UNIVERSIDADE FEDERAL DO PAMPA

Lucas Antunes de Almeida

Desenvolvimento de um Agente Inteligente para Jogar Jogos Genéricos de Atari 2600

Alegrete
2019

Lucas Antunes de Almeida

Desenvolvimento de um Agente Inteligente para Jogar Jogos Genéricos de Atari 2600

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marcelo Resende Thielo

Alegrete
2019

Lucas Antunes de Almeida

Desenvolvimento de um Agente Inteligente para Jogar Jogos Genéricos de Atari 2600

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 29 de 11 de 19
Banca examinadora:

Prof. Dr. Marcelo Resende Thielo
Orientador
UNIPAMPA

Prof. Dra. Alice Fonseca Finger
UNIPAMPA

Prof. Me. Jean Felipe Patikowski Cheiran
UNIPAMPA

Este trabalho é dedicado à minha mãe e irmã,
que me apoiaram durante toda a minha graduação.

AGRADECIMENTOS

Agradeço à minha mãe e irmã por todo o apoio, o qual possibilitou o meu envolvimento com a universidade. Agradeço por tudo que tiveram que abdicar para que eu tivesse a oportunidade de estar no curso.

Agradeço também aos meus amigos Felipe Homrich, Guilherme Neri, Lucas Melo, Rafael Fernandes e Victor Bifoni os quais me ajudaram em momentos que considerei abandonar o curso, possibilitando a minha estadia por mais tempo na universidade.

Agradeço ao meu orientador e amigo, professor Marcelo Thielo, que além de me instruir durante a graduação, em um momento no qual eu provavelmente iria trancar o curso por problemas financeiros, me indicou a um estágio que possibilitou o desenvolvimento deste trabalho.

“Thank you Mario! But our princess is in another castle!”.
(Super Mario Bros)

RESUMO

Dentre as áreas abordadas na Computação, a evolução da Inteligência Artificial vem se destacando, em especial na área de jogos. O objetivo deste trabalho é o desenvolvimento de um agente inteligente capaz de jogar múltiplos jogos. Para isto, primeiramente foi realizado um estudo sobre os conceitos necessários, os quais constituem toda a fundamentação teórica deste trabalho. Visando a abstração de conceitos de aprendizado de máquina e processamento de imagens, foram utilizadas as bibliotecas Keras e Gym. São apresentados modelos de Inteligência Artificial, Aprendizado de Máquina, Aprendizagem Profunda, Aprendizagem por Reforço, Aprendizagem por Reforço Profundo, Aprendizagem Q e Ator Crítico de Vantagem Assíncrono, adaptados para o contexto deste trabalho. Este trabalho realizou o levantamento do estado da arte na área de algoritmos para jogar jogos genéricos, apresentando os trabalhos que possuem maior relevância para o contexto atual. Agentes projetados para jogar efetivamente múltiplos jogos devem possuir a capacidade de adaptação ao ambiente no qual se encontram, determinando assim quais ações melhor se adequam a cada situação. A qualidade de um agente é medida pela distância entre suas pontuações e as pontuações obtidas por jogadores humanos. Este trabalho apresenta o desenvolvimento de um agente capaz de jogar múltiplos jogos, sem a necessidade de definir regras previamente, recebendo apenas a imagem da tela do jogo, a pontuação atual e se o estado atual representa um estado de fim de jogo. Para isto são apresentados os diagramas do desenvolvimento do agente, em conjunto com suas repectivas informações. Para provar a eficiência do agente, as pontuações obtidas nos jogos Space Invaders, Breakout, Demon Attack e Kung Fu Master foram comparadas com as pontuações obtidas por jogadores humanos.

Palavras-chave: Algoritmos para Jogar Jogos Genéricos. Inteligência Artificial. Atari 2600.

ABSTRACT

Among the areas addressed in Computing, the evolution of Artificial Intelligence has been highlighting, especially in the area of games. The aim of this paper is the development of an intelligent agent capable of multiplayer games. For this, a study was first made on the necessary concepts, which constitute all the theoretical foundations of this work. For the abstraction of machine learning and image processing concepts, we used the Keras and Gym libraries. This paper presents the definition of models of Artificial Intelligence, Machine Learning, Deep Learning, Reinforcement Learning, Deep Reinforcement Learning, Q Learning, and Asynchronous Advantage Actor-Critic adapted to the context of this work. This paper surveyed the state of the art in the area of algorithms for playing generic games, presenting the works that are most relevant to the current context. Agents designed to effectively play multiple games must have the ability to adapt to the environment in which they find themselves, thus determining which actions best suit each situation. The quality of an agent is measured by the distance between its scores and the scores obtained by human players. This paper presents the development of an agent capable of playing multiple games without the need to define rules in advance, receiving only the game screen image, the current score and whether the current state represents an endgame state. For this, the diagrams of agent development are presented, together with their respective information. To prove the effectiveness of the agent, the scores obtained in the Space Invaders, Breakout, Demon Attack, and Kung Fu Master games were compared to the scores obtained by human players.

Key-words: General Game Playing. Artificial Intelligence. Atari 2600.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

A3C Ator Crítico de Vantagem Assíncrono

IA Inteligência Artificial

AP Aprendizagem Profunda

RNP Redes Neurais Profundas

RPQ Rede Profunda Q

ARP Aprendizado por Reforço Profundo

FPS Tiro em Primeira Pessoa

GGP Algoritmos para Jogar Jogos Genéricos

MAME Multiple Arcade Machine Emulator

AM Aprendizado de Máquina

AQ Aprendizagem Q

AR Aprendizado por Reforço

SUMÁRIO

1 INTRODUÇÃO

A implementação de um algoritmo capaz de jogar jogos adequadamente de modo similar a um jogador humano experiente ainda é um desafio para a computação atual. Este desafio é constituído principalmente na definição de como um programa adquire conhecimento. Nas últimas décadas, as abordagens utilizadas para que o agente possa determinar seu próximo passo, têm apresentado uma constante evolução, a maior parte deste crescimento ocorrendo por conta do foco na área de Inteligência Artificial (IA) e sua crescente necessidade de ambientes de testes adequados. Dentro da área de IA, os jogos representam ambientes de testes para os modelos desenvolvidos, possibilitando a realização de testes em inúmeras situações distintas apenas trocando o jogo selecionado.

A necessidade de definir regras e características específicas de um jogo para desenvolver algum algoritmo capaz de jogá-lo pode impedir a reutilização do mesmo em jogos com regras diferentes. Mesmo que a mudança de jogo não seja tão brusca, visualmente jogos como Xadrez e Damas são similares, já que ocorrem exatamente no mesmo tabuleiro e possuem um número similar de peças, mas pequenas diferenças entre a aparência destas peças e suas respectivas formas de movimentação acarretam em jogos completamente distintos. Por outro lado, proporcionar ao algoritmo a capacidade de se adaptar a um jogo do qual as regras nunca lhe foram apresentadas é uma tarefa complexa e desafiadora.

Atualmente, o desafio consiste em determinar quais técnicas de detecção dos elementos presentes no ambiente do jogo e quais métodos de aprendizado serão utilizadas em conjunto. Assim um agente pode determinar quais elementos do ambiente deve levar em consideração no momento, possibilitando a obtenção da melhor ação que deve selecionar para maximizar suas chances de vitória ou progresso no jogo a longo prazo. Para diminuir a distância do desempenho em jogos entre jogadores humanos e agentes de IA, muitos profissionais da área vêm empregando constante esforço para desenvolver otimizações heurísticas e modelos de Aprendizado de Máquina (AM) capazes de atingir resultados satisfatórios.

