Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Sistema para o Aprendizado Automático de Jogos Eletônicos baseado em Redes Neurais e Q-Learning usando Interface Natural

Rafael A. S. Lopes
Victor Gueresi de Mello Braga

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Marcus Vinicius Lamar

Brasília
2017

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Rodrigo Bonifácio de Almeida

Banca examinadora composta por:

Prof. Dr. Marcus Vinicius Lamar (Orientador) — CIC/UnB
Prof. Dr. Marcelo Grandi Mandelli — CIC/UnB
Prof. Msc. Marcos Fagundes Caetano — CIC/UnB

CIP — Catalogação Internacional na Publicação

Lopes, Rafael A. S..

Um Sistema para o Aprendizado Automático de Jogos Eletônicos baseado em Redes Neurais e Q-Learning usando Interface Natural / Rafael A. S. Lopes, Victor Gueresi de Mello Braga. Brasília : UnB, 2017.
56 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2017.

1. redes neurais, 2. inteligência artificial, 3. jogos

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Sistema para o Aprendizado Automático de Jogos Eletônicos baseado em Redes Neurais e Q-Learning usando Interface Natural

Rafael A. S. Lopes
Victor Gueresi de Mello Braga

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Marcus Vinicius Lamar (Orientador)
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
CIC/UnB

Prof. Msc. Marcos Fagundes Caetano
CIC/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
Coordenador do Bacharelado em Ciência da Computação

Brasília, 16 de fevereiro de 2017

Dedicatória

Dedicamos às nossas famílias, pelo grande apoio e por oferecerem condições suficientes para concluirmos o curso de graduação.

Agradecimentos

Agradecemos ao professor orientador Marcus Vinicius Lamar por todo o apoio dado ao projeto.

Agradecemos os professores Marcelo Grandi Mandelli e Marcos Fagundes Caetano por se disponibilizarem a participar da banca avaliadora deste projeto.

Resumo

Jogos eletrônicos conquistaram espaço no meio acadêmico à medida que se tornaram parte de pesquisas na área de Inteligência Artificial. Nos estágios iniciais, tais jogos são, geralmente, simples e de fácil aprendizado de sua mecânica. Porém, à medida que o jogador avança nos nesses, geralmente se tornam difíceis e complexos, passando a exigir altos níveis de habilidade e/ou raciocínio. Sob a hipótese de que a utilização de técnicas de processamento de sinais de vídeo e treinamento de redes neurais a fim de reconhecer as mecânicas dos jogos levará ao sucesso nesses jogos. Foi desenvolvido um sistema, a partir do algoritmo *Q-learning* com uso de rede neurais, capaz de aprender o padrão de diferentes jogos, cujo o desenvolvimento é detalhado neste trabalho. O modelo proposto supera abordagem aleatórias em jogos como *Galaga*, *Enduro* e *Beamrider*. Nos jogos *Enduro* e *Beamider* o sistema se aproxima do resultados obtidos por projetos feito com acesso à memória do emulador. No entanto, nos jogos *Breakout* e *Pong*, devido à falta de precisão do sistema de entrada e aos atrasos do sistema, não houve progresso em relação a um algoritmo que joga aleatoriamente.

Palavras-chave: redes neurais, inteligência artificial, jogos

Abstract

Electronic games have gained space in academia as they have become part of research in the area of Artificial Intelligence. In the early stages, electronic games are usually simple and easy to learn from their mechanics. However, as the player progresses in games, they usually become difficult and complex, requiring high levels of skill and / or reasoning. Under the hypothesis that the use of video signal processing techniques and training of neural networks to recognize the mechanics of games lead to success. A system was developed from the *Q-learning* algorithm using neural network, able to learn the pattern of different games, which development is detailed in this work. The proposed model overcomes random approaches in games such as Galaga, Enduro and B.Rider. In games Enduro and B.Rider the system approaches the results obtained by projects done with access to the memory of the emulator. However, in Breakout and Pong games, due to lack of precision and delays, when compared to other projects and random algorithm, test results were lower.

Keywords: neural networks, artificial intelligence, games

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

FPS *Frames* por segundo. 24

MDP Processo de Decisão de Markov (do Inglês, *Markov Decision Process*). 8, 11–13

RNA Rede Neural Artificial. 9, 33–37, 39

RNAs Redes Neurais Artificiais. 8

TM Casamento de Padrões (do Inglês, *Template Matching*). 5

Capítulo 1

Introdução

Em 1958, o físico Willy Higinbotham criou, nos Estados Unidos, o primeiro jogo para computador, chamado *Tennis for two*. Era exibido na tela de um osciloscópio que representava uma simulação bem simplificada do esporte. Em 1971 foi lançado o primeiro fliperama da história, chamado *Computer Space*, criado para suportar a versão mais recente de *SpaceWar!*, desenvolvida por Nolan Bushnell. [1]

Em 1972, Raph Baer lançou o primeiro console, chamado *Odyssey 100*. Desde então, surgiram diversas gerações de videogames, o mercado de jogos eletrônicos sofreu uma rápida expansão e esses se fizeram mais presentes no cotidiano das pessoas. Em razão desse crescimento, os jogos eletrônicos conquistaram espaço no meio acadêmico e se tornaram parte de pesquisas nas áreas de Inteligência Artificial (IA) e Robótica. Muitas aplicações foram encontradas no âmbito da educação, saúde, economia, engenharia, e outras. A pesquisa *Rehabilitation Robotics and Serious Games: An Initial Architecture for Simultaneous Players* [2], por exemplo, apresenta uma proposta para a arquitetura de um sistema de tratamento e reabilitação com robôs, que utiliza jogos eletrônicos para motivar e prender atenção dos pacientes, permitindo a participação simultânea de usuários na realização de exercícios físicos de tratamento.

A criação de algoritmos eficientes de solução de jogos representa um desafio. A criação de máquinas capazes de atingir desempenhos próximos aos que já foram alcançados por jogadores especialistas é uma meta difícil e remete a significativos avanços na área da robótica.

1.1 Definição do problema

Nos estágios iniciais, os jogos eletrônicos são, geralmente, simples e de fácil aprendizado de sua mecânica. Porém, à medida que o jogador avança nos jogos, geralmente se tornam difíceis e complexos, passando a exigir altos níveis de habilidade e/ou raciocínio. Como

construir uma sistema baseado em inteligência artificial capaz de aprender os padrões de diferentes jogos eletrônicos populares de modo autônomo?

1.2 Justificativa

O desenvolvimento de técnicas de inteligência artificial que sejam capazes de aprender tarefas tão complicadas quanto aprender a solução de jogos a partir apenas de informações visuais, poderão ser utilizadas para a solução de outros problemas similares, tais como os enfrentados no desenvolvimento de robôs autônomos ou veículos inteligentes.

1.3 Hipótese

Uma solução para o problema pode ser obtida utilizando técnicas de processamento de sinais de vídeo e treinamento de redes neurais para reconhecer as mecânicas dos jogos que possam conduzir ao sucesso.

1.4 Objetivo Geral

Desenvolver um programa capaz de aprender e jogar diversos jogos eletrônicos, tais como a versão de Playstation 2 do jogo Galaga e diferentes jogos da plataforma *Atari*, com a finalidade de obter a maior pontuação possível.

1.5 Objetivos Específicos

1. Encontrar um forma eficiente de capturar *frames* da tela de um emulador da plataforma em tempo real;
2. Implementar um algoritmo eficiente para obter a pontuação do jogo a partir dos *frames* obtidos;
3. Propor uma medida de recompensa para o treinamento da rede neural a partir da variação da pontuação obtida com a ação tomada;
4. Processar os *frames* obtidos visando diminuir o custo computacional necessário ao aprendizado;
5. Definir e treinar uma estrutura de rede neural para prever a ação que maximiza a recompensa;
6. Avaliar a eficácia das técnicas propostas em jogos eletrônicos simples.

1.6 Organização do Trabalho

Esta monografia foi organizada em cinco capítulos. No Capítulo 2 é realizada a revisão teórica a respeito do tema tratado e apresentados trabalhos relacionados a automação de jogos e projetos de inteligência artificial utilizadas em jogos. No Capítulo 3 é apresentada a descrição do problema enfrentado e o sistema proposto como solução. No Capítulo 4 são apresentados os resultados obtidos pela implementação do sistema. Por fim, no Capítulo 5 são apresentadas as conclusões e as propostas para trabalhos futuros.

Capítulo 2

Revisão Teórica

Este capítulo apresenta alguns conceitos básicos de aprendizado de máquina utilizados para o desenvolvimento do projeto. Em seguida é feito uma revisão do estado-da-arte em inteligência artificial aplicada ao aprendizado automático em jogos eletrônicos.

2.1 Aprendizagem de Maquina

A aprendizagem de maquina é um subcampo da ciência da computação que dá aos computadores a capacidade de aprender sem serem explicitamente programados [3]. Desenvolvido a partir do estudo do reconhecimento de padrões e da teoria da aprendizagem computacional na inteligência artificial, o aprendizado de máquina explora o estudo e construção de algoritmos que podem aprender e fazer previsões sobre dados. Previsões ou decisões baseadas em dados, através da construção de um modelo a partir de entradas de amostra. Aprendizagem de máquina é empregada em uma ampla gama de tarefas de computação onde o projeto e programação de algoritmos explícitos é inviável. Aplicações exemplo incluem filtragem de spam, detecção de intrusos de rede, reconhecimento óptico de caracteres (OCR), motores de busca e visão de computador.

No campo da análise de dados, a aprendizagem mecânica é um método utilizado para conceber modelos e algoritmos complexos que se prestam à previsão. Em uso comercial é conhecida como análise preditiva. Esses modelos analíticos permitem que pesquisadores, cientistas de dados, engenheiros e analistas produzam decisões e resultados confiáveis e repetíveis e descobrem informações escondidas através da aprendizagem a partir de relações históricas e tendências nos dados.[4]

As tarefas de aprendizagem de máquina são tipicamente classificadas em três grandes categorias, dependendo da natureza do tipo de aprendizagem ou *feedback* disponível para um sistema de aprendizagem. São ditas aprendizagem supervisionada, não supervisionada e por reforço.

2.1.1 Casamento de Padrões

O Casamento de Padrões (do Inglês, *Template Matching*) (TM), também conhecido como 1-NN (*Nearest Neighbor*) consiste na classificação de um dado de entrada pela comparação com um conjunto de dados previamente classificados, chamados de modelos. [5].

O TM também pode ser utilizado em processamento de imagens digitais para encontrar pequenas partes de uma imagem que correspondem a uma imagem de modelo. Para modelos sem muitos detalhes, ou quando o grande parte da imagem modelo constitui a imagem correspondente, uma abordagem baseada com TM pode ser eficaz.

Dado um conjunto de K modelos, $T_k(l, c)$, correspondes ao valor do pixel na posição (l, c) da imagem modelo T_k de $L \times C$ pixeis. Uma imagem de teste S, de mesmo tamanho, pode ser classificada como pertencendo à classe c de acordo com

$$c = k|min\{D(T_k, S)\} \tag{2.1}$$

com $k = 1...K$, e D uma dada medida de distorção calculada entre a imagem S e os modelos T_k. Uma medida de distorção bastante utilizada é a distância Euclidiana, definida por

$$D(T_k, S) = \sum_{i=0}^{L-1} \sum_{j=0}^{C-1} \left(T_k(i, j) - S(i, j)\right)^2 \tag{2.2}$$

Neste trabalho o TM será usado para o reconhecimento de dígitos em uma imagem, Assim, teremos $K = 10$ classes, correspondendo aos dígitos $0, 1, 2, ...9$, onde cada classe é representada por um modelo T_k.

