece recompensas positivas quando o carro sobe sua posição na corrida, ou seja, ultrapassa um carro a sua frente. Quando o jogador é ultrapassado por outro carro, ou seja, quando sua posição geral na corrida diminui, recebe uma recompensa

negativa. Não alterou-se este comportamento, utilizando as recompensas assim como vinham da interface.

A evolução da pontuação do agente foi armazenada ao longo dos episódios e está representada no gráfico da Figura 4.2, do episódio 0 ao 1000. No gráfico, apresentam-se todas as informações armazenadas ao longo dos 1000 episódios, e a regressão de todos os pontos adquiridos ao longo do experimento.

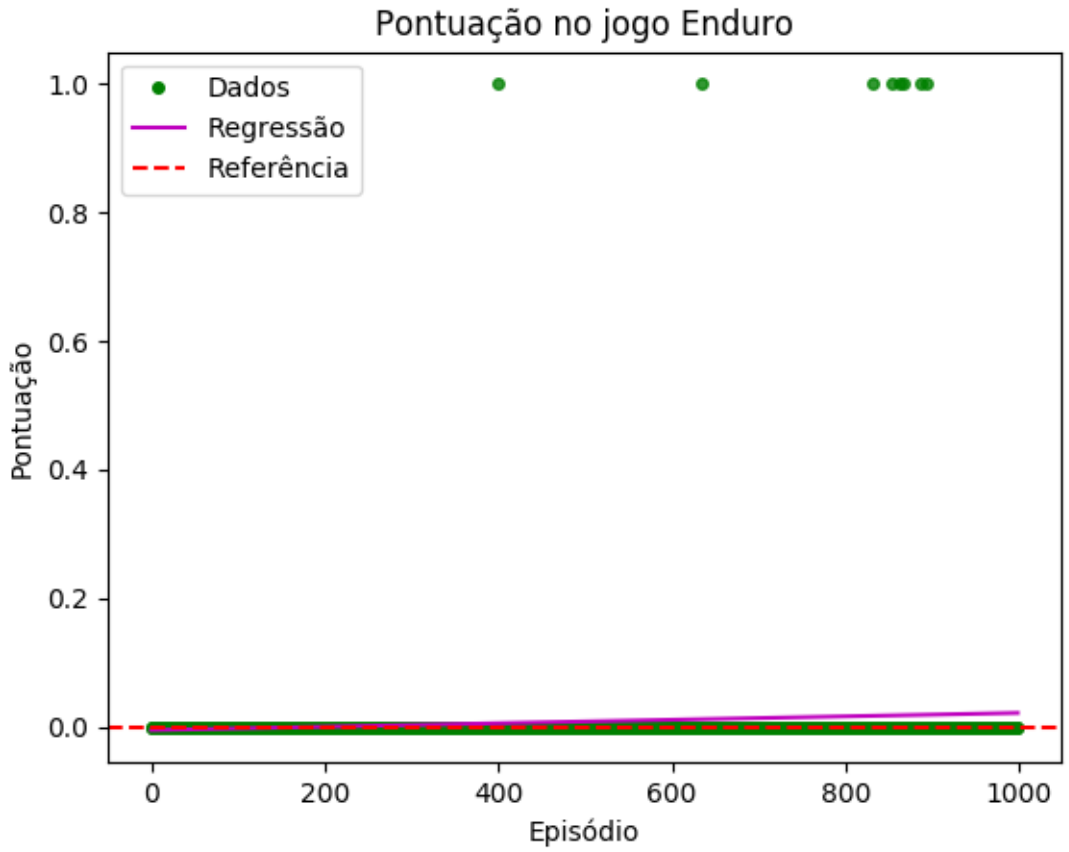


Figura 4.2: Evolução do agente no jogo Enduro no Experimento 1

A Figura 4.2 mostra claramente que não houve aprendizagem do agente ao longo do experimento realizado. A inclinação da curva que representa o seu crescimento é de 2.68×10^{-5}, um valor que pode ser interpretado como 0.

Este resultado é compreensível ao observar-se a distribuição dos resultados obtidos. No início do experimento, quando o agente possuía um comportamento completamente aleatório, o agente não conseguiu obter nenhuma recompensa. Por isso, não foi possível aprender a selecionar a melhor ação, visto que, na visão do agente, qualquer ação selecionada corresponde a um mesmo valor de recompensa esperada, 0. Ao longo de todos os 1000 episódios, em apenas 7 deles o agente obteve uma pontuação diferente de 0, e mesmo nestes casos a pontuação foi de apenas 1 ponto, a menor recompensa distinta de 0 possível de se obter no jogo.

Para observar algum aprendizado do agente, seria necessária uma adaptação da função de

recompensa para utilizar mais do que a pontuação do jogo, para que o agente possua algum *feedback* ao selecionar ações e possa, de fato, aprender com elas. Uma possibilidade é utilizar a distância percorrida como recompensa, para que o agente aprenda a ir mais rápido, visto que este é um jogo de corrida. Quanto maior a diferença entre a distância percorrida atual e a anterior, mais rápido está o carro, e consequentemente, maior a recompensa. Desta forma, talvez o agente consiga, eventualmente, aprender a ultrapassar outros carros e, de fato, conseguir vencer o jogo. Contudo, a interface do OpenAI Gym não fornece estas informações diretamente, sendo necessário o tratamento da imagem original do jogo para obter estas informações [3].