O grupo DeepMind (2019), líder em pesquisa de IA e sua aplicação para um impacto positivo, desenvolve programas que podem aprender e resolver qualquer problema complexo sem precisar ser ensinado. Ao implementar suas pesquisas no campo dos jogos, um campo de treinamento útil e flexível, o agente desenvolvido foi capaz de aprender a jogar mais de 40 títulos de Atari 2600 completamente diferentes, apenas com o fornecimento de *pixels* brutos de entrada para o agente. Na matéria publicada pelo AlphaStar (2019), membros do grupo DeepMind, é apresentado um programa para jogar StarCraft II, a primeira IA a derrotar um grande jogador profissional. Embora antes disso tenham havido sucessos significativos em *video games*, até o momento, as técnicas de IA têm lutado para lidar com alguns jogos de estratégia.

1.1 Motivação

Determinar a eficácia de um modelo é uma tarefa complexa, uma vez que o grau de dificuldade envolvido em executar testes não é trivial. Jogos proporcionam um ambiente livre de influência externa, perfeito para validar a eficácia de um modelo, permitindo que o agente desenvolvido seja submetido a múltiplos ambientes de testes com contextos e objetivos distintos. *Benchmarks* em jogos possibilitam a detecção de pontos de melhoria ou correções que podem ser necessárias em um modelo de AM.

Conforme Bowling et al. (2006), os jogos são criados para entretenimento, simulação ou educação, embora eles proporcionem ótimas oportunidades para aplicações de algoritmos de AM. A variedade de contextos possíveis e os problemas decorrentes dos mesmos são limitados apenas pela implementação do próprio jogo, além é claro da industria de *games* estar em um estado de evolução constante. Consequentemente, os jogos que utilizam de técnicas de AM atrairiam a atenção para o campo.

Segundo Serafim et al. (2017), os jogos são frequentemente usados como ambientes para testar diferentes algoritmos e técnicas. Algumas características que contribuem para a popularidade destes ambientes estão ligadas à complexidade, ao determinismo, a entrada limitada de dados, a quantidade de *pixels* presentes na tela e ao grande impacto em relação à exposição ao público, acarretando em um aumento no interesse e importância da pesquisa utilizando jogos digitais com ambiente de testes.

Segundo AlphaStar (2019), os jogos são usados há décadas como uma maneira importante de testar e avaliar o desempenho dos sistemas de IA. À medida que as capacidades crescem, a comunidade acadêmica busca jogos com complexidade crescente que incorporem elementos de IA de maior complexidade, necessários para resolver problemas científicos e do mundo real.

1.2 Objetivos

Tem-se como objetivo geral deste trabalho a construção de um agente voltado ao tema de Algoritmos para Jogar Jogos Genéricos (GGP) utilizando AM capaz de determinar o contexto do jogo ao qual foi submetido. Este agente deve receber apenas a imagem do estado atual do jogo e a quantidade de vidas restantes na tentativa atual, eliminando a necessidade de desenvolver métricas para que o agente determine os elementos centrais do jogo, de forma que o foco do agente permaneça em como jogar adequadamente o jogo e não em extrair as *features*.

Este trabalho é fortemente inspirado por Mnih et al. (2015), no qual os autores apresentam um modelo de agente capaz de obter resultados similares ao de um humano experiente em muitos dos jogos testados. Como objetivo específico, este trabalho visa desenvolver um agente que demonstre um desempenho similar aos de um humano em múltiplos jogos.

1.3 Planejamento do Trabalho

Nesta seção, será apresentado o planejamento utilizado para atingir os objetivos propostos por este trabalho. As etapas são as que seguem.

1. **Definir um tema:** Selecionar um tema de pesquisa para o trabalho de conclusão de curso;
2. **Investigar o estado da arte:** Fazer um estudo em relação aos trabalhos relacionados ao tema selecionado;
3. **Selecionar trabalhos relacionados:** Determinar a relevância dos trabalhos relacionados para este trabalho;
4. **Estudar conceitos:** Estudar os conceitos apresentados nos trabalhos relacionados selecionados;
5. **Investigar possível abordagem:** Determinar quais técnicas devem ser utilizadas neste trabalho;
6. **Desenvolver protótipo:** A partir dos conceitos estudados, desenvolver um protótipo;
7. **Expandir protótipo:** Incrementar o protótipo desenvolvido;
8. **Realizar testes:** Validar o agente proposto em múltiplos jogos;
9. **Registrar trabalho realizado:** Registrar os conceitos, técnicas utilizadas e formas de validação do agente proposto no trabalho de conclusão de curso.

1.4 Organização Deste Trabalho

- Capítulo 2 **Fundamentação Teórica:** Conceitos ligados ao trabalho proposto.
- Capítulo 3 **Trabalhos Relacionados:** Apresentação da metodologia de pesquisa e seu resultado.
- Capítulo 4 **Desenvolvimento:** Apresentação do conjunto de atividades que foram realizadas.
- Capítulo 5 **Resultados:** Apresentação dos resultados obtidos neste trabalho.
- Capítulo 6 **Considerações Finais:** Contém as considerações finais sobre o trabalho realizado.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os fundamentos teóricos necessários para o entendimento do presente trabalho. O capítulo será estruturado da seguinte maneira: na seção 2.1 é apresentado o conceito de GGP; na seção 2.3 é apresentado o conceito de IA para jogos e, em sequência, técnicas utilizadas no contexto GGP; na seção 2.4 serão apresentadas as bibliotecas utilizadas neste trabalho.

2.1 Algoritmos para Jogar Jogos Genéricos

O termo GGP é designado para algoritmos que sejam aptos a jogar mais de um jogo com sucesso. Em jogos como o xadrez geralmente são empregados algoritmos específicos, com heurísticas projetadas para maximizar suas próprias chances de vitória utilizando características específicas do jogo, barrando assim a utilização deste algoritmo em algum outro jogo. A menos que o jogo seja alguma variação do Xadrez, o algoritmo heurístico tende a não conseguir jogá-lo adequadamente.

Um algoritmo heurístico desenvolvido para jogar Xadrez não possui a capacidade de jogar adequadamente o jogo Othello, mesmo que ambos ocorram em ambientes similares. Um algoritmo proposto dentro do contexto de GGP possui a capacidade de jogar eficientemente jogos pouco similares. Algoritmos voltados para competições de GGP geralmente são testados em jogos distintos comprovando, assim, a capacidade do algoritmo de se adaptar de maneira eficiente ao ambiente em que foi submetido.

Segundo os autores Browne et al. (2019), ao longo da maior parte da história, as regras dos jogos eram transmitidas verbalmente. Um dos aspectos que pode contribuir para comprovar a veracidade de um conjunto de regras de um jogo é a facilidade com a qual um conjunto de regras é explicado e compreendido. Jogos com descrições de regras extremamente longas e detalhadas são menos prováveis de terem sido jogados ao longo da história do que jogos com regras que podem ser explicadas facilmente.

De acordo com os autores Sheng e Thuente (2011), um agente destinado a GGP é projetado para cobrir uma grande variedade de jogos, incluindo jogos *single-player*, jogos para dois jogadores, jogos *multiplayer*, jogos de turnos ou simultâneos, jogos competitivos ou cooperativos. Os agentes devem se adaptar automaticamente não só para jogar adequadamente os jogos, mas devem jogá-los de maneira eficiente, geralmente superando a capacidade de um jogador humano.