Uma vez que o TM envolve a comparação de todos os *pixels* entre as imagens modelos e a imagem de teste, a mesma pode se tornar custosa para imagens de alta resolução. A fim de tornar a aplicação do TM menos custosa computacionalmente, pode-se reduzir o tamanho da imagem ou simplificá-la, como, por exemplo, transformando-a para escala de cinza.

2.1.2 Aprendizagem Supervisionada

Aprendizagem supervisionada é a tarefa de aprendizagem da máquina de inferir uma função a partir de dados de treinamento rotulados [6]. Os dados de treinamento consistem em um conjunto de exemplos de treinamento. Na aprendizagem supervisionada, cada exemplo é um par constituído por um objeto de entrada (tipicamente um vetor) e um valor de saída desejado (também chamado de sinal de supervisão). Um algoritmo de aprendizado supervisionado analisa os dados de treinamento e produz uma função de inferência, que pode ser usada para mapear novos exemplos. Um cenário ideal permitirá

que o algoritmo determine corretamente os rótulos de classe para instâncias não presentes no conjunto de treino.

A seguir serão apresentados alguns exemplos de aprendizagem supervisionada.

1. Retropropagação de Erros, *Error Backpropagation*[7]: Técnica comum de treinamento de redes neurais artificiais usada em conjunto com um método de otimização, tal como o gradiente descendente. O algoritmo repete um ciclo de duas fases, a propagação e a atualização dos pesos. Quando um vetor de entrada é apresentado à rede, ele é propagado para a frente através da rede, camada por camada, até atingir a camada de saída. A saída fornecida pela rede é então comparada com a saída desejada e um valor de erro é calculado para cada um dos neurônios na camada de saída. Os valores de erro são então retropropagados a partir da saída, até que cada neurônio tenha um valor de erro associado que representa aproximadamente a sua contribuição para a saída.

   O algoritmo *Error Backpropagation* usa esses valores de erro para calcular o gradiente da função de perda em relação aos pesos na rede. Na segunda fase, este gradiente é usado no método de otimização para atualizar os pesos, visando minimizar a função de erro.

2. Aprendizagem por árvores de decisão: Uma técnica comumente usada na mineração de dados [8]. O objetivo é criar um modelo que prevê o valor de uma variável de destino com base em várias variáveis de entrada. Um exemplo é mostrado na Figura 2.1.

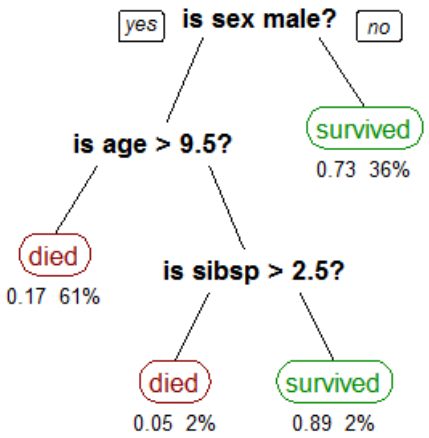

Figura 2.1: Árvore de decisão que mostra a sobrevivência dos passageiros no *Titanic* (Fonte: [9]).

Cada nó interior corresponde a uma das variáveis de entrada. Cada folha representa um valor da variável alvo, dados os valores das variáveis de entrada representadas pelo caminho da raiz para a folha.

2.1.3 Aprendizagem Não Supervisionada

A aprendizagem não supervisionada é a tarefa de aprendizagem da máquina de inferir uma função para descrever a estrutura escondida a partir de dados não rotulados.

A aprendizagem não supervisionada está intimamente relacionada com o problema da estimação da densidade na estatística. No entanto, a aprendizagem não supervisionada também abrange muitas outras técnicas que buscam resumir e explicar as principais características dos dados.

A seguir serão apresentados alguns exemplos de aprendizagem não supervisionada.

O método de aprendizagem não supervisionado mais comum é a análise de agrupamentos (*cluster*), que é usada para análise exploratória de dados para encontrar padrões ocultos ou agrupamentos em dados. Os *clusters* são modelados usando medidas de similaridades tais como a distância euclidiana. Os algoritmos comuns de *clustering* incluem:

1. Agrupamento hierárquico ou *Hierarchical Clustering*: Na mineração de dados e estatísticas, o *Hierarchical Clustering* é um método de análise que procura construir uma hierarquia de *clusters*. As estratégias para agrupamento hierárquico geralmente se dividem em dois tipos[10]:

   - Aglomerativo: Trata-se de uma abordagem de baixo para cima: cada observação começa em seu próprio *cluster*, e os pares de *clusters* são mesclados à medida que se move para cima da hierarquia.
   - Divisivo: Trata-se de uma abordagem de cima para baixo: todas as observações começam num *cluster*, e as divisões são executadas recursivamente à medida que se desloca para baixo na hierarquia.

   Em geral, as fusões e divisões são determinadas de forma gananciosa. Os resultados do *Hierarchical Clustering* são geralmente apresentados em um dendrograma.

2. Agrupamento em K-médias ou *K-Means Clustering* [11]: Uma técnica de quantização vetorial, originalmente da área processamento de sinais, que é popular para análise de *cluster* em mineração de dados. O *K-means Clustering* visa a particionar n observações em k *clusters* em que cada observação pertence ao *cluster* com a média mais próxima, servindo como um protótipo do *cluster*.

2.1.4 Aprendizagem por Reforço

Aprendizagem por reforço é uma área de aprendizagem de máquina inspirada pela psicologia behaviorista, preocupado com como os agentes de software devem tomar ações em um ambiente de modo a maximizar alguma noção de recompensa cumulativa. O problema, devido à sua generalidade, é estudado em muitas outras disciplinas, tais como teoria de jogos, teoria de controle, pesquisa de operações, teoria da informação, otimização baseada em simulação, sistemas multi-agente, estatísticas e algoritmos genéticos.

Na aprendizagem de máquina, o ambiente é tipicamente formulado como um Processo de Decisão de Markov (do Inglês, *Markov Decision Process*) (MDP), já que muitos algoritmos de aprendizado por reforço para este contexto utilizam técnicas de programação dinâmica. A principal diferença entre as técnicas clássicas e os algoritmos de aprendizado de reforço é que estes últimos não precisam de conhecimento sobre o MDP e se destinam a MDPs grandes onde os métodos exatos tornam-se inviáveis.

O aprendizado por reforço difere do aprendizado supervisionado padrão, pois nunca são apresentados pares de entrada / saída corretos, nem ações sub-ótimas explicitamente corrigidas. Além disso, há um foco no desempenho dinâmico atual que envolve encontrar um equilíbrio entre a exploração do território inexplorado e a exploração do conhecimento atual.

2.2 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNAs) são modelos computacionais baseados na estrutura do cérebro, elas são capazes de realizar aprendizagem de máquina e reconhecimento de padrões. São constituídas de um ou mais neurônios artificiais interconectados. Esses neurônios costumam ser divididos em camadas, onde existe uma camada de neurônios de entrada que pode ser conectada a uma ou diversas camadas intermediárias que então se conectam aos neurônios de saída, conforme mostrado na Figura 2.2 . As RNAs sofrem um processo de aprendizagem através de sua interação com o meio.

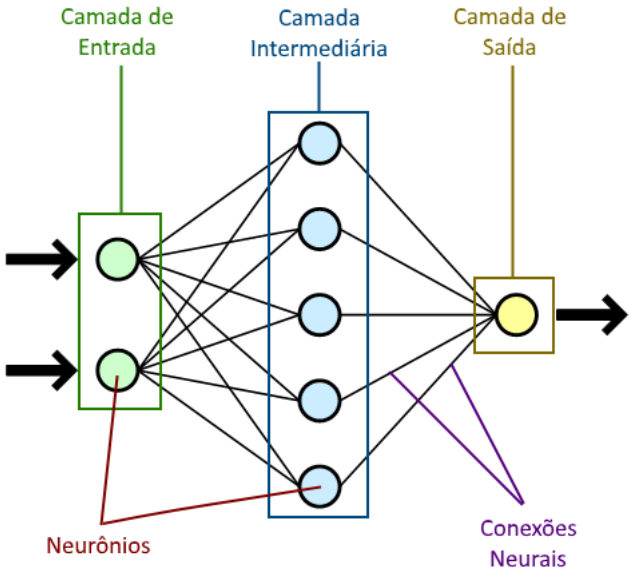

Figura 2.2: Rede neural com uma camada intermediaria

Cada neurônio de uma RNA é constituído por uma função matemática denominada função de ativação. Esses neurônios possuem conexões entre si, e essas conexões possuem pesos, são esses pesos que armazenam o conhecimento adquirido pela rede.

O procedimento utilizado para fazer o processo de aprendizado é chamado de algoritmo de aprendizagem, ele modifica os pesos das conexões entre os neurônios para alcançar o objetivo desejado [12].

2.3 Aprendizagem profunda

Aprendizagem profunda ou *Deep Learning* [13], também conhecida como aprendizagem estruturada profunda, aprendizagem hierárquica ou aprendizagem profunda de máquina, é um ramo do aprendizado de máquina baseado em um conjunto de algoritmos que tentam modelar abstrações de alto nível em dados. Em um caso simples, pode-se ter dois conjuntos de neurônios: aqueles que recebem um sinal de entrada e aqueles que enviam um sinal de saída. Quando a camada de entrada recebe uma entrada, ela passa uma versão modificada da entrada para a próxima camada. Em uma rede profunda, existem duas ou mais camadas entre a entrada e a saída, como mostrado na Figura 2.3, permitindo que o algoritmo use múltiplas camadas de processamento.

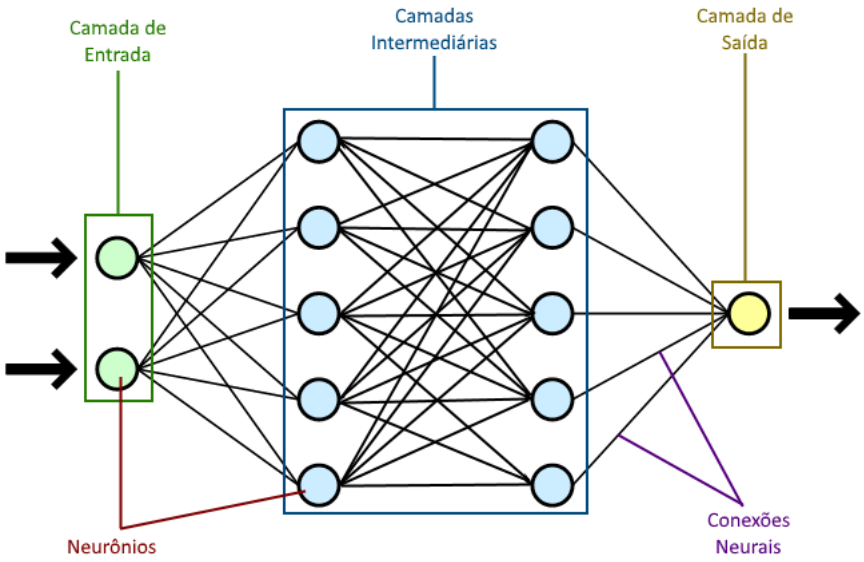

Figura 2.3: Rede neural com duas camada intermediaria

Deep Learning é parte de uma família mais ampla de métodos de aprendizado de máquina baseada em representações de aprendizagem de dados. Uma observação como, por exemplo, uma imagem, pode ser representada de várias maneiras, como um vetor de valores de intensidade por pixel, ou, de uma forma mais abstrata, como um conjunto de arestas, regiões de forma particular, etc. Algumas representações são melhores do que outras, simplificando a tarefa de aprendizagem, por exemplo, reconhecimento de face ou reconhecimento de expressão facial. Uma das promessas do *Deep Learning* é a substituição de recursos artesanais com algoritmos eficientes para a aprendizagem de recursos não-supervisionados ou semi-supervisionados e extração de recursos hierárquicos. [14]

Pesquisas nesta área tentam fazer representações melhores e criar modelos para aprender essas representações a partir de dados em grande escala sem rótulo. Algumas das representações são inspiradas nos avanços da neurociência e são vagamente baseadas na interpretação do processamento da informação e padrões de comunicação em um sistema nervoso.

