UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PAULO BRUNO DE SOUSA SERAFIM

AVALIAÇÃO DA COMPETIÇÃO NO TREINO DE AGENTES AUTÔNOMOS COM APRENDIZADO PROFUNDO POR REFORÇO EM JOGOS DE TIRO EM PRIMEIRA PESSOA

FORTALEZA

2018

PAULO BRUNO DE SOUSA SERAFIM

AVALIAÇÃO DA COMPETIÇÃO NO TREINO DE AGENTES AUTÔNOMOS COM APRENDIZADO PROFUNDO POR REFORÇO EM JOGOS DE TIRO EM PRIMEIRA PESSOA

Dissertação apresentada ao Curso de do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência Computacional

Orientador: Prof. Dr. Joaquim Bento Cavalcante Neto

Coorientador: Prof. Dr. Creto Augusto Vidal

Coorientador: Prof. Dr. Yuri Lenon Barbosa Nogueira

FORTALEZA

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S487a Serafim, Paulo Bruno de Sousa.
Avaliação da competição no treino de agentes autônomos com Aprendizado Profundo por Reforço em jogos de Tiro em Primeira Pessoa / Paulo Bruno de Sousa Serafim. – 2018.
83 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2018.
Orientação: Prof. Dr. Joaquim Bento Cavalcante Neto.
Coorientação: Prof. Dr. Creto Augusto Vidal.

1. Agentes Autônomos. 2. Aprendizado por Reforço. 3. Deep Learning. 4. Aprendizado Competitivo. 5. Jogos de Tiro em Primeira Pessoa. I. Título.

CDD 005

PAULO BRUNO DE SOUSA SERAFIM

AVALIAÇÃO DA COMPETIÇÃO NO TREINO DE AGENTES AUTÔNOMOS COM APRENDIZADO PROFUNDO POR REFORÇO EM JOGOS DE TIRO EM PRIMEIRA PESSOA

Dissertação apresentada ao Curso de do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência Computacional

Aprovada em: 20 de abril de 2018

BANCA EXAMINADORA

Prof. Dr. Joaquim Bento Cavalcante Neto (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Creto Augusto Vidal (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Yuri Lenon Barbosa Nogueira (Coorientador)
Universidade Federal do Ceará (UFC)

Profa. Dra. Soraia Raupp Musse
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

AGRADECIMENTOS

À minha futura esposa Geisa, que me apoia (e me aguenta) há mais de 8 anos, dando-me forças e incentivo para chegar até aqui. Muito obrigado por tudo, te amo!

Aos meus pais, Telma e Wellington, ao meu irmão Paulo Vítor, ao meu sobrinho Gustavo e a toda a minha família por serem essenciais na construção da pessoa que me tornei.

Aos meus orientadores, Prof. Bento e Prof. Creto, por terem me guiado e me ajudado ao longo dos últimos 5 anos. Gostaria de agradecer em especial ao Yuri, que além de orientador, foi um companheiro em toda esta jornada.

A todos os professores que tive e conheci na Computação e passei a admirar mais ainda essa profissão. Particularmente, meu muito obrigado aos professores Pablo, Carlos, João Paulo e Marcelino, e à professora Emanuele, que sempre me deram a liberdade de consultá-los quando precisei, para tratar dos mais diversos assuntos. Agradeço à professora Soraia, por ter aceito fazer parte da banca de avaliação e cujas observações enriqueceram esta dissertação.

Agradeço em especial à família de amigos que fiz ao longo desses quase 9 anos de UFC, que me ajudaram várias vezes ao longo desse tempo. Aos amigos Arthur Walraven, Raphaell Dyego e André Coêlho, principalmente pelas conversas constantes que temos.

Gostaria de dar um beijo em todos os amigos da ZMS que "considero" muito (ordem alfabética para ninguém reclamar): Alisson, Anderson, Bruno, Caio, César, Daniel, Davi, Eduardo, Esdras, Gabriel, Islane, Italo, José Augusto, Jean, João Victor, Juan, Karlos, Marcos, Matheus, Paulo Arthur, Pedro, Roginerio, Stenio, Taynan e Thiago.

Ao grupo de Pesquisa em Computação Gráfica, Realidade Virtual, Animação e Visualização (CRAb), e aos amigos de laboratório que muito me ajudaram: Markos, Daniel, Tiago, Danilo, Florencio, Elias, Suzana, Anthônio, Artur, Bustamante, Alexandre, Anderson, Matheus, Eduardo, Gabriel e a todos os outros que não foram mencionados.

Agradeço ao programa de Mestrado e Doutorado em Ciência da Computação (MDCC) da Universidade Federal do Ceará (UFC). Aos funcionário da secretaria do MDCC, Gláucia e Jonatas, e da administração do Laboratório de Pesquisa em Computação (LIA), Vlademiro e Daniel, que sempre foram bastante solícitos. Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pelo apoio financeiro para a realização desta pesquisa.

Para estas e todas as outras pessoas que contribuíram na minha vida:

Muito Obrigado!

RESUMO

Este trabalho avalia a competição no treino de agentes autônomos imersos em jogos de Tiro em Primeira em Pessoa utilizando Aprendizado Profundo por Reforço. Os agentes são compostos por uma *Deep Neural Network*, treinada através da técnica *Deep Q-Learning*. Eles recebem como entrada somente os *pixels* da tela, possibilitando a criação de jogadores generalistas, capazes de jogar diversos cenários sem a necessidade de modificações adicionais. O ambiente *ViZDoom*, baseado no jogo *Doom*, foi utilizado por possuir características apropriadas como uma base de testes. Quinze agentes autônomos são divididos em três equipes, em que duas delas são treinadas através de competição entre si e a terceira treina contra inimigos que agem de maneira completamente aleatória. Os agentes desenvolvidos foram capazes de aprender comportamentos adequados para sobreviver em um cenário personalizado para partidas de um contra um. Através dos testes realizados, verificou-se que o treinamento por competição entre agentes autônomos levou a um maior número de vitórias se comparado ao treinamento contra agentes não-inteligentes.

Palavras-chave: Agentes Autônomos. Aprendizado por Reforço. *Deep Learning*. Aprendizado Competitivo. Jogos de Tiro em Primeira Pessoa.

ABSTRACT

This work evaluates competition in training of autonomous agents immersed in First-Person Shooter games using Deep Reinforcement Learning. The agents are composed of a Deep Neural Network, which is trained using Deep Q-Learning. The input of the networks is only the pixels of the screen, allowing the creation of general players, capable of handling several environments without the need for further modifications. ViZDoom, an Application Programming Interface based on the game Doom, is used as the testbed because of its appropriate features. Fifteen agents were divided into three groups, two of which were trained by competing with each other, and the third was trained by competing against opponents that act randomly. The developed agents were able to learn adequate behaviors to survive in a custom one-on-one scenario. The tests showed that the competitive training of autonomous agents leads to a greater number of wins compared to training against non-intelligent agents.

Keywords: Autonomous agents. Reinforcement Learning. Deep Learning. Competitive Learning. First-Person Shooter games.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CMA-ES	*Covariance Matrix Adaptation Evolution Strategy*
CNN	*Convolutional Neural Networks*
DNN	*Deep Neural Networks*
DQN	*Deep Q-Network*
DRL	*Deep Reinforcement Learning*
DRON	*Deep Reinforcement Opponent Network*
FPS	*First-Person Shooter*
GPU	*Graphical Processor Unit*
HRA	*Hybrid Reward Architecture*
LSTM	*Long-Short Term Memory*
MADRL	*Multi-Agent Deep Reinforcement Learning*
MARL	*Multi-Agent Reinforcement Learning*
MDP	*Markov Decision Process*

LISTA DE SÍMBOLOS

A	Conjunto de ações
P	Função de transição
R	Função de recompensa
S	Conjunto de estados
n_{in}	Quantidade de neurônios conectados à entrada
n_{out}	Quantidade de neurônios conectados à saída
Q	Função-Q, ou função de valor-ação
U	Representa uma distribuição uniforme
V	Função de valor-estado
W	Vetor de pesos da rede neural
α	Taxa de aprendizado
γ	Fator de desconto do *Q-Learning*
η	Taxa de aprendizado padrão
π	Política

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

Jogos de Tiro em Primeira Pessoa (*First-Person Shooter First-Person Shooter* (FPS)) têm bastante apelo popular. Esse subgênero de jogos de ação teve um grande impacto perante o público desde os primeiros lançamentos, como *Wolfenstein3D* (*id Software*, 1992) e *Doom* (*id Software*, 1993), e continuam impactando jogadores até hoje com séries populares, como *Call of Duty* (*Activision*), *Battlefield* (*Electronic Arts*) e *Counter-Strike* (*Valve*) (Figura 1).

Figura 1 – Jogos FPS recentes. Da esquerda para a direita: *Counter Strike: Global Offensive* (*Valve*, 2012), *Wolfenstein: The New Order* (*Machine Games*, 2014), *Doom* (*id Software*, 2016) e *Call of Duty: WWII* (*Activision*, 2017).

Fonte: Elaborado pelo autor.

O desenvolvimento da indústria de jogos trouxe uma necessidade de melhores personagens controlados por Inteligência Artificial nesse gênero. Embora o avanço de tecnologias e técnicas relacionadas tenha simplificado a construção de personagens com comportamento pré-programado, eles ainda necessitam ter todos os seus procedimentos explicitamente desenvolvidos por um programador.

Para acabar com essa necessidade, o ideal seria criar personagens que pudessem aprender os comportamentos adequados por si só. Entretanto, desenvolver um agente autônomo para um jogo do gênero FPS não é uma tarefa simples, pois ele deve ser capaz de aprender a executar diversos tipos de ações complexas com o mínimo de informações fornecidas diretamente, isto é, pré-programadas. Além disso, também é desejável que o agente desenvolvido seja genérico, possibilitando o uso de uma mesma técnica em diversos jogos sem a necessidade de alterações.

Outra característica desejada é que a entrada recebida por um personagem do jogo seja a mesma de um jogador humano, de modo a garantir a equidade das informações recebidas. Uma maneira direta de atingir esse objetivo é utilizar como entrada para ambos a visão da tela. Porém, a construção de jogadores autônomos que recebam somente os *pixels* da tela como informação é uma tarefa bastante complexa (SCHMIDHUBER, 2015).

Recentemente, modelos de *Deep Learning* (LECUN *et al.*, 2015) vêm sendo utilizados para lidar com esse tipo de dificuldade. As tarefas em que o objetivo de um agente é aprender comportamentos adequados recebendo um valor após cada ação realizada são descritas pelo paradigma de Aprendizado por Reforço (SUTTON; BARTO, 1998). Da combinação dessas duas áreas surgem modelos suficientemente robustos para lidar com os problemas citados, numa abordagem chamada de Aprendizado Profundo por Reforço (do inglês *Deep Reinforcement Learning* (DRL)) (MNIH *et al.*, 2013, 2015).

Considerando as abordagens que utilizam técnicas de *Deep Learning* em jogos digitais, apesar de diversos trabalhos desenvolverem jogadores autônomos em ambientes de um único jogador, como é apresentado na Seção 3.2, são poucas as avaliações das habilidades dos agentes em situações com múltiplos jogadores, como é visto na Seção 3.3. O foco deste trabalho é o desenvolvimento de agentes utilizando DRL em um jogo FPS e competindo entre si.

1.2 Objetivos do trabalho

Este trabalho tem como objetivo principal avaliar o comportamento de agentes autônomos dentro de um ambiente FPS e verificar que o treinamento através da competição entre eles melhora o desempenho em um confronto entre agentes. A tarefa é modelada como um problema de Aprendizado por Reforço e é abordada utilizando um modelo de DRL. Para alcançar o objetivo proposto, os seguintes requisitos devem ser satisfeitos:

- Utilizar um cenário que possibilite o confronto entre dois ou mais personagens;
- Criar um controlador que permita o aprendizado de comportamentos adequados;
- Limitar a entrada somente à visão do jogo, para garantir a generalização do modelo;
- Definir os aspectos específicos do controlador que possibilitem o aprendizado;
- Implementar regimes de treinamento que proporcionem a verificação do aprendizado; e
- Executar testes que viabilizem a competição e a avaliação dos resultados obtidos.

Assim, garantindo todos esses requisitos, cada agente desenvolvido deve apresentar as seguintes características:

- Jogar o cenário competitivo de maneira satisfatória, isto é, apresentar comportamentos adequados;
- Melhorar seu resultado, ou seja, a recompensa recebida, ao longo do tempo; e
- Ter um desempenho melhor do que na abordagem sem competição contra outros agentes, de modo a validar a técnica proposta.

Uma vez que os requisitos tenham sido atingidos, deve-se ainda realizar um amplo levantamento de resultados e uma análise minuciosa do funcionamento de cada uma das partes utilizadas no desenvolvimento deste trabalho. O conjunto dos resultados e das avaliações serve para validar a técnica proposta, isto é, para assegurar que o treinamento através de competição entre agentes autônomos leva de fato a um desempenho melhor se comparado ao treinamento em competição contra agentes não-inteligentes.

1.3 Organização do trabalho

Este trabalho está organizado em cinco capítulos. No Capítulo 2, são expostos alguns modelos e técnicas que servem de base para o restante do trabalho. O Capítulo 3 traz trabalhos relacionados aos temas, mostrando as principais características de cada um. No Capítulo 4, há a definição da técnica proposta, especificando cada parte principal do trabalho e seus objetivos. No Capítulo 5, são apresentados os testes realizados e são discutidos os resultados obtidos. Por fim, no Capítulo 6, é feita uma análise geral deste trabalho e são indicados alguns trabalhos futuros que podem ser desenvolvidos.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Introdução

Neste capítulo, são introduzidas as técnicas e os modelos relevantes para o entendimento deste trabalho. Na Seção 2.2, é descrito o paradigma de Aprendizado por Reforço, bem como suas características e técnicas utilizadas para resolver tarefas desse tipo. Na Seção 2.3, é apresentado o modelo de *Deep Neural Networks* (DNN), o tipo utilizado neste trabalho e a sua utilização na resolução de problemas de Aprendizado por Reforço. Por fim, na Seção 2.4, é descrita a ferramenta *ViZDoom*, utilizada neste trabalho como ambiente de um jogo FPS.

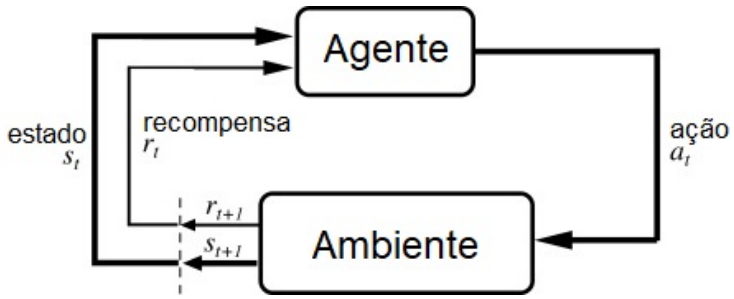
2.2 Aprendizado por Reforço

O Aprendizado por Reforço é uma ferramenta utilizada para descrever problemas de aprendizado baseado em interações e recompensas. Ele é inspirado nos experimentos de condicionamento clássico e operante onde um sujeito aprende certos comportamentos com base em seus resultados (CASTAÑEDA, 2016). Diferentemente de outros paradigmas de aprendizado, no Aprendizado por Reforço não se sabe qual o resultado esperado.

Como descrito por Sutton e Barto (1998), o tomador de decisões e aprendiz é chamado de *agente*. O objeto com que ele se relaciona é chamado de *ambiente*. O agente realiza *ações* continuamente e o ambiente responde às ações e apresenta novas situações com as quais o agente deve lidar. As situações são chamadas de *estados*. Para cada ação realizada, o ambiente dá ao agente um valor chamado de recompensa. O conjunto de especificações do ambiente é chamado de *tarefa*.

Em cada passo de tempo $t = 0, 1, 2, 3...$, o agente observa um estado $s_t \in S$, onde S é o conjunto de estados possíveis, e seleciona uma ação $a_t \in A(s_t)$, onde $A(s_t)$ é o conjunto de ações possíveis no estado s_t. No passo seguinte, o agente recebe uma recompensa r_{t+1} e encontra-se em um novo estado, s_{t+1} (Figura 2). Em outras palavras, o agente deve explorar continuamente o ambiente para aprender a obter a maior recompensa ao longo do tempo em uma tarefa. Uma maneira adequada para resolver problemas de Aprendizado por Reforço é uma formulação matemática de processos estocásticos chamada de Processo de Decisão de Markov (*Markov Decision Process* (MDP)).

Figura 2 – Diagrama de interação agente-ambiente.



Fonte: Adaptado de Sutton e Barto (1998).

2.2.1 Processos de Decisão de Markov (MDP)

Um MDP é uma ferramenta matemática utilizada para modelar problemas de tomada de decisão em situações nas quais os resultados são incertos, ou seja, processos estocásticos em que um agente precisa realizar ações repetidas sobre o que fazer e como lidar com os resultados que surgiram. Formalmente, um MDP é definido por uma tupla $\langle S, A, P, R, \gamma \rangle$, onde:

- S é um conjunto finito de estados $s_i \in S \mid i = 1...n$ e n é a quantidade de estados;
- A é um conjunto finito de ações $a_j \in A \mid j = 1...m$ e m é a quantidade de ações. A_s é o conjunto finito de ações possíveis no estado s;
- $P : S \times A \times S \rightarrow [0,1]$ é a função de transição, que determina a probabilidade de sair do estado s_t, realizando a ação a_t, e chegar no estado s_{t+1};
- $R : S \times A \rightarrow$ é a função de recompensa que determina o valor recebido imediatamente após realizar uma ação a, a partir do estado s; e
- γ é um escalar que especifica quanta prioridade deve ser dada a uma recompensa imediata.

Uma característica importante é que um MDP satisfaz a *Propriedade de Markov*, isto é, um processo estocástico que não possui memória. Na prática, isto significa que, para a determinação do estado a ser alcançado, a função de transição só depende do estado atual e da ação realizada. Portanto, a probabilidade de se realizar uma ação baseado nos passos de tempo anteriores pode ser definida por:

$$P(s_{t+1} \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_0, a_0, r_0) = P(s_{t+1} \mid s_t, a_t). \tag{2.1}$$

Com isso, o problema de Aprendizado por Reforço, isto é, a maximização da recompensa, pode ser descrito de acordo com a estrutura do MDP.

2.2.2 Maximização da recompensa recebida

A recompensa esperada de um agente ao longo do tempo é dada por:

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_{t+T}, \tag{2.2}$$

onde T é o último passo do processo.

De modo geral, uma recompensa recente é mais importante do que as anteriores. Para isso, há um desconto no valor obtido ao longo do tempo. Assim, o objetivo do agente passa a ser maximizar a *recompensa descontada*:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.3}$$

onde γ é chamado de fator de desconto, com um valor $0 \leq \gamma \leq 1$, que define a importância dada a uma recompensa imediata sobre as recompensas de ações que potencialmente serão realizadas.

Para chegar a uma solução ótima, um agente então necessita descobrir uma sequência de ações que dê a maior recompensa descontada possível. Esse mapeamento de ações é chamado de *política*.

2.2.3 Política

O comportamento do agente traduz-se em uma *política* π, um mapa que determina qual ação o agente deverá realizar em cada estado (MARTA, 2016). Uma política *determinística* mapeia um estado em uma única ação sendo representada matematicamente por:

$$\pi(s_t) = a_t, \tag{2.4}$$

onde $s_t \in S$ e $a_t \in A$. Já uma política *estocástica* define uma distribuição de probabilidades no conjunto de ações A e é representada matematicamente por:

$$\pi(s_t, a_t) = p, \tag{2.5}$$

onde $0 \leq p \leq 1$.

O objetivo final do Aprendizado por Reforço é encontrar uma política ótima π^* que maximiza a recompensa total recebida ao longo de uma tarefa. Como uma forma de avaliar e medir quão bem uma política se comporta, de modo a diferenciá-la objetivamente de uma outra política,, utilizam-se duas funções: a *função de valor-estado*, $V(s)$, e a *função de valor-ação*, $Q(s,a)$, também chamada de *função-Q*.

2.2.4 Função de valor-estado

Dada uma política π, o valor esperado de um estado é a recompensa descontada esperada no começo de um estado s usando a política π:

$$V^{\pi}(s) = \mathbb{E}_{\pi}(R_t \mid s_t = s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right]. \tag{2.6}$$

A função estado-valor possui propriedade recursiva. Assim, pode-se utilizá-la para solucionar o MDP através da *Equação de Bellman* de V^{π} (SUTTON; BARTO, 1998):

$$V^{\pi}(s_t) = R(s_t, a_t) + \sum_{s_{t+1} \in S} P(s_{t+1} \mid s_t, a_t)\gamma V^{\pi}(s_{t+1}). \tag{2.7}$$

Logo, conhecendo o valor ótimo, a política ótima será dada por:

$$\pi^*(s_t) = arg\ max_{a \in A} r_t(s,a) + \sum_{s_{t+1} \in S} P(s_{t+1} \mid s, a)\gamma V^{\pi}(s_{t+1}). \tag{2.8}$$

2.2.5 Função-Q

A função que descreve o valor recebido esperado ao realizar uma ação a em um estado s de acordo com um política π é chamada de função de valor-ação, ou função-Q:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_t \mid s_t = s, a_t = a] = \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}. \tag{2.9}$$

Assim como a função estado-valor, a função-Q também pode ser reescrita de forma recursiva:

$$Q^{\pi}(s,a) = \sum_{s_{t+1} \in S} P(s_{t+1} \mid s, a)[R(s,a) + \gamma Q^{\pi}(s_{t+1}, a_{t+1})]. \tag{2.10}$$

Essa equação define a *Equação de Bellman* para $Q^{\pi}(s,a)$ e pode ser usada para solucionar o MDP (SUTTON; BARTO, 1998). A função-Q é utilizada em uma técnica para resolver problemas de Aprendizado por Reforço chamada de *Q-Learning*.

2.2.6 Q-Learning

Embora existam diversos métodos de Aprendizado por Reforço, como *actor-critic* (BARTO *et al.*, 1983), REINFORCE (WILLIAMS, 1992) e SARSA (RUMMERY; NIRANJAN, 1994), será evidenciada a técnica *Q-Learning*, que é utilizada como base para o modelo utilizado neste trabalho. O *Q-Learning* (WATKINS, 1989; WATKINS; DAYAN, 1992) é uma técnica livre de modelos, ou seja, não depende do modelo do ambiente utilizado, e é também *off-policy*, isto é, aproxima o valor ótimo Q^* independentemente da política que está sendo seguida. Essa técnica se mostrou adequada na resolução de problemas de DRL (MNIH *et al.*, 2013, 2015).