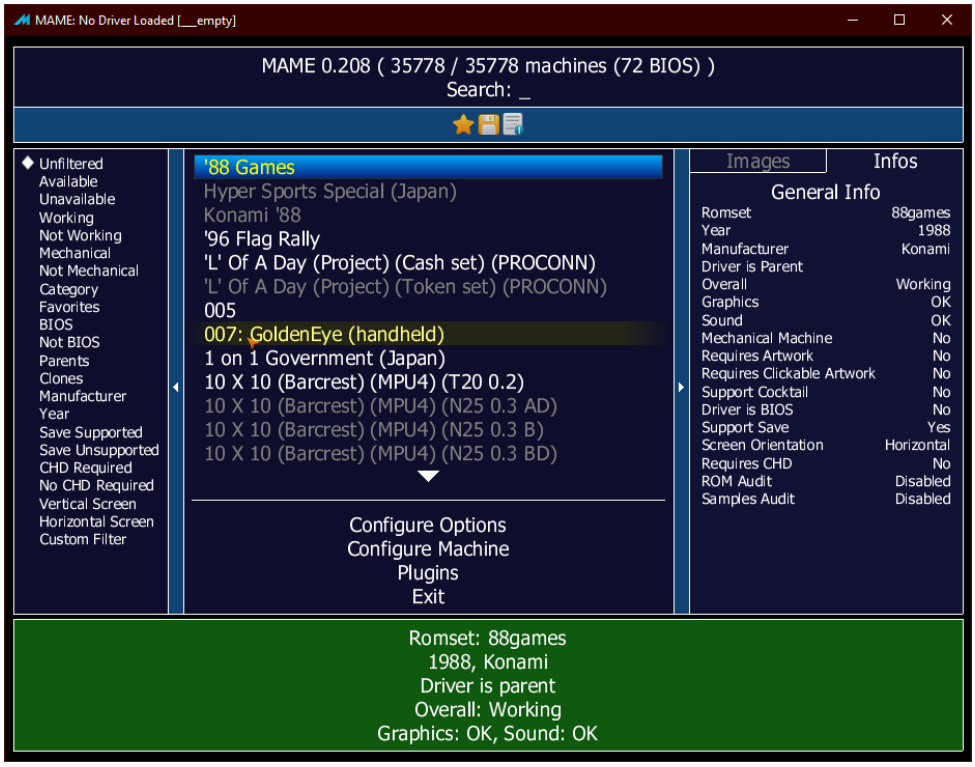
Os autores Dockhorn e Apeldoorn (2018), apresentam o desenvolvimento de agente para GGP capaz de lidar com múltiplos tipos de problemas diferentes, interpretar de maneira adequada o ambiente e determinar de forma automática quais ações devem ser selecionadas considerando as informações do ambiente que estejam a sua disposição. É um dos principais objetivos de longo prazo nas pesquisas na área de IA.

2.2 Multiple Arcade Machine Emulator

Emuladores proporcionam a oportunidade de se virtualizar uma máquina, seja ela um computador ou um *console*, possibilitando a execução de programas e softwares que tenham sido desenvolvidos especificamente para aquela arquitetura.

O Multiple Arcade Machine Emulator (MAME) é um emulador de código aberto. O qual tem como principal objetivo preservar preservar décadas de jogos históricos de *arcade*, computadores e *consoles*. Como a tecnologia continua a avançar, o MAME impede que estes importantes sistemas *vintage* sejam perdidos e esquecidos (MAMEdev Team, 2019). Na Figura 1 pode-se observar a tela inicial do emulador.

Figura 1 – Tela inicial do emulador MAME



Fonte: Elaborado pelo autor.

2.3 Inteligência Artificial

O domínio de IA consiste principalmente em ações nas quais o computador não tem facilidade em realizar, como reconhecer rostos, falar algum idioma ou ser criativo. Na Figura 2, encontra-se a tela inicial do jogo *Golden Axe*, que para sua época representava um grande avanço no campo de IA para jogos. O jogo contava com inimigos que passavam correndo pelo jogador e depois atacavam o jogador por trás.

Figura 2 – Telas iniciais do jogo Golden Axe MAME

Fonte: Elaborado pelo autor.

De acordo com os autores Galway, Charles e Black (2008), a área de IA para jogos representa a implementação de um conjunto de algoritmos e técnicas da IA tradicional e moderna, com o objetivo de fornecer soluções para uma série de problemas dependentes do ambiente do jogo.

Segundo os autores Millington e Funge (2009), a IA, na maioria dos jogos modernos, aborda três tópicos fundamentais: a capacidade de mover personagens, a capacidade de tomar decisões sobre onde se mover e a capacidade de pensar de maneira tática ou estratégica em relação ao ambiente em que se encontram.

2.3.1 Aprendizado de Máquina

O AM é uma das muitas aplicações da área de IA que fornece aos sistemas a capacidade de aprender e se aprimorar de forma automática a partir de ciclos de iterações, onde por meio destas iterações e da análise dos dados fornecidos à aplicação, a mesma passa a se ajustar, possuindo assim a capacidade de determinar qual a ação mais adequada a tomar no momento de forma automática.

Segundo Fürnkranz (2001), o AM tornou-se uma das principais áreas em IA. Onde inevitavelmente os algoritmos que utilizam técnicas de AM rivalizam e superam as capacidades humanas na maioria dos jogos relacionados a estratégia ou coordenação motora.

2.3.2 Aprendizagem Profunda

A definição de Aprendizagem Profunda (AP) consiste em um tema emergente dentro do campo de IA. Consiste em uma subcategoria de AM que enfatiza a possibilidade de aprendizado com uso em redes neurais. A AP realiza o treinamento de um modelo computacional para que o mesmo possa detectar padrões nos dados aos quais foi submetido, visando assim aproximar soluções com base na generalização destes dados.

De acordo com Bengio et al. (2009), técnicas ligadas a algum modelo de AP visam o aprendizado de hierarquias com recursos de níveis mais altos em relação a hierarquia formada pela composição de níveis inferiores. Os recursos de aprendizado em vários níveis de abstração permitem que o sistema aproxime respostas de funções complexas que mapeiam a entrada para uma saída diretamente ligada aos dados, sem depender de recursos criados manualmente.

Segundo os autores Justesen et al. (2019), a rápida evolução dos métodos de aprendizagem profunda é, sem dúvida, devido à convenção de comparar os resultados em conjuntos de dados disponíveis publicamente. Uma convenção semelhante na área de IA, é a de usar ambientes de jogo para comparar algoritmos de jogo, nos quais métodos são classificados com base em sua capacidade de marcar pontos e vencer em jogos.

2.3.3 Aprendizado por Reforço

Técnicas de Aprendizado por Reforço (AR) permitem que o agente possua a capacidade de alterar o seu comportamento baseando-se em respostas obtidas do ambiente. Em alguns casos, o agente pode atingir o mínimo global para um determinado problema, ou seja, existe a possibilidade de minimizar a quantidade de erros cometidos em um determinado ambiente de jogo. Um esquema simples do funcionamento da um agente de AR pode ser visto na Figura 3.

Figura 3 – Esquema simplificado de um modelo de Aprendizagem por Reforço

Fonte: Elaborado pelo autor.

A Figura 3, apresenta um esquema clássico de AR, adaptado para jogos digitais, onde é fornecido ao agente um estado e uma recompensa, os quais podem ser representados por uma imagem da tela e uma pontuação respectivamente. Como resposta o agente devolve uma ação para o jogo.

Os autores Millington e Funge (2009), apresentam que o AR é o nome dado a uma variedade de técnicas de aprendizado baseadas na experiência. Um algoritmo baseado em

reforço possui três componentes fundamentais, sendo elas uma estratégia de exploração para testar diferentes ações ligadas ao jogo, uma função de reforço que determina de quanto é a recompensa de uma determinada ação e uma regra de aprendizado que une as duas primeiras características.