Várias arquiteturas de *Deep Learning*, como redes neurais profundas, redes neurais profundas convolucionais, redes de crenças profundas e redes neurais recorrentes têm sido aplicadas a campos como visão computacional, reconhecimento automático de fala, processamento de linguagem natural, reconhecimento de áudio e bioinformática.

Os algoritmos de *Deep Learning* são baseados em representações distribuídas. A suposição subjacente por trás das representações distribuídas é que os dados observados são gerados pelas interações de fatores organizados em camadas. A aprendizagem profunda acrescenta a suposição de que essas camadas de fatores correspondem a níveis de abs-

tração ou composição. Vários números de camadas e tamanhos de camadas podem ser usados para fornecer diferentes quantidades de abstração. [15]

O *Deep Learning* explora essa ideia de fatores explicativos hierárquicos onde os conceitos de nível superior, mais abstratos, são aprendidos com os de nível inferior. Essas arquiteturas são muitas vezes construídas com um método ganancioso camada por camada. *Deep Learning* ajuda a separar essas abstrações e escolher quais recursos são úteis para a aprendizagem. [16]

Para as tarefas de aprendizagem supervisionada, os métodos de *Deep Learning* torna clara a engenharia de características, traduzindo os dados em representações intermediárias compactas semelhantes aos componentes principais e derivando estruturas em camadas que removem a redundância na representação.

Muitos algoritmos de aprendizagem profunda são aplicados a tarefas de aprendizagem não supervisionadas. Este é um benefício importante porque os dados não marcados são geralmente mais abundantes do que os dados rotulados.

2.4 Processos de Decisão de Markov

Um Processo de Decisão de Markov (do Inglês, *Markov Decision Process*) (MDP) [17] fornece uma estrutura matemática para modelagem de tomada de decisão em situações onde os resultados são em parte aleatórios e em parte sob o controle de um tomador de decisão. Os MDPs são úteis para estudar uma ampla gama de problemas de otimização resolvidos através de programação dinâmica e aprendizado por reforço.

Mais precisamente, um MDP é um processo de controle estocástico de tempo discreto. Em cada etapa de tempo, o processo está em algum estado s, e o agente pode escolher qualquer ação a que esteja disponível. O processo responde no próximo passo de tempo, movendo-se aleatoriamente para um novo estado s', e dando ao agente uma recompensa correspondente $R_a(s, s')$. A Figura 2.4 apresenta uma máquina de estados utilizando um MDP.

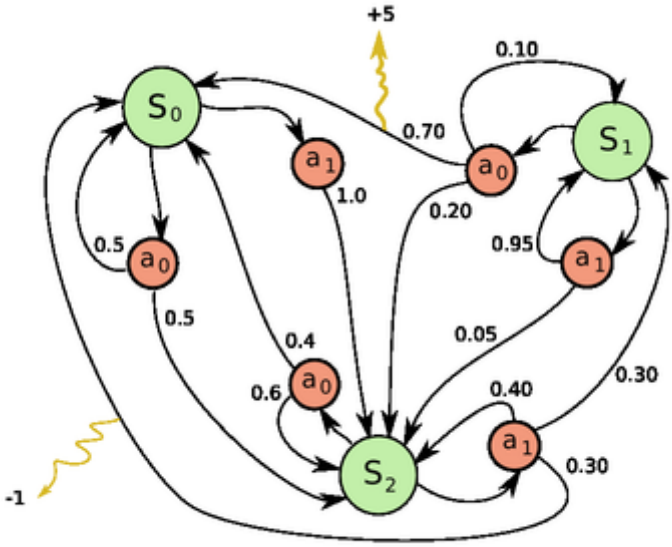

Figura 2.4: Exemplo de um simples MDP com três estados e duas ações (Fonte: [18]).

A probabilidade do processo se mover para seu novo estado s' é influenciada pela ação escolhida. Especificamente, é dado pela função de transição de estado $P_a(s, s')$. Assim, o próximo estado s' depende do estado atual s e da ação do decisor a. Mas dado s e a, é condicionalmente independente de todos os estados e ações anteriores. Em outras palavras, as transições de estado de um processo MDP satisfazem a propriedade Markov.

Na teoria da probabilidade e na estatística, o termo propriedade de Markov refere-se à propriedade sem memória de um processo estocástico. Um processo estocástico tem a propriedade de Markov se a distribuição de probabilidade condicional de estados futuros do processo, depende somente do estado presente, não da sequência de eventos que a precederam. Um processo com esta propriedade é chamado um processo de Markov.

2.5 Q-Learning

O *Q-learning* é uma técnica de aprendizado de reforço sem modelo. Especificamente, *Q-learning* pode ser usado para encontrar uma política ótima de seleção de ação para qualquer dado (finito) processo de decisão de Markov (MDP). Ele funciona aprendendo uma função de valor de ação que, em última análise, dá a utilidade esperada de tomar uma determinada ação em um determinado estado e seguir a política otimizada a partir daí. Uma política é uma regra que o agente segue na seleção de ações, dado o estado em que está. Quando essa função de valor de ação é aprendida, a política ideal pode ser construída simplesmente selecionando a ação com o valor mais alto em cada estado. Um

dos pontos fortes do *Q-learning* é que ele é capaz de comparar a utilidade esperada das ações disponíveis sem exigir um modelo do ambiente.

O *Q-learning* possui constantes arbitrarias que podem modificar o comportamento do algoritmo:

1. *Learning Rate* α : determina em que medida a informação recém-adquirida irá substituir as informações antigas. Um fator de 0 fará com que o agente não aprenda nada, enquanto um fator de 1 faria com que o agente considerasse apenas as informações mais recentes.

2. *Discount Factor* γ : determina a importância das recompensas futuras. Um fator 0 fará com que o agente considere apenas as recompensas atuais, enquanto um fator que se aproxima de 1 fará com que ele se esforce para uma recompensa de longo prazo.

O *Q-learning*, define uma função $Q(s, a)$ que representa a recompensa futura máxima descontada quando realizado a no estado s, e continua a partir desse ponto.

$$Q(s_t, a_t) = max\{R_{t+1}\} \tag{2.3}$$

onde $R_t = r_t + r_{t+1} + r_{t+2} + ... + r_n$ do ponto t em diante e r é a recompensa indicada pela transição de cada MDP.

$Q(s, a)$ é a melhor pontuação possível depois de realizar uma ação no estado final s. É chamado de Q-função, porque representa a eficácia de uma determinada ação em um determinado estado.

Para estimar a pontuação no final do jogo, sabendo apenas o estado atual e a ação, e não as ações e recompensas futuras, foca-se em apenas uma transição $< s, a, r, s' >$. É possível expressar o Q-valor do estado s e ação a em termos do Q-valor do próximo estado s'.

$$Q(s, a) = r + \gamma \times max_{a'}\{Q(s', a')\} \tag{2.4}$$

Isso é chamado de equação de Bellman. A recompensa máxima futura para este estado e ação é a recompensa imediata mais a recompensa máxima do futuro para o estado seguinte.

A ideia principal no *Q-learning* é que podemos aproximar iterativamente a função Q usando a equação de Bellman. No caso mais simples, a função Q é implementada como uma tabela, com estados como linhas e ações como colunas. A essência do algoritmo *Q-learning* pode ser representada pelo Algoritmo 1 a seguir:

O $max_{a'}\{Q(s', a')\}$ utilizado para atualizar $Q(s, a)$ é apenas uma aproximação e, em estágios iniciais de aprendizagem, pode ser completamente errado. No entanto, a apro-

Algoritmo 1 Q-learning

1: inicialize Q arbitrariamente
2: observe o estado inicial s
3: **while** não chegar ao fim **do**
4: escolha e execute uma acao
5: observe a recompensa r e o novo estado s'
6: $Q(s,a) = Q(s,a) + \alpha * (r + \gamma * max_{a'}\{Q(s',a') - Q(s,a))\}$
7: $s = s'$
8: **end while**

ximação se torna mais precisa a cada iteração e tem sido mostrado que, se executamos esta atualização o número suficiente de vezes, a função Q irá convergir e representar o verdadeiro Q-valor.

2.6 Revisão bibliográfica

Nesta seção são apresentados trabalhos na área de automação e inteligência artificial aplicada a jogos utilizando diferentes tecnologias.

2.6.1 O Turco

Em 1770 foi construída a primeira máquina capaz de jogar xadrez. Várias partidas foram disputadas pela Europa e América e durante muitos anos não se soube como era possível uma máquina jogar melhor que um humano. Era chamada O Turco [19], foi criada por Wolfgang von Kempelen, que buscava impressionar a imperatriz Maria Teresa da Áustria. A máquina, apresentada na Figura 2.5 continha uma mesa com o tabuleiro de xadrez e no seu interior havia um operador que movimentava os braços de um boneco. Essa informação só fora revelada em 1820, após 50 anos da sua criação; até então os oponentes acreditavam que estavam jogando apenas contra uma máquina.

Desde então, pesquisas nas áreas da robótica e inteligência artificial envolvendo a criação de robôs que executam trabalhos humanos são cada dia mais frequentes e amplamente aplicadas em diversas áreas de atuação como na indústria (automobilística, aeronáutica, alimentícia, farmacêutica, siderúrgica, dentre outras), na medicina, uso doméstico, em trabalhos de difícil acesso, perigosos e que envolvem situações de risco, otimizando esses processos e oferecendo proteção às pessoas. Os resultados obtidos com essas pesquisas representam grandes avanços do conhecimento humano.

Figura 2.5: O Turco: Robô jogador de xadrez (Fonte: [19]).

2.6.2 Atari 2600

O projeto utiliza uma rede neural convolucional, treinada com uma variação do *Q-learning*, que tem como entrada *pixels* originais do jogo, o método foi aplicado em sete jogos diferentes da plataforma Atari 2600. A Figura 2.6 mostra a tela de diferentes jogos de Atari.

Figura 2.6: Cinco jogos do Atari2600 (da esquerda para a direita): Pong, Breakout, Space Invaders, Seaquest, Beam Rider (Fonte: [20]).

O projeto proposto pela *DeepMind Technologies* [21] considera tarefas em que um agente interage com um ambiente E, neste caso, o emulador de atari em uma sequência de ações, observações e recompensas. A cada período de tempo o agente seleciona uma a_t ação a partir do conjunto de ações possíveis no jogo. $A = 1, ..., K$. A ação é passada

para o emulador e modifica o seu estado interno e o resultado do jogo. Em geral E pode ser estocástico. O estado interno do emulador não é observado pelo agente, em vez disso, observa uma imagem x_t do emulador, que é um vetor de valores de pixel em bruto que representam a tela atual. Além disso, recebe-se uma recompensa r_t representando a mudança na pontuação do jogo. Nota-se que, em geral, o resultado do jogo pode depender de toda a sequência de ações e observações. O *feedback* sobre uma ação só pode ser recebido depois de muitos milhares de períodos de tempo terem decorrido.

O objetivo do agente é interagir com o emulador pela seleção das ações de uma forma que maximiza a recompensas futuras. A suposição padrão, utilizada no projeto, é que futuras recompensas são descontados por um fator de γ por período de tempo, e define o futuro retorno descontado no tempo t como

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} \times r_{t'} \tag{2.5}$$

onde T é o período de tempo em que o jogo termina. É definido valor de ação ótima pela função $Q^*(s, a)$ como o retorno máximo esperado que pode ser obtido seguindo qualquer estratégia, Depois de ver algumas sequências s e , em seguida, tomar alguma ação a, $Q^*(s, a) = max_l\{R_t|s_t = s, a = a, l\}$, onde l é uma politica de sequência de mapeamento para ações (ou distribuições sobre as ações).