O agente utiliza suas experiências para atualizar uma tabela contendo os valores-Q para todas as possibilidades de pares estado-ação. Os valores são atualizados através da seguinte equação de atualização, correspondente à função-Q apresentada na Seção 2.2.5:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha\left[r_t + \gamma max_a Q(s_{t+1},a) - Q(s_t,a_t)\right], \tag{2.11}$$

onde α é a taxa de aprendizado, $0 \leq \alpha \leq 1$, e determina a taxa na qual as novas informações irão sobrepor os valores antigos. Assim, a política do *Q-Learning* será, para cada estado, realizar a ação com o maior valor-Q, isto é:

$$\pi(s) = arg\ max_a Q^{\pi}(s,a). \tag{2.12}$$

Essa política levará a uma função-Q ótima:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{2.13}$$

Uma vez que o valor-Q ótimo do próximo passo de tempo $Q^*(s_{t+1},a_{t+1})$ seja conhecido, então a estratégia ótima no estado s_{t+1} é realizar a ação a_{t+1}, maximizando o valor de

recompensa esperado. Assim, no estado s_t deverá ser realizada a ação a_t que leve ao estado s_{t+1}. Dessa forma, a função-Q ótima pode ser escrita como:

$$Q^*(s_t, a_t) = \mathbb{E}\left[r + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \mid s, a\right]. \tag{2.14}$$

O Algoritmo 1 descreve o processo de atualização. Dadas uma taxa de aprendizado adequada (KAELBLING *et al.*, 1996) e um MDP finito, o modelo irá convergir e encontrará o valor ótimo, isto é, $\lim_{t\to\infty} Q(s,a) = Q^*(s,a)$ (SUTTON; BARTO, 1998; CASTAÑEDA, 2016). Dessa forma, um agente será capaz de aprender comportamentos adequados, isto é, uma política ótima, dentro de um problema de Aprendizado por Reforço.

Algoritmo 1: *Q-Learning*

```
início
    inicialize todos os Q(s,a) arbitrariamente;
    repita
        s_t ← estado inicial;
        repita
            selecione a_t = π(s_t), de acordo com a Equação 2.12;
            realize a ação a_t, observe r_{t+1} e s_{t+1};
            atualize Q(s,a) de acordo com a Equação 2.11;
            s_t ← s_{t+1}
        até s_t ser terminal;
    até convergir;
fim
```

2.3 Deep Neural Networks (DNN)

Em problemas de Aprendizado por Reforço reais, tanto o conjunto de estados quanto o conjunto de ações podem ser muito grandes, a ponto de se tornar impraticável resolver esses problemas utilizando uma tabela de valores-Q (DIJK, 2017). Nesses casos, uma solução adequada é utilizar uma representação através de uma função parametrizada Q_θ, em que Q^π_θ é próxima à função-Q ótima Q^π (LAMPLE; CHAPLOT, 2017). Assim, o objetivo passa a ser encontrar valores θ tais que:

$$Q_\theta(s,a) \approx Q^*(s,a). \tag{2.15}$$

A função de aproximação pode ser linear ou não-linear, como uma rede neural. Recentemente, modelos de redes neurais com um grande número de camadas, chamados de DNN, mostraram resultados bem melhores do que os métodos anteriores (SCHMIDHUBER, 2015). Foi o avanço das tecnologias computacionais que possibilitou o aprendizado de uma rede neural com dezenas de camadas (LECUN *et al.*, 2015).

De maneira geral, a maioria dos modelos de *Deep Learning* possuem algumas características em comum que ajudam a explicar o seu melhor desempenho (LECUN *et al.*, 2015; SCHMIDHUBER, 2015), tais como: uso de um grande volume dados, inicialização de pesos não-aleatórios, técnicas dinâmicas de atualização dos pesos das redes neurais e aumento do poderio computacional através da utilização de *Graphical Processor Unit* (GPU). Outra característica marcante das DNN é sua adequação a tarefas que utilizam visão, através do uso de CNN, que são especialmente úteis em tarefas que utilizam como entrada os *pixels* da tela (LECUN *et al.*, 2015).

2.3.1 Redes Neurais Convolucionais (CNN)

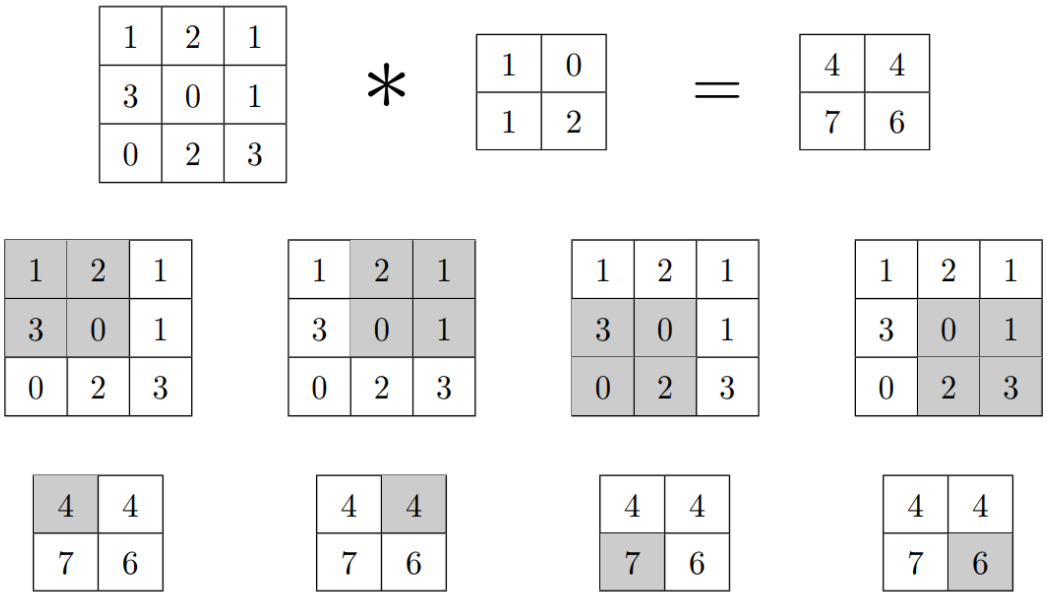
As CNN's (LECUN *et al.*, 1998) são a chave da inovação no aprendizado através de imagens (HAUSKNECHT, 2016). Elas são projetadas de modo a processar dados de entrada na forma de vetores multidimensionais, como imagens em duas dimensões, ou três canais de cores de imagens (LECUN *et al.*, 2015). Uma Rede Neural Convolucional é composta de quatro tipos de camadas (LECUN *et al.*, 2015): *camadas convolucionais*, *camadas de pooling*, *camadas completamente conectadas* e a *camada de saída*.

Como visto em Elst (2016), uma camada convolucional consiste de diversos *mapas de características*. Cada um deles recebe apenas um pedaço da saída da camada anterior, chamado de *campo receptivo*. Essa propriedade é chamada de *conectividade local*, uma vez que nem todas as unidades são conectadas entre si. Outra propriedade, chamada de *compartilhamento de pesos*, é que as unidades de um mesmo mapa de características compartilham o mesmo conjunto de pesos, chamados de *filtro de convolução*. A operação de multiplicação realizada entre o filtro de convolução e o campo receptivo é chamada de *convolução discreta*.

Uma camada convolucional possui alguns parâmetros comuns que podem variar ao longo de camadas diferentes (ELST, 2016). O primeiro é o *tamanho do filtro*, que determina o tamanho do campo receptivo conectado a cada camada. O *tamanho do passo* controla o deslocamento em que o filtro esquadrinha a camada anterior, ou seja, se o tamanho do passo

é 1×1, o deslocamento do filtro será de uma unidade horizontalmente e de uma unidade verticalmente. Por fim, a entrada pode ser *preenchida com zeros* em um certo número de linhas e colunas, ou seja, serão criadas linhas e colunas virtuais de valores zero virtuais que aumentarão o tamanho da entrada sem influenciar o cálculo de saída. A Figura 3 ilustra a convolução discreta em uma camada convolucional.

Figura 3 – Ilustração de uma convolução discreta que ocorre em uma camada convolucional. O filtro de convolução tem tamanho 2×2 e o tamanho do passo é 1×1.

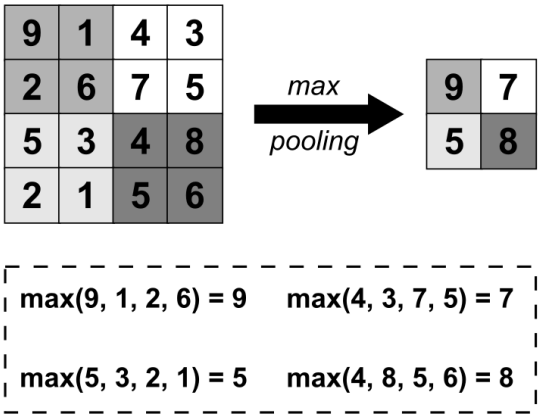


Fonte: Elst (2016).

A camada de *pooling* realiza uma subamostragem da entrada. Ao invés de multiplicar os pesos recebidos por um certo filtro, é aplicada uma função ao valores (ELST, 2016). As funções mais comuns são a de média e a de valor máximo, chamadas de *mean pooling* e *max pooling*, respectivamente (SCHMIDHUBER, 2015). O processo de *max pooling* é exemplificado na Figura 4.

De modo geral, uma camada convolucional identifica características específicas da imagem de entrada e uma camada de *pooling* reduz a dimensionalidade e a complexidade dessas características, indicando se elas estão presentes ou não na imagem. Com a introdução de invariância às translações realizadas, isto é, a irrelevância do posicionamento, informações espaciais a respeito da localização das características podem ser perdidas após a camada de *pooling*. Portanto, pode ser mais adequado não utilizá-la em problemas em que a localização das propriedades aprendidas sejam importantes, como em um jogo.

Figura 4 – Ilustração de uma operação de *max pooling* que ocorre em uma camada de *pooling*. O filtro de convolução e o passo têm tamanho 2×2.

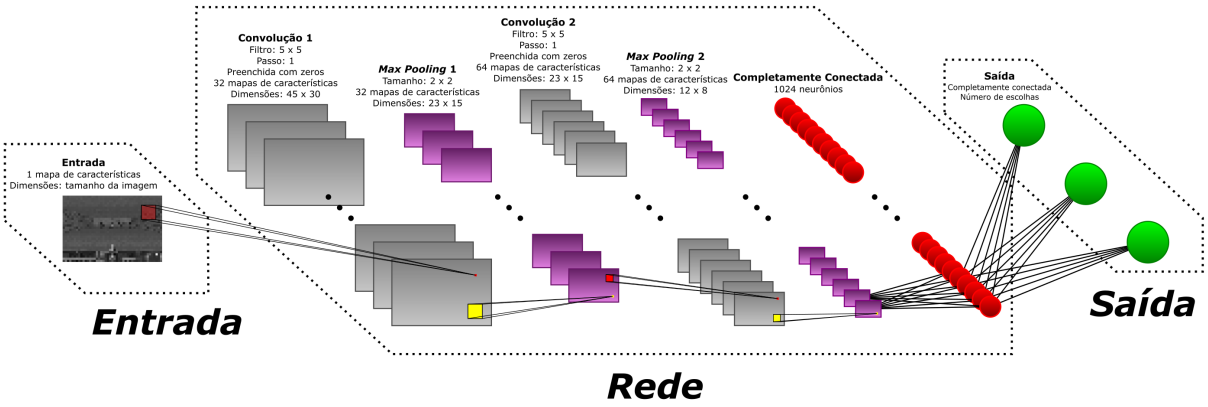


Fonte: Elaborado pelo autor.

O conjunto das duas camadas anteriores é chamado de *camada de extração de características*, pois elas irão retirar as informações de propriedades de uma determinada entrada (ELST, 2016). Vários conjuntos de extração são postos em sequência de modo que várias características sejam absorvidas pela rede neural antes das camadas finais.

As camadas completamente conectadas são as responsáveis por reunir todas as características descobertas anteriormente e passar os resultados para as camadas mais externas. Por fim, a camada de saída recebe os valores descobertos ao longo da rede e dão o resultado final. Uma arquitetura típica de CNN é mostrada na Figura 5

Figura 5 – Exemplo de uma arquitetura típica de CNN, contendo duas camadas convolucionais seguidas de camadas de *max pooling*.



Fonte: Adaptado de Serafim *et al.* (2017b).

2.3.2 Deep Q-Network (DQN)

O modelo que combina *Deep Learning* e *Q-Learning* foi desenvolvido por Mnih *et al.* (2013, 2015) e é chamado de DQN. Nele, o aproximador da função-Q utilizado é uma DNN.

Se $Q_\theta(s,a) \approx Q^*(s,a)$, é natural pensar que Q_θ deve estar próximo de satisfazer a equação de otimalidade de Bellman, o que leva à função de custo:

$$L_t(\theta_t) = \mathbb{E}\left[(y_t - Q_{\theta_t}(s_t,a_t))^2 \mid s_t, a_t, r_t, s_{t+1}\right], \tag{2.16}$$

onde:

$$y_t = r_t + \gamma \cdot \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}). \tag{2.17}$$

Portanto,

$$L_t(\theta_t) = \mathbb{E}\left[(r_t + \gamma \max_{a_{t+1}} Q_{\theta_{t-1}}(s_{t+1}, a_{t+1}) - Q_{\theta_t}(s_t,a_t))^2 \mid s_t, a_t, r_{t+1}, s_{t+1}\right]. \tag{2.18}$$

O valor de y é fixo para as iterações anteriores, o que leva ao seguinte gradiente através da diferenciação do custo em relação aos pesos:

$$\nabla_{\theta_t} L_t(\theta_t) = \mathbb{E}\left[(y_t - Q_{\theta_t}(s_t,a_t)) \nabla_{\theta_t} Q_{\theta_t}(s_t,a_t) \mid s_t, a_t, r_t, s_{t+1}\right]. \tag{2.19}$$

Ao invés de usar uma estimativa precisa do gradiente acima, ele é calculado usando a aproximação:

$$\nabla_{\theta_t} L_t(\theta_t) \approx (y_t - Q_{\theta_t}(s_t,a_t)) \nabla_{\theta_t} Q_{\theta_t}(s_t,a_t). \tag{2.20}$$

As atualizações do *Deep Q-Learning*, usando as estimativas da função de custo da Equação 2.16, são estáveis e se comportam bem na prática (MNIH *et al.*, 2015).

2.4 Ambiente ViZDoom

Desenvolvido por Kempka *et al.* (2016), o *ViZdoom* é um ambiente de pesquisa para problemas de Aprendizado por Reforço que utilizam visão e é baseado no jogo FPS *Doom* (id Software, 1993). Essa plataforma fornece um mundo virtual em três dimensões, semi-realista e com uma perspectiva de visão em primeira-pessoa. Ela permite o desenvolvimento de jogadores controlados por inteligência artificial utilizando o *buffer* da tela como visão (Figura 6).

Figura 6 – Captura de tela do ambiente *ViZDoom*.

Fonte: Kempka *et al.* (2016).

Uma das principais características dessa plataforma é a facilidade de personalização. Podem ser criados mapas, cenários, inimigos e elementos do ambiente sem nenhuma restrição de acesso ao código-fonte. Também podem ser modificados os parâmetros dos cenários, como valores de recompensa, objetivo, ações disponíveis e vários outros.

A Interface de Programação de Aplicação (*Application Programming Interface* (API)) do *ViZDoom* é flexível e fácil de manipular. Ela foi projetada tendo em mente tarefas de Aprendizado por Reforço. Ela permite o acesso ao *buffer* da tela e realizar ações correspondentes às ações de teclado e *mouse*. Também dá acesso direto às variáveis de estado do jogo, como a saúde do jogador ou a quantidade de munição disponível. A API foi escrita em linguagem *C++* e oferece suporte às linguagens *Python*, *Java* e *Lua*.

O *ViZDoom* implementa quatro modos de controle: jogador síncrono, em que o jogo executa a uma taxa constante de 35 quadros por segundo, de forma que os agentes atuam em seu próprio ritmo, sendo o modo mais adequado para as tarefas de Aprendizado por Reforço; jogador assíncrono, em que o jogo executa uma taxa de atualização variável, fazendo com que os jogadores possam vivenciar diferentes velocidades do jogo; e os modos de espectador síncrono e assíncrono, em que exclusivamente um jogador humano controla as ações.

O ambiente também permite a execução de cenários personalizados. Qualquer cenário que siga o padrão estabelecido pode ser executado com suas próprias regras, ações possíveis, valores recebidos e condições terminais. A criação de cenários personalizados deve ser feita através de ferramentas externas desenvolvidas pela comunidade.

Outra característica muito importante do *ViZDoom* é a possibilidade de uma renderização fora da tela, ou seja, a visão do jogo não é mostrada. A tela é representada por matrizes

de valores no intervalo [0, 1], que correspondem aos *pixels* da imagem, e essas matrizes são passadas para os agentes, mas não são renderizadas na tela. Dessa forma, o jogo processa as ações levando em conta a visão, mas o desempenho é bastante acelerado.

2.5 Considerações finais

Para resolver um problema em que um agente deve aprender quais ações ele deve realizar dentro de um ambiente que dê uma recompensa para cada escolha, em especial um jogo FPS, utiliza-se o paradigma de Aprendizado por Reforço. Dentre as maneiras de gerar comportamentos adequados, as redes neurais também podem ser utilizadas. Neste capítulo, foram apresentados alguns conceitos e modelos relacionados aos problemas de Aprendizado por Reforço. Especificamente, foram mostradas técnicas para resolver esse tipo de problema através da utilização de DNN. No próximo capítulo, são abordados os trabalhos relacionados à técnica proposta neste trabalho.

3 TRABALHOS RELACIONADOS

3.1 Introdução

O foco deste trabalho encontra-se no desenvolvimento e na avaliação de agentes autônomos cujo controlador é construído através de uma DNN e competem entre si em um jogo digital como ambiente de testes. Dessa forma, neste capítulo são apresentados os trabalhos que desenvolvem agentes autônomos utilizando aprendizado por reforço através do modelo de DQN em jogos digitais de modo geral.

Na Seção 3.2 são apresentados os trabalhos em que um único agente autônomo está presente. Ela é dividida de acordo com o jogo escolhido como ambiente de testes. A Seção 3.3 mostra os trabalhos em que múltiplos agentes autônomos competem nos ambientes. Alguns trabalhos apresentam agentes estritamente competitivos, enquanto outros desenvolvem características simultâneas de competição e cooperação.

3.2 Aprendizado Profundo por Reforço em único agente autônomo

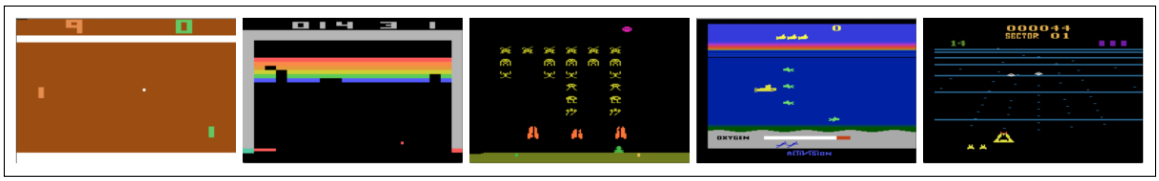
Nesta seção são apresentados alguns trabalhos que utilizam *Deep Learning* como arquitetura de redes neurais no desenvolvimento de inteligência artificial para o comportamento de um único personagem de jogos digitais. Inicialmente são mostrados aqueles cujos escopos limitam-se a jogos do console Atari 2600, bastante populares desde o artigo de Mnih *et al.* (2013). Posteriormente, são visitados os trabalhos aplicados utilizando *ViZDoom* (KEMPKA *et al.*, 2016). Esses trabalhos são particularmente relevantes pelo fato do *ViZDoom* ter servido como ambiente de testes para agentes autônomos que utilizam DNN (ARULKUMARAN *et al.*, 2017). Por fim, são mostrados trabalhos em outros jogos.

3.2.1 Jogos de Atari 2600

Os jogos do console Atari 2600 (Figura 7) são bastante utilizados como plataformas de teste para arquiteturas de *Deep Learning*. Isso se deve principalmente ao limite de *pixels* de entrada, à complexidade apresentada e à grande exposição ao público que possuem ainda hoje.

Um artigo pioneiro na utilização de *Deep Learning* em jogos foi o trabalho de Mnih *et al.* (2013). Nele, foi desenvolvido o primeiro modelo de *Deep Learning* a ser aplicado com sucesso usando uma entrada bruta de alta dimensão - ou seja, somente os *pixels* da tela -

Figura 7 – Cinco jogos de Atari 2600, da esquerda para a direita: *Pong*, *Breakout*, *Space Invaders*, *Seaquest*, *Beam Rider*.



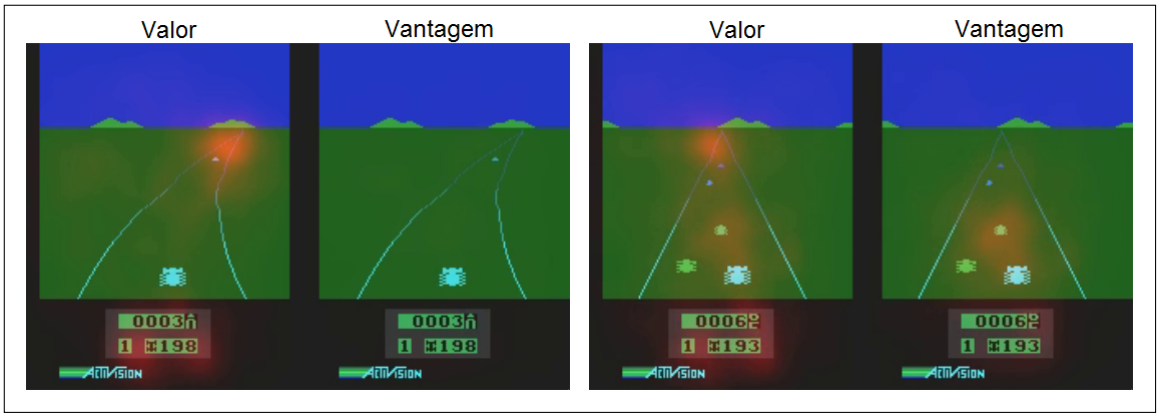
Fonte: Adaptado de Mnih *et al.* (2013).

utilizando aprendizado por reforço e uma rede neural convolucional treinada através da variante da técnica de aprendizado *Q-Learning* para DNN chamada de DQN.