4.3.2 Ms. Pacman

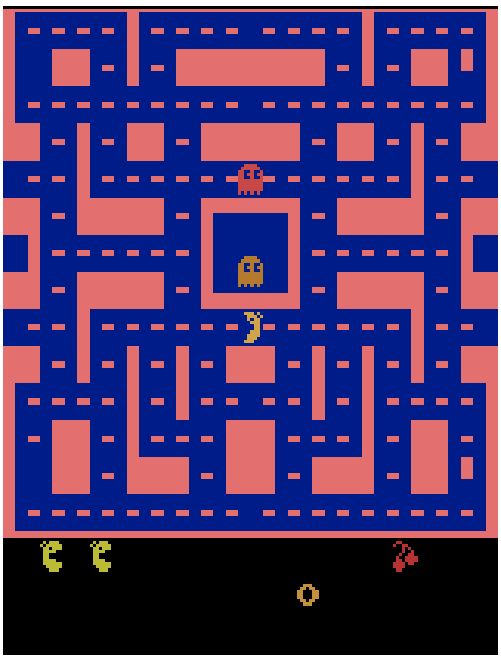
Ms. Pacman é um jogo eletrônico de 2 dimensões no qual um jogador visa coletar moedas ao longo do cenário sem ser capturado pelos inimigos, fantasmas que o seguem. O jogador controla o personagem verticalmente e horizontalmente, e os inimigos são controlados pelo sistema do jogo. Caso um inimigo entre em contato com o jogador, este perde uma vida. Ao perder todas as vidas o jogo se encerra. Além de passar por cima das moedas, outro modo de coletar pontos no jogo é capturando um fantasma. O jogador consegue capturar os fantasmas caso consuma uma *power pill*, que são representadas por moedas maiores espalhadas nos quatro cantos da tela. Contudo, o efeito da *power pill* se esgota após um intervalo de tempo, e o jogador perde a capacidade de capturar fantasmas. Enquanto o efeito da *power pill* está ativo, os fantasmas fogem do jogador ao invés de persegui-lo.

- Nome do ambiente no OpenAI: MsPacman-v0
- Tamanho da imagem original: 210×160×3
- Tamanho de S_t^a: 58×53
- Espaço de ações: [0,1,2,3,4,5,6,7,8]

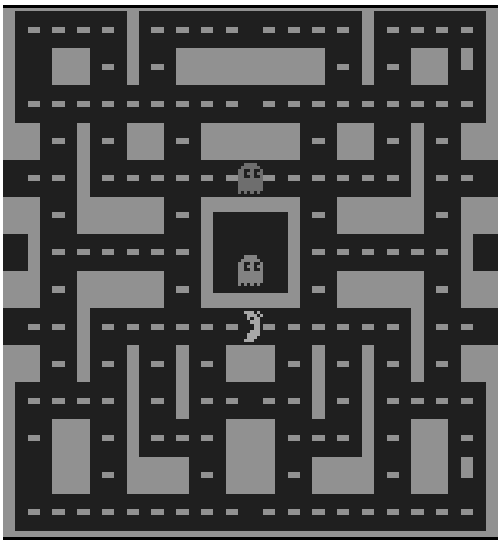
A Figura 4.3a mostra o jogo assim como é percebido, ou seja, mostra a imagem fornecida pelo ambiente, a interface do OpenAI Gym. Já a Figura 4.3b mostra a imagem do jogo após o pré-processamento, a imagem que o agente realmente utiliza para tomar suas decisões.

O jogo recompensa o jogador sempre que o mesmo passa por um espaço onde há uma moeda, o concedendo 1 ponto positivo de recompensa. Caso o jogador consiga capturar um fantasma com o uso de uma *power pill*, o jogo o recompensa com 300 pontos positivos. A interface do OpenAI Gym não fornece recompensas negativas quando o personagem morre, visto que a morte não afeta o placar do jogo. Contudo, para ensinar ao agente que o mesmo não deve ser capturado pelos fantasmas, atribuiu-se uma recompensa de -10 pontos quando o jogador perde todas as três vidas.

A evolução da pontuação do agente foi armazenada ao longo dos episódios e está representada nos gráficos das Figuras 4.4a e 4.4b, que representam os experimentos 1 e 2, respectivamente. No gráfico, apresentam-se as informações armazenadas em intervalos de 10 episódios para não causar poluição visual, e a regressão de todos os pontos adquiridos ao longo dos 1000 episódios.



(a) Ms. Pacman simulado no OpenAI



(b) Ms. Pacman após o tratamento de imagem

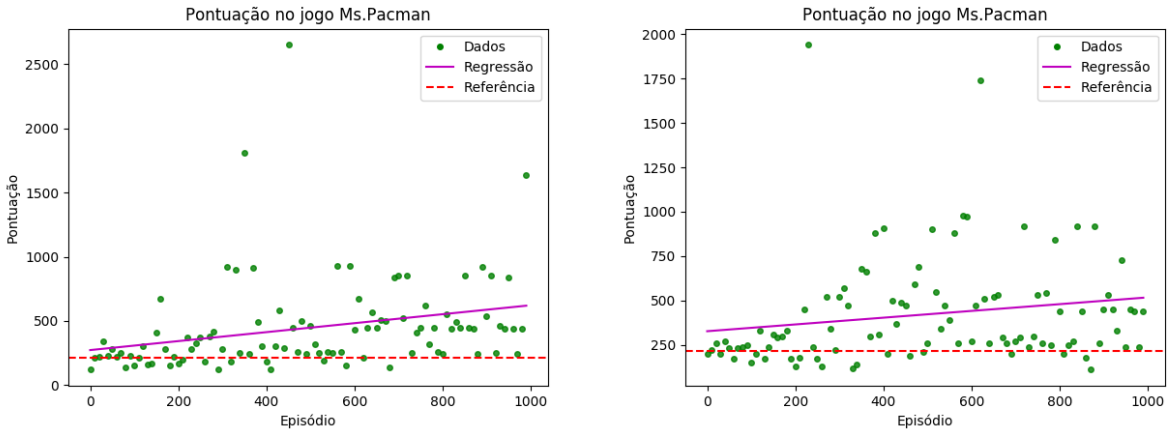
Figura 4.3: Capturas de tela do jogo Ms. Pacman

As Figuras 4.4a e 4.4b mostram a evolução do desempenho do agente ao longo de 1000 episódios do jogo *Ms. Pacman*. Em ambos os experimentos, a regressão linear dos dados obtidos possui inclinação positiva, com valores aproximados de 0.35 e 0.27 para os experimentos 1 e 2. Esta inclinação é uma indicação de que houve um crescimento na capacidade do agente de obter pontuação no jogo, pois o agente tende a obter uma pontuação maior com o passar dos episódios.