Este projeto, diferente do projeto apresentado nesta monografia, tem acesso a memória do emulador o que permite uma melhor controle dos *frames* obtidos, controle do tempo do jogo e a obtenção do placar. Porém, se distancia da visão humana do jogo que se limita apenas a tela propriamente dita.

2.6.3 ViZDoom

O *ViZDoom* é uma plataforma baseada no jogo de tiro em primeira pessoa Doom para pesquisa em aprendizagem por reforço baseado em visão, como ilustrado pela Figura 2.7. É simples de utilizar, altamente flexível, multi-plataforma, leve e eficiente. Em contraste com outros ambientes populares de aprendizagem visual como o do *Atari 2600*, o *ViZDoom* fornece uma perspectiva semi-realística, em primeira pessoa, do mundo virtual 3D. A *API* do *ViZDoom* dá ao usuário total controle sobre o ambiente. Múltiplos modos de operação facilitam a experimentação com diferentes paradigmas de aprendizagem como a aprendizagem por reforço, aprendizado por formação, aprendizado por demonstração e o aprendizado supervisionado. Utilizando apenas uma captura de tela, a plataforma consegue criar um *bot* que jogue o jogo. Para testes da plataforma, foram utilizados dois estágios customizados do jogo: um simples de movimentação e tiros e um mais complexo de navegação em um labirinto.

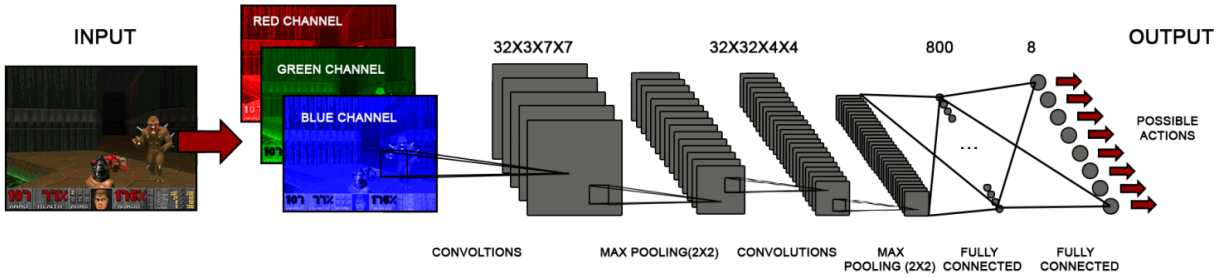

Figura 2.7: Funcionamento do projeto ViZDoom (Fonte: [22]).

O *ViZDoom* fornece recursos que podem ser explorados em diferentes tipos de experimentos AI. As principais características incluem diferentes modos de controle , cenários personalizados , acesso ao *buffer* de profundidade, mostrado na Figura 2.8, e renderização fora da tela , eliminando a necessidade de usar uma interface gráfica.

1. Modos de controle : *ViZDoom* implementa quatro modos de controle : i) jogador síncrono, ii) espectador síncrono, iii) jogador assíncrona, e iv) o público assíncrona.

   No modo assíncrono, o jogo corre a constantes 35 quadros por segundo e se o agente reage muito lentamente, ele pode perder alguns *frames*. Por outro lado, se ele toma uma decisão muito rapidamente, ele é bloqueado até que o próximo quadro chegar da *engine*. Assim, para reforço de aprendizagem da pesquisa os mais úteis são os modos síncronos, em que o motor de jogo espera o tomador de decisões. Desta forma, o sistema de aprendizagem pode aprender no seu ritmo e não é limitado por quaisquer constrangimentos temporais. É importante observar que, para fins de reprodutibilidade e depuração experimentais, o modo síncrono é executado de forma determinística. Nos modos de jogador, é o agente que faz ações durante o jogo. Em contraste, nos modos de expectador, um jogador humano está no controle, e o agente só observa as ações do jogador . Além disso, *ViZDoom* fornece um modo *multiplayer* assíncrono, que permite jogos envolvendo até oito jogadores em uma rede.

2. Cenários: Uma das características mais importantes de *ViZDoom* é a capacidade de executar cenários personalizados. Isso inclui a criação de mapas apropriados, programando a mecânica de ambiente (“quando e como as coisas acontecem”), definindo condições terminais (por exemplo, “matando um certo monstro”, “chegar a um determinado lugar”, “morreu”) e recompensas (por exemplo, por “matar um monstro”, “Se machucar”, “pegar um objeto”). Este mecanismo abre uma ampla gama de possibilidades de experimentação.

3. *Depth Buffer Acces*: *ViZDoom* fornece acesso ao *buffer* de profundidade do processador, como demonstrados na Figura 2.8, o que pode ajudar um agente a compreender a informação visual recebida.

Figura 2.8: ViZDoom permite acesso ao buffer de profundidade (Fonte: [22]).

Este recurso dá a oportunidade de testar se os algoritmos de aprendizagem podem, autonomamente, reconhecer a posição dos objetos no ambiente. A informação de profundidade pode também ser utilizada para simular os sensores de distância comuns em robôs móveis.

4. *Off -Screen Rendering* e *Frame Skipping*: Para facilitar experiências de aprendizado de máquina computacionalmente pesadas, *ViZDoom* é equipado com renderização fora da tela e *Frameskipping*. A renderização fora da tela diminui a carga computacional tornando possível executar os experimentos em servidores sem terminal gráfico. *Frameskipping*, por outro lado, permite omitir renderização de quadros selecionados. Intuitivamente, um robô eficaz não necessita visualizar cada *frame* [22].

   Vizdoom apesar de não ter acesso a nenhum outro dado além do da tela do jogo, só funciona em um jogo, no caso *Doom*.

2.6.4 Learnfun and Playfun

O *Nintendo Entertainment System* (NES), apresentado na Figura 2.9, é um console lançado pela *Nintendo* na América do Norte, Europa, Ásia, Austrália e Brasil. Originalmente lançado no Japão em 1983 com o nome de *Nintendo Family Computer* (Famicom), o sistema foi redesenhado e recebeu o novo nome para ser lançado no mercado em 1985.

Figura 2.9: Console NES (Fonte: [23]).

Na conferência SIGBOVIK 2013, Tom Murphy apresentou seu projeto para jogar qualquer jogo do console NES de forma autônoma. [24] A ideia central do projeto é monitorar valores na memória de jogo, através de emulação, para deduzir quando o jogador está vencendo. Os estímulos que um jogador humano percebe, como a tela de vídeo e efeitos sonoros, são completamente ignorados. Como uma simplificação adicional, é assumido que ganhar sempre consiste em um valor que esteja crescendo como: posição no nível ficando maior, a maior número de vidas, o mundo ou nível número ficando maior, e assim por diante.

O projeto necessita que um jogador humano jogue o jogo manualmente uma vez enquanto o sistema monitora esta interação. Ele, então, verifica a memória durante essa interação a fim de deduzir os valores crescentes que têm a maior probabilidade de indicar uma vitória no jogo. Então, comandos são enviados ao jogo a fim de maximizar esses valores deduzidos.

Esse projeto, apesar de funcionar para diferentes jogos do NES, também tem acesso à memória do emulador, além de precisar de uma interação manual com o jogo a fim de deduzir os endereços de memória que podem indicar um bom resultado no jogo.

2.6.5 MarI/O

MarI/O é um algoritmo genético utilizando redes de neurais, desenvolvido por Seth Bling's [25], capaz de aprender a jogar *Super Mario World*.

A implementação de Seth baseia-se no conceito de *NeuroEvolution of Augmenting Topologies* (NEAT) [26]. NEAT é um tipo de algoritmo genético que gera redes neurais artificiais (RNAs) eficientes a partir de uma rede inicial muito simples.

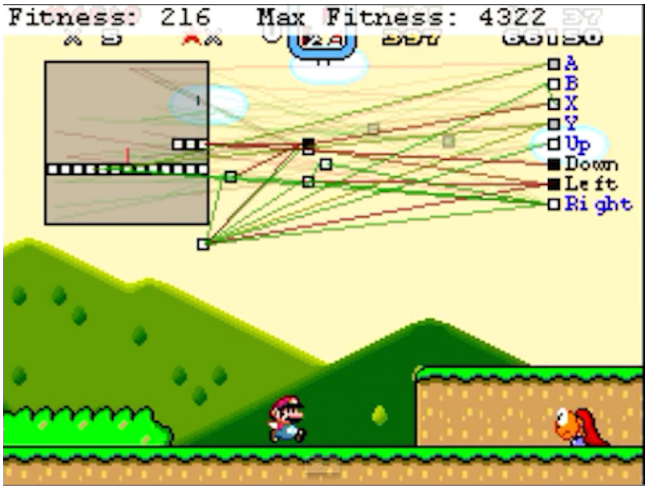

Figura 2.10: Visão do MarI/O (Fonte: [27]).

A Figura 2.10 demonstra a forma como o *MarI/O* trabalha. O quadrado do lado esquerdo representa a entrada da rede neural, ou seja, o que o programa "enxerga". Na tela é exibida uma simples representação da fase em que o jogador se encontra, os quadrados brancos representam lugarem estáticos onde o jogador pode pisar e os quadrados pretos representam objetos que se movem como: inimigos ou cogumelos. No lado direito estão representadas as saídas da rede, em outras palavras, os oito botões presentes no controle do *Super Nintendo*. Entre as entrada e as saídas está representada a rede neural, os quadrados flutuantes representam os neurônios e as linha as conexões entre eles. A linha verde corresponde a uma conexão positiva e a linha vermelha uma conexão negativa. Uma linha verde a um quadrado branco ou preto torna a saída conectada a outra ponta um quadrado da mesma cor, enquanto a linha vermelha torna a cor oposta. Uma saída branca representa uma botão que deve ser apertado neste dado instante, enquanto uma saída preta representa um botão que não deve ser pressionado. No canto superior direito existe um parâmetro chamado *Fitness*. Este parâmetro representa o quão longe o jogador está do ponto de partida da fase. Apenas os testes com maior *Fitness* são escolhidos para criar uma nova geração. Os autores reportam que foram necessárias trinta e quatro gerações para que a rede fosse capaz de completar uma fase.

O *MarI/O* acessa a memória do emulador para identificar quais blocos são permitidos pisar e quais não são, além do projeto só funcionar para o jogo *Super Mario Word*.

Diferentemente dos projetos apresentados neste capítulo, o sistema proposto foi construído sem acesso a memória do emulador, ou seja, apenas observando diretamente os

pixels presentes na tela. Assim, não há controle exato de quantos *frames* passaram e nenhuma outra informação numérica como placar do jogo. Além da interação com emulador ser feita a partir de comando externos (via teclado) o que prove uma abordagem mais próxima à experiência de um ser humano ao jogar um jogo. No próximo capítulo será apresentado o sistema proposto.

Capítulo 3

Modelo Proposto

Este capítulo aborda questões encontradas durante a construção do sistema computacional que aprende a jogar diferentes jogos eletrônicos. Serão apresentadas as técnicas utilizadas e o funcionamento dos mecanismos desenvolvidos para atingir o objetivo inicial do projeto.

A Figura 3.1 mostra uma visão da interação do sistema proposto com o jogo.