O método foi aplicado a sete jogos de Atari 2600 dentro do Ambiente de Aprendizado de Jogos Arcade (do inglês, *Arcade Learning Environment*, (BELLEMARE *et al.*, 2013)). Esse ambiente é um emulador e uma plataforma de avaliação de tecnologias de Inteligência Artificial em mais de 50 jogos de Atari 2600. Em seis dos sete jogos testados, a técnica utilizada ultrapassou os resultados alcançados em abordagens anteriores e em três o método superou um humano especialista.

Desde então, muitos trabalhos utilizaram *Deep Learning* em jogos de Atari 2600. Wang *et al.* (2016) apresentaram uma abordagem utilizando uma rede que representa dois estimadores separados, um para a função de valor do estado e outra para a função de vantagem de ação dependente do estado. Na prática, o primeiro estimador se preocupa com o resultado a ser obtido (portanto atenta-se mais ao futuro), enquanto o segundo prioriza informações de ações imediatas (Figura 8). Essa abordagem levou a resultados que mostram uma melhor avaliação de ações, além de superar o estado da arte.

Figura 8 – Mapas de dois momentos distintos do jogo *Enduro* mostrando em vermelho os pontos em evidência de cada estimador, valor e vantagem.



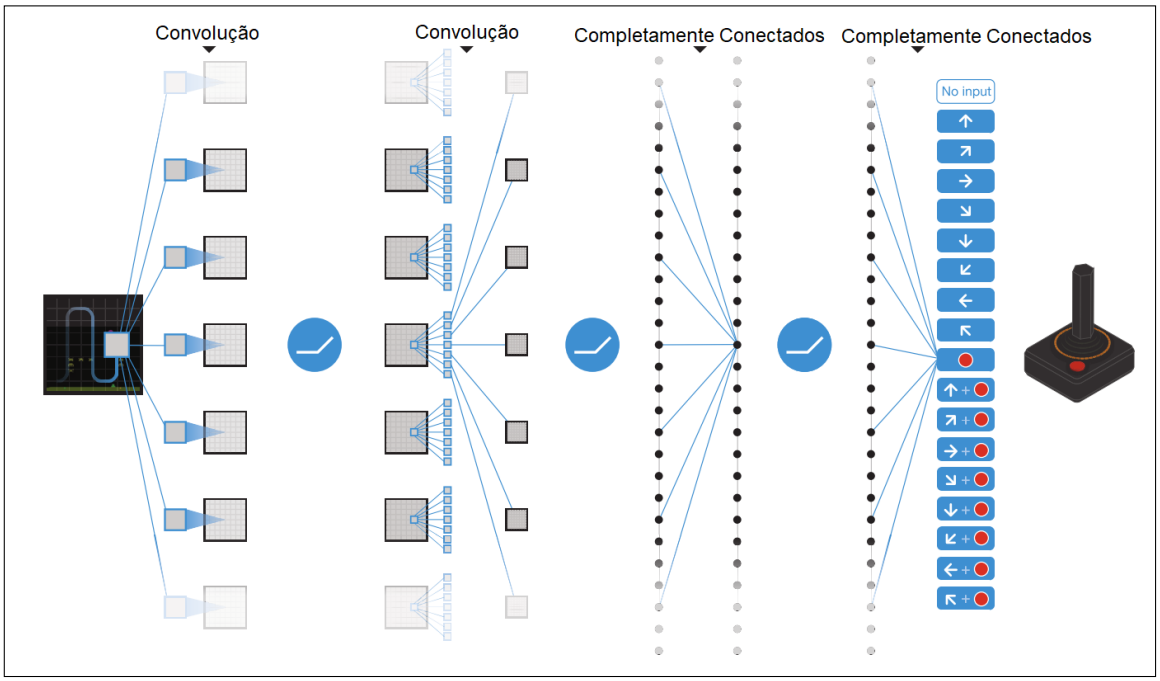
Fonte: Adaptado de Wang *et al.* (2016).

Parisotto *et al.* (2016) propuseram um novo método de treinamento de uma única rede profunda de ações sobre um conjunto de tarefas relacionadas. O método proposto, chamado de *Actor Mimic*, consiste no aprendizado da tomada de decisão de um agente autônomo através da orientação de professores especialistas, ou seja, redes neurais especializadas em certos objetivos. Estas redes neurais guiam o agente de modo a atingir os seus próprios objetivos. Os autores então aplicam a técnica a jogos de Atari 2600 de modo a comprovar a sua eficácia, mostrando que o método é capaz de atingir um desempenho de especialista em múltiplos jogos simultaneamente.

Hasselt *et al.* (2016) levantam uma série de questões em relação ao modo como as DQN funcionam e as limitações que elas impõem ao aprendizado. Os autores propõem uma modificação utilizando *Double Q-Learning*, que consiste no aprendizado simultâneo de duas funções de valor - o que resulta em dois conjuntos de pesos. Essa modificação gera resultados menos superestimados e levam a um melhor desempenho da rede.

Mnih *et al.* (2015) atualizaram seu trabalho prévio, realizando uma avaliação mais aprofundada dos modelos. Os autores aplicaram a mesma técnica de *Deep Q-Learning* em jogos de Atari 2600, porém foram usados 49 jogos dessa vez. Os autores utilizaram o mesmo algoritmo, a mesma arquitetura de rede (Figura 9) e os mesmos hiperparâmetros. A entrada foi, mais uma vez, somente os *pixels* da tela sem nenhum tratamento. Em 29 jogos o desempenho foi melhor do que o de humanos especialistas.

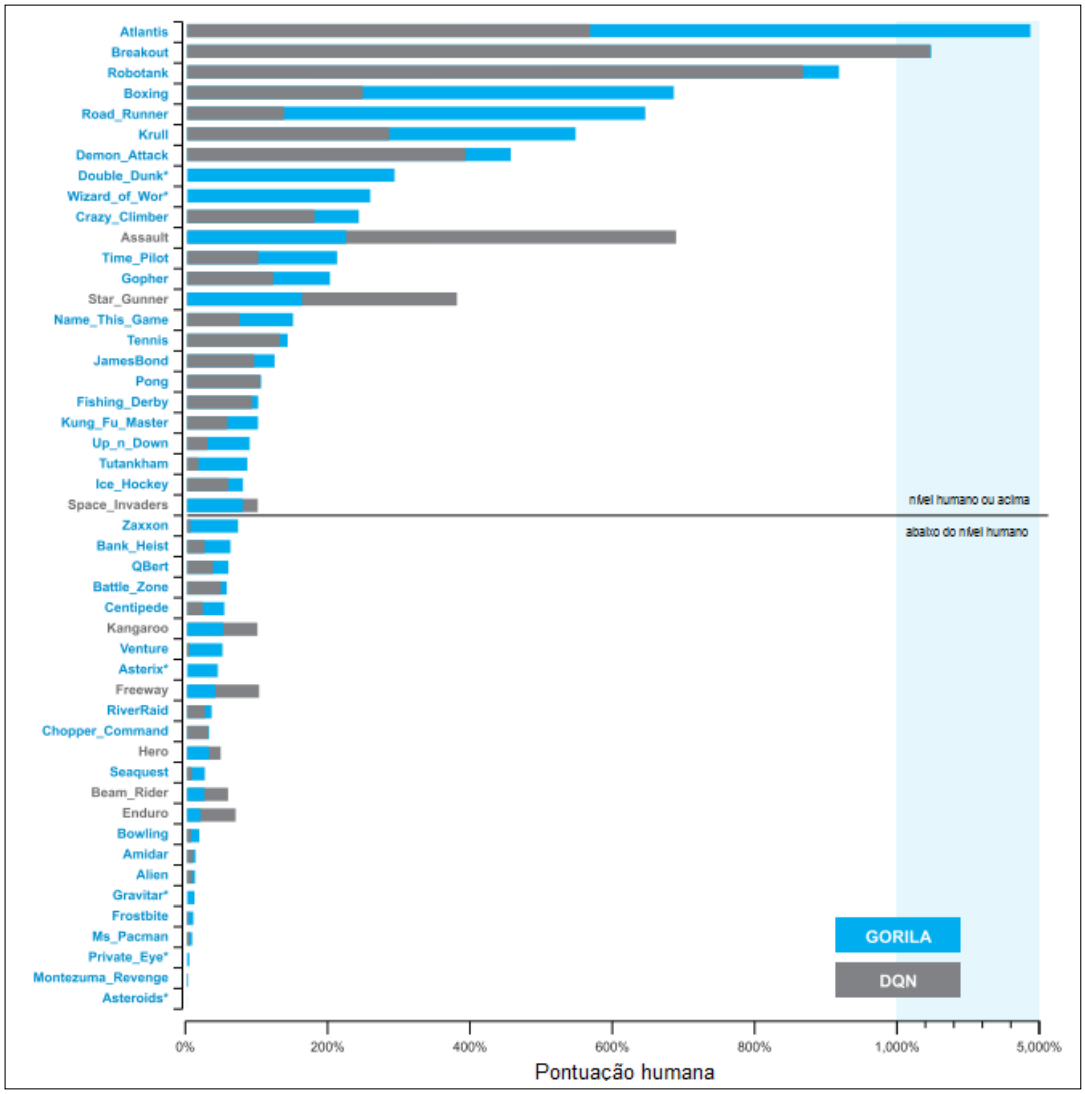
Figura 9 – Arquitetura da rede neural utilizada por Mnih *et al.* (2015).



Fonte: Adaptado de Mnih *et al.* (2015).

Nair *et al.* (2015) modificaram a técnica de Mnih *et al.* (2013, 2015) utilizando uma arquitetura distribuída e a chamaram de *Gorila*. A paralelização foi capaz de aumentar o desempenho em comparação com as abordagens anteriores. O modelo foi testado nos mesmos 49 jogos de Atari 2600 e em 41 deles a técnica em paralelo superou as não-distribuídas (Figura 10).

Figura 10 – Comparação do desempenho da técnica *Gorila* com DQN. No eixo das abscissas a pontuação de jogadores humanos.



Fonte: Adaptado de Nair *et al.* (2015).

Liang *et al.* (2016) realizaram uma análise minuciosa das principais características de DQN que garantem seu elevado desempenho. Os autores propuseram um modelo de representação linear com uma arquitetura rasa de rede neural que incorporava essas características. A técnica foi verificada em nove jogos de Atari 2600 e os autores obtiveram resultados competitivos com os de *Deep Learning* em um ambiente específico desenvolvido por eles mesmos.

Mnih *et al.* (2016) propuseram uma abordagem utilizando gradiente descendente assíncrono para a otimização dos controles. Os autores apresentaram variações de quatro técnicas de aprendizado por reforço e mostraram que o melhor método obtido superou o estado da arte. Além disso, o esforço computacional foi bastante reduzido, pois o melhor modelo foi treinado em um único processador de múltiplos núcleos em metade do tempo necessário ao treinamento do modelo convencional.

Romoff *et al.* (2016) propuseram uma nova abordagem aos modelos anteriores: utilizar Redes Neurais Condicionais (BENGIO *et al.*, 2013; DAVIS; AREL, 2013), um tipo de rede neural que seletivamente ativa somente partes da rede por vez. Essas redes seriam utilizadas para quebrar a grande rede neural principal em blocos que possam ser individualmente especializados em resolver um problema particular. A técnica foi testada em três jogos de Atari 2600 e os resultados obtidos foram inconclusivos, pois algumas vezes há uma melhora no desempenho, enquanto em outras há uma piora. Os autores afirmam que esse comportamento ocorre devido à aleatoriedade presente no modelo e novas investigações seriam realizadas.

3.2.2 Plataforma ViZDoom

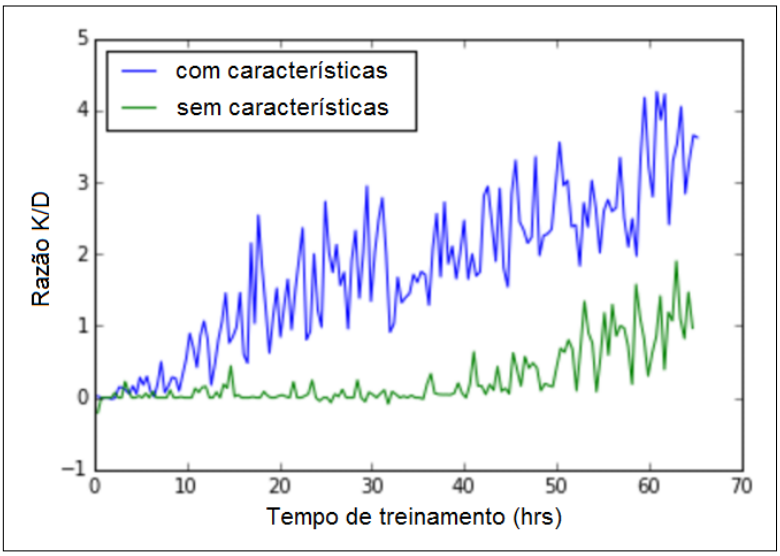
Recentemente, o jogo *Doom* (*id Software, 1993*) vem sendo bastante utilizado em aplicações de DRL. Lample e Chaplot (2017) apresentaram a primeira arquitetura adequada a ambientes de três dimensões em *Doom*, um jogo do subgênero FPS (Figura 11). Os autores utilizaram arquiteturas de uma *Deep Recurrent Q-Network* conectada a uma Rede Neural de Memória de Curto-Longo Prazo (*Long-Short Term Memory* (LSTM)).

Figura 11 – Captura de tela do jogo *Doom*.

Fonte: Lample e Chaplot (2017).

A abordagem explora informações de características do jogo, como presença ou ausência de inimigos e itens, durante a fase de treinamento. Simultaneamente ao aprendizado dessas particularidades, ocorre a minimização da função objetivo do *Q-Learning*. Esse aspecto evidenciou um aumento dramático da velocidade de aprendizado e desempenho do agente em relação a uma versão sem exploração das características. A relação de mortes causadas por mortes sofridas, a razão K/D, melhora bem mais quando há informações de características ao longo do treinamento (Figura 12).

Figura 12 – Diferença de resultados com e sem informações de características.

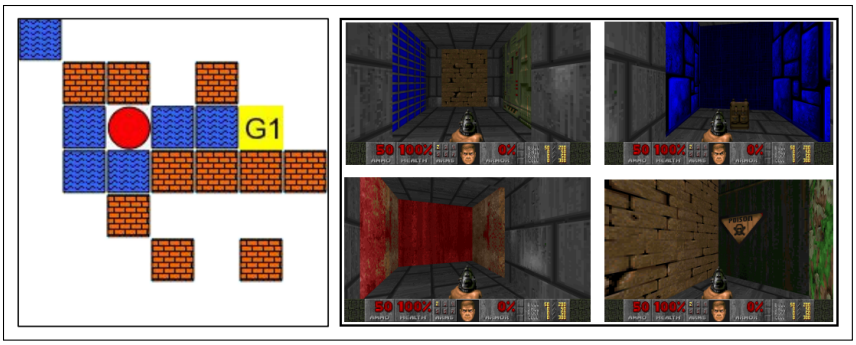


Fonte: Adaptado de Lample e Chaplot (2017).

Outra aplicação usando DRL na plataforma *ViZDoom* foi realizada por Kulkarni *et al.* (2016). Os autores usaram um modelo chamado Representações Sucessivas, que decompõe a função de valor em dois componentes: um preditor de recompensa e um mapa de sucessão. O agente foi treinado para resolver dois problemas: encontrar a saída de um labirinto na plataforma *MazeBase* (SUKHBAATAR *et al.*, 2015) e alcançar uma sala do ambiente *ViZDoom* (Figura 13).

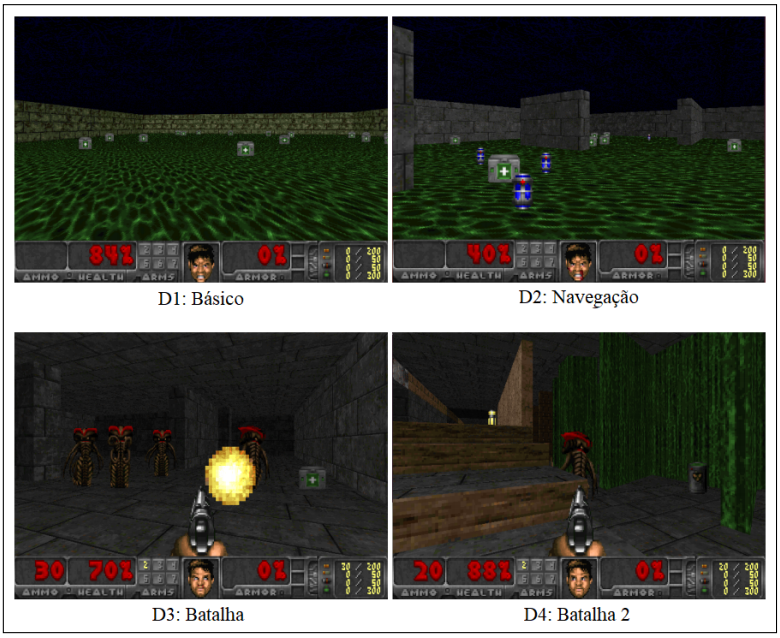
Dosovitskiy e Koltun (2017) combinaram um fluxo sensorial de alta dimensão, a imagem recebida como entrada, com um fluxo de medição de baixa dimensão, dados externos como saúde e quantidade de munição, para criar um modelo que fornece um rico sinal de supervisão, criando uma "visão de futuro". O agente foi treinado para jogar quatro cenários diferentes do *ViZDoom* usando técnicas de aprendizado supervisionado (Figura 14). O personagem treinado venceu a trilha *Full Deathmatch* da Competição de *ViZDoom* (*ViZDoom AI Competition*) em 2016.

Figura 13 – Visão do ambiente *MazeBase* (esquerda) e do labirinto no jogo *Doom* (direita).



Fonte: Kulkarni *et al.* (2016).

Figura 14 – Captura de tela dos quatro cenários utilizados.

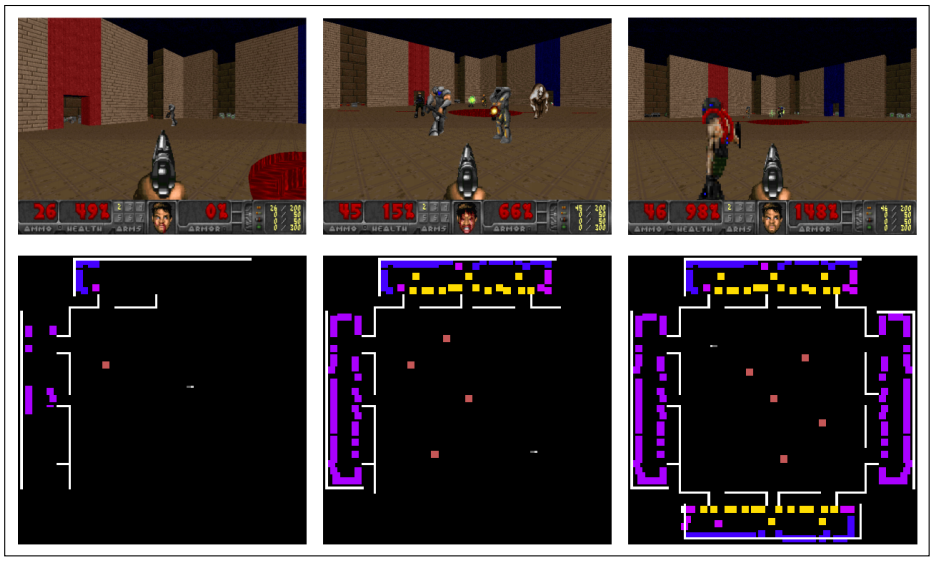


Fonte: Dosovitskiy e Koltun (2017).

Bhatti *et al.* (2016) estenderam a imagem de entrada bruta da rede neural adicionando informações de localização do jogador, e detalhes de objetos e elementos estruturais encontrados. Dessa forma, um "mapa semântico" é criado e fornecido ao personagem, de modo que ele tenha uma visão completa do ambiente (Figura 15). Eles usaram um agente autônomo na plataforma *ViZDoom* para avaliar a eficácia dessa abordagem. Os autores mostraram que o *framework* de extensão proposto aprende consistentemente políticas eficientes.

Uma abordagem usando Neuroevolução foi realizada por Alvernaz e Togelius (2017). Os autores treinaram uma rede *autoencoder* para criar uma representação em baixa dimensão da observação do ambiente e usá-la como entrada de uma Estratégia de Evolução por Adaptação da Matriz de Covariância (*Covariance Matrix Adaptation Evolution Strategy* (CMA-ES)) para

Figura 15 – A visão limitada do personagem (topo) e o mapa semântico gerado (baixo).



Fonte: Bhatti *et al.* (2016).

treinar os controladores. Eles testaram o modelo no cenário de coleta de vida da plataforma *ViZDoom* (KEMPKA *et al.*, 2016) em que os jogadores perdem vida ao longo do tempo e precisam coletar caixas de vida para sobreviver por mais tempo, apresentando resultados variados.

Pathak *et al.* (2017) utilizaram uma exploração dirigida por curiosidade em agentes desenvolvidos para jogar *Super Mario Bros.* e *Doom*. Os autores mostraram que a abordagem aumentou o desempenho de outros modelos quando usada em conjunto com estes. Eles também mostraram que um agente é capaz de atingir um comportamento adequado mesmo se não receber recompensas.

Wu e Tian (2017) utilizaram o modelo assíncrono proposto por Mnih *et al.* (2016) juntamente com *Curriculum Learning* (BENGIO *et al.*, 2009) no ambiente *ViZDoom* usando apenas os estados do jogo. O treinamento foi realizado de maneira progressiva através de disputas contra agentes pré-programados embutidos no ambiente. O agente desenvolvido foi o vencedor da trilha *Limited Deathmatch* da Competição de *ViZDoom* (*ViZDoom AI Competition*) em 2016.