O jogo *Ms. Pacman* não exige precisão ou velocidade de resposta do jogador. Além disso, o mapa inteiro é inicialmente coberto de moedas, de forma que o jogador recebe recompensas imediatas a cada ação, a não ser que retorne a um espaço já percorrido. Por estes motivos, o ambiente é favorável a avaliação do desempenho de ações discretas, de forma que o agente é capaz de julgar com maior facilidade ações individuais, em detrimento de jogos que avaliam combinações de ações.

Outro aspecto do jogo é que o jogador não precisa executar ações com precisão, visto que o jogo não envolve reflexos do jogador. O agente consegue aprender gradativamente ao longo dos episódios ao executar ações que embora não o levem a solucionar o jogo com maestria, o deixem um pouco mais próximo de seu objetivo, o que pode ser observado com o crescimento gradual da pontuação.

Devido aos resultados positivos obtidos com os experimentos realizados com o jogo *Ms. Pacman*, acredita-se que a representação do estado do agente S_t^a escolhida tenha sido suficiente para que o agente obtivesse as informações necessárias para aprender o jogo. Em outras palavras, os estados S_{ti}^a eram markovianos.



(a) Experimento 1 do Ms. Pacman

(b) Experimento 2 do Ms. Pacman

Figura 4.4: Experimentos do Ms. Pacman

4.3.3 Breakout

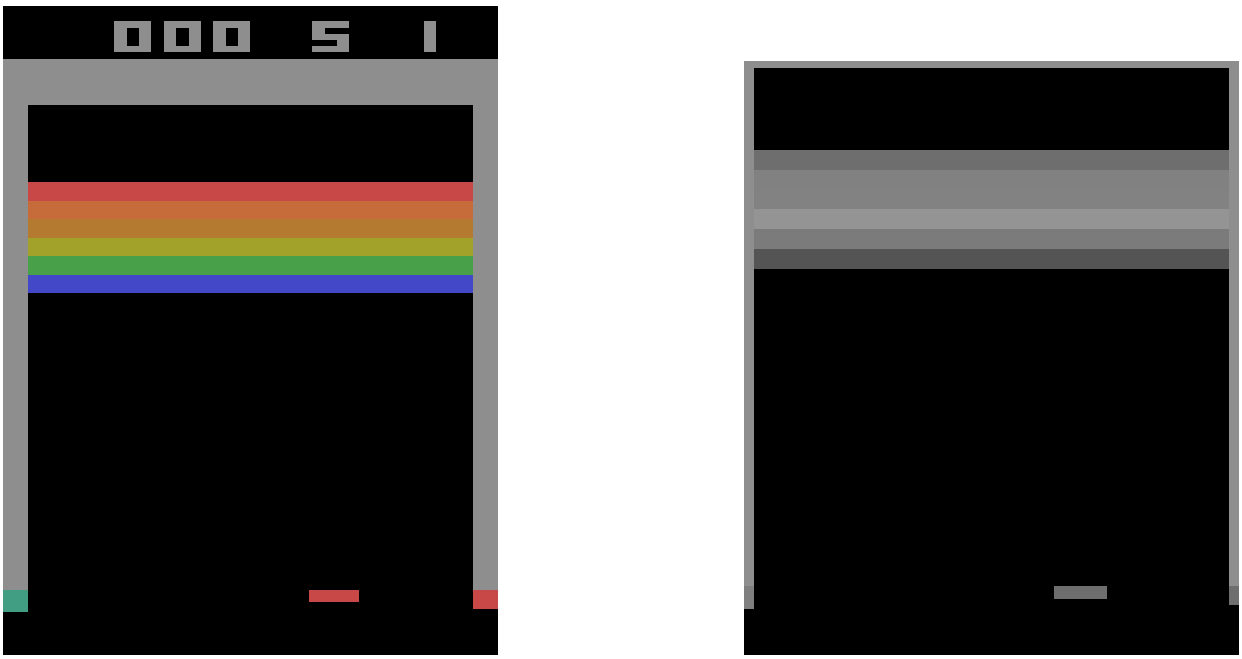
Breakout é um jogo eletrônico de 2 dimensões no qual um jogador busca destruir uma camada de tijolos alinhados ao na parte superior da tela. Uma bola quica pela tela e destrói tijolos ao entrar em contato com eles. É um jogo individual, onde o jogador controla uma raquete, movendo-a horizontalmente na parte inferior da tela. O jogador perde uma vida caso não rebata a bola e ela passe do limite inferior da tela. Ao perder todas suas vidas, o jogo acaba. Quanto mais tijolos o jogador destruir, maior a quantidade de pontos que fará.

A Figura 4.5a mostra o jogo assim como é percebido, ou seja, mostra a imagem fornecida pelo ambiente, a interface do OpenAI Gym. Já a Figura 4.5b mostra a imagem do jogo após o pré-processamento, a imagem que o agente realmente utiliza para tomar suas decisões.

- Nome do ambiente no OpenAI: Breakout-v0
- Tamanho da imagem original: 210×160×3
- Tamanho de S_t^a: 60×50
- Espaço de ações: [0,1,2,3]

O jogo recompensa o jogador sempre que o mesmo destrói um tijolo na área superior da tela, concedendo-o 1 ponto positivo de recompensa. A interface do OpenAI Gym não fornece recompensas negativas quando o personagem perde todas as vidas, visto que a morte não afeta o placar do jogo. Contudo, para ensinar ao agente que o mesmo deve continuar jogando o máximo de tempo possível, atribuiu-se à perde de todas as vidas uma recompensa de -10 pontos.