Assim como apresentado por Jeerige, Bein e Verma (2019), o AR é uma classe de modelos de AM onde o processo de aprendizagem é baseado em respostas avaliativas sem qualquer sinal supervisionado. O AR tem como objetivo criar agentes semelhantes a humanos, que aprendem por tentativa e erro, apenas com recompensas ou punições, para desenvolver estratégias bem sucedidas que eventualmente levem às maiores recompensas de longo prazo.

2.3.4 Aprendizado por Reforço Profundo

Modelos de Aprendizado por Reforço Profundo (ARP) consistem em combinações de modelos tradicionais como AR e AP, proporcionando redes com a capacidade de aprendizagem profunda, mas com características de pontuação. A Figura 4 mostra que a única diferença entre um agente de ARP e AR tradicional, está na presença de uma rede utilizada no aprendizado do próprio agente.

Figura 4 – Esquema simplificado de um modelo de Aprendizagem por Reforço Profundo

Fonte: Elaborado pelo autor.

Segundo os autores Torrado et al. (2018) um agente de ARP aprende através de interações com um ambiente dinâmico e equilibra a troca de recompensas entre o planejamento de longo prazo e curto prazo, ou seja, determina quais ações são vantajosas a curto e longo prazo. Agentes de ARP são constituídos essencialmente pela combinação entre modelos de AR e AP.

2.3.5 Aprendizagem Q

Aprendizagem Q (AQ) é um modelo de AR aprimorado com políticas que procuram

encontrar a melhor ação a ser tomada em relação ao estado atual do ambiente fornecido. A função de aprendizagem utiliza ações aleatórias e, portanto, procura ações que maximizem a recompensa total. Modelos de AQ prezam pela qualidade da solução, mesmo que para obter soluções de maior qualidade, o tempo gasto em busca da solução seja maior. Um agente de AQ armazena os estados encontrados e sua melhor resposta em uma tabela para permitir comparações entre estados, o que pode ser visto na Figura 5.

Figura 5 – Funcionamento de um agente de Aprendizagem Q

Fonte: Elaborado pelo autor.

Segundo os autores Millington e Funge (2009), técnicas de AQ dependem de possuir o problema representado de uma maneira particular. Com essa representação, ele pode armazenar e atualizar informações relevantes à medida que explora ações possíveis. Modelos de AQ tratam o mundo do jogo como uma máquina de estados. A qualquer momento, o algoritmo está em algum estado, o qual deve codificar todos os detalhes relevantes sobre o ambiente e os dados internos do personagem. Assim, se a saúde do personagem é significativa para o aprendizado, e se o personagem se encontra em duas situações idênticas com dois níveis de saúde diferentes, então ele os considerará como estados diferentes. Qualquer informação não incluída no estado não pode ser aprendida.

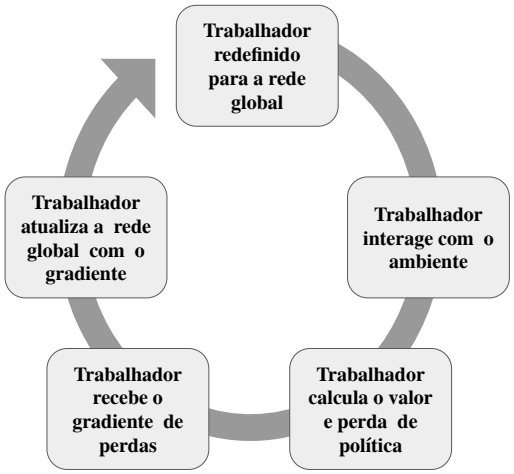
2.3.6 Ator Crítico de Vantagem Assíncrono

De acordo com os autores Mnih et al. (2016) o modelo de Ator Crítico de Vantagem Assíncrono (A3C) é mais rápido, mais simples, mais robusto e capaz de obter pontuações superiores quando comparado com técnicas de tradicionais ARP. O modelo é capaz de funcionar em espaços de ações contínuos e discretos.

A estrutura de um agente A3C consiste em ciclos de iterações nos quais um agente denominado de trabalhador repete as mesmas ações até que o resultado seja satisfatório. Uma estrutura simplificada para um agente A3C pode ser vista na Figura 6.

As principais características de uma rede A3C são:

Figura 6 – Funcionamento de um agente A3C



Fonte: Elaborado pelo autor.

- **Rede Global:** No A3C, há uma rede global e vários agentes, cada um com seu próprio conjunto de parâmetros de rede
- **Agentes:** Cada um desses agentes interage com sua própria cópia do ambiente ao mesmo tempo em que os outros agentes estão interagindo com seus ambientes.
- **Gradiente de perdas:** Vetor que aponta para a concentração de perdas do agente.
- **Perda de política:** Corresponde à propagação das probabilidades de ação

2.4 Bibliotecas

Sendo as bibliotecas uma coleção de classes, funções e variáveis escritas para facilitar o desenvolvimento de aplicações, foi decidido que a utilização das mesmas seria necessária para o desenvolvimento deste trabalho. A utilização de bibliotecas possibilita a abstração de conceitos básicos de programação, possibilitando que os esforços gastos com a implementação de um agente para GGP sejam feitos de forma objetiva. O uso de bibliotecas permite que o tempo que seria gasto desenvolvendo funções como detectores de posições para personagens ou a base de um modelo de AM seja utilizado para realizar testes e otimizações que possivelmente terão impacto no desempenho do agente.

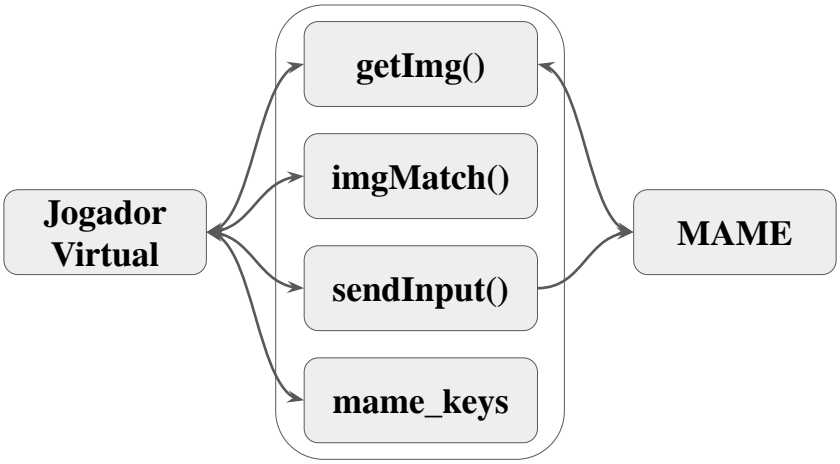
2.4.1 MameVirtualPlayer

Para utilizar o emulador MAME como ambiente de simulação em conjunto com a linguagem de programação *Python*, é necessária a utilização de algum intermediador. Levando em conta a facilidade de utilização do intermediador, foi inicialmente escolhida

a biblioteca *MameVirtualPlayer*. Proposta e desenvolvida por Dora (2018), a *MameVirtualPlayer* é uma biblioteca para substituição de um jogador no emulador MAME por uma IA, utilizável na prática. Por ter sido desenvolvida com o intuito de ser utilizada em conjunto com o emulador MAME a biblioteca é de fácil utilização no contexto atual.

A biblioteca *MameVirtualPlayer* permite abstrair a parte da comunicação com o emulador MAME e o agente desenvolvido, além de possibilitar a utilização da linguagem de programação *Python* para o desenvolvimento do agente. A biblioteca fornece funções de detecção de *Sprites* e localizações dos mesmos no ambiente do jogo em tempo de execução. A Figura 7 mostra como as funções da biblioteca *MameVirtualPlayer* interagem com o emulador MAME.