3.2.3 Outros jogos

Outros trabalhos surgiram na área usando diferentes jogos como bases de testes. Chen e Yi (2016) apresentaram um modelo convolucional para jogos 3D mais complexos. A técnica foi aplicada nos jogos Super Smash Bros. e Mario Tennis, ambos para o console Nintendo 64 (Figura 16).

Figura 16 – Captura de tela dos jogos Mario Tennis (esquerda) e Super Smash Bros. (direita) executando na adaptação realizada pelos autores.

Fonte: Chen e Yi (2016).

Para a construção da rede neural, houve inicialmente uma diminuição das dimensões do jogo, de 334 $\times$ 246 *pixels* para 128 $\times$ 128 *pixels*. Esses novos *pixels* foram passados como entrada da rede neural. No trabalho, eles demonstram que, apesar de não ser capaz de vencer um humano consistentemente, a aplicação foi capaz de rivalizar o desempenho da Inteligência Artificial pré-programada nos jogos com apenas duas épocas de treinamento.

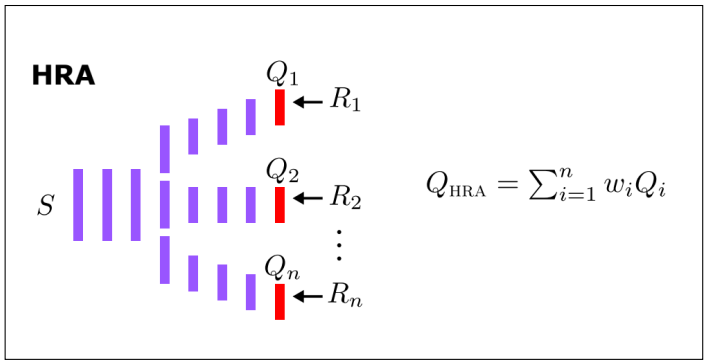
Para tentar resolver o problema da generalização, Seijen *et al.* (2017) propõem um novo método chamado Arquitetura de Recompensa Híbrida (*Hybrid Reward Architecture* (HRA)). Uma HRA recebe como entrada uma função de recompensa decomposta e aprende uma função de valor separada para cada componente da função de recompensa, permitindo um aprendizado mais eficiente.

O método HRA consiste de vários agentes que aprendem diferentes tipos de comportamento em paralelo baseados em uma política individual. A coleção de agentes pode ser vista como um único agente com múltiplas "cabeças", em que cada uma delas produz um valor de ação do estado atual baseada em uma função de recompensa diferente (Figura 17). Desse modo, há uma união dos comportamentos em um único agente, que apresenta uma capacidade maior de generalização. No trabalho, os autores utilizam essa técnica para resolver uma única tarefa complexa. O método foi testado em um jogo personalizado e no jogo de Atari 2600 Ms. Pac-Man. Um desempenho acima do humano foi obtido no jogo Ms. Pac-Man, uma realização inédita até então.

3.3 Aprendizado Profundo por Reforço em múltiplos agentes

Recentemente, os modelos de *Deep Learning* passaram a ser utilizados em experimentos de Aprendizado por Reforço com Múltiplos Agentes (*Multi-Agent Reinforcement Learning* (MARL)). Diversas abordagens combinadas de *Deep Q-Learning* e MARL, também chamada de

Figura 17 – Ilustração da Arquitetura de Recompensa Híbrida.



Fonte: Seijen *et al.* (2017).

Multi-Agent Deep Reinforcement Learning (MADRL) estão sendo testadas, incluindo aplicações em robótica (LEVINE *et al.*, 2016), resolução de enigmas (FOERSTER *et al.*, 2016), jogos matriciais (LANCTOT *et al.*, 2017) e pôquer (HEINRICH *et al.*, 2015; YAKOVENKO *et al.*, 2016; MORAVČÍK *et al.*, 2017). De modo geral, elas podem ser divididas em: cooperação entre agentes, competição entre agentes e abordagens mistas.

Vários trabalhos utilizam jogos digitais como ambientes de testes. Muitos deles foram desenvolvidos avaliando unicamente a cooperação entre agentes (DAS *et al.*, 2017; FOERSTER *et al.*, 2016; FOERSTER *et al.*, 2017; GUPTA *et al.*, 2017; HAUSKNECHT, 2016; KONG *et al.*, 2017; PALMER *et al.*, 2017; SUKHBAATAR *et al.*, 2016). Os trabalhos desenvolvidos com competição são descritos a seguir.

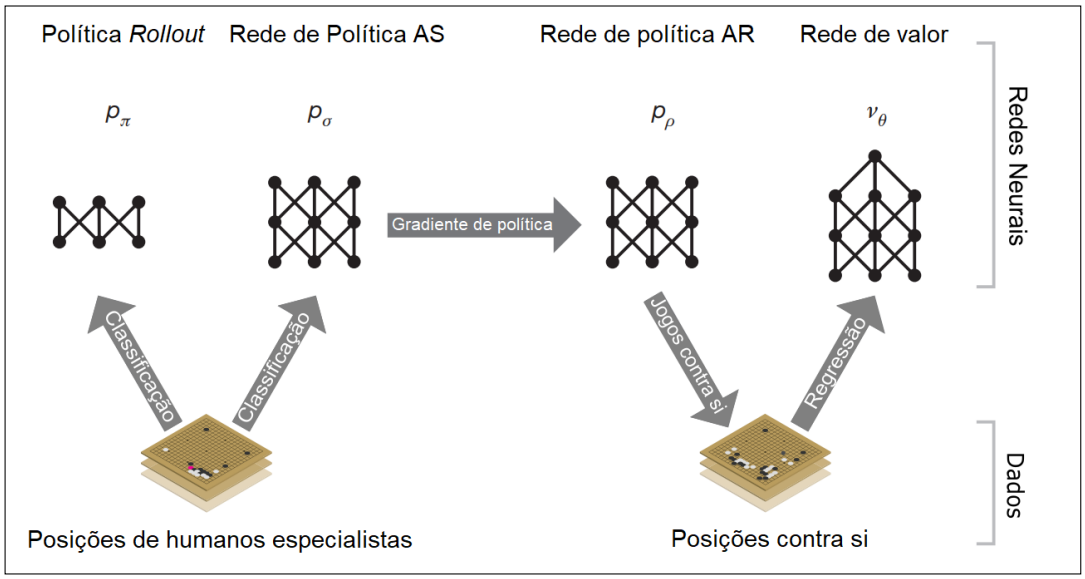
3.3.1 Estritamente competitivos

Maddison *et al.* (2014) utilizaram a técnica de *Deep Learning* para avaliação de jogadas e construção de um jogador para o jogo Go. Os resultados mostraram que a rede treinada foi capaz de vencer jogadores amadores humanos e rivalizar com o estado da arte em árvore de busca de Monte-Carlo. Embora esse trabalho não se enquadre como MARL, uma vez que utiliza somente o paradigma de aprendizado supervisionado para treinamento do agente, ele foi o passo inicial para o trabalho revolucionário descrito a seguir.

Em outubro de 2016, um jogador autônomo de Go supreendeu ao vencer o então campeão europeu por 5 vitórias a 0. O modelo, batizado de AlphaGo, foi desenvolvido por Silver *et al.* (2016). A abordagem utilizada combinou aprendizado supervisionado, aprendizado por reforço e árvore de busca de Monte-Carlo para escolher a melhor ação a ser realizada em um dado momento do jogo.

O treinamento ocorreu em etapas. O treinamento inicial utilizou aprendizado supervisionado em duas redes de políticas diferentes para predizer os movimentos seguintes, a partir de movimentos de jogadores humanos especialistas. Em seguida, ocorreu um treinamento da rede utilizando aprendizado por reforço, cujo objetivo era maximizar o número de jogos ganhos, através de partidas contra uma versão anterior de si-mesmo. Por fim, uma rede de avaliação do valor das jogadas foi treinada através de regressão para predizer qual jogador seria o vencedor, utilizando os dados da etapa anterior (Figura 18).

Figura 18 – Etapas de treinamento das redes neurais.



Fonte: Adaptado de Silver *et al.* (2016).

Após o treinamento, as redes de política e valor foram combinadas utilizando árvore de busca de Monte-Carlo. Essa abordagem alcançou uma porcentagem de 99.8% de vitórias contra outros programas de Go e venceu, em março de 2016, o então campeão mundial de Go por 4 a 1. Essa foi a primeira vez que um programa de computador venceu um jogador profissional no jogo completo de Go.

He *et al.* (2016) observaram que em ambientes competitivos com múltiplos agentes, as variáveis e informações preditivas de comportamento dos adversários dependem bastante da aplicação específica em que os testes ocorrem. Os autores buscaram desenvolver um *framework* geral para a modelagem de oponentes em tarefas de natureza de aprendizado por reforço, o que permitiria aos agentes aprenderem características específicas de seus adversários.

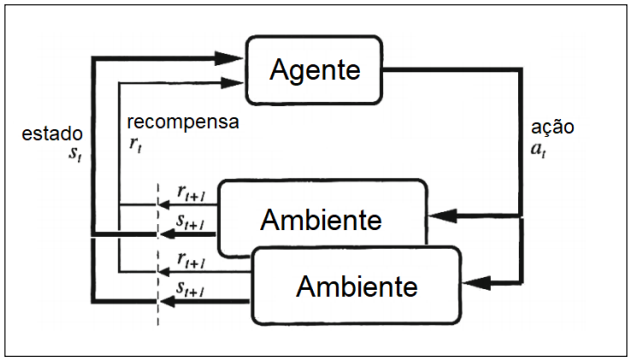
Os autores desenvolveram um modelo chamado de *Deep Reinforcement Opponent Network* (DRON), que possui um módulo de predição dos valores-Q e um módulo de aprendizado

do oponente que infere estratégias dos adversários. Em vez de predizer explicitamente as propriedades dos oponentes, o DRON aprende representações ocultas dos adversários baseado em observações passadas e as usa para calcular uma resposta adaptativa.

A abordagem de modelagem de oponentes proposta incorporou, implicitamente, predições dos comportamentos dos adversários em sua política de aprendizado, sem conhecimento prévio do domínio. O modelo foi avaliado em um jogo de futebol simulado personalizado e em um jogo de perguntas e respostas, mostrando um desempenho superior ao dos modelos puramente de DQN e suas variantes.

McKenzie *et al.* (2017) observaram a habilidade de um agente usando o modelo de DQN aprender múltiplas tarefas. Diferentemente dos trabalhos de Mnih *et al.* (2013, 2015), em que o agente treinava e jogava um único jogo por vez, o objetivo seria jogar múltiplos jogos com um mesmo conjunto de treinamento. Para tanto, os autores utilizaram uma abordagem de treinamento em que o agente jogava dois jogos simultaneamente com uma única ação, isto é, a rede neural recebia a entrada de dois jogos diferentes, escolhia uma única ação a ser realizada e essa ação era reproduzida nos dois (Figura 19). Dessa forma, os autores indicam que há uma "competição" interna ocorrendo na rede neural.

Figura 19 – Adaptação do paradigma de aprendizado por reforço para uma ação simultânea em dois jogos.



Fonte: Adaptado de McKenzie *et al.* (2017).

Os resultados mostraram que o desempenho do agente foi pior do que o observado nas situações com um único jogo, porém melhor do que quando um agente é treinado em dois jogos de maneira separada. Esse fato mostra que a habilidade de generalização de aprendizado do modelo de DQN é limitada, mas pode ser melhorada através do treinamento competitivo em dois jogos simultaneamente.

3.3.2 Abordagens mistas de competição e cooperação

Uma característica comum em diversas tarefas coletivas é que vários problemas possuem objetivos tanto de competição quanto de cooperação. Os trabalhos que utilizam abordagens mistas de competição e cooperação entre múltiplos agentes desenvolvidos utilizando aprendizado por reforço serão descritos a seguir.

Utilizando o modelo de DQN, Tampuu *et al.* (2017) mostraram que ajustes nas estratégias de recompensas dos agentes fazem com que comportamentos competitivos e cooperativos emerjam. Os autores utilizaram o jogo *Pong* do console Atari 2600 como ambiente de testes.

Três modelos de recompensas foram utilizados. Um modelo completamente competitivo dava uma recompensa positiva para o agente que marcasse um ponto e penalizava o outro, de modo que o objetivo dos jogadores era marcar o maior número de pontos. Um modelo completamente cooperativo penalizava os dois agentes caso algum deles pontuasse, assim, o objetivo de ambos era manter a bola em jogo o máximo de tempo possível. E um terceiro modelo dava uma bonificação variável para o agente que pontuasse, para verificar a transição da competição para a cooperação dentro do jogo. Os agentes competitivos aprenderam a jogar e a marcar pontos de maneira eficiente, enquanto os agentes treinados em regime de recompensas colaborativas encontraram uma estratégia ótima para manter a bola em jogo o máximo de tempo possível.

Hausknecht e Stone (2016) utilizaram um modelo de redes neurais em um jogo de futebol simplificado. O jogo utilizado foi o Ataque em Metade do Campo (do inglês, Half Field Offense), um dos domínios que compõem a Liga de Simulação 2D (2D Simulation League) do Campeonato de Futebol RoboCup (KALYANAKRISHNAN *et al.*, 2007). Nesse jogo, um número determinado de agentes é posicionado aleatoriamente no campo, metade dos quais são atacantes e os demais são defensores. O objetivo do ataque é marcar um gol enquanto o dos defensores é tomar a bola.

Os autores desenvolveram um modelo de redes neurais para resolução de problemas de decisão de Markov com espaços de ações parametrizados. Esse tipo de problema é definido por um conjunto de ações discretas, cada uma contendo um número de parâmetros contínuos.

No artigo, foram treinados agentes atacantes independentes, isto é, que não compartilhavam informações entre si. Seus atributos constituem um espaço de ações parametrizado em que as ações possíveis são: arrancada (poder, direção), em que o agente se moverá na direção indicada com um poder escalar no intervalo [0, 100]; giro (direção), que indica um giro na

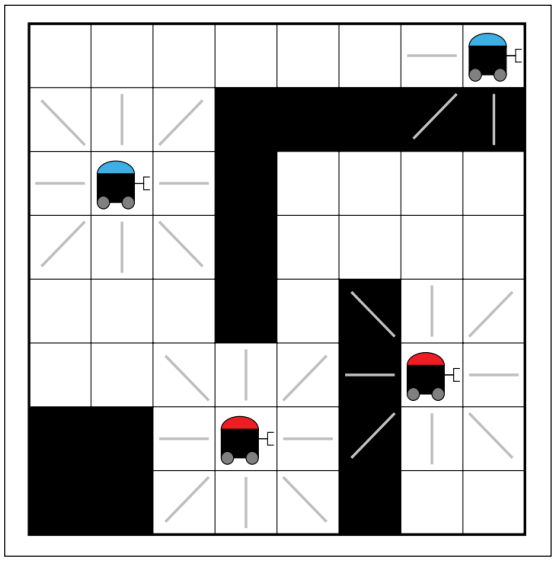
direção designada; investida (direção), onde o indivíduo irá investir contra o adversário para tentar roubar-lhe a bola; e chute (poder, direção), que consiste em um chute na direção indicada com poder no intervalo [0, 100]. O aprendizado ocorreu através de um modelo de aprendizado por reforço utilizando *Deep Learning*. Os agentes treinados com a técnica foram capazes de superar o campeão da competição RoboCup Soccer de 2012 na porcentagem de gols feitos.

Em sua dissertação de mestrado, Castañeda (2016) desenvolveu extensões de aprendizado por reforço para situações com múltiplos agentes. Os modelos foram testados em configurações com um único ou dois agentes autônomos, e foram comparados entre jogadores controlados por um modelo tradicional de DQN e as extensões desenvolvidas no trabalho.

O jogo *Pong* do console Atari 2600 foi utilizado como ambiente de testes e comparação em diversos experimentos competitivos e cooperativos. Os resultados demonstraram que para algumas tarefas os modelos estendidos superaram a técnica de DQN tradicional, o que sugere uma necessidade de desenvolver e utilizar arquiteturas capazes de copiar características com dinâmicas mais ricas e complexas.

Egorov (2016) propõe uma reformulação nas técnicas de DRL tradicionais para que o estado do sistema seja representado como imagens de entrada. O autor utiliza um jogo de perseguição e fuga de carros, em que cada carro é controlado por um agente. Dois agentes devem cooperar para fugir, enquanto outros dois devem cooperar para pegar o outro grupo primeiro (Figura 20).

Figura 20 – Ilustração do jogo criado por Egorov (2016). Os agentes vermelhos devem tentar capturar os agentes azuis. Os quadrados pretos são obstáculos.



Fonte: Egorov (2016).

O autor apresenta um trabalho inicial de generalização entre ambientes em um sistema de múltiplos agentes e de generalização entre agentes. Além disso, o autor mostra que pode-se usar transferência de aprendizado para aumentar o tempo de convergência dos resultados.

Leibo *et al.* (2017) estenderam as abordagens com jogos matriciais e criaram dois jogos digitais para avaliar características sociais de competição e cooperação entre agentes autônomos independentes, ou seja, sem compartilhamento de informações. Os autores analisaram as dinâmicas das políticas aprendidas pelos agentes em dois ambientes: um jogo de coleta de frutas e um jogo de caçada em grupo.

Os agentes são controlados por modelos de DQN. Como são independentes, cada um vê os outros como parte do ambiente, isto é, da perspectiva de um jogador o aprendizado dos outros se mostra como um ambiente não estacionário. Além disso, os agentes não raciocinam sobre o aprendizado dos outros.

Os autores observaram a emergência de comportamentos agressivos, no primeiro experimento, e colaborativos, no segundo. Fazendo ajustes nos modelos de recompensas, também foi possível observar como os conflitos podem surgir e ajudar a esclarecer como a natureza sequencial dos dilemas do mundo real afeta a cooperação.

3.4 Considerações finais

Neste capítulo foram apresentados os trabalhos relevantes que utilizam DRL em jogos digitais e trabalhos que apresentam competição entre agentes de jogos digitais. No próximo capítulo, é apresentada a técnica proposta neste trabalho. São mostrados trabalhos prévios a este e são detalhados: o jogo usado como ambiente de testes, o controlador dos agentes autônomos e as características específicas dos experimentos realizados.

4 TÉCNICA PROPOSTA

4.1 Visão Geral

A técnica proposta nesse trabalho busca desenvolver um agente autônomo de um jogo FPS capaz de aprender a se comportar de maneira adequada dentro do ambiente e avaliar se o desempenho desse tipo de agente pode ser melhorado pela utilização de competição durante seu treinamento. Primeiramente, são apresentados dois trabalhos prévios que desenvolvem agentes autônomos capazes de lidar com a tarefa em ambientes com um único jogador. Para a tarefa competitiva, pode-se observar a necessidade de se definir algumas questões relativas à maneira como o problema é atacado. Cada uma dessas questões é tratada individualmente.

Inicialmente, deve-se definir qual jogo será usado como base de testes para o ambiente em que o personagem estará inserido. Além disso, o agente deve possuir um controlador próprio capaz de lhe propiciar um aprendizado adequado. Os experimentos realizados devem possuir características específicas de acordo com o problema. Por fim, múltiplos agentes devem ser avaliados ao longo de seus treinamentos e testes competitivos. Portanto, o modelo geral da técnica proposta nesse trabalho segue a forma da Figura 21.

Figura 21 – Visão geral da técnica proposta.

Fonte: Elaborado pelo autor.

4.2 Trabalhos prévios

4.2.1 Usando uma arquitetura de classificação em um jogo de Tiro em Primeira Pessoa

Serafim *et al.* (2017b) utilizaram uma arquitetura de rede neural planejada para resolver um problema de aprendizado supervisionado. A arquitetura foi validada em uma tarefa de classificação de um conjunto de dígitos escritos à mão e foi aplicada a um problema de aprendizado por reforço, um jogo FPS (Figura 22). Os autores utilizaram um modelo de DNN para resolver ambas as tarefas. Nos dois casos, a entrada foi apenas o conjunto de *pixels* de uma imagem.

Figura 22 – Exemplos de entradas da rede neural.

(a) Imagens da tarefa de classificação. (b) Captura de tela dos cenários do jogo.

Fonte: Serafim *et al.* (2017b).

A plataforma *ViZDoom* foi utilizada como ambiente de testes para o segundo problema, usando DRL. Um agente autônomo foi construído para jogar dois cenários diferentes do jogo de tiro: um cenário simples, chamado *Basic*, contra apenas um inimigo estático foi rapidamente solucionado; e um cenário mais complexo, chamado *Defend the Line*, contra 6 inimigos de dois tipos diferentes, em que o agente foi capaz de aprender a matar diversos monstros sem nenhuma informação adicional além dos *pixels* de cada quadro.

Mostrou-se que uma mesma arquitetura de rede neural foi adequada para a tarefa de classificação e foi capaz de aprender a jogar os dois cenários de maneira adequada. Assim, abre-se uma perspectiva para resolver problemas de diferentes paradigmas de aprendizado com uma única arquitetura de rede, o que pode levar a uma solução potencialmente generalizada e a uma abordagem robusta para lidar com tarefas complexas.

4.2.2 Desenvolvimento de um agente autônomo de um jogo de Tiro em Primeira Pessoa usando Aprendizado Profundo por Reforço

Serafim *et al.* (2017a) se concentraram no desenvolvimento de um agente autônomo para um jogo FPS. A arquitetura da rede neural foi construída de maneira mais apropriada para o jogo e resultou em um agente com comportamentos mais adequados nos ambientes verificados.

Foram utilizados três cenários da plataforma *ViZDoom*: um cenário em que o agente enfrenta apenas um inimigo estático, chamado *Basic*, para validar se o aprendizado ocorria ou não; um cenário mais complexo contra 6 inimigos simultaneamente, chamado *Defend the Line*, para avaliar a capacidade de aprendizado contra múltiplos inimigos; e um cenário personalizado, chamado *Medikits and Poisons*, construído para avaliar a capacidade de exploração do agente, cujo objetivo era permanecer vivo o máximo de tempo possível (Figura 23). Em todos os casos, a entrada foi somente os *pixels* da tela.

Figura 23 – Os três cenários jogados: *Basic* (esquerda), *Defend the Line* (centro) e *Medikits and Poisons* (direita).

Fonte: Serafim *et al.* (2017a).

No primeiro cenário o agente desenvolvido foi capaz de aprender rapidamente um comportamento ótimo. Na segunda tarefa, o agente aprendeu não só a girar em direção a um inimigo e matá-lo, como também aprendeu a sobreviver tempo suficiente para matar diversos deles. O agente também mostrou-se adequado em um cenário planejado para exploração do ambiente, em que ele aprendeu a pegar itens que aumentasse seus pontos de vida e a evitar itens que poderiam matá-lo.