A evolução da pontuação do agente foi armazenada ao longo dos episódios e está representada nos gráficos das Figuras 4.6a e 4.6b, que representam os experimentos 1 e 2, respectivamente. No

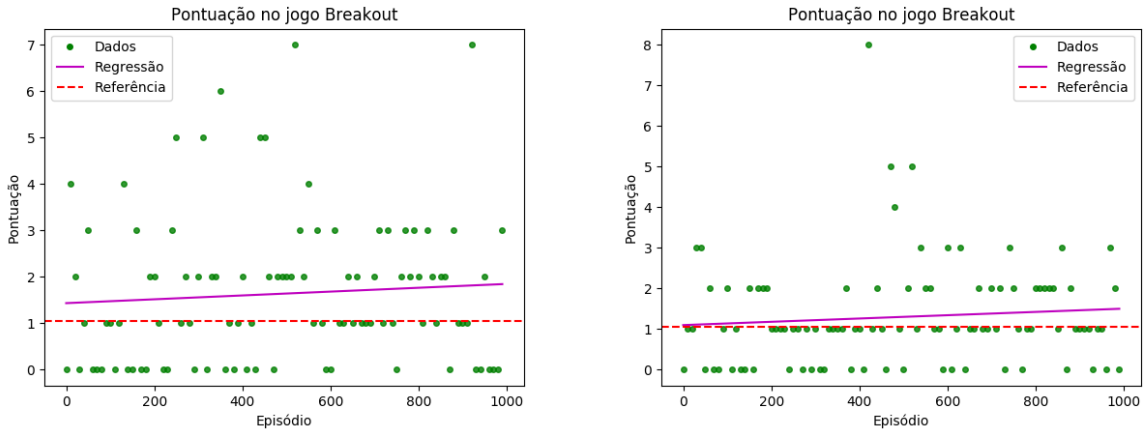


(a) Breakout simulado no OpenAI

(b) Breakout após o tratamento de imagem

Figura 4.5: Capturas de tela do jogo Breakout

gráfico, apresentam-se as informações armazenadas em intervalos de 10 episódios para não causar poluição visual, e a regressão de todos os pontos adquiridos ao longo dos 1000 episódios.



(a) Experimento 1 do Breakout

(b) Experimento 2 do Breakout

Figura 4.6: Experimentos do Breakout

Ambos experimentos obtiverem resultados semelhantes. Conforme pode-se observar nos gráficos de desempenho do agente, a pontuação do agente no jogo Breakout não superou de forma significativa a pontuação de um agente aleatório. Embora possa-se observar que a reta que define a pontuação obtida pelo agente tenha inclinação positiva em ambos os resultados, a inclinação da reta é aproximadamente de apenas 0.0004 para os dois experimentos . Este valor é tão próximo de 0 que não pode-se de fato afirmar que o agente tenha aprendido a melhorar seu desempenho com

o passar do tempo. Muito provavelmente, este valor é positivo pois é extremamente improvável que a média de um número finito de experimentos aleatórios, como os realizados, seja exatamente 0.

Ao longo dos experimentos, observou-se que o agente priorizava sempre a tomada de uma mesma ação, independente do estado alcançado. Possivelmente, a RNA treinada não foi capaz de diferenciar entre os diversos estados, e desta forma configurou pesos menores para as associações dos neurônios voltados ao estado do ambiente. Como o agente não aprende a diferenciar entre os estados do jogo, os pesos dos neurônios relacionados às entradas do estado do agente tornam-se pequenos. Sem diferenciar entre os estados, o agente aprende simplesmente que uma ação representa uma recompensa esperada maior que as demais. Como o agente utiliza uma política determinística, tende a selecionar sempre a mesma ação neste caso, exceto quando explora o ambiente. Contudo, como o caráter exploratório diminui com o tempo, a seleção de ações diferentes tornam-se muito breves para ter um impacto real no jogo.

Acredita-se que a RNA não tenha observado relações entre o estado do agente e as recompensas obtidas porque o estado do agente S_t^a não foi markoviano para este jogo. Por utilizar-se uma única imagem da tela do jogo como estado do agente, o agente não consegue inferir informações como velocidade e aceleração da bola. Para o jogo *Breakout*, estas informações são de extrema importância, visto que o jogador deve mover-se em direção a bola para poder rebatê-la.

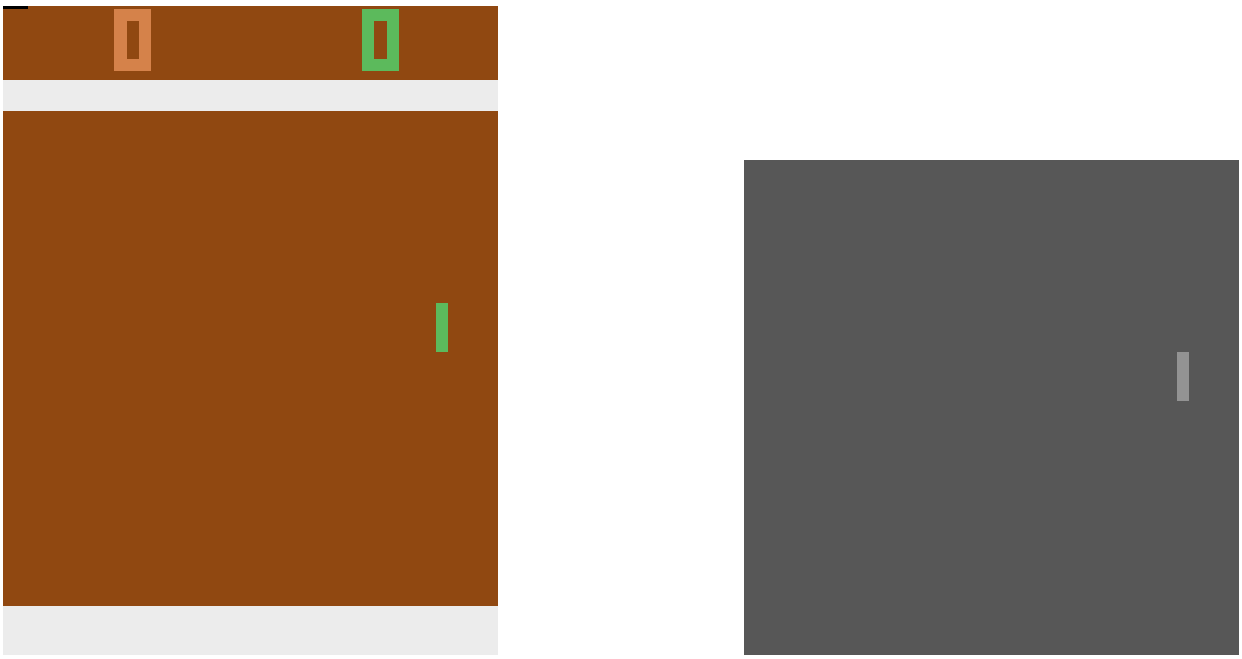
Além disso, o agente pode não ter sido capaz de realizar a associação do estado com a ação porque o jogo *Breakout* não recompensa o agente por ações que deixem o agente mais próximo do seu objetivo, a não ser que o agente de fato o alcance. O agente não é recompensado por andar em direção a bola caso não consiga alcançá-la a tempo de rebatê-la. Desta forma, não é possível o aprendizado gradual do agente, visto que o agente apenas aprenderá caso execute, ao explorar o ambiente, ações que o façam marcar o ponto, que é a única forma de obter recompensas positivas neste jogo.