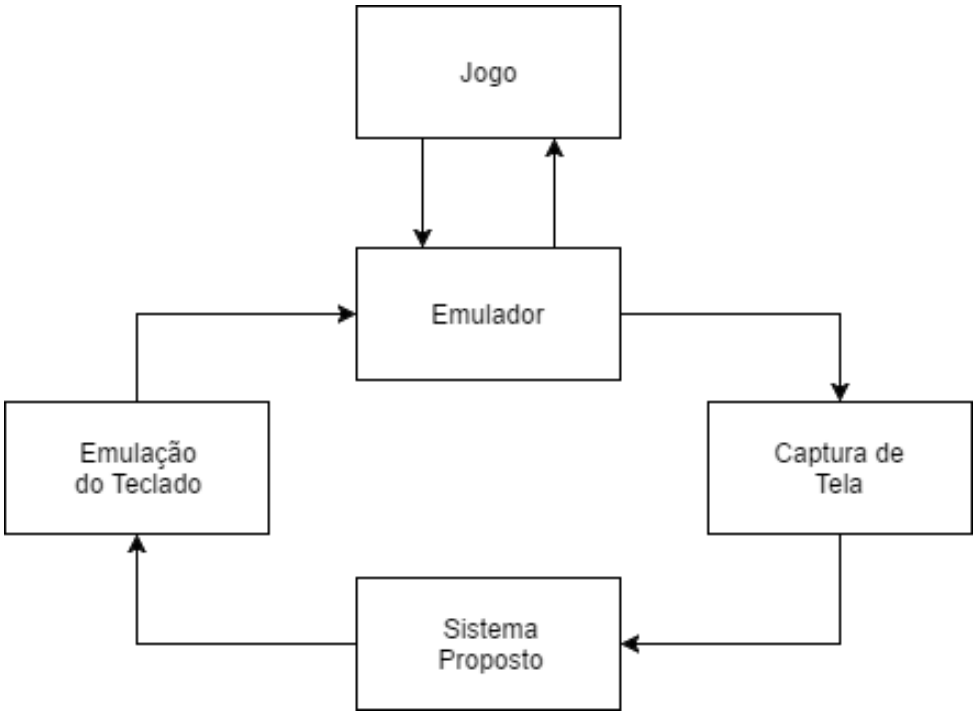

Figura 3.1: Interação do sistema proposto com o jogo

O sistema pode ser definido de forma resumida como:

1. Executar o jogo numa janela
2. Obter todos os *pixels* do jogo através de uma captura de tela

3. Criar um *buffer* concatenando 4 desses conjuntos de *pixels*
4. Utilizar a rede neural com esse *buffer* para obter uma ação a ser tomada
5. Com probabilidade e, tomar uma ação aleatória no jogo. Senão tomar a ação definida pela rede.
6. Observar novamente os *pixels* do jogo, obtendo o placar
7. Salvar o placar obtido em 6, o *buffer* do item 3 e a ação tomada em 5 como uma experiência na memória do *Experience Replay*
8. Sortear uma experiência na memória do *Experience Replay*
9. Treinar a rede com base na experiência sorteada

Enquanto uma pessoa joga, os diversos estímulos dados pelos jogos são processados pelo cérebro sem que ela perceba. A imagem capturada pelos olhos é processada pelo cérebro, para então ser mandado um estímulo às mãos para que apertem os botões, assim controlando o jogo.

Estas ações deverão ser simuladas pelo sistema a fim de obter um bom despenho nos jogos em questão. A seguir serão apresentadas as técnicas para implementar os mecanismos que permitirão o sistema simular as ações de interpretar as imagens do jogo e enviar os comandos ao emulador.

3.1 Captura da tela do jogo

O primeiro passo para processar as informações do jogo, como demonstrado na Figura Figura 3.1, é a captura da imagem gerada pelo emulador. Algumas alternativas são: captura através de câmeras ou captura direta dos *pixels* da janela do emulador.

Câmera

Vantagens:

- A grande variedade de câmeras permite ao desenvolvedor do projeto escolher o tipo do sensor que melhor se enquadra às necessidades e ao orçamento do projeto.
- As técnicas de visão computacional aplicadas ao projeto podem ser mais facilmente aproveitadas em outros trabalhos, já que a implementação não fica restrita a um padrão de tela específico.

Desvantagens:

- O desenvolvimento de aplicações que ocorrem em tempo real e exigem processamento rápido de imagem necessita de câmeras altamente específicas. Dessa forma, o custo/benefício desta abordagem pode se tornar muito alto.
- A imagem adquirida por este meio possui um certo atraso que prejudica a tomada de decisões.
- A imagem adquirida sofre inerentemente a ação de ruídos de aquisição e variações devido à iluminação ambiente.

Captura da janela do emulador

A janela do emulador pode ser colocada em uma posição fixa da tela do computador. Com a janela posicionada, é feito um recorte da tela no formato retangular com a posição e as dimensões da janela obtendo-se assim a imagem do jogo.

Vantagens:

- Não necessita de um sistema de captura do sinal a partir do monitor, os dados estão presente na tela do computador e podem ser processado diretamente pela inteligência artificial; reduzindo, assim, o tempo de aquisição das informações.
- Não é afetado pelas características do meio.
- Não possui atraso na captura da imagem

Desvantagens:

- Por se tratar de uma tela especifica, caso a resolução ou a posição da janela mude, é preciso recalcular todas as coordenadas do recorte que será feito.

É importante salientar que ao fazer a captura da tela do jogo deseja-se capturar diferentes *frames*. Foi considerando que todos os jogos são executados a 60 *Frames* por segundo (FPS) logo para calcular o tempo de um *frame* basta fazer $1/60 = 0.01666666$ assim a captura da tela é feita a cada 16.6 milissegundos.

Cada tela capturada será a entrada para o sistema proposto como ilustrado pela Figura 3.1.

3.2 Sistema proposto

A Figura 3.2 mostra uma versão mais detalhada do sistema proposto.

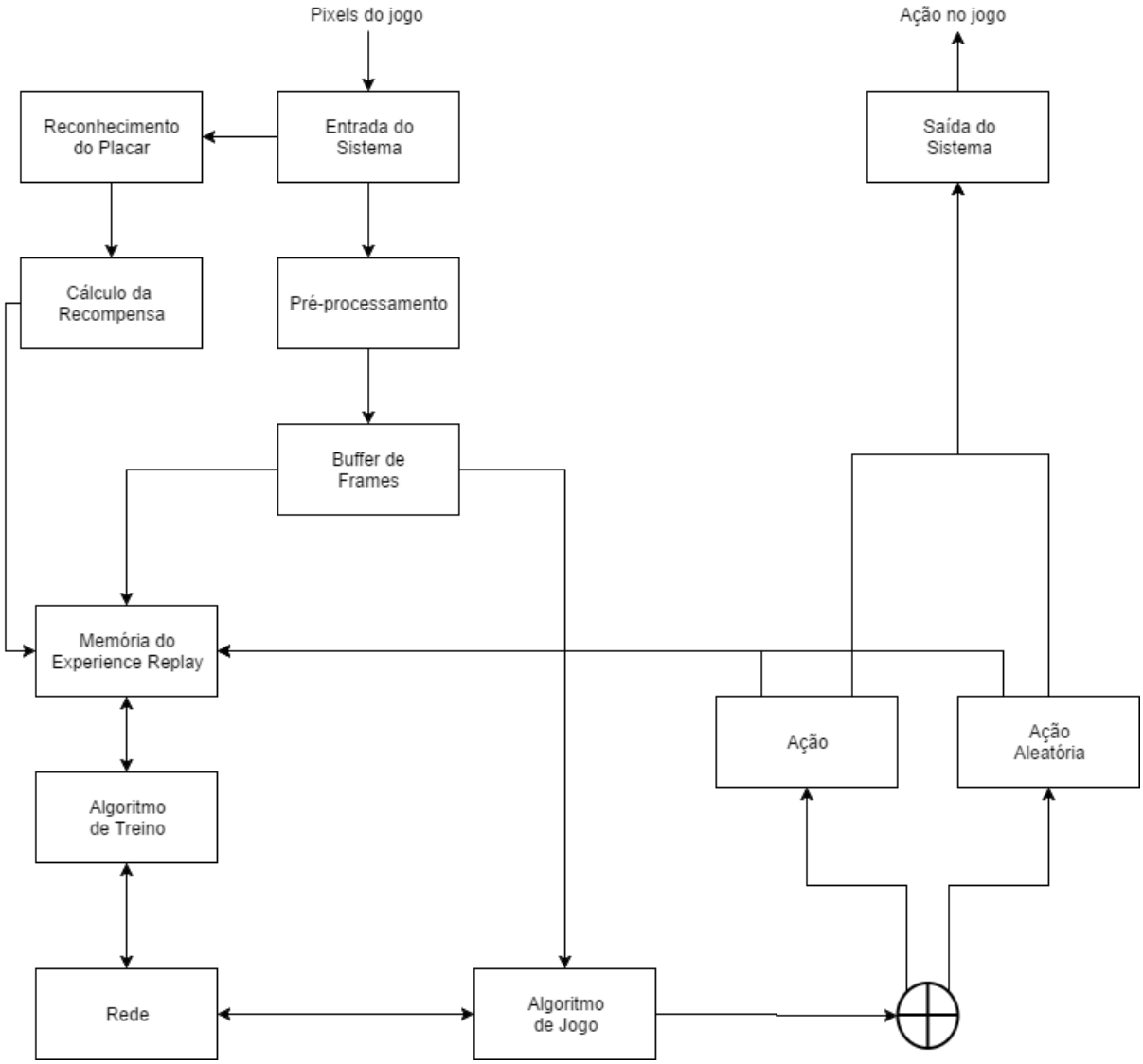

Figura 3.2: Diagrama de blocos do sistema proposto

A seguir serão descritos com mais detalhes cada parte de sistema apresentada pela Figura 3.1.

3.2.1 Pré-processamento

Após a captura da imagem do jogo é realizado um pré-processamento, como demonstrado na Figura 3.2, visando diminuir a quantidade de dados, de forma a utilizar apenas dados relevantes, tornando o algoritmo mais computacionalmente eficiente. O primeiro passo é eliminar cores presentes na imagem transformando-a em uma imagem em escala de cinza. A Figura 3.3 apresenta a saída do pré-processamento feito sobre um *frame* do jogo *Breakout*.

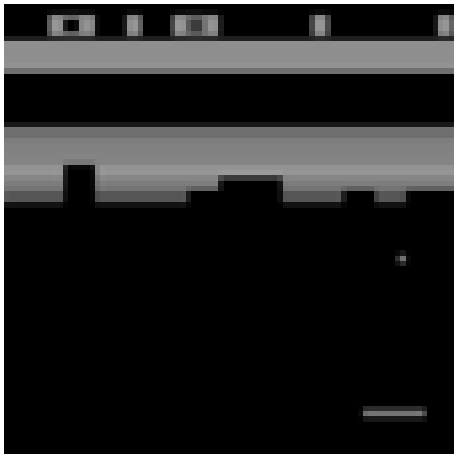

Figura 3.3: Pré-processamento de um *frame* do jogo *breakout*

A remoção das cores foi adotada pois, para os jogos testados, tal informação não é um fator fundamental para se conseguir jogá-los. Em seguida, é feito um recorde da área real do jogo. Muitos jogos possuem faixas pretas ou ilustrações que são irrelevantes para o jogo. Este recorte foi feito em um tamanho padrão 84×84 *pixels* para todos os jogos de *Atari*. No jogo *Galaga*, o recorte foi de 175×186 *pixels* pois, como foi utilizada a versão de *Playstation 2*, o tamanho da tela é maior.

3.2.2 *Buffer* de *frames*

Para utilizar o *Q-Learning* precisa-se definir os estados. A escolha óbvia seria um estado ser composto pelos pixels que compõem um *frame* do jogo. Um quadro contém as informações relevantes sobre a situação do jogo, porém não contém as informações relativas à velocidade e a direção do movimento dos objetos presentes no jogo. No entanto, o conjunto de dois quadros consecutivos contém estas informações. Neste trabalho, para obter estas informações com maior alcance temporal, e assim definir o estado, foram utilizadas um conjunto de quatro quadros.