Os exemplos mostraram que o modelo proposto foi capaz de aprender comportamentos complexos em todos os ambientes. Desse modo, os autores apresentam um ponto inicial na construção de agentes autônomos de um jogo FPS capaz de lidar com situações bem distintas utilizando somente os *pixels* da tela como entrada.

4.3 O Jogo

Neste trabalho, devido à sua grande popularidade e visibilidade como base de testes no desenvolvimento de agentes autônomos utilizando DRL, como visto na Seção 3.2.2, o jogo utilizado foi *Doom* (Figura 11). Para isso, o desenvolvimento se dá dentro do ambiente *ViZDoom* (KEMPKA *et al.*, 2016), uma plataforma de pesquisa baseada no jogo *Doom*, projetada para pesquisa em Aprendizado de Máquina Visual e DRL. Essa ferramenta permite acesso direto ao motor de jogo e possibilita a construção de um agente personalizado que envia comandos e recebe informações acerca do estado atual do jogo.

4.3.1 Análise do Jogo Doom

Cardona *et al.* (2013) investigaram a adequação do jogo Ms. Pac-Man como uma base de testes para Coevolução Competitiva. Para tanto, os autores estabelecem três características que definem um jogo adequado para a tarefa:

- **Profundidade de desafio**: um desafio não deve ser trivial, para que certas estratégias vencedoras sejam alcançadas não pela facilidade de atingir o objetivo, mas sim como consequência de um aprendizado consistente;
- **Suavidade do desafio**: deve ser possível sair de uma estratégia trivial até uma muito boa em pequenos incrementos;
- **Transitividade da função de *fitness***: quanto melhor for uma estratégia contra alguma outra, melhor ela deve ser em média contra todas as outras, ou seja, os desempenhos de um agente contra cada um dos outros precisam estar correlacionados.

Embora a coevolução não seja utilizada neste trabalho, a avaliação de melhora adotada fundamenta-se na competição. Dessa forma, a adequação do jogo *Doom* é analisada dentro dos atributos indicados. São eles:

- **Profundidade de desafio**: o agente recebe somente os *pixels* da tela como entrada e deverá aprender comportamentos adequados por si só. Além disso, o modo de jogo será multi-jogador competitivo, aumentando a dificuldade do desafio, pois o desempenho de um personagem dependerá também da destreza de seu adversário. Assim, o desafio não será trivial, pois para alcançar um resultado adequado o agente deve passar por um aprendizado consistente;
- **Suavidade do desafio**: os dois adversários iniciarão o jogo comportando-se de maneira

aleatória. A melhora no comportamento de cada um se dará aos poucos de acordo com o aprendizado. Portanto, ao longo de cada partida os agentes terão alcançado um resultado um pouco superior ao que tinham antes, caracterizando a suavidade do desafio;

- **Transitividade da função de *fitness***: como não há evolução, não existe uma função de *fitness*, porém ainda há um resultado quantitativo associado ao desempenho de cada agente: a recompensa. O resultado do comportamento de cada agente é mensurado através de uma pontuação de recompensa ao fim de cada episódio. Assim, uma melhora no comportamento dos personagens indica que eles foram capazes de obter uma pontuação superior à anterior. Logo, pode-se afirmar que há transitividade na pontuação de recompensa, pois uma pontuação maior indica um desempenho melhor.

Observa-se que o jogo satisfaz às três características indicadas por Cardona *et al.* (2013). Desse modo, o jogo *Doom* é considerado adequado à utilização da estratégia de melhora competitiva desejada.

4.4 Controlador

Para que um agente seja capaz de se comportar adequadamente no ambiente do jogo, ele precisa ter um controlador capaz de guiar seu comportamento e aprendizado. Uma característica desejada é que a entrada recebida pelo personagem seja a visão da tela do jogo, pois usar somente os *pixels* da tela garante uma equidade entre todos os jogadores, sejam eles humanos ou controlados por Inteligência Artificial.

Entretanto, o desenvolvimento de um jogador virtual autônomo que receba como entrada somente os *pixels* da tela é uma tarefa bastante difícil. Uma técnica recente que pode lidar com este tipo de problema é a *Deep Learning* (LECUN *et al.*, 2015). Desde seu desenvolvimento (HINTON *et al.*, 2006) ela foi estendida para resolver diferentes tipos de problemas. Recentemente, como visto no Capítulo 3, ela também vem sendo bastante utilizada para modelar agentes autônomos que recebem como entrada somente os *pixels* da tela. Assim, a criação de um jogador autônomo que aprende comportamentos complexos por si só começa a se tornar factível.

O objetivo neste trabalho é criar um personagem que aprenda comportamentos adequados recebendo uma recompensa para cada ação realizada em um dado momento. O Aprendizado por Reforço tem se mostrado como um paradigma apropriado para resolver esse tipo de tarefa (SUTTON; BARTO, 1998). O processo de aprendizado se move em direção a uma solução que maximiza a recompensa numérica escolhendo a melhor ação dentro daquele

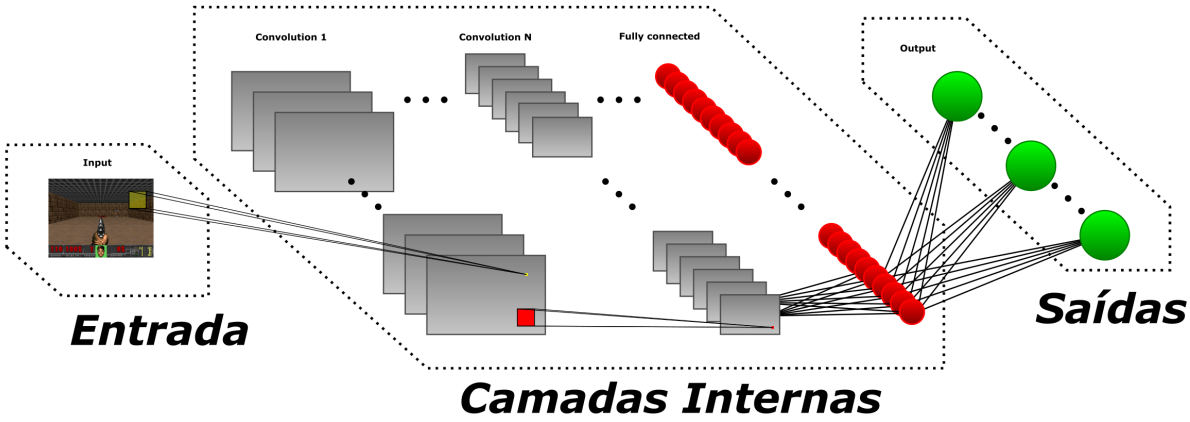
estado. Em outras palavras, o agente precisa aprender a melhor ação a ser realizada em cada estado. Da combinação de *Deep Learning* e Aprendizado por Reforço surge uma técnica robusta capaz de lidar com esse tipo de tarefa. Para tanto, essa combinação faz uso do modelo de DQN (LAMPLE; CHAPLOT, 2017; MNIH *et al.*, 2013), usado para treinar agentes em tarefas de Aprendizado por Reforço.

4.4.1 Arquitetura da Rede Neural do Controlador

A Rede Neural do Controlador possui uma arquitetura que pode ser generalizada com as seguintes características (Figura 24):

- **Entrada**: a entrada é uma imagem, em cores ou em escala de cinza, representada por uma matriz de valores reais dentro do intervalo [0,1]. A imagem fornecida pela plataforma *ViZDoom* pode ser pré-processada antes de ser passada como entrada para a rede neural;
- **Camadas Internas**: a rede é constituída por diversas camadas internas convolucionais, que reduzirão a dimensão de suas entradas e detectarão características espaciais. Após sucessivas camadas convolucionais, haverá uma camada de neurônios completamente conectados à camada anterior para unir todas as características descobertas anteriormente;
- **Saída**: a camada de saída será constituída pelo número de neurônios igual à quantidade de ações possíveis de cada cenário jogado.

Figura 24 – Resumo da arquitetura da Rede Neural usada no controlador do agente autônomo.



Fonte: Elaborado pelo autor.

4.5 Características dos experimentos

4.5.1 Ambiente de execução

Um cenário personalizado foi criado para observar a habilidade de um agente em um ambiente de competição entre múltiplos jogadores . É bastante comum em jogos FPS um modo de jogo entre diversos oponentes, em que dois ou mais competem para ser aquele que mais pontua, o que significa em geral matar mais e morrer menos do que os outros competidores.

Neste ambiente, existem dois competidores (Figura 25) posicionados em lados opostos de uma sala retangular (Figura 26). Cada agente pode realizar uma de três ações possíveis: mover-se lateralmente para a esquerda, mover-se lateralmente para a direita ou atirar.

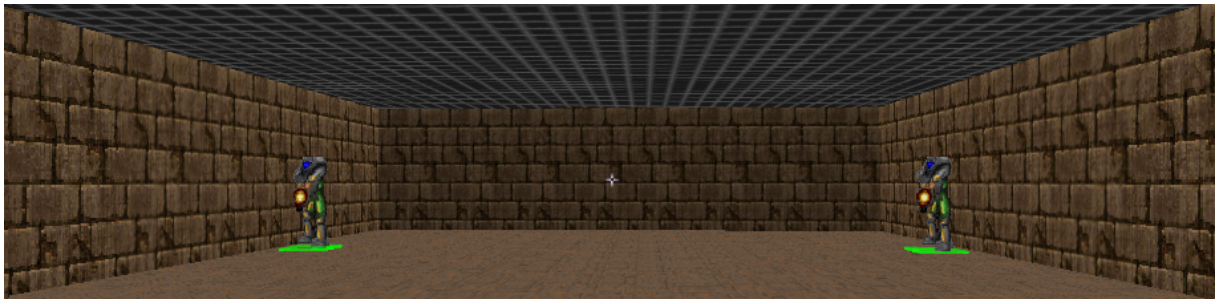
Figura 25 – Os dois agentes vistos através da visão do oponente.

(a) Jogador verde.

(b) Jogador vermelho.

Fonte: Elaborado pelo autor.

Figura 26 – Visão lateral do cenário.

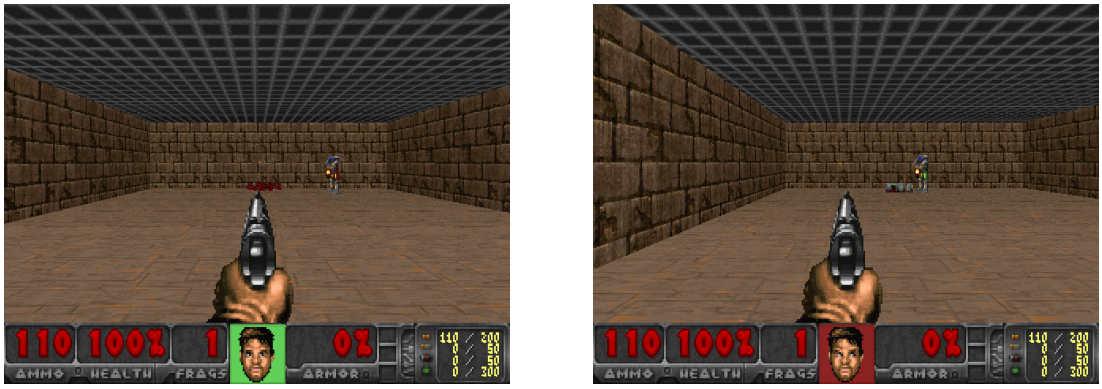


Fonte: Elaborado pelo autor.

Como este cenário é executado através de uma conexão cliente-servidor, alguns ajustes são necessários para garantir a equidade entre os jogadores. O ambiente é executado em modo síncrono, onde ambos os agentes precisam realizar uma ação antes do próximo passo. Para prevenir que um agente mova-se para o local da morte de seu oponente esperando que ele renasça no mesmo ponto para matá-lo logo que ele surja, os agentes renascem em locais aleatórios de seus lados da sala.

Um episódio começa com os agentes em lados opostos do cenário (Figura 27). Um único tiro é suficiente para matar um agente. O objetivo dos jogadores é maximizar o número de vezes que eles matam o oponente. Um objetivo secundário implícito é permanecer vivo o máximo de tempo possível, uma vez que morrer significa aumentar a pontuação do oponente.

Figura 27 – Visões dos agentes no cenário.



(a) Visão do agente verde. (b) Visão do agente vermelho.

Fonte: Elaborado pelo autor.

4.5.1.1 Métricas de avaliação

Os valores das métricas de avaliação foram escolhidos de modo a manter a consistência com os cenários já presentes no *ViZDoom*. Cada agente ganha 60 pontos toda vez que mata o oponente. Um agente morrerá se for atingido uma vez, e renascerá logo depois em um local aleatório de seu lado do cenário. Um episódio termina após corridos 300 passos de execução, momento em que cada agente recebe sua pontuação correspondente.

Nos trabalhos descritos na Seção 4.2, percebeu-se que, se os agentes executarem uma ação por quadro, a diferença entre os estados anteriores e o estado atual será tão sutil que o aprendizado se tornará difícil. Para resolver esse problema, foi utilizado um valor de repetição de oito quadros (BELLEMARE *et al.*, 2013). Isso significa que uma ação é escolhida, repetida por oito quadros e, então, uma nova ação será realizada.

4.5.2 Hiperparâmetros

Os hiperparâmetros apresentados a seguir foram escolhidos empiricamente, baseados nos resultados obtidos dos trabalhos mostrados na Seção 4.2. O fator de desconto do *Q-Learning* escolhido, γ, foi 0,99 e a taxa de aprendizado foi igual a 0,0001.

4.5.2.1 Inicialização de Xavier

Todos os *bias* da rede neural são inicializados com o valor de 0,1 e todos os pesos são inicializados com valores aleatórios utilizando a Inicialização de Xavier (GLOROT; BENGIO, 2010).

Glorot e Bengio (2010) fizeram um estudo aprofundado e realizaram diversos experimentos para entender por que o método do gradiente descendente obtinha um desempenho ruim em DNN cuja inicialização de pesos se dava de maneira aleatória. Após suas análises, os autores sugerem que uma inicialização de pesos que mantenha as variâncias de ativação e de gradiente ao longo de toda a rede seja dada por:

$$W \sim U\left[-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right], \tag{4.1}$$

para uma distribuição uniforme U, em que W significa a distribuição de inicialização do neurônio n em questão, n_{in} é o número de neurônios conectados à entrada de n e n_{out} é a quantidade de neurônios conectados à saída de n. Já para uma distribuição normal, deve ser usada uma variância de:

$$Var[W] = \frac{2}{n_{in}+n_{out}}. \tag{4.2}$$

Esse tipo de inicialização de pesos de redes neurais obteve bons resultados nos experimentos realizados pelos autores e hoje é conhecido como Inicialização de Xavier.

4.5.2.2 Algoritmo RMSProp

O algoritmo *RMSProp* (TIELEMAN; HINTON, 2012) foi usado para treinar a rede neural usando mini-lotes de tamanho 64. Ele consiste em um método de adaptação dinâmica do gradiente descendente utilizando mini-lotes, que são pequenos conjuntos de dados usados na atualização dos pesos (LI *et al.*, 2014).

O método do gradiente descendente realiza mudanças nos pesos de acordo com o valor da primeira derivada parcial. Assim, quanto maior o valor da derivada, maior será a mudança. Entretanto, essa característica pode ser problemática, pois uma derivada pequena pode indicar um área larga, rasa e pouco curvada da superfície em que um grande passo é necessário, enquanto que uma derivada grande pode indicar uma parte íngreme e bastante curvada em que o tamanho do passo deve ser reduzido para prevenir instabilidade (SUTTON, 1986).

Para minimizar os efeitos da dependência da primeira derivada parcial dos pesos e aumentar a velocidade e a estabilidade do aprendizado, Tieleman e Hinton (2012) propuseram um método de mudança dinâmica no valor da taxa de aprendizado da rede. Assim, a atualização dos pesos da rede deve ser guiada pelo cálculo de uma média móvel do gradiente, dada por:

$$EQ(w(t)) = 0,9 \cdot EQ(w(t-1)) + 0,1 \left(\frac{\partial EQ}{\partial w(t)} \right)^2, \tag{4.3}$$

onde EQ é o erro quadrático médio e w é o vetor de pesos no passo t. A atualização dos pesos seria então realizada através da equação:

$$w(t+1) = w(t) - \frac{\eta}{\sqrt{EQ(w,t)}} \left(\frac{\partial EQ}{\partial w(t)} \right), \tag{4.4}$$

em que η é a taxa de aprendizado padrão.

4.5.2.3 Experience Replay

Também foi utilizado *Experience Replay* (LIN, 1993) para reduzir a correlação entre quadros consecutivos. Uma memória de *replay* mantém os últimos cem mil quadros e uma amostra é escolhida aleatoriamente e passada para a rede neural a cada atualização. Desse modo, é possível obter vantagens como maior eficiência no uso de dados, aumento da velocidade de aprendizado e melhora na escolha de parâmetros que evitem mínimos locais (MNIH *et al.*, 2013).

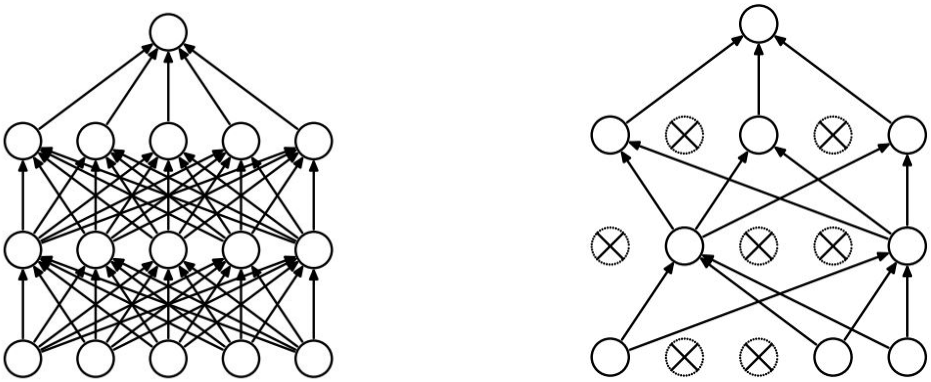
4.5.2.4 Dropout

Para diminuir o *overfitting*, ou seja, um ajuste excessivo da rede neural aos dados de entrada, foi utilizada a técnica de *Dropout* durante o treinamento, com uma probabilidade de 0,7. A ideia principal do *Dropout* (SRIVASTAVA *et al.*, 2014) consiste na remoção aleatória de neurônios e de suas conexões durante a fase de treinamento (Figura 28). Como mostrado por esses autores, essas adaptações são suficientes para prevenir que os neurônios se adaptem demasiadamente ao conjunto de dados.

4.5.2.5 Política ε-greedy

Uma política ε-*greedy* (SUTTON; BARTO, 1998) com decaimento linear de 1,0 até 0,1 foi usada para fazer o equilíbrio entre prospecção e exploração. De acordo com essa política, o agente tem

Figura 28 – Ilustração de *Dropout* em uma rede neural.



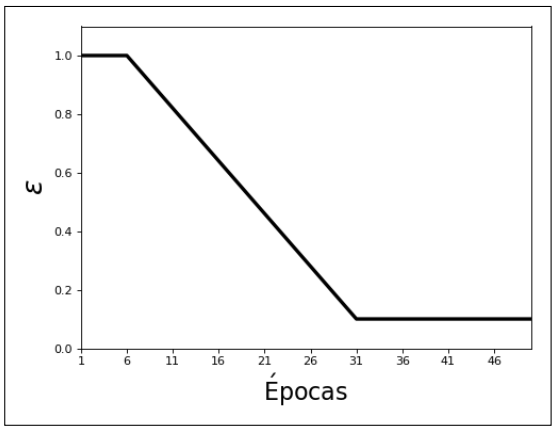
(a) Rede neural normal.

(b) Após o *Dropout*.

Fonte: Adaptado de Srivastava *et al.* (2014).

uma probabilidade ε, de acordo com a época em que se encontra (Figura 29), de realizar uma ação aleatória ao invés de escolher a considerada melhor no momento.

Figura 29 – Valor de ε a cada época.



Fonte: Elaborado pelo autor.

4.5.3 Topologia da rede neural

4.5.3.1 Camadas iniciais

A entrada da rede neural é uma matriz de números reais, representando uma imagem em escala de cinza de 64 por 48 *pixels*, que é então passada para uma primeira camada convolucional. Esta camada possui um filtro de largura e altura de tamanho quatro, e um passo de largura e altura de tamanho dois. Esta configuração divide o tamanho da entrada ao meio. Nesta camada, 32 características são computadas.

As saídas da primeira camada convolucional são passadas para uma segunda camada convolucional que irá computar 64 características. Esta camada também tem um filtro de largura e altura de tamanho quatro, e um passo de largura e altura de tamanho dois, dividindo pela metade o tamanho das entradas recebidas.

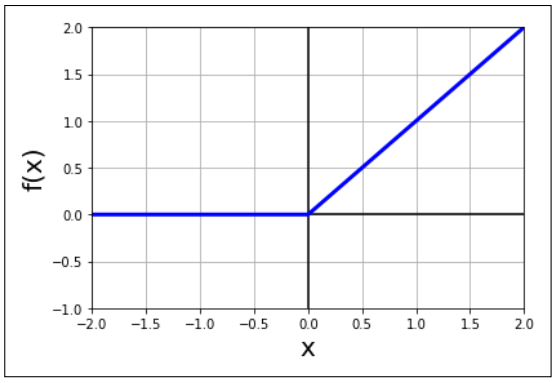
4.5.3.2 Função ReLU

Ambas as camadas convolucionais utilizam a função de ativação ReLU (Unidade de Retificação Linear, do inglês, *Rectifier Linear Unit*). As unidades ReLU (HAHNLOSER *et al.*, 2000) são neurônios artificiais cuja função de ativação é dada pela equação:

$$f(x) = max(0,x),\ x \in \mathbb{R}. \tag{4.5}$$

Glorot *et al.* (2011) mostraram que esses tipos de neurônios têm um desempenho igual ou melhor do que as funções comumente utilizadas anteriormente, como funções sigmoides ou tangentes hiperbólicas. Mais ainda, redes neurais utilizando ReLU's (Figura 30) alcançam seus melhores resultados mesmo sem nenhum tipo de pré-treino.