4.3.4 Pong

Pong é um jogo eletrônico que simula um jogo de tênis de mesa em 2 dimensões. O jogador controla uma das raquetes, movendo-a verticalmente no lado direito da tela. O jogador compete contra outro que controla a outra raquete do lado esquerdo da tela, neste caso, uma inteligência artificial pré-implementada pelo jogo. Ao acertar a bola, um jogador a arremessa para o outro lado do campo, na intenção de fazer um ponto caso o adversário não consiga rebater a bola. A bola aumenta de velocidade cada vez que rebatida, reiniciando sua velocidade velocidade caso um jogador falhe em acertá-la. O objetivo do jogo é fazer 21 pontos. Um ponto é feito quando o adversário falha em rebater a bola, que some ao atravessar o limite do seu lado do campo.

A Figura 4.7a mostra o jogo assim como é percebido, ou seja, mostra a imagem fornecida pelo ambiente, a interface do OpenAI Gym. Já a Figura 4.7b mostra a imagem do jogo após o pré-processamento, a imagem que o agente realmente utiliza para tomar suas decisões.

- Nome do ambiente no OpenAI: Pong-v0



(a) Pong simulado no OpenAI

(b) Pong após o tratamento de imagem

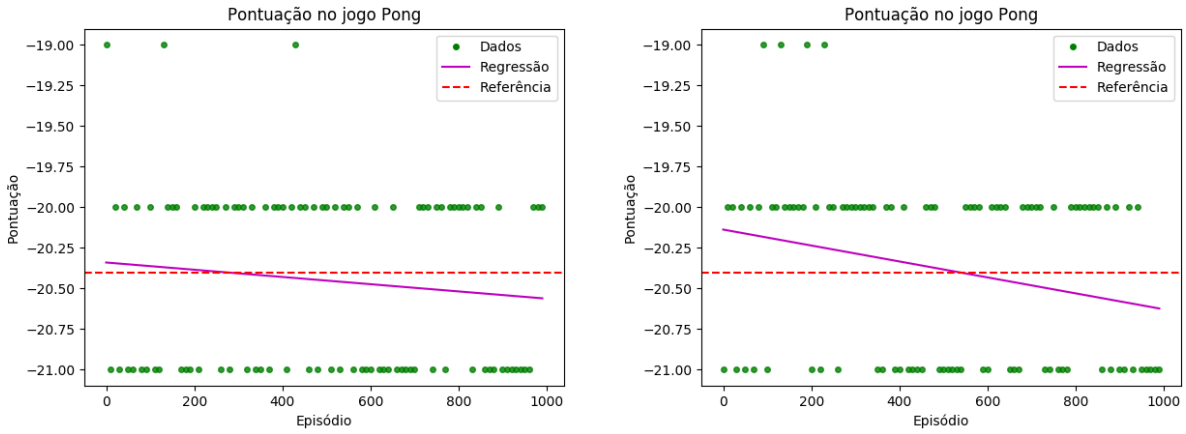
Figura 4.7: Capturas de tela do jogo Pong

- Tamanho da imagem original: 210×160×3
- Tamanho de S_t^a: 65×53
- Espaço de ações: [0,1,2,3,4,5]

O jogo recompensa o jogador sempre que o mesmo marca um ponto, ou seja, quando a bola atravessa o lado esquerdo da tela devido a falha do adversário em rebatê-la, concedendo ao jogador 1 ponto positivo de recompensa. Além disso, o jogo fornece uma recompensa negativa de -1 ponto quando o jogador falha em rebater a bola. Neste trabalhou, utilizou-se a função de recompensa fornecida pela interface do OpenAI Gym conforme descrita.

A evolução da pontuação do agente foi armazenada ao longo dos episódios e está representada nos gráficos das Figuras 4.8a e 4.8b, que representam os experimentos 1 e 2, respectivamente. No gráfico, apresentam-se as informações armazenadas em intervalos de 10 episódios para não causar poluição visual, e a regressão de todos os pontos adquiridos ao longo dos 1000 episódios.

Embora a pontuação obtida através de uma política aleatória não sirva para indicar se houve aprendizagem, o fato do agente ter obtidos pontuações menores que a de um agente aleatório é uma indicação de que o agente não aprendeu a tomar as ações corretas para maximizar sua recompensa. Mesmo que a inclinação da curva tenha sido negativa nos experimentos realizados, com um valores de -0.0002 e -0.0004, isto não significa que houve um aprendizado negativo, ou que o agente tenha aprendido a não fazer pontos, pois o valor foi muito próximo de 0. É muito improvável que a inclinação de uma reta obtida experimentalmente seja exatamente 0, por isso é compreensível um agente que não demonstre aprendizado obter uma inclinação levemente positiva ou negativa.