Ao ser pré-processada a matriz de *pixels* da imagem resultante é transformada em vetor. Como o estado é composto por um grupo de quatro quadros, após o segundo quadro do grupo ser transformado em vetor este é concatenado ao final do primeiro vetor gerado e assim por diante até serem concatenados quatro vetores.

3.2.3 Reconhecimento do placar

A fim de medir o desempenho do sistema ao jogar um determinado jogo, faz-se necessário a definição de alguma métrica que demonstre esse desempenho. Nos jogos atuais existem diferentes métricas para medir este desempenho, tais como: tempo de jogo, número de

eliminações/mortes em jogos competitivos, entre outras. Nem sempre o jogo possui uma métrica numérica exata. Nos jogos testado a métrica escolhida foi o placar do jogo, exemplificado na Figura 3.4, que de forma simples, atribui um valor numérico que cresce de acordo com o progresso no jogo.

Figura 3.4: Placar do jogo Galaga

Para que o valor numérico do placar possa ser utilizado para a medição do desempenho, neste trabalho, o sistema inteligente deverá reconhecer esse número por meios visuais.

Neste trabalho o *Template Matching* foi utilizado para comparar a imagem capturada com uma série de *templates* que foram previamente classificadas de forma manual de acordo com dígito representado.

O processamento segue os seguintes passos:

1. Para cada pixel da imagem a se identificar, faz-se a diferença absoluta entre esse valor e o valor correspondente do primeiro template
2. Repetir o passo anterior para todas as cores de todos os pixels da imagem, sempre somando o resultado a um valor que será a diferença da imagem para o determinado *template*
3. Repetir os passos anteriores até se esgotarem os *templates*
4. Verificar as somas obtidas e selecionar a menor
5. O número reconhecido é o representado pelo *template* cuja diferença foi selecionada no passo anterior

Esta implementação possui como vantagem sua simplicidade, que pode ser feita em pouco tempo. Porém, a comparação exige que todos os *templates* sejam armazenados em memória.

O reconhecimento do placar, como demostrado na Figura 3.2, é feito uma vez por estado usando a imagem do primeiro quadro antes de ser pré-processada.

Cálculo da recompensa

Seja r_1 o o valor obtido no placar a partir do primeiro quadro o de s, r_2 o valor do placar obtido no próximo estado s' e r o valor da recompensa gerada pela a ultima ação tomada. Para calcular r toma-se $r = r_2 - r_1$. Caso r fosse tomado apenas como o valor

do placar em um instante de tempo, este seria o valor acumulado da recompensa de todas as ações tomadas desde o inicio do jogo, não se adequando bem à metodologia adotada neste trabalho.

3.2.4 Algoritmo de jogo

Ao aplicar o pré-processamento padrão, ou seja, tomar as quatro últimas imagens de tela, redimensioná-las para 84×84 e converter em escala de cinza com 256 níveis de cinza, resultariam em $256 \times 84 \times 84 \times 4 \approx 1067970$ possíveis estados de jogo. Isso significa 1067970 linhas na *Q-tabela*. Mais do que o número de átomos no universo conhecido. Pode-se argumentar que muitas combinações de pixels (e, portanto, estados) nunca ocorrem. Assim, é possível representá-la como uma tabela esparsa contendo apenas estados visitados. No entanto, a maioria dos estados são muito raramente visitados e seria inviável fazer a *Q-tabela* convergir. Idealmente, é preciso ter boa estimativa de *Q-valores* para estados que nunca vimos antes.

As redes neurais são excepcionalmente boas em encontrar padrões em dados altamente estruturados. Pode-se representar a função Q com uma rede neural que leva o estado (quatro telas de jogo) e ação como entrada e produz o valor Q correspondente. Alternativamente, é possível ter apenas um estado como entrada e *Q-valores* de cada ação possível como saída. Essa abordagem tem a vantagem de que, se quisermos executar uma atualização de Q-valor ou escolher a ação com o maior Q-valor, apenas teremos que fazer uma passagem pela rede e ter todos os valores de Q para todas as ações disponíveis imediatamente. A Figura 3.5 ilustra estas duas abordagens.

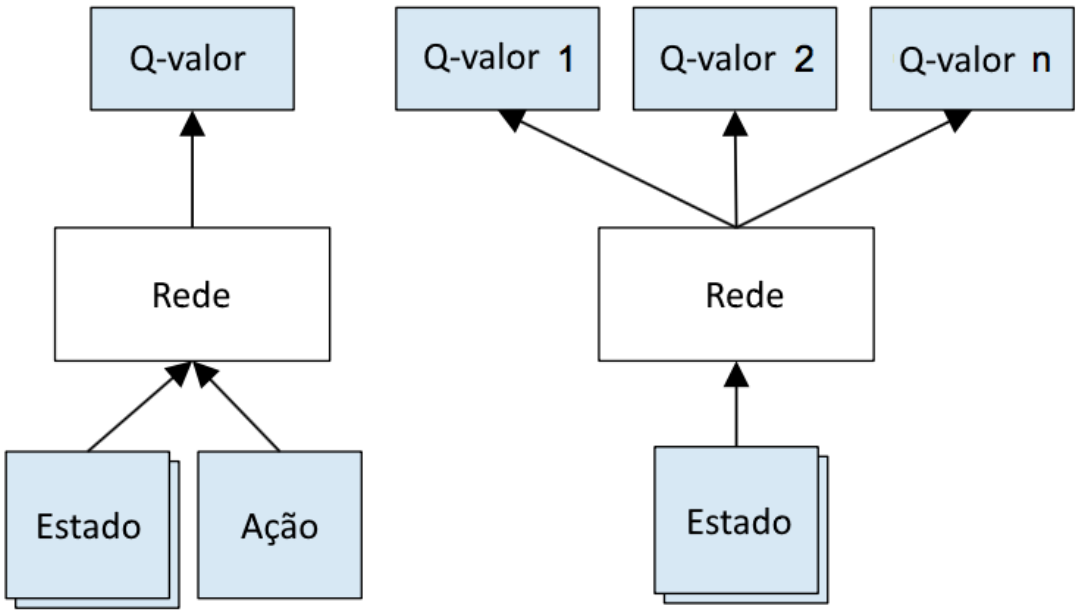

Figura 3.5: Duas possíveis abordagens de utilização da rede neural

A entrada para a rede neural corresponde a um estado. As saídas da rede são os valores de Q para cada ação possível (18 em *Atari* e 6 no *Galaga*). Os valores de Q podem ser quaisquer valores reais.

A ação que possui o maior Q-valor gerado pela rede é tida como saída do sistema. Esta ação é mantida durante toda obtenção dos quatro quadros do próximo estado.

Exploration-Exploitation

Q-learning tenta resolver o problema de atribuição de crédito - ele propaga recompensas de volta no tempo, até que ele atinja o ponto de decisão crucial que foi a causa real para a recompensa obtida.

Observe que quando uma Q-tabela ou rede neural são inicializada aleatoriamente suas previsões são inicialmente aleatórias. Se escolhermos uma ação com o maior valor de Q, a ação será aleatória e o agente executa "exploração bruta". À medida que uma função Q converge, ela retorna valores Q mais consistentes e a quantidade de exploração diminui. Assim, pode-se dizer que o *Q-learning* incorpora a exploração como parte do algoritmo. Mas esta exploração é não gananciosa, ela se contenta com a primeira estratégia eficaz que encontra.

Uma correção simples e eficaz para o problema acima é a exploração ϵ-gulosa, com probabilidade ϵ escolher uma ação aleatória, caso contrário vá com a ação "gananciosa" com o maior valor Q. Ao iniciar o programa ϵ possui valor 1 ao longo do tempo diminui de 1 a 0,1. No início o sistema faz movimentos completamente aleatórios para explorar o

espaço de estados de forma máxima e, em seguida, estabelece-se a uma taxa de exploração fixa.

Seguindo este conceito, como mostra a Figura 3.2, a saída do sistema pode ser tanto uma ação completamente aleatória como a ação que possui o maior Q-valor gerado pela rede.

3.2.5 Algoritmo de treino

Dada uma transição $< s, a, r, s' >$, sendo s o estado atual, a a ação, r a recompensa e s' o próximo estado, a regra de atualização da Q-tabela do *Q-learning* deve ser substituída pelo seguinte:

1. Faça uma passagem na rede para o estado atual s para obter valores de Q previstos para todas as ações.
2. Faça uma passagem na rede para o próximo estado s' e calcule as saídas máximas da rede $max_a\{Q(s', a')\}$.
3. Defina o alvo de Q-valor da ação como $r + \gamma.max_a\{Q(s', a')\}$ usando o valor máximo calculado na etapa 2. Para todas as outras ações, defina o destino Q-valor para o mesmo que originalmente retornado da etapa 1
4. Atualize os pesos usando o algoritmo de *BackPropagation*.

Experience Replay

Com o algoritmo descrito anteriormente é possível ter uma ideia sobre como estimar a recompensa futura em cada estado usando *Q-learning* e aproximar a função Q usando uma rede neural. Mas verifica-se que a aproximação dos valores Q usando funções não-lineares não é muito estável. Existem diversas formas de faze-la convergir.

A forma utilizada foi a *Experience Replay*. Durante o jogo, todas as experiências $< s, a, r, s' >$ são armazenadas em uma memória de repetição. Ao treinar a rede, experiências aleatórias da memória de repetição são utilizadas em vez da transição mais recente. Isso quebra a similaridade de amostras de treinamento subsequentes, que de outra forma poderiam direcionar a rede para um mínimo local. A repetição da experiência também torna a tarefa de treinamento mais semelhante à aprendizagem supervisionada usual, que simplifica a depuração e o teste do algoritmo.

3.3 Simulação do teclado

O sistema proposto interage com o emulador através da simulação do pressionamento de teclas do teclado. Para isso, foi utilizada a classe *Robot* do *Java*, sendo que o *Java* mapeia o teclado conforme a Figura 3.6.

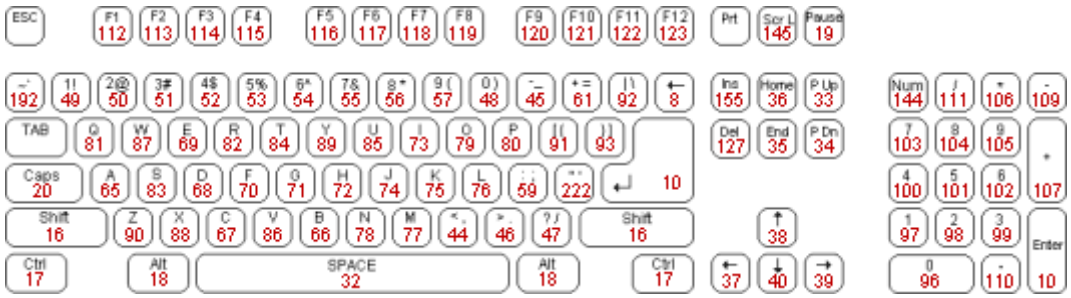

Figura 3.6: Mapeamento feito pelo *Java* de teclas do teclado

A classe *Robot* funciona simulando comandos externos de interação com o teclado do computador, com métodos que simulam o aperto e a soltura de teclas. Essa forma de interação adiciona alguns atrasos no tempo de envio de comandos, assim aproxima-se mais de uma interação humana do que um envio direto de comandos pela memória, por exemplo.

Neste capítulo foi apresentado e detalhado o sistema proposto, que é baseado em *Q-Learning* e utiliza uma rede neural a fim de otimizar o algoritmo. No capítulo seguinte serão apresentados diversos testes deste sistema proposto, quando aplicado a diferentes jogos.

Capítulo 4

Resultados Obtidos

Neste capítulo são apresentados os resultados obtidos ao se aplicar o modelo proposto em diversos casos de teste.