Figura 30 – Função ReLU.



Fonte: Elaborado pelo autor.

4.5.3.3 Camadas finais

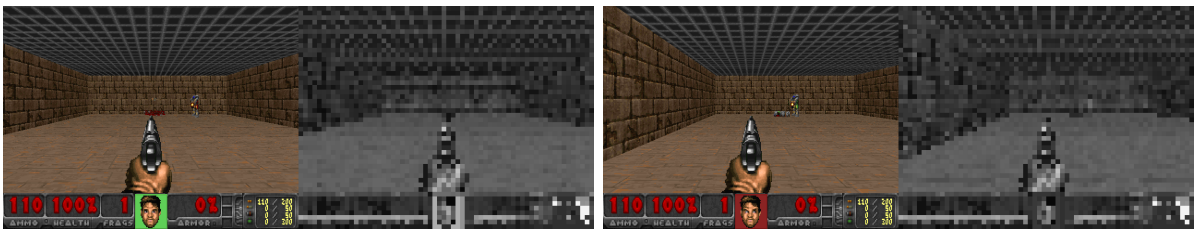
Após a passagem pela segunda camada convolucional, as 64 imagens de tamanho 16 por 12 *pixels* são "achatadas" em vetores de tamanho 192, e são completamente conectadas a uma camada de 512 neurônios. Esta camada tem o objetivo de juntar todas as características descobertas

previamente nas camadas convolucionais. Os resultados são enviados à camada de saída, que possui três neurônios, um para cada ação possível a ser realizada por um agente.

4.5.3.4 Entradas e saídas

A imagem bruta colorida de 640 por 480 *pixels* da tela é reduzida para uma imagem de 64 por 48 *pixels* em escala de cinza, que é a entrada da rede neural (Figura 31). Dessa forma, a entrada é uma camada de 64 por 48 neurônios cujos valores variam de 0,0 a 1,0.

Figura 31 – Visões original (esquerda) e pré-processada (direita) dos agentes.



(a) Visão do agente verde. (b) Visão do agente vermelho.

Fonte: Elaborado pelo autor.

Cada ação possui seu próprio *q-value* associado na camada de saída, assim, cada ação implica em um neurônio. A camada de saída possui três neurônios, pois o cenário utilizado permite três ações possíves: mover-se lateralmente para a esquerda, mover-se lateralmente para a direita ou atirar.

4.5.3.5 Resumo

A topologia da rede é resumida na Tabela 1.

Tabela 1 – Topologia da rede neural.

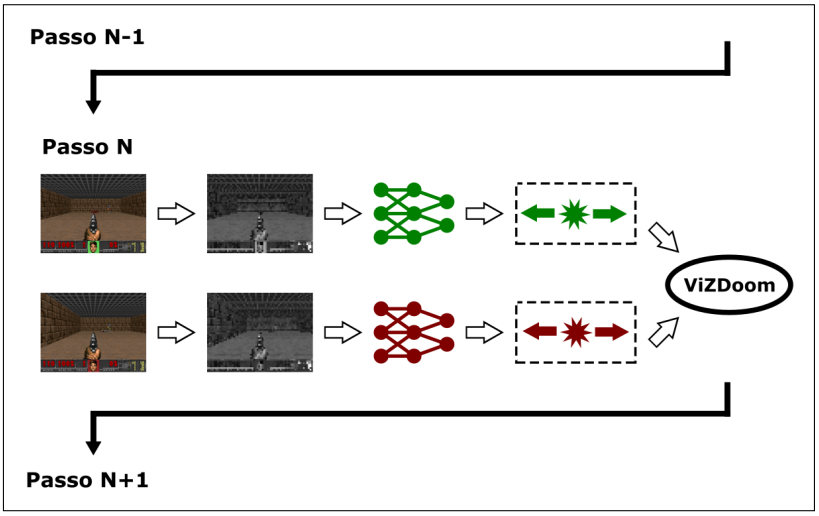
Camada	Dimensão do filtro	Passo	Número de filtros	Entrada	Saída
1ª convolucional	4×4	2×2	32	64×48	32×24
2ª convolucional	4×4	2×2	64	32×24	16×12
Completamente conectada				12288	512
Saída				512	3

Fonte: Elaborado pelo autor.

4.5.4 Visão geral de um passo de execução

De uma maneira geral, a cada passo de execução do modelo, a visão das telas são pré-processadas e passadas para as redes neurais individuais de cada agente. Por sua vez, a rede neural indica qual ação deve ser realizada. As duas ações, uma de cada agente, são enviadas ao ambiente *ViZDoom*, que executará as ações e seguirá para o passo seguinte. Essa descrição está ilustrada na Figura 32.

Figura 32 – Esquematização de um passo da execução no ambiente.



Fonte: Elaborado pelo autor.

4.6 Considerações Finais

Neste capítulo, foram apresentados trabalhos anteriores que fizeram parte do desenvolvimento desta dissertação e foram discutidos os objetivos que se desejam atingir, explicando todos os três principais componentes do modelo avaliado: o jogo, o controlador de um agente e a execução dos experimentos. Foram detalhadas as configurações específicas dos modelos utilizados e as técnicas auxiliares empregadas.

A validação da técnica proposta se dará através da avaliação do comportamento e do desempenho dos agentes treinados através da competição contra outros agentes autônomos, em comparação com outros agentes treinados sem competição contra outros agentes autônomos.

No próximo capítulo são apresentados os resultados dos experimentos realizados com os grupos de agentes, com o intuito de avaliar o modelo proposto, e de discutir suas implicações.

5 RESULTADOS

5.1 Introdução

Neste capítulo são apresentados os resultados obtidos com a implementação e execução de competição entre dois agentes autônomos em um jogo FPS. Os resultados englobam análises de cada um dos times de agentes e são divididos em três partes: os aspectos observados durante a fase de treinamento, a execução dos testes contra oponentes completamente aleatórios e os testes realizados nas disputas dos times entre si.

As análises são realizadas com base nos gráficos de pontuação média dos agentes e dos times, e nos resultados dos confrontos ocorridos. São avaliadas as questões de sucesso no aprendizado, comportamentos exibidos e desempenho em competição, em comparação com os resultados obtidos pelos outros times.

A fase de treinamento de cada time é avaliada na Seção 5.2. Na Seção 5.3 é analisado o desempenho dos agentes contra oponentes completamente aleatórios. A competição entre os times de agentes autônomos é realizada na Seção 5.4. Na Seção 5.5 são discutidos os resultados obtidos e algumas questões observadas. Por fim, na Seção 5.6 são feitas algumas considerações finais.

5.1.1 Aspectos técnicos

O trabalho foi implementado utilizando a linguagem de programação *Python*, versão 2.7. A biblioteca *TensorFlow* ™ foi utilizada para criação e execução das DNN's. O ambiente *ViZDoom* foi utilizado através da biblioteca *vizdoom* para Python.

O computador utilizado para a realização dos testes possui um processador Intel ® Core i7 7700 3.60 GHz, dois pentes de memória RAM DDR4 8GB 2133 MHz e uma placa de vídeo NVIDIA ® GeForce GTX 1080 8GB DDR5. Todos os treinamentos e testes foram executados na Unidade de Processamento Gráfico (GPU).

5.2 Treinamento

Quinze agentes foram treinados e divididos em três grupos de cinco agentes cada. Os grupos foram denominados de Time Azul, Time Vermelho e Time Laranja (Figura 33). Os agentes do Time Azul e do Time Vermelho foram treinados através da competição entre os dois grupos, em

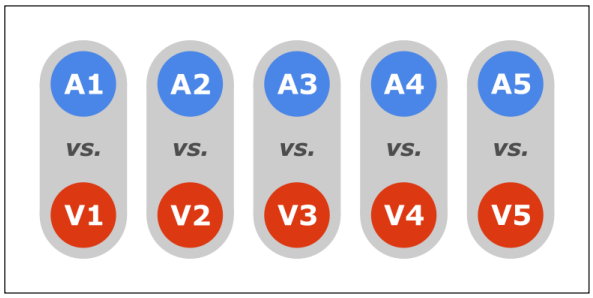
que cada agente do Time Azul enfrentou um único agente do Time Vermelho (Figura 34). Dessa forma, os agentes dos times Azul e Vermelho aprenderam seus comportamentos baseados na competição contra outros agentes autônomos.

Figura 33 – Divisão dos agentes em times.

(a) Time Azul. (b) Time Vermelho. (c) Time Laranja.

Fonte: Elaborado pelo autor.

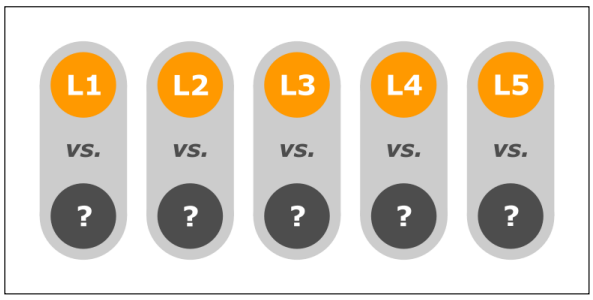
Figura 34 – Esquema de treinamento dos times Azul e Vermelho.



Fonte: Elaborado pelo autor.

O terceiro grupo foi treinado contra oponentes que agiam de maneira completamente aleatória, ou seja, escolhiam aleatoriamente, com igual probabilidade, qual das três possíveis ações realizariam em seguida (Figura 35).

Figura 35 – Esquema de treinamento do time Laranja.



Fonte: Elaborado pelo autor.

Cada agente foi treinado por 50 épocas de 200 episódios cada. Um episódio termina após corridos 300 passos de execução. Ao fim de cada episódio, os agentes são testados por 50 episódios, ou seja, jogam novamente, mas utilizando somente os comportamentos aprendidos. Antes do treinamento, os agentes foram inicializados com pesos aleatórios de acordo com as características descritas na Seção 4.5, o que significa que eles não possuíam nenhum conhecimento prévio a respeitos dos comportamentos adequados.

5.2.1 Time Azul vs. Time Vermelho

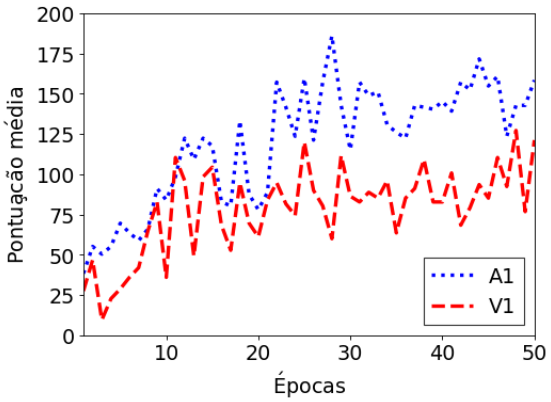
Neste treino, os dois jogadores eram agentes autônomos com capacidade de aprendizado, isto é, possuíam as características apresentadas na Seção 4.5. Como é possível observar nos gráficos de treinamento na Figura 36, os agentes apresentaram uma curva de pontuação média ascendente, indicando que foram capazes de aprender comportamentos adequados para alcançar o objetivo do cenário proposto: matar o oponente o maior número de vezes. Assim, pode-se afirmar que o treinamento por competição entre agentes autônomos é adequado para o aprendizado dentro de um ambiente FPS como o apresentado.

É interessante notar que os agentes do Time Azul obtiveram desempenho superior ao do Time Vermelho durante quase todo o treino, ou seja, houve uma certa dominância do Time Azul em relação ao Time Vermelho. Entretanto, houve um único caso em que um agente do Time Vermelho foi superior ao fim do treinamento: o agente V2 teve um desempenho melhor em relação ao agente A2. Isto aconteceu ou por causa de uma piora do A2 ou por uma melhora significativa e rápida de V2. Esse caso é analisado com mais detalhes na Seção 5.5.

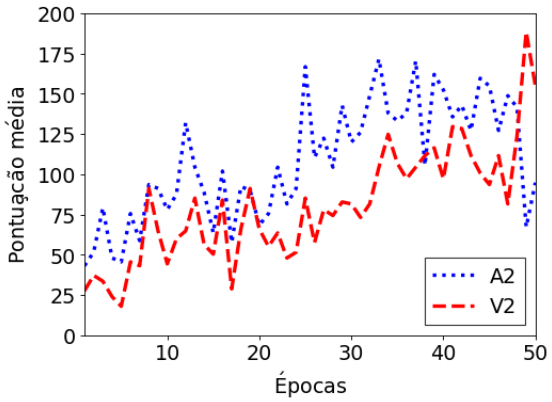
5.2.2 Time Laranja vs. Oponentes Completamente Aleatórios

Observa-se nos gráficos de treinamento da Figura 37 que a curva ascendente indica que ocorreu aprendizado por parte dos agentes, ou seja, eles foram capazes de desenvolver comportamentos adequados para jogar o ambiente. Além disso, de maneira geral, todos os agentes terminaram o treinamento com uma pontuação média por volta de 150, denotando que eles foram capazes de matar os oponentes múltiplas vezes em um mesmo episódio. Com isso, conclui-se que o treinamento por competição contra oponentes que agem de maneira completamente aleatório também é adequado para o aprendizado de um agente autônomo dentro do cenário apresentado.

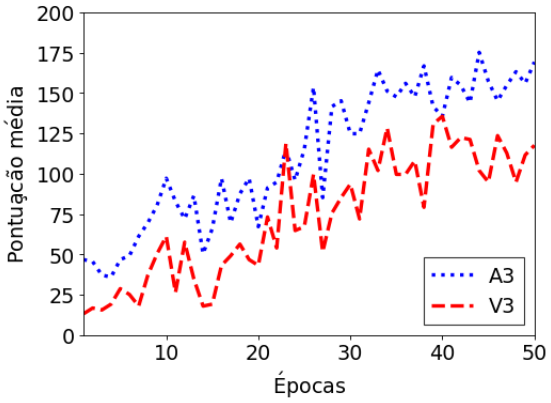
Figura 36 – Média da pontuação dos jogadores dos times Azul e Vermelho durante o treinamento. Em cada um dos gráficos a linha pontilhada azul indica a pontuação média de um agente do Time Azul e a linha tracejada vermelha indica a pontuação média de um agente do Time Vermelho.



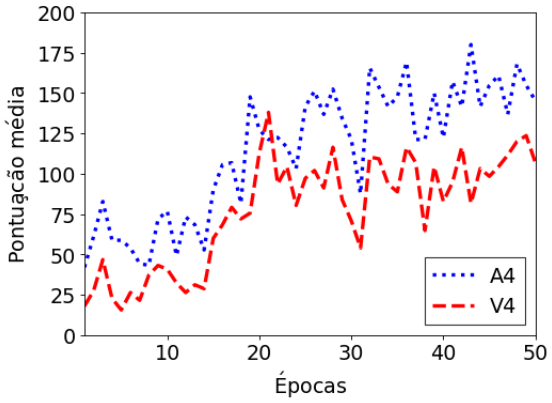
(a) Agente A1 vs. Agente V1.



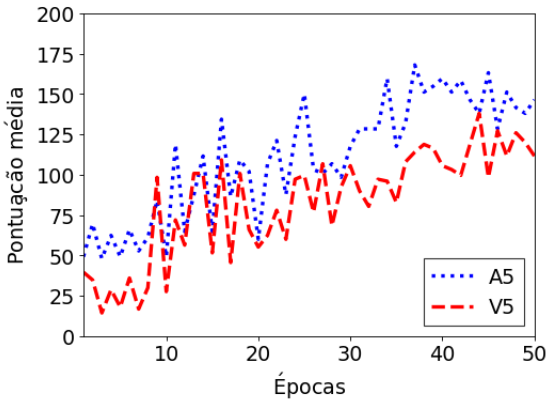
(b) Agente A2 vs. Agente V2.



(c) Agente A3 vs. Agente V3.



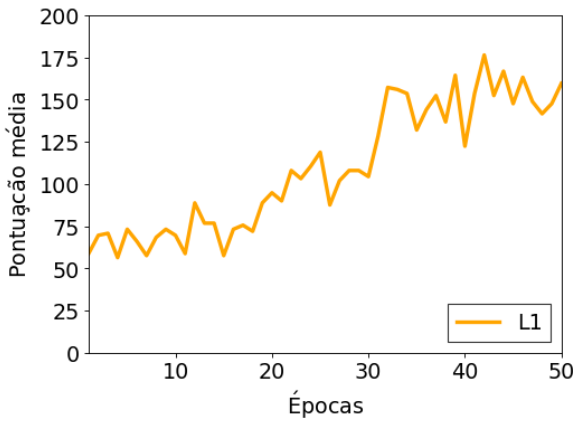
(d) Agente A4 vs. Agente V4.



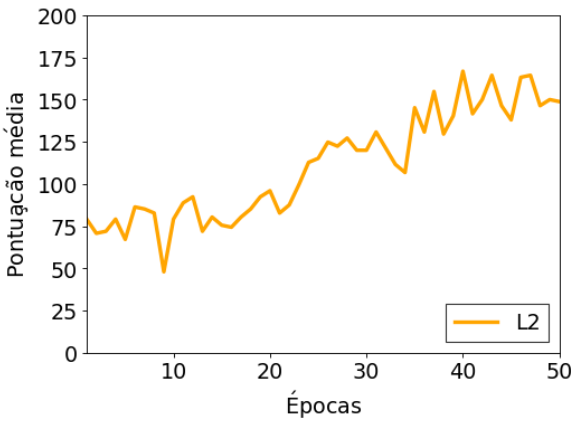
(e) Agente A5 vs. Agente V5.

Fonte: Elaborado pelo autor.

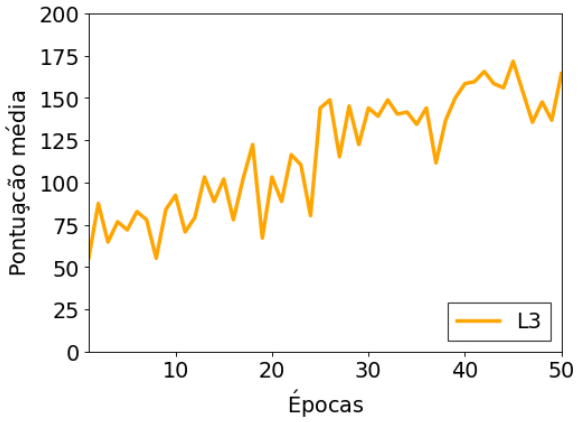
Figura 37 – Média da pontuação dos jogadores do Time Laranja durante o treinamento. Em cada um dos gráficos a linha contínua laranja indica a pontuação média de um agente do Time Laranja.



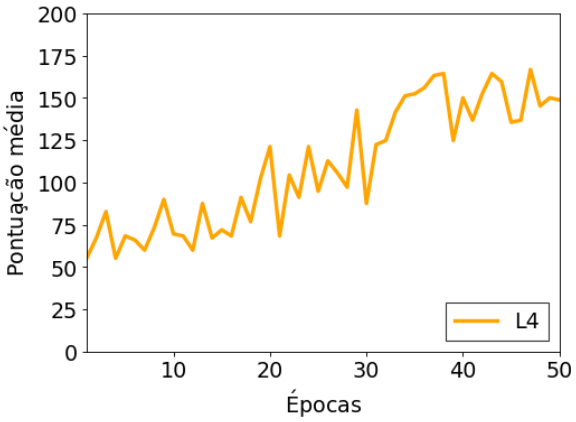
(a) Agente L1.



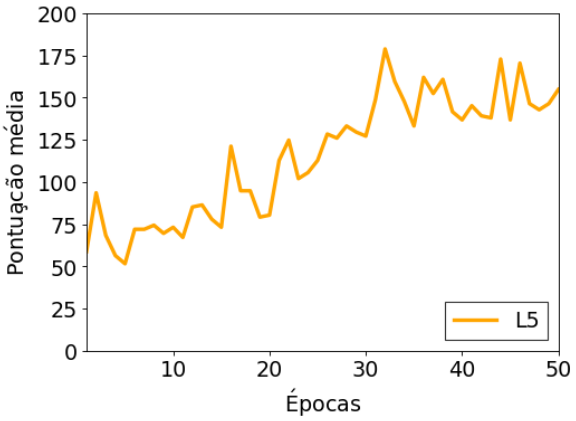
(b) Agente L2.



(c) Agente L3.



(d) Agente L4.



(e) Agente L5.

Fonte: Elaborado pelo autor.

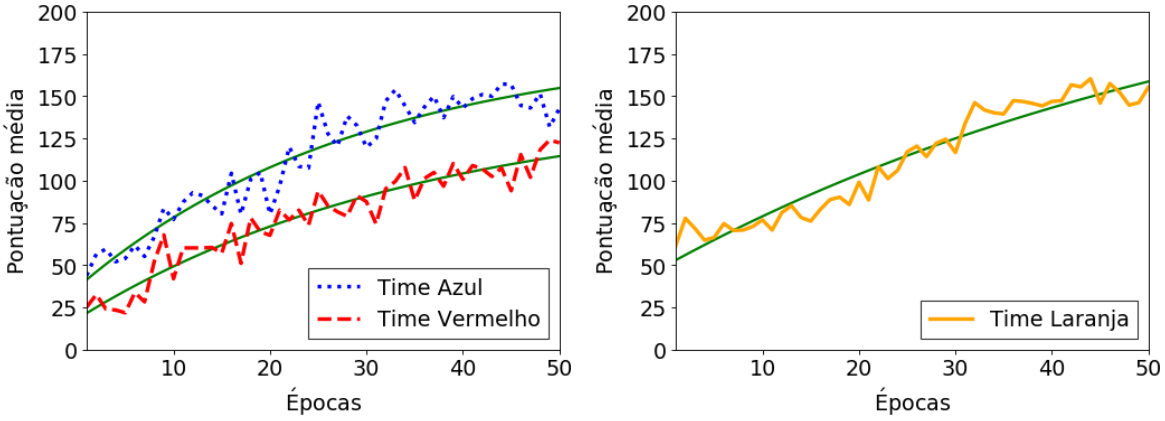
5.2.3 Comparação entre os times

Observe que os dados de treinamento para os indivíduos do Time Laranja e do Time Azul, que foram os melhores no treinamento contra o Time Vermelho, são bastante semelhantes. Assim, pode-se inicialmente inferir que o aprendizado contra agentes não-inteligentes é melhor do que treinar contra agentes autônomos. Essa discussão será mais elaborada na Seção 5.5.