(a) Experimento 1 do Pong

(b) Experimento 2 do Pong

Figura 4.8: Experimentos do Pong

Assim como no *Breakout*, observou-se que a RNA aprendeu a executar uma única ação, independente do S_t^a em que o agente se encontra. Mais uma vez, acredita-se que o S_t^a escolhido não foi markoviano. O estado utilizado possui apenas uma única imagem da tela do jogo, de forma que o agente consiga inferir a velocidade e aceleração da bola. Por ser uma simulação eletrônica de um jogo de tênis de mesa, o *Pong* precisaria destas informações para saber a direção da bola e onde melhor posicionar-se para rebater a bola.

O jogo *Pong* não recompensa o agente por ações que deixem o agente mais próximo do seu objetivo, a não ser que o agente de fato o alcance. Caso o agente não faça o ponto, não será recompensado, mesmo que tenha rebatido a bola corretamente previamente. Desta forma, não é possível o aprendizado gradual do agente, visto que o agente apenas aprenderá caso execute, ao explorar o ambiente, ações que o façam marcar o ponto.

4.4 Análise Geral

Os resultados obtidos nos experimentos realizados foram variados, levantando diversas hipóteses sobre o trabalho.

A metodologia implementada treina o agente para executar ações discretas a cada instante de tempo t, conforme o escopo do jogo selecionado. Jogos como o *Breakout* ou *Pong* exigem precisão de movimentação e não recompensam o jogador por movimentos individuais, mas por jogadas completas. Nestes jogos, o agente apenas é recompensado caso faça uma jogada correta, porém não jogadas bem direcionadas, muitas vezes quase corretas, não são recompensadas. Isto dificulta o aprendizado gradual do agente, que apenas aprende caso execute, ao explorar, jogadas complexas e combinadas, pois este deve aprender a executar comportamentos complexos de uma vez. Observou-se que a metodologia aplicada não foi bem sucedida ao trabalhar com jogos desta natureza.

Ao aplicar-se o experimento no jogo *Enduro*, observou-se que o agente não foi capaz de explorar o ambiente de forma a obter recompensas. Sem adquirir recompensas, o agente foi incapaz de demonstrar qualquer nível de aprendizagem. O jogo exige que o jogador execute ações contínuas, como por exemplo acelerar constantemente, o que dificulta a exploração de ações discretas.

Observou-se que, nos jogos *Breakout* e *Pong*, o agente não aprendeu a estimar o Q-valor corretamente. Uma possível explicação para este fenômeno é que o estado do agente utilizado não foi markoviano nestes jogos. Devido a falta de informação essencial do ambiente, a RNA não é capaz de aprender a correlacionar os estados com as recompensas adquiridas. Nestes casos, a RNA passa a desconsiderar a informação do estado atual nas suas previsões, atribuindo pesos pequenos a estas informações.

Embora não tenha sido observado aprendizagem nestes 3 jogos, o agente obteve maior êxito no jogo *Ms. Pacman*. Neste jogo, como o agente é frequentemente recompensado a cada ação tomada e a mecânica do jogo prioriza a estratégia frente à precisão da movimentação, o aprendizado foi mais efetivo.

Outro fator limitante do experimento foi o fato de que o agente era executado de forma assíncrona do ambiente. Para simular um ambiente mais próximo da realidade, o agente precisa interpretar a observação e selecionar uma ação enquanto o jogo continua sendo executado continuamente. Enquanto o agente não seleciona uma nova ação, o ambiente continua recebendo a última ação selecionada pelo agente. O atraso decorrente do treinamento da rede neural e da comunicação entre os sistemas pode também ter sido um fator que afetou o aprendizado do agente nos experimentos realizados.

4.4.1 Comparação dos resultados

A Tabela 4.2 apresenta um comparativo dos resultados obtidos neste trabalho com outros trabalhos da literatura obtidos por sistemas autônomos que jogam os mesmos jogos aqui apresentados. Compara-se os resultados obtidos após o treinamento do agente, e não a curva de aprendizagem dos mesmos.

Tabela 4.2: Resultado dos experimentos

	Ms Pacman	Enduro	Breakout	Pong
Aleatório	213	0	1.06	-20.4
Experimento 1	379.8	0	1.26	-20.84
Experimento 2	467	-	1.26	-20.8
Sarsa [35]	-	129	5.2	-19
Contingency [35]	-	159	6	-17
DQN [35]	-	470	168	20
Ms. Pacman [34]	502.5	-	-	-

Os experimentos de [35] e [34] representados na Tabela 4.2 foram executados utilizando um simulador diferente do apresentado neste trabalho. Por isso, a tabela é utilizada apenas de forma qualitativa, para demonstrar em quais experimentos houve e em quais não houve aprendizado. Através da comparação qualitativa dos experimentos, sabendo quais obtiveram resultados positivos e quais obtiveram resultados negativos, é possível comparar as abordagens e decisões tomadas, mesmo que os algoritmos em si não possam ser comparados.

Na primeira linha da tabela, apresenta-se os resultados obtidos com a implementação de uma política aleatória implementada neste trabalho, para servir de comparativo. As linhas seguintes da tabela apresentam os resultados obtidos nos experimentos realizados neste trabalho conforme a Tabela 4.1. Por fim, apresenta-se os resultados obtidos nos trabalhos [35] e [34].

Os experimentos realizados pela *Deep Mind Technologies* [35] nos jogos *Enduro*, *Breakout* e *Pong* também realizam a conversão da imagem para escala de cinza e recortam-a para utilizar apenas a área útil do jogo. Diferente dos experimentos descritos neste trabalho, esses experimentos utilizam redes neurais convolucionais. Além disso, os autores fixam todas as recompensas positivas em 1 e as recompensas negativas em -1, na expectativa de facilitar a utilização dos mesmos parâmetros de aprendizagem para todos os jogos, porém abrindo mão da diferenciação entre recompensas de diferente escala de grandeza. Por fim, a representação do estado do agente escolhida por [35] é o agrupamento das últimas 4 imagens observados pelo agente, permitindo que o agente obtenha informações de velocidade e aceleração dos objetos na tela.