4.1 Toy Problem

Para comprovar a viabilidade do modelo proposto, foi criado um *Toy Problem*, ou seja, um problema mais simples, mas que se assemelha ao problema escolhido para o projeto. O *Toy Problem* está representado na Figura 4.1

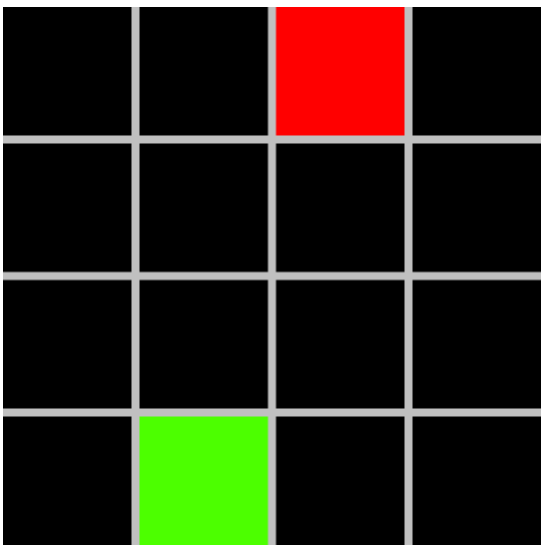

Figura 4.1: Visão do *Toy Problem*

Foi criado um campo de 4×4 espaços com um jogador e um inimigo, ambos ocupando 1 espaço. O jogador, representado pelo quadrado verde, permanece sempre na linha inferior do campo e pode se mover livremente, 1 posição por vez, na horizontal dentro do limite do campo. O inimigo, representado pelo quadrado vermelho, começa na linha superior

do campo e desce em direção ao jogador, sempre 1 posição por vez, em linha reta ou na diagonal consecutiva, sem ultrapassar os limites laterais do campo. O objetivo do jogador é evitar que o inimigo o atinja, ganhando 1 ponto se conseguir desviar e perdendo 3 pontos ao ser atingido.

O modelo proposto foi aplicado ao *Toy Problem*, à exceção da captura de imagem e da simulação do pressionamento de teclas no teclado, pois a leitura do campo e as ações do jogador eram feitos diretamente pelo programa. O tamanho de *frames* utilizados por vez na rede também foi diminuído para 3 pois um número maior impossibilitaria que jogador se movesse em tempo hábil a fim de desviar do inimigo.

Foram feitas duas tentativas distintas de treinamento: uma com utilização exclusiva da RNA a fim de movimentar o jogador e outra com a inclusão de uma chance de 10% de jogador fazer uma ação completamente aleatória. Após treinar a RNA por 3000 jogos utilizando ambas as técnicas, observou-se que a RNA treinada com o último método apresentou um desempenho superior, não deixando o jogador ser atingido pelo inimigo nenhuma vez. Assim, essa rede com o melhor resultado foi selecionada para uma bateria maior de testes e após o teste por 10000 jogos completos (até que o inimigo desça até a linha onde está o jogador), não houve nenhuma colisão do inimigo com o jogador.

4.2 Pong

Pong é um jogo eletrônico de esporte em duas dimensões que simula tênis de mesa. [28] O jogador controla uma raquete no jogo, movendo-a verticalmente no lado direito da tela, e compete contra o computador ou outro jogador que controlam uma segunda raquete no lado oposto. Os jogadores usam suas raquetes para acertar a bola e mandá-la para o outro lado. A bola aumenta de velocidade cada vez que é rebatida, reiniciando a velocidade caso algum dos jogadores não acerte a bola. O objetivo é fazer mais pontos que seu oponente, fazendo com que o oponente não consiga retornar a bola para o outro lado. A Figura 4.2 mostra uma imagem do *Pong* na sua versão para *Atari 2600*.

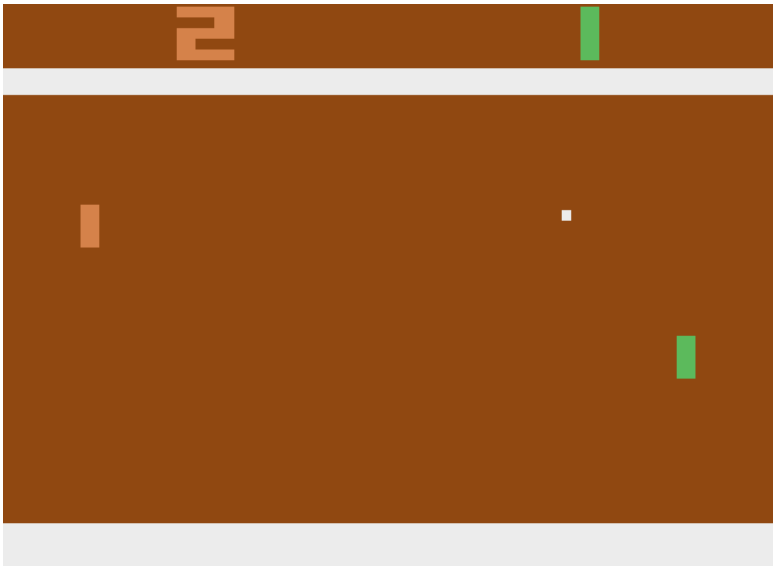

Figura 4.2: Imagem do jogo Pong

O sistema proposto foi aplicado à versão de *Atari* do jogo *Pong*, utilizando o emulador *Stella* [29].

A mesma RNA foi treinada por 500 jogos consecutivos e a RNA ao fim desses treinos foi salva e testada com 10 partidas do jogo. Também colocou-se um algoritmo que toma ações completamente randômicas para jogar 10 partidas do jogo. Os resultados obtidos desses testes estão representados na tabela abaixo:

Iterações	**Número da Partida**										**Média**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Randômico	-21	-21	-21	-21	-21	-21	-21	-20	-21	-21	-20.9
500 Treinos	-21	-21	-21	-21	-21	-19	-21	-21	-19	-21	-20.6

Tabela 4.1: Resultados referentes ao jogo *Pong*

Os resultados do jogo não foram satisfatórios. Observou-se que um dos fatores que prejudicou o sistema adotado foi a baixa precisão do sistema de entrada, isto é devido ao atraso envolvendo todo o processo desde a captura da tela do jogo até o envio do comando, observou-se que os comandos enviados ao jogo repetiam-se por muito tempo, dificultando o acerto da bola, que exige bastante precisão. Outro fator prejudicial observado foi que o jogo não delimita a área na qual o jogador pode se movimentar. Assim, o jogador acabou muitas vezes fora da área de jogo e, por isso, perdeu muitos pontos.

4.3 Breakout

Breakout é um jogo eletrônico onde o objetivo é destruir uma camada de tijolos alinhados no topo da tela, a fim de acumular pontos. Isso é alcançado pelo jogador ao rebater uma bola, que quica pela tela e destrói os tijolos ao se chocar contra eles. O jogador perde uma vida quando a bola toca a parte inferior da tela. A Figura 4.3 mostra uma imagem do *Breakout* na sua versão para *Atari 2600*.

Figura 4.3: Imagem do jogo Breakout

O sistema proposto foi aplicado à versão de *Atari* do jogo *Breakout*, utilizando o emulador *Stella* [29]. A mesma RNA foi treinada por 500 jogos consecutivos e a RNA ao fim desses treinos foi salva e testada com 10 partidas do jogo. Também colocou-se um algoritmo que toma ações completamente randômicas para jogar 10 partidas do jogo. Os resultados obtidos desses testes estão representados na tabela abaixo:

Rede	**Número da Partida**										**Média**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Randômico	4	1	3	2	3	2	1	2	1	3	2.2
500 Treinos	3	1	1	1	0	3	4	2	3	3	2.1

Tabela 4.2: Resultados referentes ao jogo *Breakout*

Assim como no jogo *Pong*, foram enfrentados os mesmos problemas de baixa precisão temporal e a não delimitação da área de jogo. Os resultados também não foram satisfatórios, pois não demonstraram melhora em comparação ao algoritmo completamente randômico.

4.4 Enduro

Enduro é um jogo eletrônico onde jogador controla um carro, visto de trás, e deve manobrá-lo numa corrida de longa distância, cujo percurso é aleatório. [30] O objetivo da corrida é ultrapassar uma certa quantidade de carros a cada nível, a fim de percorrer a maior distância possível e assim obter pontos. A Figura 4.4 mostra uma tela do jogo no sistema *Atari 2600*.

Figura 4.4: Imagem do jogo Enduro

O sistema proposto foi aplicado à versão de *Atari* do jogo *Enduro*, utilizando o emulador *Stella* [29]. A mesma RNA foi treinada por 500 jogos consecutivos e a RNA ao fim desses treinos foi salva e testada com 10 partidas do jogo. Também colocou-se um algoritmo que toma ações completamente randômicas para jogar 10 partidas do jogo. Os resultados obtidos desses testes estão representados na tabela abaixo:

Rede	**Número da Partida**										**Média**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Randômico	490	430	530	480	470	470	500	470	470	480	479
500 Treinos	700	720	740	730	690	740	740	680	730	740	721

Tabela 4.3: Resultados referentes ao jogo *Enduro*

O jogo *Enduro*, ao contrário de jogos como o *Pong* e o *Breakout*, exige menos precisão do jogador, além de delimitar a área de possível movimento no jogo. Assim, o sistema obteve uma alta pontuação apesar do mecanismo de entrada não muito preciso. Desta forma, os resultados obtidos neste jogo foram considerados satisfatórios, superando o

algoritmo que joga de forma completamente aleatória.

4.5 Beamrider

Beamrider é um jogo de tiro espacial de aspecto futurista, lançado pela Activision. O objetivo básico é destruir uma frota de naves alienígenas que aparecem do fundo da tela. A Figura 4.5 mostra uma tela do jogo no sistema *Atari 2600*.

Figura 4.5: Imagem do jogo Beamrider

O sistema proposto foi aplicado à versão de *Atari* do jogo *Beamrider*, utilizando o emulador *Stella* [29]. A mesma RNA foi treinada por 500 jogos consecutivos e a RNA ao fim desses treinos foi salva e testada com 10 partidas do jogo. Também colocou-se um algoritmo que toma ações completamente randômicas para jogar 10 partidas do jogo. Os resultados obtidos desses testes estão representados na tabela abaixo:

Rede	**Número da Partida**										**Média**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Randômico	132	264	308	176	352	352	396	308	396	176	286
500 Treinos	440	440	440	352	616	220	264	264	352	308	370

Tabela 4.4: Resultados referentes ao jogo *Beamrider*

O jogo *Beamrider*, assim como o jogo *Enduro*, não exige grande precisão de movimento do jogador, ele também tem sua área de movimentação limitada a apenas 5 espaços. Assim, o sistema obteve uma alta pontuação apesar do mecanismo de entrada não muito preciso. Entretanto, o ganho de desempenho em relação ao algoritmo aleatório não se

igualou ao ganho no jogo *Enduro*, uma explicação para isso é que o *Beamrider* é muito mais rigoroso com as falhas do jogador, que perde uma vida ao ser atingido, o *Enduro* apenas rebate o jogador para trás numa colisão. Ainda sim, os resultados obtidos neste jogo foram considerados satisfatórios, superando o algoritmo que joga de forma completamente aleatória.