Figura 7 – Arquitetura da biblioteca MameVirtualPlayer



Fonte: Adaptado de Dora (2018)

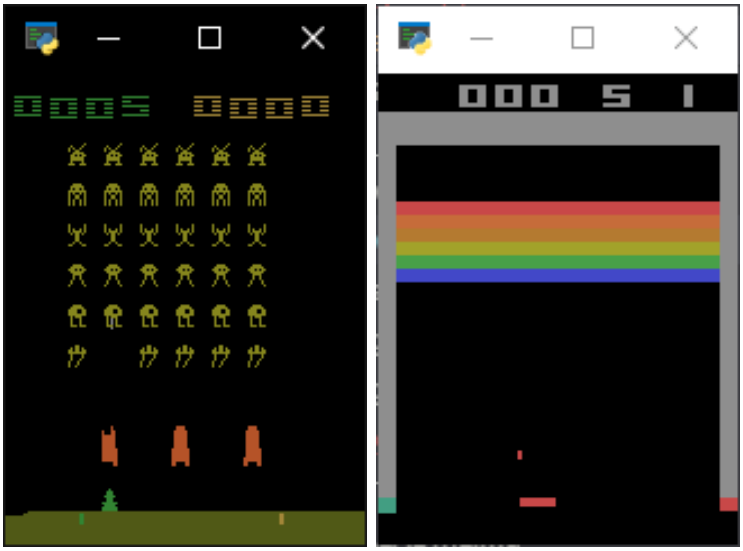
2.4.2 Gym

Segundo Gym (2019), o Gym é um kit de ferramentas para desenvolver e comparar algoritmos de aprendizado por reforço. Ele não faz suposições sobre a estrutura do seu agente e é compatível com qualquer biblioteca de computação numérica, como TensorFlow ou Theano. A biblioteca é uma coleção de problemas de teste e ambientes que você pode usar para elaborar seus algoritmos de aprendizado por reforço. Esses ambientes têm uma interface compartilhada, permitindo a escrita de algoritmos gerais.

A biblioteca Gym possui ambientes de emulação que permitem aos desenvolvedores efetuarem testes em seus algoritmos de AR. Dentre estes ambientes, encontra-se um emulador de Atari 2600. A Figura 8 mostra os jogos Space Invaders e Breakout emulados na biblioteca Gym.

Por se tratar de uma biblioteca com foco em testes de algoritmos de AR, a biblioteca Gym facilita a obtenção de informações que estão presentes em múltiplos jogos.

Figura 8 – Space invaders e breakout na biblioteca Gym



Fonte: Elaborado pelo autor.

Sempre que um agente envia uma ação para a biblioteca, recebe como resposta a imagem da tela, a recompensa, a confirmação de que a partida ainda não terminou e informações específicas do jogo em execução.

2.4.3 Keras

Para a implementação das redes necessárias neste trabalho, foi selecionada a biblioteca *Keras* com base na facilidade de uso e variações de redes disponíveis para a utilização, possibilitando testes com múltiplos tipos de modelos e *Frameworks* sem grandes alterações no código do agente.

O *Keras* é uma API de alto nível de redes neurais com código aberto, escrito em *Python* e capaz de executar em cima de *TensorFlow*, *CNTK*, ou *Theano*. Foi desenvolvido com foco em permitir a experimentação rápida. Ser capaz de ir da ideia ao resultado com o menor atraso possível é a chave para fazer uma boa pesquisa (KERAS, 2019). Os possíveis *Frameworks* para o *Keras* possuem as seguintes características:

- **CNTK:** O *CNTK* é o *Framework* de aprendizagem profunda de código aberto da *Microsoft*. A biblioteca inclui rede neurais profundas de *feed-forward*, redes convolucionais e redes recorrentes.

- **Theano:** Escrito em *Python*, *Theano* é uma biblioteca que lida com matrizes multi-dimensionais, como o *Numpy*. Usado com outras bibliotecas, é bem adequado para exploração de dados e destinado a pesquisa.

- **Tensorflow:** O *Tensorflow* foi criado pela Google com o intuito de substituir o *Theano*. Possui ferramentas para apoiar o aprendizado por reforço.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta a abordagem utilizada na busca de trabalhos relacionados ao tópico principal deste trabalho. Para isto, a seção 3.1 apresenta os critérios utilizados durante a pesquisa. Os resultados da pesquisa são apresentados na seção 3.2. Por fim as considerações do capítulo são apresentadas na seção 3.3.

3.1 Método de Pesquisa

Foi realizada uma pesquisa em torno do tema de GGP objetivando fazer um levantamento do estado da arte. Para este fim, foram utilizadas as plataformas *Google Scholar*[1] e *IEEE Xplore Digital Library*[2], nas quais foi utilizada a string de busca *((general game playing) AND ((deep learning) OR (deep reinforcement learning) OR (machine learning) OR (convolutional neural network) OR (reinforcement learning)))*

O critério de filtragem utilizado consiste em uma escala de relevância em relação ao presente trabalho, como pode ser visto a seguir:

- **Não Relevante:** Artigos que apresentassem algo relacionado a GGP mas não se adequavam ao objetivo deste trabalho.
- **Relevante:** Artigos que apresentavam contribuições para este trabalho, como aplicações de algoritmos de AM em GGP.
- **Muito Relevante:** Artigos que apresentam seus próprios modelos com foco em GGP com desempenhos superiores à outros modelos.

3.2 Resultados da Pesquisa

Após determinar qual a relevância de cada um dos artigos obtidos em relação ao objetivo deste trabalho, foram selecionados 10 artigos que possuem fundamentos e propostas importantes para a realização do trabalho. Segue a análise dos artigos selecionados.

Os autores Sheng e Thuente (2011) apresentam um agente destinado a GGP que isola a cobertura de pesquisa heurística para a tomada de decisões contextuais, criando assim uma árvore de decisão. Os autores avaliam a influência de características do jogo em relação ao contexto de decisão e não de todo o jogo. O agente proposto apresenta melhorias significativas em relação a tomada de decisões baseada em contexto para os jogos testados, os quais incluíam Tic-Tac-Toe e Othello.

No trabalho de Mnih et al. (2013), é apresentado um modelo de AP utilizando técnicas de ARP para GGP chamado de Rede Profunda Q (RPQ), o qual é submetido a testes em jogos do Atari 2600 sem a necessidade de incorporar informações específicas dos

[1] Disponível em <https://scholar.google.com.br/>
[2] Disponível em <https://ieeexplore.ieee.org/Xplore/home.jsp>

jogos que seriam executados, usando apenas *pixels* brutos como entrada para os testes. Para demonstrar a eficácia de seu algoritmo, os autores apresentam que o desempenho obtido supera o desempenho de um jogador humano experiente nos jogos Breakout, Enduro e Pong, mas mostram também que os resultados em jogos como Q * bert, Seaquest e Space Invaders não superam os de um jogador humano.

O modelo apresentado por Guo et al. (2014) tem como objetivo principal construir um agente de Atari 2600 capaz de jogar em tempo real superior a agentes baseados no modelo RPQ. A ideia central é utilizar agentes baseados em planejamento lento para fornecer dados de treinamento para uma arquitetura de AP capaz de reproduzir ações em tempo real. São propostos três modelos de agentes distintos, onde todos o três agentes utilizam a mesma base, entretanto possuem pequenas diferenças em relação a forma que tratam os dados de entrada. As abordagens apresentadas podem atingir resultados superiores aos obtidos com RPQ, entretanto para isto é necessário desacelerar o jogo, proporcionando mais tempo de processamento para o agente desenvolvido.