O trabalho apresentado por [34] baseia-se em [35], e por isto realiza as mesmas decisões descritas acima para testar no ambiente do jogo *Ms. Pacman* fornecido pela interface do OpenAI Gym.

Nos jogos *Enduro*, *Breakout* e *Pong*, o agente desenvolvido neste trabalho obteve pontuações inferiores do que em [35], que utilizou quatro imagens como estado do agente. Já no jogo *Ms. Pacman*, o agente obteve pontuações próximas as observadas por [34], o que sugere que para este jogo o estado do agente com uma única imagem tenha sido markoviano, e que, portanto, este número de imagens seja suficiente para obter as informações no jogo, o que não ocorre nos demais jogos investigados.

Neste capítulo, apresentou-se os resultados obtidos com a metodologia proposta a 4 jogos distintos. O próximo capítulo apresenta a conclusão do projeto como um todo e apresenta sugestões de estudos a serem feitos tendo com base neste trabalho e aponta hipóteses para melhorias a serem feitas no projeto para melhorar os resultados aqui apresentados.

Capítulo 5

Conclusões

Este trabalho apresentou o desenvolvimento de um agente que utiliza aprendizagem por reforço para seleção de ações de um agente autônomo em distintos ambientes simulados desconhecidos. Embora o trabalho tenha sido desenvolvido utilizando ambientes de jogos eletrônicos, a ferramenta de simulação utilizada também permite a simulação em ambientes mais próximos do mundo real, tal como ambientes de robótica tridimensional.

Para validar o algoritmo desenvolvido, escolheu-se trabalhar com a ferramente do OpenAI Gym para simular os jogos eletrônicos a qual o agente foi submetido. A escolha do OpenAI deve-se à padronização do ambiente que a ferramente fornece e pela modularidade inerente da ferramenta.

Como o trabalho foi realizado em ambiente simulado, foi necessário implementar uma forma de comunicação entre o agente e o ambiente. Para isso, usou-se a ferramenta de robótica ROS, permitindo a comunicação assíncrona e modular entre os dois.

O agente desenvolvido foi treinado para tomar decisões a partir de observações da imagem da tela do jogo. As imagens eram pré-processadas pelo agente para acelerar o algoritmo de treino, eliminando informações de cor e reduzindo a qualidade das imagens.

Implementou-se um algoritmo de aprendizagem por reforço, treinando uma rede neural artificial para realizar o mapeamento de um par estado e ação à recompensa esperada por executar aquela ação naquele estado. Desta forma, o agente pode utilizar a rede neural para testar todas as ações possíveis no estado no qual se encontra e selecionar a que o forneça com a maior recompensa.

Tendo desenvolvido o agente, experimentou-se a eficácia de aprendizagem do agente em quatro jogos: *Enduro*, *Ms. Pacman*, *Breakout* e *Pong*. Dentre os ambientes experimentados, o agente apenas mostrou aprender no jogo *Ms. Pacman*. A modelagem do estado do agente como a imagem do jogo no momento atual não foi suficiente para os jogos *Breakout* e *Pong*, que precisavam estimar o vetor velocidade dos componentes na tela. Já no *Enduro*, o agente foi incapaz de explorar estados que o recompensasse, visto que o jogo exige comportamentos complexos para recompensar o jogador.

5.1 Trabalhos Futuros

Para os jogos em que não observou-se aprendizado, espera-se que um resultado melhor seja obtido caso adapte-se a função de recompensa para recompensar o jogador por ações que, embora não o façam marcar pontos no jogo, o deixem mais próximos de fazê-lo.

O agente desenvolvido neste trabalho foi testado com jogos eletrônicos simples, da plataforma Atari. Espera-se que o algoritmo seja capaz de, ou possa ser adaptado para solucionar jogos de maior complexidade.

O agente foi desenvolvido utilizando uma política determinística. Ao ser treinado, sempre selecionava a ação que julgava ser melhor para maximizar suas recompensas. Recomenda-se a adaptação do agente para utilizar uma política estocástica, que ao invés de determinar qual ação o agente deverá tomar, determine a probabilidade do agente em tomar cada ação, de forma a manter naturalmente uma característica exploratória.

Outra adaptação que pode vir a melhorar o desempenho do agente é utilizar as imagens anteriores à atual para modelar-se o estado do agente. Desta forma, o agente pode vir a induzir informações como velocidade e aceleração dos personagens na tela [3]. Para lidar com o processamento adicionar resultante de um uso de mais imagens, pode-se utilizar técnicas de *experience replay* [36], e realizar o aprendizado da rede após o episódio por inteiro, para que o treinamento da rede não afete a velocidade de resposta do agente.

Futuramente, sugere-se utilizar de redes neurais convolucionais para lidar com as imagens de entrada, visto que estas redes costumam possuir bom desempenho e são amplamente utilizadas para lidar com imagens [37].

Aproveitando a implementação modular e uso de um sistema de comunicação altamente utilizado no campo da robótica, o ROS, sugere-se a substituição do ambiente simulado com o OpenAI Gym por um videogame real. A plataforma PlayStation2 possui um compilado de jogos de Atari Namco Museum [3] que pode ser utilizado para comparar os resultados dos mesmos jogos no mundo físico e ambiente simulado.

Para utilizar o agente desenvolvido em um sistema no mundo físico, é necessário que se construa um sistema robótico para que o agente possa realizar as observações sobre o ambiente com sensores e atue sobre o mesmo. Uma câmera como sensor para filmar a tela e captar as imagens do jogo seria uma aproximação à realidade de um jogador humano, e um mecanismo robótico que realize o pressionamento dos botões do controle introduziria considerações como tempo de resposta para o acionamento de botões.

Outra possibilidade no campo da robótica é integrar o OpenAI Gym com simuladores tradicionais de robótica, tais como o Gazebo [38], e experimentar o desempenho do agente em ambientes mais complexos e estocásticos.