4.6 Galaga

Galaga é um jogo eletrônico que tem como objetivo obter a maior pontuação possível, destruindo os inimigos com forma similar a de insetos. O jogador controla uma nave que pode se mover de para esquerda e para a direita ao, longo do fundo da tela, ou atirar. Os inimigos ocupam o espaço ao topo da tela e, de tempos em tempos, voam para baixo em direção à nave do jogador, lançando tiros e bombas em tentativa de colisão. Caso a nave do jogador colida com um tiro ou inimigo, esta é destruída e o jogador perde uma vida. Uma outra ação tomada pelos inimigos é a tentativa de captura. Quando a nave do jogador é capturada, este perde uma vida, no entanto, a nave correspondente aquela vida fica grudada ao inimigo que a capturou. Uma vez acertado pelo tiro do jogador, o inimigo é destruído e a pontuação do jogador aumenta. Os inimigos que possuem a ação de de captura possuem uma vida maior logo precisão de dois tiros para serem destruídos. Ao destruir o inimigo que capturou sua nave, o jogador ganha sua nave de volta passando a utilizar duas naves, uma ao lado a outra, fazendo com que o tiro saia dobrado. No caso de jogador, na tentativa de salvar sua nave capturada, acabe acertando sua própria nave esta é destruída e perdida até o final da partida. A Figura 4.6 ilustra uma partida da versão de Playstation 2 do Galaga.

Figura 4.6: Tela de uma partida do Galaga na versão de PlayStation 2

O sistema proposto foi aplicado na versão do *Galaga* para o console *Playstation 2*. Uma mesma RNA foi treinada durante 500 partidas do jogo, nas quais foi utilizada apenas uma vida do jogo. A cada 10 partidas, uma RNA foi salva. Também colocou-se um algoritmo que toma ações completamente randômicas para jogar 10 partidas do jogo. A Tabela 4.5 resume os resultados obtidos.

Rede	**Número da Partida**										**Média**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Randômico	2390	510	1510	560	1690	2740	360	1260	1110	1060	1319
340 Treinos	420	2780	2290	2290	1800	2660	2660	2660	2110	2420	2209
500 Treinos	2430	2130	1980	1560	720	540	1500	1630	1990	1390	1587

Tabela 4.5: Resultados referentes ao jogo *Galaga*

Os resultados obtidos foram satisfatórios pois demonstraram uma melhora sobre o algoritmo que joga de forma completamente aleatória. Um fato observado foi que a rede que passou pela maior quantidade de treinos não foi que obteve a maior pontuação no jogo, uma possível explicação para esse fato é de que a aleatoriedade do jogo pode gerar estados para os quais a rede ainda não está preparada para lidar.

4.7 Comparação com Outros Sistemas

A tabela 4.6 apresenta um resumo de todos os resultados obtidos, juntamente com os resultados obtidos por outros sistemas autônomos que jogam jogos de Atari.

	B. Rider	Enduro	Breakout	Pong	Galaga
Randômico	286	479	2.2	-20.9	1319
Randômico [21]	354	0	1.2	-20.4	
Sarsa [21]	996	129	5.2	-19	
Contingency [21]	1743	159	6	-17	
DQN [21]	4092	470	168	20	
Humano [21]	7456	368	31	-3	3364
Modelo Proposto	370	721	2.1	-20.6	1587

Tabela 4.6: Resumo dos Resultados Obtidos

Na primeira linha da Tabela 4.6 estão representados os resultados do algoritmo randômico obtidos no teste do modelo proposto. Os resultados randômicos apresentados na segunda linha da mesma tabela referem-se aos resultados descritos no artigo feito pela *DeepMind Technologies* [21]. Os resultados referentes ao jogo Galaga apresentam apenas os resultados obtidos nos testes do sistema proposto.

Pode-se observar que o algoritmo proposto apresentou resultados satisfatórios em 3 dos 5 jogos no qual foi aplicado. Vale ressaltar que o algoritmo proposto foi o único a realizar a obtenção das imagens e envio dos comandos sem acesso à memória dos emuladores dos respectivos consoles, assim simulando uma interação real de um jogador com o jogo. No entanto, os sistemas computacionais projetados para esta tarefa ainda encontra-se muito aquém dos resultados obtidos por um ser humano, indicando que há muito espaço para desenvolvimentos na área.

Neste capítulo foram apresentados os resultados obtidos ao aplicar-se o sistema proposto nesta monografia a 5 diferentes jogos. O próximo capítulo apresentará uma breve conclusão sobre o projeto, assim como considerações finais e possíveis melhorias.

Capítulo 5

Conclusão

Este trabalho apresentou o desenvolvimento de um sistema capaz de jogar as diferentes jogos em emuladores dos consoles Atari e PlayStation 2. O sistema é baseado em uma rede neural que recebe como entrada um conjunto de *frames* capturados da tela do emulador, detecta o placar de cada jogo e o transforma em número, a partir da saída da rede, um sistema baseado em *Q-Learning* observa qual ação deve ser tomada; e envia comandos para o emulador simulando um aperto de tecla do teclado do computador. Feito isso, uma experiência (conjunto de estado, ação, recompensa e próximo estado) é gerada e armazenada em uma memoria de experiencias. Essas experiências, então, são utilizadas para treinar a rede, sendo que o treinamento pode ser feito em tempo real ou não.

Considerando os resultados obtidos, verificou-se que o projeto apresentou resultados satisfatórios para certos jogos. O protótipo construído e apresentado neste trabalho foi capaz de aprender a jogar 3 dos cinco 5 jogos testados com um desempenho satisfatório. Nos 3 jogos os resultados foram melhores do que uma solução baseada em escolha aleatória da ação, e quando comparado a outros algoritmos propostos na literatura obteve resultados comparáveis. No entanto, deve-se notar que os algoritmos publicados usam os dados retirados diretamente da memória do emulador.

A implementação do reconhecimento placar por *Template Matching* apresentou-se adequada. Por meio dessa abordagem se tornou possível detectar os números com alto nível de precisão a partir da imagem do placar do jogo, não tendo sido reportado aqui análise destes resultados.

A adoção de uma de rede neural para prever os resultados do *Q-learning* tornou viável a utilização deste algoritmo que permite a analise temporal de uma ação tomada.

O treinamento da rede se mostrou e eficaz, no entanto, computacionalmente custoso. Como a parte relevante da tela dos jogos possui um grande número de *pixels* e a entrada da rede é composta por um conjunto de *frames*, a camada de entrada da rede ficou com grande número de neurônios, aumentando o custo computacional do treinamento.

Buscando aumentar o desempenho do sistema proposto e incorporar novas funcionalidades, propomos os seguintes trabalhos futuros:

- Simulação mais real de um jogador humano, utilizando-se de captura de imagens através de uma câmera filmando a tela e um mecanismo robótico para o pressionamento de botões em um controle a fim de interagir com o jogo.
- Adaptação do algoritmo para um circuito integrado do tipo FPGA. Com isso, é possível diminuir em grande quantidade as latências envolvidas no sistema apresentado, assim como facilitar a integração com outros dispositivos de entrada e saída, como os mencionados no item anterior.
- Alterar a rede neural utilizada para uma rede neural convolucional. Rede neural convolucionais são tem um melhor desempenho para trabalhar com imagens além dos *frames* obtidos não precisariam ser convertidos em vetores e poderiam ser empilhados em um matriz quadridimensional.

Referências

[1] M. de L. S. Batista, S. M. B. Lima P. L. Quintão, L. C. D. Campos, e T. J. de S. Batista. Um estudo sobre as história dos jogos eletrônicos. Janeiro 2007. Revista Eletrônica Faculdade Metodista Granbery. 1

[2] K. de O Andrade, G. Fernandes, J. Jr. Martins, V.C. Roma, R.C. Joaquim, e G.A.P. Caurin. *Rehabilitation robotics and serious games: An initial architecture for simultaneous players.* ISSNIP, 2013. 1

[3] Phil Simon. *Too Big to Ignore: The Business Case for Big Data.* Wiley, 2013. p. 89. 4

[4] Machine learning: What it is and why it matters. www.sas.com. 4

[5] R. Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice.* Wiley, 2009. 5

[6] Mehryar Mohri, Afshin Rostamizadeh, e Ameet Talwalkar. *Foundations of Machine Learning.* The MIT Press, 2012. 5

[7] David E Rumelhart, Geoffrey E Hinton, e Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988. 6

[8] Rokach e Lior. *Data mining with decision trees: theory and applications.* World Scientific Pub Co Inc., 2008. 6

[9] Wikipedia. Decision tree learning — Wikipédia, a enciclopédia livre. `https://en.wikipedia.org/wiki/Decision_tree_learning`, 2017. [Online; acessado 04-03-2017]. 6

[10] Rokach Lior e Oded Maimon. *"Clustering methods."Data mining and knowledge discovery handbook.* Springer US, 2005. 7

[11] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967. 7

[12] Simon Haykin. *Neural Networks: A Comprehensive Foundation, segunda edição.* McMaster University, Hamilton, Ontario, Canada, 1999. 9

[13] Ian Goodfellow, Yoshua Bengio, e Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`. 9

[14] Song, H.A., Lee, e S. Y. Hierarchical representation using NMF. 2013. Neural Information Processing. Lectures Notes in Computer Sciences. 10

[15] Bengio, Y., Courville, A., Vincent, e P. Representation learning: A review and new perspectives. 2013. IEEE Transactions on Pattern Analysis and Machine Intelligence. 11

[16] Deng, H.A., Lee, e S. Y. Deep learning: Methods and applications. 2014. Foundations and Trends in Signal Processing. 11

[17] J.L. Doob. *Stochastic Processes*. Wiley Classics Library. Wiley, 1990. 11

[18] Wikipedia. Markov decision process — Wikipédia, a enciclopédia livre. `https://en.wikipedia.org/wiki/Markov_decision_process`, 2017. [Online; acessado 04-03-2017]. 12

[19] Wikipedia. O turco — Wikipédia, a enciclopédia livre. `https://pt.wikipedia.org/wiki/O_Turco`, 2017. [Online; acessado 04-03-2017]. 14, 15

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, e Ioannis Antonoglou. Playing atari with deep reinforcement learning. May 2015. DeepMind Technologies. 15

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, e Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 15, 40

[22] Michal Kempka, Marek Wydmuch, Gregory Runc, Jakub Toczek, e Wojciech Jaskowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. May 2016. Institute of Computing Science, Poznan University of Technology, Poznan, Poland. 17, 18

[23] Wikipedia. Nintendo entertainment system — Wikipédia, a enciclopédia livre. `https://pt.wikipedia.org/wiki/Nintendo_Entertainment_System`, 2017. [Online; acessado 04-03-2017]. 19

[24] T Murphy VII. The first level of super mario bros. is easy with lexicographic orderings and time travel. *The Association for Computational Heresy (SIGBOVIK) 2013*, 2013. [page 112]. 19

[25] Seth Bling's. Mari/o. `https://www.youtube.com/watch?v=qv6UVOQ0F44&t=37s`, 2015. Acesso em 19/01/2017. 19

[26] Kenneth O. Stanley e Risto Miikkulainen. Evolving neural networks through augmenting topologies. November 2002. 19

[27] Google. Mari/o - machine learning for video — Google. `https://www.google.com.br/search?q=marI/O&espv=2&biw=1366&bih=662&source=lnms&tbm=isch&sa=X&ved=0ahUKEwi7p4Pm7L3SAhUJEZAKHVMoB9EQ_AUICCgD#imgrc=7e9U457VxQXj3M:`, 2017. [Online; acessado 04-03-2017]. 20

[28] Wikipedia. Pong — Wikipédia, a enciclopédia livre. https://pt.wikipedia.org/wiki/Pong, 2017. [Online; acessado 03-02-2017]. 33

[29] Stephen Anthony Bradford W. Mott e The Stella Team. Stella: " A Multi-Platform Atari 2600 VCS Emulator". https://stella-emu.github.io/, 2017. [Online; acessado 03-02-2017]. 34, 35, 36, 37

[30] Wikipedia. Enduro (jogo eletrônico) — Wikipédia, a enciclopédia livre. https://pt.wikipedia.org/wiki/Enduro_(jogo_eletrônico), 2017. [Online; acessado 03-02-2017]. 36