5.2.4 Considerações sobre o aprendizado

Como é possível observar pelas curvas ascendentes nos gráficos de treinamento de cada time (Figura 38), todos os agentes foram capazes de aprender comportamentos adequados para a competição no ambiente de testes deste trabalho. De maneira mais específica, os agentes foram capazes de aprender comportamentos como mover-se em direção ao oponente (Figura 39) e mover-se em direção ao oponente e atirar para matá-lo (Figura 40).

Figura 38 – Comparação da média da pontuação do treinamento de cada time. A linha pontilhada azul indica a pontuação média do Time Azul, a linha tracejada vermelha indica a pontuação média do Time Vermelho e a linha contínua laranja indica a pontuação média do Time Laranja. As linhas contínuas verdes indicam as linhas de tendência da pontuação de cada time.



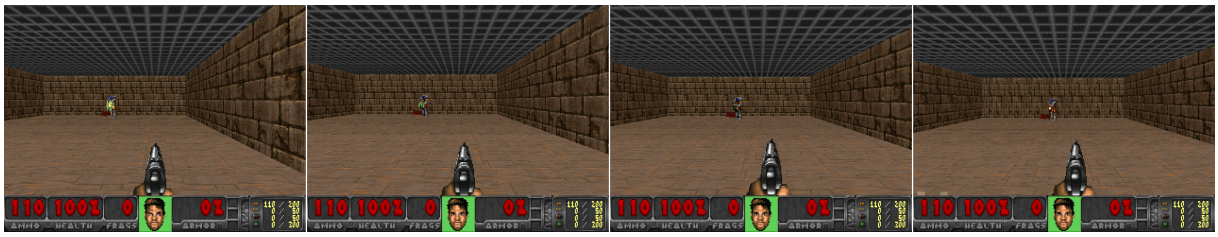
(a) Times Azul e Vermelho.

(b) Time Laranja.

Fonte: Elaborado pelo autor.

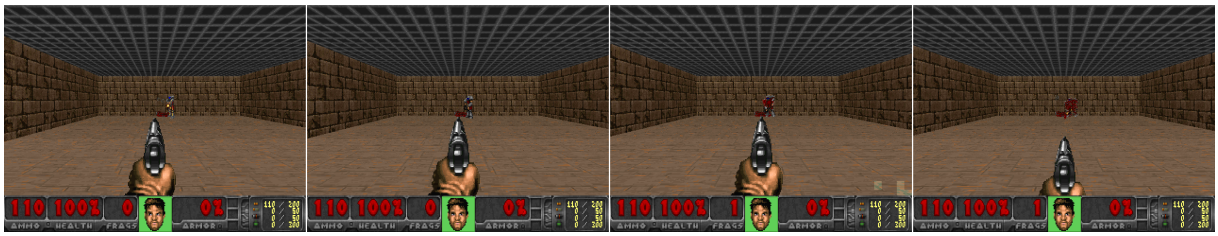
É importante salientar que um mesmo comportamento de mover-se em direção ao oponente e atirar pode levar a um acerto (Figura 40) ou a uma morte (Figura 41). A única diferença neste caso é a velocidade de reação, ou seja, quem atirou primeiro no momento correto. Essa característica deixa claro que a dificuldade de aprender um comportamento para aumentar a pontuação é maior do que pode parecer a princípio.

Figura 39 – Da esquerda para a direita, **1º quadro**: o agente inicia na parte direita do cenário. **2º ao 4º quadro**: ele se move para a esquerda em direção ao oponente.



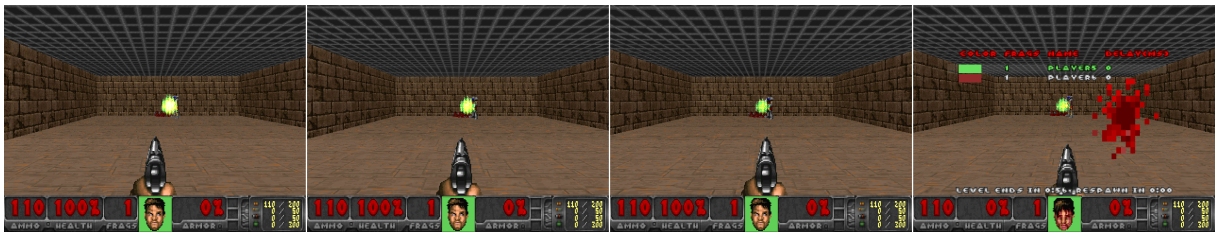
Fonte: Elaborado pelo autor.

Figura 40 – Da esquerda para a direita, **1º quadro**: o agente inicia na parte esquerda do cenário. **2º quadro**: ele se move para a direita em direção ao oponente. **3º quadro**: ele encontra o oponente e atira. **4º quadro**: o oponente morre.



Fonte: Elaborado pelo autor.

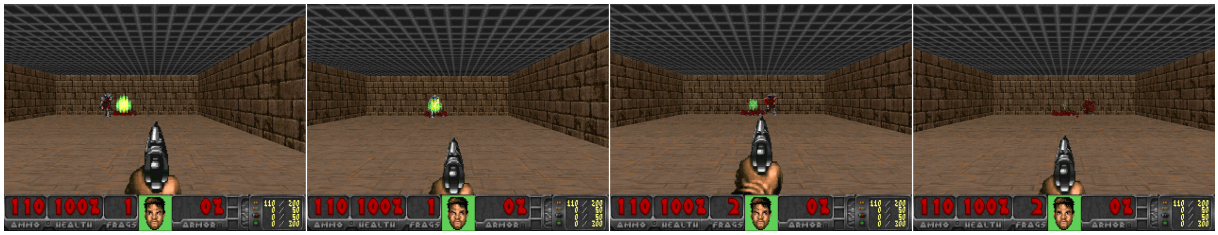
Figura 41 – Da esquerda para a direita, **1º quadro**: o agente inicia na parte esquerda do cenário. **2º quadro**: ele se move para a direita em direção ao oponente. **3º quadro**: ele encontra o oponente e não atira. **4º quadro**: ele é morto antes de atirar no oponente.



Fonte: Elaborado pelo autor.

No caso da disputa entre os times Azul e Vermelho, ambos os agentes aprenderam a mover-se em direção ao oponente e atirar quando estiverem em frente a ele. Quando os dois agentes estão se movendo em direção ao adversário simultaneamente, eles se movem em direções opostas. Assim, no momento em que um agente atirar ao se cruzarem, errará o tiro, pois no momento seguinte em que o tiro deveria atingir o adversário, o oponente já terá passado por ele. Como observado na Figura 42, os agentes também foram capazes de aprender a antecipar o movimento do adversário, de modo a conseguir acertá-lo.

Figura 42 – Da esquerda para a direita, **1º quadro**: o agente inicia na parte direita do cenário. **2º quadro**: ele se move para a esquerda em direção ao oponente. **3º quadro**: ele atira no limite de passar direto pelo oponente. **4º quadro**: o oponente morre.



Fonte: Elaborado pelo autor.

5.3 Competição contra oponentes completamente aleatórios

Uma vez terminados todos os treinamentos, os agentes foram testados através de competições contra adversários que agem de maneira completamente aleatória. Durante os testes não houve aprendizado, ou seja, os pesos das redes neurais não foram modificados, somente utilizados.

De um certo modo, pode-se considerar esse tipo de tarefa como essencialmente de um único jogador, uma vez que só está presente um agente autônomo e os oponentes escolhem aleatoriamente uma das três ações possíveis com igual probabilidade. Eles não têm comportamentos predefinidos e agem de modo pouco adequado para o cenário.

Cada um dos agentes de todos os times jogou 200 partidas contra oponentes que agem de maneira completamente aleatória. Ao final, a média da pontuação de cada time foi calculada. O Time Azul obteve uma pontuação média de **113,64**, a média de pontuação do Time Vermelho foi **78,60** e a do Time Laranja foi **156,84**.

Observa-se que o Time Vermelho teve o pior desempenho, o que está de acordo com os resultados mostrados nas curvas de treinamento da Figura 38. Entretanto, os agentes do Time Laranja tiveram resultados bem melhores do que os do Time Azul, diferentemente dos desempenhos bem próximos mostrados ao longo do treinamento.

Inicialmente, esse resultado pode ser considerado inesperado, uma vez que acredita-se em uma melhora de desempenho com a competição entre agentes autônomos. Entretanto, há a possibilidade da indicação que o grupo laranja teve um melhor desempenho somente por estar acostumado ao comportamento de agentes completamente aleatórios. Uma discussão maior será feita na Seção 5.5.

5.4 Competição contra outros agentes autônomos

Agora será considerado um ambiente de competição entre agentes autônomos de fato. Dois agentes autônomos de times diferentes são postos no cenário para uma disputa entre si. Novamente não houve mais aprendizado nesse momento, somente a utilização da rede neural treinada na Seção 5.2.

Cada um dos agentes jogou 200 partidas contra todos os 10 agentes dos outros dois times. Em cada uma das partidas foi estabelecido um resultado de vitória, se o agente matou mais do que o adversário, derrota, se o agente matou menos do que o oponente, ou empate, se ambos tiveram o mesmo número de mortes. Em seguida, foram calculadas as porcentagens de vitória em cada um dos confrontos individuais. Os resultados obtidos são avaliados a seguir.

5.4.1 Time Azul vs. Time Vermelho

No treinamento por competição entre os dois times, o Time Azul obteve melhores resultados, exceto o agente A2. Na competição contra oponentes completamente aleatórios, o Time Azul também teve um desempenho melhor do que o Time Vermelho. Assim, espera-se que o Time Azul tenha um desempenho melhor nos testes contra os agentes do Time Vermelho.

A Tabela 2 apresenta as porcentagens de vitória de cada um dos agentes em seus confrontos. Os resultados estão de acordo com o esperado, uma vez que, de modo geral, o desempenho do Time Azul foi melhor do que o desempenho do Time Vermelho. A única exceção foi o agente A2, que teve um desempenho pior tanto no treinamento quanto nos testes. O caso do A2 será avaliado com mais detalhes na Seção 5.5.

Assim, conclui-se que os agentes autônomos que têm um melhor desempenho contra outros agentes autônomos no treinamento, de fato obtêm melhores resultados quando postos em competição entre si. Analogamente, agentes que têm um desempenho pior ao fim do treinamento, caso do A2, obtêm resultados piores.

5.4.2 Time Azul vs. Time Laranja

Quando observados os resultados do treinamento, constata-se que os times Azul e Laranja tiveram curvas de aprendizado bem próximas. Apesar disso, o Time Laranja foi melhor contra oponentes completamente aleatórios. O próximo passo é verificar se o desempenho do Time Laranja foi de fato melhor do que o do Time Azul em uma competição direta.

Tabela 2 – Porcentagem de vitória em cada confronto entre os agentes dos times Azul e Vermelho. A porcentagem da esquerda representa o agente da coluna (azul) e a porcentagem da direita representa o agente da linha (vermelho). O complemento de 100% da soma é a porcentagem de empates. Em negrito são destacados os melhores desempenhos no confronto.

	A1	A2	A3	A4	A5
V1	**53,5%** × 31,0%	38,0% × **48,0%**	**60,5%** × 27,0%	**58,0%** × 30,5%	**57,0%** × 28,5%
V2	**53,5%** × 33,0%	33,0% × **56,5%**	**58,0%** × 29,0%	**57,5%** × 30,5%	**62,5%** × 29,0%
V3	**57,5%** × 25,0%	25,5% × **57,5%**	**54,5%** × 30,0%	**55,5%** × 27,0%	**56,0%** × 29,5%
V4	**53,0%** × 27,5%	30,0% × **57,0%**	**53,0%** × 31,0%	**53,5%** × 32,5%	**54,5%** × 33,0%
V5	**58,0%** × 27,0%	33,0% × **47,5%**	**50,0%** × 31,5%	**51,5%** × 32,0%	**56,0%** × 30,5%

Fonte: Elaborado pelo autor

Pelos resultados mostrados na Tabela 3, os agentes do Time Azul tiveram um desempenho melhor do que os do Time Laranja, exceto o agente A2. Esse agente já foi exceção desde o treinamento e será avaliado com mais detalhes na Seção 5.5.

Desse modo, o Time Azul, apesar de ter tido um desempenho inferior contra agentes completamente aleatórios, teve um desempenho melhor do que o Time Laranja em um confronto direto. Assim, observa-se que o treinamento contra outros agentes autônomos faz com que o Time Azul tenha um melhor desempenho contra outros agentes autônomos.

Como o Time Azul teve um desempenho superior ao Time Vermelho durante o treinamento, conclui-se que os agentes que obtiveram os melhores resultados no treino contra outros agentes autônomos terão um melhor desempenho em uma competição contra outros jogadores inteligentes. Por outro lado, o treinamento contra oponentes que agem de maneira completamente aleatória não garante um resultado superior contra jogadores inteligentes.

5.4.3 Time Vermelho vs. Time Laranja

Tanto no treinamento quanto nos resultados contra oponentes completamente aleatórios, o Time Vermelho teve um desempenho bem inferior ao do Time Laranja. Dessa forma, espera-se que o Time Vermelho também obtenha resultados piores em um confronto direto entre os times.

Entretanto, como observado na Tabela 4, o desempenho do Time Vermelho foi melhor em todos os casos do que o Time Laranja. Dessa forma, tem-se que o treinamento contra outros agentes autônomos, o Time Azul, fez com que o Time Vermelho obtivesse melhores resultados

Tabela 3 – Porcentagem de vitória em cada confronto entre os agentes dos times Azul e Laranja. A porcentagem da esquerda representa o agente da coluna (azul) e a porcentagem da direita representa o agente da linha (laranja). O complemento de 100% da soma é a porcentagem de empates. Em negrito são destacados os melhores desempenhos no confronto.

	A1	A2	A3	A4	A5
L1	**53,0%** × 37,5%	29,5% × **55,0%**	**49,5%** × 35,5%	**53,0%** × 33,5%	**48,0%** × 35,5%
L2	**46,0%** × 38,5%	31,5% × **54,5%**	**55,5%** × 31,0%	**60,0%** × 29,0%	**45,0%** × 39,0%
L3	**53,0%** × 29,0%	28,0% × **59,0%**	**48,5%** × 35,0%	**51,5%** × 35,0%	**48,5%** × 35,5%
L4	**51,5%** × 35,5%	28,0% × **59,5%**	**57,0%** × 31,5%	**54,5%** × 33,0%	**49,0%** × 34,5%
L5	**60,0%** × 26,0%	28,0% × **55,5%**	**52,5%** × 32,5%	**64,5%** × 25,5%	**57,5%** × 29,5%

Fonte: Elaborado pelo autor

em um confronto contra outros agentes autônomos treinados contra oponentes completamente aleatórios.

Como o Time Vermelho teve um desempenho inferior ao Time Azul durante o treinamento, conclui-se que os agentes que obtiveram os piores resultados no treino contra outros agentes autônomos terão um pior desempenho em uma competição contra outros jogadores inteligentes. Entretanto, o treinamento contra oponentes que agem de maneira completamente aleatória não garante um desempenho superior contra jogadores inteligentes.

Tabela 4 – Porcentagem de vitória em cada confronto entre os agentes dos times Vermelho e Laranja. A porcentagem da esquerda representa o agente da coluna (vermelho) e a porcentagem da direita representa o agente da linha (laranja). O complemento de 100% da soma é a porcentagem de empates. Em negrito são destacados os melhores desempenhos no confronto.

	V1	V2	V3	V4	V5
L1	**52,0%** × 25,0%	**46,5%** × 34,0%	**56,0%** × 28,0%	**57,5%** × 29,5%	**43,5%** × 38,0%
L2	**58,5%** × 23,0%	**56,0%** × 29,0%	**50,0%** × 38,0%	**51,5%** × 34,0%	**43,5%** × 34,0%
L3	**55,5%** × 28,0%	**55,0%** × 26,5%	**55,0%** × 29,0%	**50,0%** × 33,5%	**43,5%** × 38,5%
L4	**47,0%** × 28,5%	**40,0%** × 37,0%	**48,0%** × 27,0%	**58,5%** × 29,5%	**50,5%** × 33,5%
L5	**56,0%** × 27,0%	**61,5%** × 25,0%	**58,0%** × 29,0%	**56,5%** × 27,0%	**49,5%** × 39,5%

Fonte: Elaborado pelo autor

5.5 Discussões Finais

5.5.1 O agente A2

Nos treinamentos realizados entre os times Azul e Vermelho, o único caso em que um agente do Time Azul terminou o treino com uma pontuação média menor foi o agente A2. Na Figura 36b, observa-se que a queda no desempenho nas últimas épocas foi brusca. Duas possibilidades serão avaliadas: ou o agente A2 teve uma piora significativa em seu comportamento aprendido, ou o agente V2 teve uma melhora em relação a seu adversário.

Nos testes realizados na Seção 5.4, ele também foi a exceção, tendo mostrado resultados bem inferiores aos outros agentes. Como o agente A2 teve um resultado pior em todos os testes e o agente V2 não mostrou nenhuma melhora significativa nos testes em relação aos outros agentes vermelhos, conclui-se que ocorreu uma degeneração do comportamento do agente A2.

As avaliações dos resultados e os testes realizados sugerem que se trata de um problema interno do ambiente *ViZDoom*. Entretanto, para comprová-lo, seria necessário fazer uma análise interna do código-fonte do ambiente, algo fora do escopo deste trabalho.

5.5.2 Melhor desempenho do Time Laranja contra os oponentes aleatórios

Quando treinados em um ambiente contra oponentes que agem de forma completamente aleatória, o melhor desempenho do Time Laranja em comparação com o Time Azul chamou atenção. Uma vez que a pontuação média de treinamento foi bastante próxima (Figura 38), esperava-se que ambos obtivessem resultados semelhantes nos testes. Outra questão importante é que o objetivo deste trabalho é justamente mostrar que o treinamento contra outros agentes autônomos gera um desempenho melhor.

Surgem duas possibilidades de explicação para essa diferença:

1. os agentes do Time Laranja de fato aprenderam melhores comportamentos;
2. eles obtiveram um melhor desempenho por já estarem acostumados com oponentes completamente aleatórios.

Para sanar essa questão definitivamente, é necessário analisar os testes realizados em competições entre os agentes autônomos.

Os resultados dos testes contra o Time Azul (Tabela 3) mostram que os agentes do Time Laranja obtiveram resultados piores, exceto no caso A2 já discutido. Com isso, a segunda

hipótese ganha mais força, uma vez que não se pode dizer que os agentes do Time Laranja aprenderam melhores comportamentos do que os do Time Azul, pois assim deveriam ter obtido resultados melhores.

Ao analisar os resultados dos testes contra o Time Vermelho, tem-se uma evidência ainda mais forte. A comparação das curvas de treinamento (Figura 38) e da pontuação média obtida contra oponentes aleatórios, mostra claramente uma vantagem para o Time Laranja em relação ao Time Vermelho. Entretanto, no confronto direto, o Time Vermelho foi melhor em todos os casos. Esses fatos reforçam ainda mais que o desempenho dos agentes do Time Laranja contra adversários aleatórios foi melhor por eles estarem mais acostumados com esse tipo de oponente ao longo do treinamento, ou seja, o controlador se ajustou a um tipo de adversário mais simplista.

5.5.3 Competição e inteligência

O objetivo deste trabalho é mostrar que o treinamento através de competição entre agentes inteligentes gera resultados melhores do que o treinamento competitivo contra um agente não-inteligente, em confrontos diretos. O grande número de partidas realizadas entre os agentes, 200 em cada confronto, dá uma segurança de afirmar que os resultados obtidos são consistentes. Assim, quando o Time Azul obtém melhores resultados do que o Time Laranja em confronto direto, é possível formular uma certa *versão fraca* do objetivo deste trabalho: agentes que treinam contra adversários inteligentes **e** obtêm um desempenho melhor, terão um melhor desempenho em confrontos diretos do que agentes que treinam contra oponentes não-inteligentes.

Uma vez que o Time Vermelho, mesmo tendo um desempenho geral bem abaixo dos times Azul e Laranja no treinamento, obteve resultados melhores do que o Time Laranja em um confronto direto, pode-se formular uma *versão forte* do objetivo deste trabalho: agentes que treinam contra adversários inteligentes terão em confrontos diretos um melhor desempenho do que agentes que treinam contra oponentes não-inteligentes.

5.6 Considerações Finais

Neste capítulo foram mostrados os treinamentos e testes realizados com o grupo de agentes autônomos desenvolvidos e utilizados em um ambiente de competição entre múltiplos jogadores. Foram apresentados e discutidos os resultados obtidos na fase de treinamento e na execução de

dois tipos diferentes de testes, contra oponentes completamente aleatórios e contra adversários inteligentes.

Com base nos resultados dos testes, verificou-se que o treinamento através de competição contra outros agentes autônomos produz um desempenho melhor em um ambiente de competição entre múltiplos jogadores, em detrimento de um treinamento contra adversários não-inteligentes. Em especial, treinar um agente autônomo contra outros agentes autônomos faz com que ele esteja mais apto a lidar com a disputa contra oponentes inteligentes.

O próximo capítulo apresenta as conclusões sobre este trabalho, destacando as suas principais contribuições, identificando em que pontos mudanças podem ser realizadas e sugerindo maneiras de como fazê-las em possíveis trabalhos futuros.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho teve como objetivo mostrar que o treinamento por competição aumenta o desempenho de agentes autônomos. Para tanto, foi avaliada a competição entre diversos agentes autônomos separados em grupos. Especificamente, os agentes autônomos que treinaram contra outros agentes autônomos obtiveram resultados melhores do que aqueles que treinaram contra oponentes não-inteligentes.

O desenvolvimento dos agentes foi mostrado desde o início. A tarefa de aprender comportamentos adequados em um jogo FPS, recebendo recompensas por suas ações, é descrita como um problema de Aprendizado por Reforço e solucionada utilizando um modelo de *Deep Learning*, chamado DQN. Essa técnica já foi utilizada com sucesso em diversos trabalhos semelhantes, como mostrado no Capítulo 3, tanto em situações com um único agente, quanto com múltiplos agentes.

O jogo FPS utilizado através do ambiente *ViZDoom* mostrou-se adequado para o aprendizado dos agentes e para a execução dos testes. Um cenário personalizado foi criado para permitir a competição entre dois agentes. Foi desenvolvido um controlador utilizando uma DNN, que permitiu o aprendizado de comportamentos adequados por parte dos agentes autônomos, utilizando como entrada somente os *pixels* da tela, através da utilização do modelo DQN. Portanto, é um controlador genérico que pode ser utilizado em outros cenários ou jogos sem a necessidade de alterações.