No trabalho desenvolvido por Mnih et al. (2015), os autores aprimoram um agente baseado no modelo RPQ, que pode aprender políticas dos jogos diretamente de dados sensoriais de alta dimensão. Sendo capaz de combinar as técnicas de ARP com Redes Neurais Profundas (RNP), o objetivo do agente proposto é o de selecionar ações de uma maneira que maximize o *score* futuro cumulativo, visando atingir a função de ótimo valor, que nada mais é do que a soma das recompensas recontada a cada iteração. O algoritmo proposto foi testado com 49 jogos distintos, nos quais jogos como Breakout e Boxing obtiveram *scores* de pelo menos dez vezes o que foi obtido por um jogador experiente.

Os autores Singal, Aggarwal e Dutt (2017) têm como principal objetivo comparar algoritmos de ARP com AR tradicionais em um ambiente virtual em relação ao seu desempenho, velocidade de aprendizado, capacidade de explicar decisões humanas e capacidade de extrair recursos do ambiente de decisão. São apresentadas implementações de modelos de AR tradicionais, como *Imitation Learning* e dois modelos recentes, sendo o primeiro deles o *Q-Learning* e o segundo modelo um algoritmo de ARP. O jogo utilizado nos testes é um jogo de plataforma desenvolvido pelos próprios autores do trabalho com a finalidade de validar os algoritmos. Neste trabalho, foram comparados os resultados obtidos pelos algoritmos com quinze jogadores humanos. Enquanto o desempenho das técnicas mais atuais se manteve constante, os resultados do algoritmo de *Imitation Learning* variavam de acordo com a experiência dos jogadores humanos.

No trabalho apresentado por Serafim et al. (2017), é introduzida uma arquitetura de rede para resolver um problema de aprendizagem supervisionada, a classificação de um conjunto de dados manuscrito e um problema de aprendizado por reforço aplicado a um jogo do gênero Tiro em Primeira Pessoa (FPS). Para jogar o jogo foi utilizada uma adaptação de uma RPQ. Para demonstrar os resultados do trabalho é utilizada uma plataforma baseada no jogo Doom, onde os autores apresentam que a arquitetura utilizada

é capaz de resolver tanto o problema de aprendizagem supervisionada e problemas de ARP, onde em um dos ambientes utilizados o agente consegue aprender sem auxílio os objetivos do jogo.

Os autores Li et al. (2018) abordam o tema de GGP para jogos de luta, com a característica corriqueiramente denominada 2.5D, a qual possui a possibilidade de movimento limitado em um eixo de profundidade. O artigo aborda o problema que a ambiguidade de aparições visuais como altura ou profundidade de um personagem do jogo podem causar nas detecções. Considerando a maneira como é tratado o estado atual de um jogo 2.5D, a similaridade entre as ações de um personagem que efetuou um pulo ou um personagem que se deslocou no eixo de profundidade se torna um problema durante a interpretação desta situação pelo algoritmo utilizado. Para solucionar este problema, os autores propõem uma arquitetura de ARP que estende uma rede A3C, à qual se referem como A3C +, que adiciona recursos de ações recorrentes como parte das entradas para a rede.

No trabalho dos autores (DOCKHORN; APELDOORN, 2018), é proposto um novo modelo de agente de aprendizado com foco em GGP. O agente proposto possui a capacidade de aprender jogando um mesmo jogo repetidamente, guardando o melhor resultado obtido através de escolhas randômicas baseadas na melhor escolha anterior para a situação atual. Após efetuar uma escolha, o agente pode adaptar seu comportamento visando obter o melhor *score* possível no momento. O agente primeiro aprende a mecânica do jogo através de técnicas de AM, em seguida extrai conhecimento simbólico baseado em cada uma das mecênicas detectadas. Quando uma nova fase é apresentada ao agente, ele é capaz de revisar seu conhecimento e desenvolver uma nova abordagem.

O trabalho dos autores (JEERIGE; BEIN; VERMA, 2019), apresenta reproduções de pesquisas existentes relacionadas a ARP e jogos. Para os testes foi utilizado o jogo Breakout para o Atari 2600 em que o agente aprende o controle das políticas usando abordagens de ARP. Os autores exploram as técnicas A3C e RPQ, as quais são utilizadas para treinar agentes inteligentes que possuam a capacidade de interagir com o ambiente. Como resultado para a comparação entre os dois algoritmos implementados os autores apresentam que a técnica de se sobressai em relação a RPQ, já que a segunda técnica necessita de uma placa de vídeo poderosa para treinar rapidamente.

Os autores (JUSTESEN et al., 2019) apresentam uma revisão dos avanços recentes recente em relação a AP e GGP. São apresentados os avanços nos modelos de Aprendizado Supervisionado, Não supervisionado, AR, Modelos Híbridos e Modelos Evolutivos. O panorama geral para os gêneros de jogos Arcade, *Racing*, FPS, *Open-World*, *Real-time Strategy*, *Team Sports*, *Text Adventure* é apresentado como um dos objetivos principais do trabalho. Alguns dos trabalhos revisados aplicam o aprendizado supervisionado para imitar comportamentos dos registros do jogo, enquanto outros são baseados em métodos que aprendem um modelo do ambiente.

3.3 Considerações do Capítulo

Através do levantamento do estado da arte realizado, nota-se a variedade de técnicas utilizadas dentro do contexto de GGP, permitindo observar a quantidade de estratégias distintas que podem obter resultados equivalentes ao de um jogador humano, comprovando assim a capacidade de aprendizado destes algoritmos. Nota-se que alguns dos trabalhos utilizam combinações de trabalhos anteriores, mostrando assim a possível árvore de combinações que pode ser feita para obter o resultado esperado do agente. Múltiplos modelos apresentam resultados satisfatórios quando comparados com jogadores humanos pouco experientes ou experientes, mas em todos os modelos apresentados existem jogos com os quais o agente não conseguiu se adaptar adequadamente ou não atingiu pontuações satisfatórias.

4 DESENVOLVIMENTO DO TRABALHO

Este capítulo apresenta o desenvolvimento do agente destinado a GGP. Para isto, a seção 4.1 apresenta as ferramentas escolhidas para a realização deste trabalho; a seção 4.2 apresenta a descrição do protótipo adotado neste trabalho; a seção 4.3 apresenta algumas das características que o agente deve seguir; na seção 4.4 é apresentada a definição do agente proposto; na seção 4.5 é apresentado o modo como as informações fornecidas pela biblioteca Gym são entregues a biblioteca Keras;a seção 4.6 apresenta o sistema de fases utilizado no agente; a seção 4.7 apresenta o modelo de treinamento empregado neste trabalho; na seção 4.8 é apresentado o sistema de controle de memória do agente; a seção 4.9 apresenta o sistema utilizado para a obtenção das pontuações dos jogos; na seção 4.10 é apresentado o algoritmo final do agente; na seção 4.11 são apresentados os critérios de escolha utilizados para cada um dos jogos que o agente jogou; por fim, na seção 4.12 são apresentadas as considerações deste capítulo.

4.1 Definição das Ferramentas e Tecnologias

Para o desenvolvimento do agente proposto, foram realizados testes utilizando o emulador MAME e a biblioteca MameVirtualPlayer. Originalmente foi desenvolvido um protótipo heurístico que fazia uso das informações fornecidas pela biblioteca para jogar o jogo Space Invaders. Entretanto, mesmo que a biblioteca fosse eficiente e simples de utilizar, devido à dificuldade enfrentada em extrair a pontuação das imagens de diferentes jogos, foi feita uma busca por ambientes de emulação que fornecessem de maneira simplificada as pontuações.