Há estudos na área da aprendizagem com sistemas multiagentes em ambientes de jogos digitais que visam observar o aprendizado de comportamentos colaborativos entre agentes [23]. Em trabalhos futuros, pode-se adaptar o sistema desenvolvido para jogar jogos *multiplayer* colaborativos.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] KEON. *Deep Q-Learning with Keras and Gym.* [s.n.], 2017. Disponível em: <https://keon.io/deep-q-learning/>.

[2] NIELSEN, M. A. *Neural networks and deep learning.* Determination Press, 2015. Disponível em: <http://neuralnetworksanddeeplearning.com>.

[3] LOPES, R. A. S.; BRAGA, V. G. de M.; LAMAR, M. V. Sistema baseado em redes neurais e q-learning para jogar jogos eletrônicos. 2017. Disponível em: <http://bdm.unb.br/handle/10483/17240>.

[4] GERA, D. L. *Ancient Greek Ideas on Speech, Language and Civilization.* [S.l.]: Oxford University Press, 2003.

[5] MATARIC, M. J. *Introdução à robótica.* [S.l.]: UNESP, 2014.

[6] NORBERT, W. *The human use of human beings: cybernectics and society.* [S.l.]: Free Association Books, 1989.

[7] RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A Modern Approach.* [S.l.]: Prentice Hall, 2009.

[8] BOJARSKI, M. et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. Disponível em: <https://arxiv.org/abs/1604.07316>.

[9] SIMON, P. *Too Big to Ignore: The Business Case for Big Data.* [S.l.]: Wiley, 2013.

[10] MITCHELL, T. M. *Machine Learning.* [S.l.]: McGraw-Hill Science/Engineering/Math, 1997.

[11] RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach.* [S.l.]: Malaysia; Pearson Education Limited,, 2016.

[12] SUTTON, R. S.; BARTO, A. G. et al. *Reinforcement learning: An introduction.* [S.l.]: MIT press, 1998.

[13] MOHRI, A. R. M.; TALWALKAR, A. *Foundations of Machine Learning.* [S.l.]: The MIT Press, 2012.

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

[15] CRAIGHEAD, J. et al. A survey of commercial & open source unmanned vehicle simulators. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, p. 852–857, 2007.

[16] TURING, A. M. Digital computers applied to games. 1953. Disponível em: <http://www.turingarchive.org/browse.php/b/7>.

[17] SILVER, D. et al. Mastering the game of go without human knowledge. *Nature*, Nature Publishing Group, v. 550, n. 7676, p. 354, 2017.

[18] OPENAI Five. [S.l.: s.n.], 2018. https://blog.openai.com/openai-five/.

[19] RAJARATNAM, D.; THIELSCHER, M. Towards general game-playing robots: Models, architecture and game controller. In: SPRINGER. *Australasian Joint Conference on Artificial Intelligence*. [S.l.], 2013. p. 271–276.

[20] BEYNIER, A. et al. *Markov Decision Processes in Artificial Intelligence: MDPs, beyond MDPs and applications*. [S.l.]: Wiley, 2010.

[21] UCL Course on RL. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

[22] HEFNY, A. et al. Recurrent predictive state policy networks. *arXiv preprint arXiv:1803.01489*, 2018. Disponível em: <https://arxiv.org/abs/1803.01489>.

[23] PORTELA, M. V.; RAMOS, G. N. Seleção de comportamentos em múltiplos agentes autônomos com aprendizagem por reforço em ambientes estocásticos. 2015. Disponível em: <http://bdm.unb.br/handle/10483/15302>.

[24] ROSENBERG, A.; SOMAPPA, G. Pixel to pinball: Using deep q learning to play atari. Disponível em: <http://www.cs.virginia.edu/~gs9ed/reports/Reinforcement_Learning_Paper.pdf>.

[25] MOORE, A. W.; ATKESON, C. G. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, v. 13, n. 1, Oct 1993. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/BF00993104>.

[26] DOOB, J. L. Stochastic processes. Wiley, 1990.

[27] TESAURO, G. Temporal difference learning and td-gammon. *Communications of the ACM*, v. 38, p. 58–68, Março 1995.

[28] MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved., v. 518, n. 7540, p. 529–533, fev. 2015. Disponível em: <http://dx.doi.org/10.1038/nature14236>.

[29] HAYKIN, S. *Neural networks: a comprehensive foundation*. [S.l.]: Prentice Hall PTR, 1994.

[30] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. MIT Press, 2016. Disponível em: <http://www.deeplearningbook.org>.

[31] BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 35, n. 8, p. 1798–1828, Aug 2013. ISSN 0162-8828.

[32] OPENAI Gym. 2018. Disponível em: <`https://gym.openai.com`>.

[33] ROS.ORG. 2018. Disponível em: <`http://www.ros.org`>.

[34] RAYAPATI, P.; VERHAM, Z. Deep reinforcement learning for atari (ms.pac-man). 2016.

[35] MNIH, V. et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. Disponível em: <`https://arxiv.org/abs/1312.5602`>.

[36] ZHANG, S.; SUTTON, R. S. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017. Disponível em: <`https://arxiv.org/pdf/1712.01275`>.

[37] SIMARD, P. Y.; STEINKRAUS, D.; PLATT, J. C. Best practices for convolutional neural networks applied to visual document analysis. In: *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* [S.l.: s.n.], 2003. p. 958–963.

[38] ZAMORA, I. et al. Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. *arXiv preprint arXiv:1608.05742*, 2016. Disponível em: <`https://arxiv.org/pdf/1608.05742.pdf`>.

ANEXOS

I. DESCRIÇÃO DO CONTEÚDO DO CD

O CD é composto por:

- leiame.pdf: Esta descrição do CD;
- instalacao.launch: Arquivo de instalação de bibliotecas e pacotes necessários;
- executar.sh: Arquivo de execução do sistema completo;
- codigo.zip: Código completo para executar o sistema compactado em formato ZIP;
- relatorio.pdf: Arquivo com este relatório em formato PDF.

II. PROGRAMAS UTILIZADOS

- OpenAI Gym
- ROS
- Pytorch
- Pandas
- Pillow