As características específicas dos experimentos e os hiperparâmetros da rede neural possibilitaram o aprendizado dos agentes autônomos utilizados, como verificado no Capítulo 4. Os regimes de treinamento através de competição levaram a resultados que mostraram o aprendizado de comportamentos adequados ao ambiente, como visto através do crescimento da curva de aprendizado (Figura 38).

Os testes executados através de competição entre agentes autônomos mostraram claramente um desempenho melhor do que agentes treinados contra jogadores não-inteligentes. Assim, os resultados validam a técnica proposta neste trabalho. Em especial, o modelo descrito no Capítulo 4 e os resultados analisados no Capítulo 5, mostram que todos os requisitos necessários citados no Capítulo 1 foram alcançados com êxito.

6.1 Trabalhos futuros

O controlador apresentado aqui não limita a arquitetura da rede neural às configurações que foram utilizadas. A utilização de outras arquiteturas diferentes não só é possível como pode gerar resultados interessantes. Entretanto, essa mudança afeta a base dos agentes desenvolvidos e todo o treinamento deve ser feito do zero, o que demandará tempo. Em particular, aumentar o tamanho da rede neural pode ser benéfico, uma vez que mais características podem ser aprendidas e passadas através das camadas. Porém, um maior tamanho leva a uma maior complexidade, o que diminuirá a velocidade do aprendizado. Além disso, o treinamento será bem mais difícil e pode inviabilizar o aprendizado.

Uma característica dos agentes mostrados neste trabalho é a capacidade de aprendizado recebendo como entrada somente os *pixels* da tela. Existem modelos que passam informações adicionais na entrada (BHATTI *et al.*, 2016; LAMPLE; CHAPLOT, 2017) e podem melhorar ainda mais o desempenho dos agentes. Entretanto, a utilização de informações aumentadas deve ser feita com cuidado, pois é interessante que um agente aprenda certas características por si só, para não haver necessidade de manipulação externa. De modo geral, utilizar modelos diferentes de redes neurais, como os vistos no Capítulo 3, pode dar resultados interessantes. Porém, a adaptação dessas técnicas não é trivial e não garante os mesmos benefícios.

Similarmente, os agentes poderiam ser testados em diferentes jogos, em especial do gênero FPS. Algumas modificações, como a adição de personagens civis (que não devem ser mortos) e a utilização de cenários com movimentação e mira tridimensionais, podem levar a resultados interessantes. Entretanto, podem ser necessárias adaptações específicas no controlador. É importante considerar também que o aumento da complexidade do problema dificultará o aprendizado.

O ambiente *ViZDoom* só exige como entrada ações correspondentes às atividades de teclado e *mouse*. Assim, outros agentes que não utilizem redes neurais podem ser avaliados no jogo. As adaptações necessárias são específicas para cada caso e os resultados não serão necessariamente semelhantes. Além disso, a comparação entre agentes com diferentes tipos de controladores pode levantar questões interessantes a serem avaliadas.

Como visto na Seção 5.5.1, um único agente apresentou resultados bem diferentes do que os outros. As avaliações dos resultados e do código-fonte dos agentes sugerem um problema interno do ambiente *ViZDoom*. Entretanto, para identificar e solucionar completamente este problema, é necessário realizar uma análise mais específica do código-fonte do ambiente.

Também devem ser considerados os fatores implícitos, controláveis ou não, que podem afetar o aprendizado, como a distribuição aleatória da posição inicial dos agentes.

Na Seção 5.2.1, é possível observar uma certa dominância apresentada pelo Time Azul. Uma vez que exceto o agente patológico A2 todos os outros do Time Azul foram os melhores, entende-se que há algum fator que implica essa dominância. Em testes realizados, observou-se que o fato dos agentes do Time Azul serem os servidores das partidas é o único elemento correlacionado a essa questão. Uma forma de avaliar esse caso seria inverter os jogadores cliente e servido ao longo do treinamento.

REFERÊNCIAS

ALVERNAZ, S.; TOGELIUS, J. Autoencoder-augmented neuroevolution for visual doom playing. In: **2017 IEEE Conference on Computational Intelligence and Games (CIG)**. [S.l.: s.n.], 2017. p. 1–8.

ARULKUMARAN, K.; DEISENROTH, M. P.; BRUNDAGE, M.; BHARATH, A. A. Deep reinforcement learning: A brief survey. **IEEE Signal Processing Magazine**, v. 34, n. 6, p. 26–38, Nov 2017. ISSN 1053-5888.

BARTO, A. G.; SUTTON, R. S.; ANDERSON, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. **IEEE Transactions on Systems, Man, and Cybernetics**, SMC-13, p. 834–846, 1983.

BELLEMARE, M. G.; NADDAF, Y.; VENESS, J.; BOWLING, M. The arcade learning environment: An evaluation platform for general agents. **Journal of Artificial Intelligence Research**, v. 47, p. 253–279, 2013.

BENGIO, Y.; LÉONARD, N.; COURVILLE, A. C. Estimating or propagating gradients through stochastic neurons for conditional computation. **CoRR**, abs/1308.3432, 2013. Disponível em: <http://arxiv.org/abs/1308.3432>.

BENGIO, Y.; LOURADOUR, J.; COLLOBERT, R.; WESTON, J. Curriculum learning. In: **Proceedings of the 26th Annual International Conference on Machine Learning**. New York, NY, USA: ACM, 2009. (ICML '09), p. 41–48. ISBN 978-1-60558-516-1.

BHATTI, S.; DESMAISON, A.; MIKSIK, O.; NARDELLI, N.; SIDDHARTH, N.; TORR, P. H. S. Playing doom with slam-augmented deep reinforcement learning. **CoRR**, abs/1612.00380, 2016. Disponível em: <http://arxiv.org/abs/1612.00380>.

CARDONA, A. B.; TOGELIUS, J.; NELSON, M. J. Competitive coevolution in ms. pac-man. In: **2013 IEEE Congress on Evolutionary Computation**. [S.l.: s.n.], 2013. p. 1403–1410. ISSN 1089-778X.

CASTAÑEDA, A. O. **Deep Reinforcement Learning Variants of Multi-Agent Learning Algorithms**. Dissertação (Mestrado) — School of Informatics University of Edinburgh, 2016.

CHEN, Z.; YI, D. **The Game Imitation: A Portable Deep Learning Model for Modern Gaming AI**. 2016. CS231n: Convolutional Neural Networks for Visual Recognition. Disponível em: <http://cs231n.stanford.edu/reports/2016/pdfs/113_Report.pdf>.

DAS, A.; KOTTUR, S.; MOURA, J. M. F.; LEE, S.; BATRA, D. Learning cooperative visual dialog agents with deep reinforcement learning. In: **2017 IEEE International Conference on Computer Vision (ICCV)**. [S.l.: s.n.], 2017. p. 2970–2979.

DAVIS, A. S.; AREL, I. Low-rank approximations for conditional feedforward computation in deep neural networks. **CoRR**, abs/1312.4461, 2013. Disponível em: <http://arxiv.org/abs/1312.4461>.

DIJK, J. van. **Recurrent Neural Networks for Reinforcement Learning: an Investigation of Relevant Design Choices**. Dissertação (Mestrado) — Universiteit van Amsterdam, Amsterdã, 2017.

DOSOVITSKIY, A.; KOLTUN, V. Learning to act by predicting the future. In: **International Conference on Learning Representations (ICLR)**. [S.l.: s.n.], 2017.

EGOROV, M. **Multi-Agent Deep Reinforcement Learning**. 2016. CS231n: Convolutional Neural Networks for Visual Recognition. Disponível em: <http://cs231n.stanford.edu/reports/2016/pdfs/122_Report.pdf>.

ELST, K. **Deep Q-learning in the Physical World**. Dissertação (Mestrado) — Universiteit Gent, Gante, 2016.

FOERSTER, J.; ASSAEL, I. A.; FREITAS, N. de; WHITESON, S. Learning to communicate with deep multi-agent reinforcement learning. In: LEE, D. D.; SUGIYAMA, M.; LUXBURG, U. V.; GUYON, I.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems 29**. [S.l.]: Curran Associates, Inc., 2016. p. 2137–2145.

FOERSTER, J.; NARDELLI, N.; FARQUHAR, G.; AFOURAS, T.; TORR, P. H. S.; KOHLI, P.; WHITESON, S. Stabilising experience replay for deep multi-agent reinforcement learning. In: PRECUP, D.; TEH, Y. W. (Ed.). **Proceedings of the 34th International Conference on Machine Learning**. International Convention Centre, Sydney, Australia: PMLR, 2017. (Proceedings of Machine Learning Research, v. 70), p. 1146–1155.

FOERSTER, J. N.; ASSAEL, Y. M.; FREITAS, N. de; WHITESON, S. Learning to communicate to solve riddles with deep distributed recurrent q-networks. **CoRR**, abs/1602.02672, 2016. Disponível em: <http://arxiv.org/abs/1602.02672>.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: TEH, Y. W.; TITTERINGTON, M. (Ed.). **Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics**. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010. (Proceedings of Machine Learning Research, v. 9), p. 249–256.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. In: GORDON, G.; DUNSON, D.; DUDíK, M. (Ed.). **Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics**. Fort Lauderdale, FL, USA: PMLR, 2011. (Proceedings of Machine Learning Research, v. 15), p. 315–323.

GUPTA, J. K.; EGOROV, M.; KOCHENDERFER, M. Cooperative multi-agent control using deep reinforcement learning. In: SUKTHANKAR, G.; RODRIGUEZ-AGUILAR, J. A. (Ed.). **Autonomous Agents and Multiagent Systems**. Cham: Springer International Publishing, 2017. p. 66–83.

HAHNLOSER, R. H. R.; SARPESHKAR, R.; MAHOWALD, M. A.; DOUGLAS, R. J.; SEUNG, H. S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. **Nature**, Macmillian Magazines Ltd., v. 405, p. 947, Jun 2000.

HASSELT, H. v.; GUEZ, A.; SILVER, D. Deep reinforcement learning with double q-learning. In: **Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence**. [S.l.]: AAAI Press, 2016. (AAAI'16), p. 2094–2100.

HAUSKNECHT, M.; STONE, P. Deep reinforcement learning in parameterized action space. In: **Proceedings of the International Conference on Learning Representations (ICLR)**. [S.l.: s.n.], 2016.

HAUSKNECHT, M. J. **Cooperation and Communication in Multiagent Deep Reinforcement Learning**. Tese (Doutorado) — Graduate School of The University of Texas at Austin, 2016.

HE, H.; BOYD-GRABER, J.; KWOK, K.; III, H. D. Opponent modeling in deep reinforcement learning. In: BALCAN, M. F.; WEINBERGER, K. Q. (Ed.). **Proceedings of The 33rd International Conference on Machine Learning**. New York, New York, USA: PMLR, 2016. (Proceedings of Machine Learning Research, v. 48), p. 1804–1813.

HEINRICH, J.; LANCTOT, M.; SILVER, D. Fictitious self-play in extensive-form games. In: **Proceedings of the 32Nd International Conference on International Conference on Machine Learning**. [S.l.]: JMLR, 2015. (ICML'15, v. 37), p. 805–813.

HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. **Neural Computation**, MIT Press, Cambridge, MA, USA, v. 18, n. 7, p. 1527–1554, jul. 2006. ISSN 0899-7667.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **Journal of Artificial Intelligence Research**, v. 4, p. 237–285, 1996.

KALYANAKRISHNAN, S.; LIU, Y.; STONE, P. Half field offense in robocup soccer: A multiagent reinforcement learning case study. In: LAKEMEYER, G.; SKLAR, E.; SORENTI, D.; TAKAHASHI, T. (Ed.). **RoboCup-2006: Robot Soccer World Cup X**. Berlin: Springer Verlag, 2007. (Lecture Notes in Artificial Intelligence, v. 4434), p. 72–85.

KEMPKA, M.; WYDMUCH, M.; RUNC, G.; TOCZEK, J.; JASKOWSKI, W. ViZDoom: A doom-based AI research platform for visual reinforcement learning. In: **IEEE Conference on Computational Intelligence and Games**. Santorini, Greece: IEEE, 2016. p. 341–348. Disponível em: <http://arxiv.org/abs/1605.02097>.

KONG, X.; XIN, B.; LIU, F.; WANG, Y. Revisiting the master-slave architecture in multi-agent deep reinforcement learning. **CoRR**, abs/1712.07305, 2017. Disponível em: <http://arxiv.org/abs/1712.07305>.

KULKARNI, T. D.; SAEEDI, A.; GAUTAM, S.; GERSHMAN, S. J. Deep successor reinforcement learning. **CoRR**, abs/1606.02396, 2016. Disponível em: <http://arxiv.org/abs/1606.02396>.

LAMPLE, G.; CHAPLOT, D. S. Playing FPS games with deep reinforcement learning. In: **Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)**. [S.l.: s.n.], 2017.

LANCTOT, M.; ZAMBALDI, V. F.; GRUSLYS, A.; LAZARIDOU, A.; TUYLS, K.; PÉROLAT, J.; SILVER, D.; GRAEPEL, T. A unified game-theoretic approach to multiagent reinforcement learning. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2017.

LECUN, Y.; BENGIO, Y.; HINTON, G.; Y., L.; Y., B.; G., H. Deep learning. **Nature**, v. 521, p. 436–444, 2015. ISSN 0028-0836.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, November 1998.

LEIBO, J. Z.; ZAMBALDI, V.; LANCTOT, M.; MARECKI, J.; GRAEPEL, T. Multi-agent reinforcement learning in sequential social dilemmas. In: **Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems**. [S.l.]: International Foundation for Autonomous Agents and Multiagent Systems, 2017. (AAMAS '17), p. 464–473.

LEVINE, S.; FINN, C.; DARRELL, T.; ABBEEL, P. End-to-end training of deep visuomotor policies. **J. Mach. Learn. Res.**, JMLR.org, v. 17, n. 1, p. 1334–1373, jan. 2016. ISSN 1532-4435.

LI, M.; ZHANG, T.; CHEN, Y.; SMOLA, A. J. Efficient mini-batch training for stochastic optimization. In: **Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: ACM, 2014. (KDD '14), p. 661–670. ISBN 978-1-4503-2956-9.

LIANG, Y.; MACHADO, M. C.; TALVITIE, E.; BOWLING, M. State of the art control of atari games using shallow reinforcement learning. **Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems**, p. 485–493, 2016.

LIN, L.-J. **Reinforcement Learning for Robots Using Neural Networks**. Tese (Doutorado) — Carnegie Mellon University, Pittsburgh, PA, USA, 1993.

MADDISON, C. J.; HUANG, A.; SUTSKEVER, I.; SILVER, D. Move evaluation in go using deep convolutional neural networks. **CoRR**, abs/1412.6564, 2014. Disponível em: <http://arxiv.org/abs/1412.6564>.

MARTA, D. L. S. **Deep Learning Methods for Reinforcement Learning**. Dissertação (Mestrado) — Instituto Superior Técnico, Lisboa, 2016.

MCKENZIE, M.; LOXLEY, P.; BILLINGSLEY, W.; WONG, S. Competitive reinforcement learning in atari games. In: **AI 2017: Advances in Artificial Intelligence - Proceedings of the 30th Australasian Joint Conference**. Melbourne, Australia: [s.n.], 2017. p. 14–26.

MNIH, V.; BADIA, A. P.; MIRZA, M.; GRAVES, A.; LILLICRAP, T. P.; HARLEY, T.; SILVER, D.; KAVUKCUOGLU, K. Asynchronous methods for deep reinforcement learning. **arXiv**, v. 48, p. 1–28, 2016. Disponível em: <http://arxiv.org/abs/1602.01783>.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. A. Playing atari with deep reinforcement learning. **CoRR**, abs/1312.5602, 2013. Disponível em: <http://arxiv.org/abs/1312.5602>.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. a.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G.; PETERSEN, S.; BEATTIE, C.; SADIK, A.; ANTONOGLOU, I.; KING, H.; KUMARAN, D.; WIERSTRA, D.; LEGG, S.; HASSABIS, D. Human-level control through deep reinforcement learning. **Nature**, v. 518, n. 7540, p. 529–533, 2015. ISSN 0028-0836.

MORAVČÍK, M.; SCHMID, M.; BURCH, N.; LISÝ, V.; MORRILL, D.; BARD, N.; DAVIS, T.; WAUGH, K.; JOHANSON, M.; BOWLING, M. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. **Science**, American Association for the Advancement of Science, v. 356, n. 6337, p. 508–513, 2017. ISSN 0036-8075.

NAIR, A.; SRINIVASAN, P.; BLACKWELL, S.; ALCICEK, C.; FEARON, R.; DE MARIA, A.; PANNEERSHELVAM, V.; SULEYMAN, M.; BEATTIE, C.; PETERSEN, S.; LEGG, S.; MNIH, V.; KAVUKCUOGLU, K.; SILVER, D. Massively parallel methods for deep reinforcement learning. **arXiv:1507.04296**, 2015. Disponível em: <http://arxiv.org/abs/1507.04296>.

PALMER, G.; TUYLS, K.; BLOEMBERGEN, D.; SAVANI, R. Lenient multi-agent deep reinforcement learning. **CoRR**, abs/1707.04402, 2017. Disponível em: <http://arxiv.org/abs/1707.04402>.

PARISOTTO, E.; BA, J.; SALAKHUTDINOV, R. Actor-mimic: Deep multitask and transfer reinforcement learning. In: **ICLR**. [S.l.: s.n.], 2016.

PATHAK, D.; AGRAWAL, P.; EFROS, A. A.; DARRELL, T. Curiosity-driven exploration by self-supervised prediction. In: **International Conference on Machine Learning (ICML)**. [S.l.: s.n.], 2017.

ROMOFF, J.; BENGIO, E.; PINEAU, J. Deep conditional multi-task learning in atari. In: **International Conference on Machine Learning (ICML)**. [S.l.: s.n.], 2016. v. 48.

RUMMERY, G.; NIRANJAN, M. **On-line Q-learning using connectionist sytems**. [S.l.], 1994. Technical Report - CUED/F-INFENG-TR 166.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. **Neural Networks**, Elsevier Ltd, v. 61, p. 85–117, 2015. ISSN 18792782.

SEIJEN, H. van; FATEMI, M.; ROMOFF, J.; LAROCHE, R.; BARNES, T.; TSANG, J. Hybrid reward architecture for reinforcement learning. **CoRR**, abs/1706.04208, 2017. Disponível em: <http://arxiv.org/abs/1706.04208>.

SERAFIM, P. B. S.; NOGUEIRA, Y. L. B.; VIDAL, C. A.; CAVALCANTE-NETO, J. B. On the development of an autonomous agent for a 3d first-person shooter game using deep reinforcement learning. In: **Anais do XVI Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames)**. [S.l.: s.n.], 2017.

SERAFIM, P. B. S.; NOGUEIRA, Y. L. B.; VIDAL, C. A.; CAVALCANTE-NETO, J. B. Towards playing a 3d first-person shooter game using a classification deep neural network architecture. In: **Proceedings of the 19th Symposium on Virtual and Augmented Reality (SVR)**. [S.l.: s.n.], 2017. p. 120–126.

SILVER, D.; HUANG, A.; MADDISON, C. J.; GUEZ, A.; SIFRE, L.; DRIESSCHE, G. van den; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M.; DIELEMAN, S.; GREWE, D.; NHAM, J.; KALCHBRENNER, N.; SUTSKEVER, I.; LILLICRAP, T.; LEACH, M.; KAVUKCUOGLU, K.; GRAEPEL, T.; HASSABIS, D. Mastering the game of go with deep neural networks and tree search. **Nature**, Nature Publishing Group, v. 529, n. 7587, p. 484–489, 2016. ISSN 0028-0836.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, p. 1929–1958, 2014.

SUKHBAATAR, S.; SZLAM, A.; FERGUS, R. Learning multiagent communication with backpropagation. In: LEE, D. D.; SUGIYAMA, M.; LUXBURG, U. V.; GUYON, I.; GARNETT,

R. (Ed.). **Advances in Neural Information Processing Systems 29**. [S.l.]: Curran Associates, Inc., 2016. p. 2244–2252.

SUKHBAATAR, S.; SZLAM, A.; SYNNAEVE, G.; CHINTALA, S.; FERGUS, R. Mazebase: A sandbox for learning from games. **CoRR**, abs/1511.07401, 2015. Disponível em: <http://arxiv.org/abs/1511.07401>.

SUTTON, R. S. Two problems with backpropagation and other steepest-descent learning procedures for networks. In: **Proceedings of the Eighth Annual Conference of the Cognitive Science Society**. [S.l.]: Hillsdale, NJ: Erlbaum, 1986.

SUTTON, R. S.; BARTO, A. G. **Introduction to Reinforcement Learning**. 1st. ed. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262193981.

TAMPUU, A.; MATIISEN, T.; KODELJA, D.; KUZOVKIN, I.; KORJUS, K.; ARU, J.; ARU, J.; VICENTE, R. Multiagent cooperation and competition with deep reinforcement learning. **PLOS ONE**, Public Library of Science, v. 12, n. 4, p. 1–15, 04 2017.

TIELEMAN, T.; HINTON, G. **Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude**. 2012. Coursera: Neural Networks for Machine Learning.

WANG, Z.; SCHAUL, T.; HESSEL, M.; HASSELT, H. V.; LANCTOT, M.; FREITAS, N. D. Dueling network architectures for deep reinforcement learning. In: **Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48**. [S.l.]: JMLR.org, 2016. (ICML'16), p. 1995–2003.

WATKINS, C. J. C. H. **Learning from Delayed Rewards**. Tese (Doutorado) — King's College, Oxford, 1989.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. **Machine Learning**, v. 8, p. 279–292, 1992.

WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. **Machine Learning**, v. 8, p. 229–256, 1992.

WU, Y.; TIAN, Y. Training agent for first-person shooter game with actor-critic curriculum learning. In: **International Conference on Learning Representations (ICLR)**. [S.l.: s.n.], 2017.

YAKOVENKO, N.; CAO, L.; RAFFEL, C.; FAN, J. Poker-cnn: A pattern learning strategy for making draws and bets in poker games using convolutional networks. In: **Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence**. [S.l.]: AAAI Press, 2016. (AAAI'16), p. 360–367.