A necessidade de obter as pontuações de forma simples para o agente, é gerada pelo contexto atual deste trabalho. Para desenvolver um agente capaz de tomar decisões em jogos distintos, é necessário utilizar informações que estejam disponíveis em todos os jogos selecionados. Neste trabalho, as características que foram utilizadas foram a pontuação e a imagem da tela do jogo, por se tratarem de informações simples presentes na maioria dos jogos.

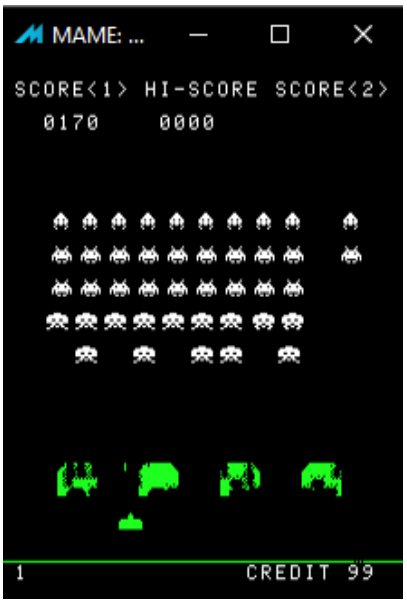
Após uma análise sobre ferramentas alternativas para este trabalho, foi encontrado o ambiente de emulação da biblioteca Gym, o qual fornece as informações necessárias para o desenvolvimento do agente. Assim como planejado no início deste trabalho, para abstrair alguns conceitos de AM, o agente desenvolvido utilizou a biblioteca Keras, a qual já conta com um ambiente para desenvolvimento de redes neurais simplificado.

4.2 Protótipo

Para determinar se o desenvolvimento de um agente inteligente capaz de jogar múltiplos jogos era factível utilizando a biblioteca MameVirtualPlayer e o emulador MAME, foi desenvolvido um agente que fazia uso de funções heurísticas para o jogo Space Inva-

ders, com o objetivo de destruir o maior número de inimigos. Para isto foram levadas em consideração as posições dos inimigos e do jogador. Na Figura 9 encontra-se a captura da tela, no momento no qual o agente foi submetido ao jogo Space Invaders.

Figura 9 – Utilizando o agente heurístico no jogo Space Invaders



Fonte: Elaborado pelo autor.

O desenvolvimento de agente que utilizasse as informações específicas dos jogos selecionados não é uma tarefa difícil, já que a biblioteca MameVirtualPlayer disponibiliza métodos que permitem a extração destas informações. Entretanto, durante o desenvolvimento do agente capaz de jogar múltiplos jogos, foram encontradas dificuldades em padronizar o formato da pontuação, já que cada jogo pode utilizar um formato distinto.

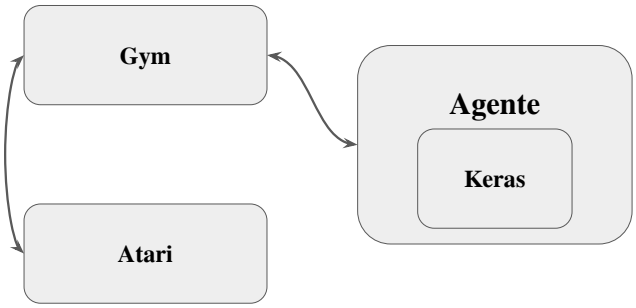
4.3 Definição do Funcionamento do Agente

O objetivo principal deste trabalho é desenvolver dentro do contexto de GGP. Para isto foi desenvolvido um agente que faz uso da comunicação adquirida ao utilizar o Gym e dos métodos adquiridos com Keras apenas fornecendo os *Frames* do jogo desejado. Na Figura 10 é apresentada a posição do agente em relação às demais ferramentas.

Uma parcela considerável dos autores apresentados na Capítulo 3 utilizaram técnicas baseadas em ARP e RPQ, demonstrando a eficácia destes dois modelos em relação ao contexto GGP. Assim como pode ser visto na Figura 11, o agente proposto utiliza a biblioteca Gym para obter os *frames* e suas respectivas informações, e como resposta fornece à biblioteca a ação que decidiu seguir naquele momento.

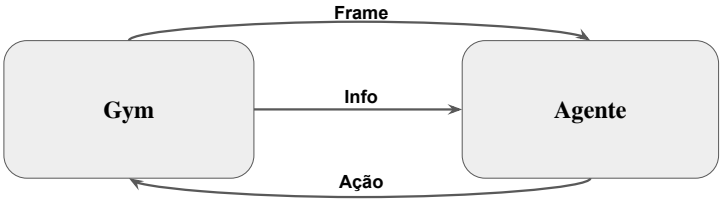
O modelo de comunicação apresentado na Figura 11 fornece ao agente a capacidade de jogar um jogo qualquer, desde que as informações corretas sejam fornecidas, portanto,

Figura 10 – Localização do agente proposto



Fonte: Elaborado pelo autor.

Figura 11 – Comunicação entre o agente e a biblioteca Gym



Fonte: Elaborado pelo autor.

a padronização do formato dos dados recebidos pelo agente, permite o teste em jogos completamente distintos sem a necessidade de fazer adaptações no código fonte do agente.

4.4 Definição da Rede Neural do Agente

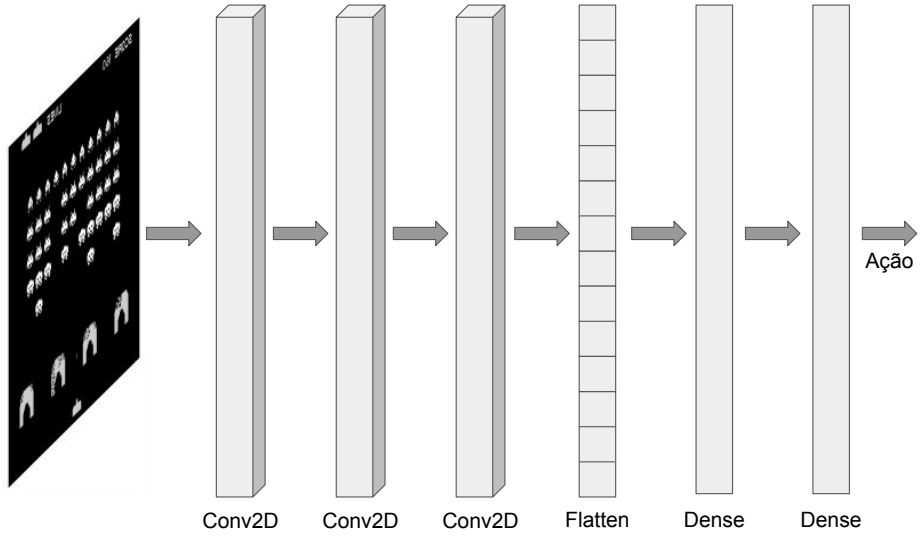
O modelo de rede neural implementado baseia-se no fato de que a entrada fornecida é um *Frame* previamente preparado, ou seja, a captura do momento atual do jogo, o qual é entregue à rede sendo representado por tons de cinza. Além do instante atual do jogo, é fornecido à rede algumas das informações obtidas com a biblioteca Gym, sendo estas a quantidade de vidas restantes e se o *Frame* atual representa um estado de *game over*.

O agente proposto e desenvolvido consiste primeiramente da inserção de camadas de processamento da biblioteca Keras. As camadas utilizadas foram essencialmente:

- **Camadas de convolução:** responsáveis por criar um *Kernel* de convolução que é convoluído com a entrada da camada para produzir um tensor de saídas
- **Camadas niveladoras:** as quais são responsáveis por nivelar a entrada, não afetando o seu tamanho
- **Camadas totalmente conectadas:** todos os neurônios destas camadas são conectados com todos os neurônios das próximas camadas

A representação destas camadas pode ser vista na Figura 12, a qual apresenta as camadas com os mesmos nomes encontrados na biblioteca Keras.

Figura 12 – Definição da rede neural do agente



Fonte: Elaborado pelo autor.

A origem desta configuração de camadas é baseada no sistema utilizado pelos autores Mnih et al. (2015), com a diferença que, neste trabalho, foram utilizadas adaptações que possibilitam a implementação da rede na biblioteca Keras, em conjunto com a possibilidade de mudar a quantidade de neurônios presente na camada de saída, permitindo assim, controlar dinamicamente a quantidade de movimentos de cada jogo.

4.5 Utilização das Bibliotecas Gym e Keras

Para interagir com o ambiente de emulação, o agente desenvolvido utiliza comandos da biblioteca Gym. O uso dessa biblioteca, possibilita obter a imagem da tela e a recompensa(pontos) do ambiente de emulação. Para isto foi necessário primeiramente selecionar o jogo desejado, o que foi realizado da seguinte maneira:

$$env = gym.make('NomeDoJogo - v0')$$

Tendo definido qual jogo executar, para iniciar uma nova partida ou recomeçar um episódio, foi utilizado o comando *reset* da biblioteca Gym. O comando *reset* pode ser utilizado sempre que o jogo atinge um ponto de parada, ou seja, sempre que o agente chegou a um estado de *game over*. Para isto o comando foi utilizado do seguinte modo:

$$env.reset()$$

Para fornecer as ações selecionadas pelo agente para o jogo em execução, basta utilizar o comando *step* e fornecer como argumento o número da ação desejada. Assim como pode ser visto a seguir, a resposta para este comando será a imagem da tela, a recompensa, o indicador de parada do jogo e informações relativas ao jogo selecionado.

$$tela, recompensa, done, info = env.step(ação)$$

Tendo obtido a imagem da tela, o agente a redimensiona para o formato da entrada da rede neural. Neste trabalho o formato de entrada da rede foi definido para (84, 84, 1), onde os dois primeiros números representam as dimensões da imagem, e o terceiro número representa a quantidade de imagens agrupadas. Após submeter a entrada para a rede neural, é obtido como resultado um número inteiro, que corresponde a alguma ação do jogo.

4.6 Fases do Agente

Para oferecer ao agente desenvolvido experiências que permitam que ele compreenda como o jogo ao qual foi submetido funciona, a função de treinamento do agente conta com um sistema de fases, viabilizando o treinamento sem a definição de regras dos jogos. O sistema é composto por três fases, sendo elas:

1. **Observação:** O agente efetua ações randômicas para observar as suas consequências, preenchendo assim a primeira versão da memória.

2. **Exploração:** Após ter efetuado a fase de observação, o agente passa a explorar ações que não se encontram na memória, visando preencher as lacunas criadas por bifurcações nas escolhas de partidas passadas.

3. **Seleção:** Na última fase, o agente passa a selecionar melhor as ações, visando as que oferecem melhor recompensa.

4.7 Treinamento do Agente

Para que a rede desenvolvida tenha algum impacto real na maneira como o agente interage com o ambiente, é necessário primeiro uma fase de treinamento, a qual pode ser vista no pseudocódigo abaixo.

Algoritmo 1: Função de treinamento do agente

```
número de episódios recebe zero;
número máximo de episódios recebe n;
while número de episódios menor que o número máximo de episódios do
    reiniciar o ambiente do jogo;
    while não morreu no jogo do
        verificar possível ação;
        enviar ação para o jogo;
        verificar o resultado desta ação;
        salvar na memória o resultado da ação;
    end
    incrementar o número de episódios;
end
```

A fase de treinamento do agente consiste em um ciclo de iterações no qual inicialmente são testadas ações randômicas. À medida que o número de episódios avança e a quantidade de estados salvos na memória crescem, as decisões do agente passam a apresentar um comportamento mais inteligente e consistente.

4.8 Otimização de Memória

O treinamento do agente necessita de um histórico de ações e resultados, possibilitando que o agente aprenda com os erros cometidos em iterações anteriores. Entretanto o problema aparece quando a memória gasta por este histórico passa a representar o maior consumo do agente. Para solucionar este problema, foi adicionado um parâmetro que permite o controle da quantidade de iterações que o agente pode guardar em sua memória. Esta ação segue o exemplo encontrado no algoritmo a seguir:

Algoritmo 2: Função de otimização de memória

```
if tamanho do histórico igual ao tamanho máximo do histórico then
    remover o estado mais antigo do histórico;
    adicionar o novo estado ao histórico;
end
```

Com esta otimização, esperasse que ao decorrer do treinamento, apenas as ações mais vantajosas continuem na memória. Conforme o agente seleciona suas ações, a memória substitui as ações mais antigas. Portanto, quando o agente passa a escolher as melhores ações, as piores passam a ser apagadas da memória.

4.9 Obtenção das Pontuações

Para medir o desempenho do agente desenvolvido, durante a função de treinamento, este também se encarrega de executar a função responsável por exibir a pontuação obtida em cada um dos jogos. Para obter a pontuação correspondente ao momento atual do jogo, o agente utiliza a biblioteca Gym. Assim como pode ser visto no algoritmo a seguir, enquanto o agente estiver executando a função de treinamento, as pontuações serão exibidas.

Algoritmo 3: Função de obtenção das pontuações

```
while agente estiver na função de treinamento do
    if episódio atual encerrado then
        obter a pontuação do episódio atual;
        exibir a pontuação do episódio atual;
    end
end
```

O sistema de pontuações foi desenvolvido visando a simplicidade, permitindo assim a utilização em todos os jogos selecionados sem a necessidade de nenhuma adaptação no código fonte do agente.

4.10 Funcionamento do agente

Após a definição de como cada parte do agente deveria funcionar, todas elas foram utilizadas em conjunto, onde o resultado foi o desenvolvimento de um agente com a mesma estrutura do algoritmo a seguir:

O agente foi desenvolvido visando o contexto GGP, ou seja, a possibilidade de submetê-lo a qualquer jogo, desde que as informações corretas sejam fornecidas. Para submeter o agente a qualquer ambiente de teste, é necessário apenas que sejam fornecidos o estado atual da tela do jogo em forma de uma imagem representada por tons de cinza e as pontuações. Para otimizar o processo de aprendizagem do agente, no algoritmo anterior, é fornecida a quantidade de ações possíveis do jogo, permitindo o afunilamento das decisões do agente.

Algoritmo 4: Funcionamento do agente

```
receber o ambiente para o treinamento;
extrair a quantidade de ações;
declarar a rede neural;
receber o número máximo de episódios;
definir o número do episódio para zero;
while número de episódios menor que o número máximo de episódios do
    reiniciar o ambiente do jogo;
    definir a fase atual;
    while não morreu no jogo do
        obter a imagem da tela;
        obter a recompensa;
        fornecer a imagem da tela e a recompensa para a rede neural;
        salvar a imagem da tela e a recompensa na memória;
        if fase atual observação then
            selecionar uma ação aleatório;
        else if fase atual exploração then
            selecionar uma ação que não foi explorada ainda;
        else
            selecionar a melhor ação baseada na memória;
        end
        enviar ação para o jogo;
    end
    obter a pontuação;
    exibir a pontuação;
    incrementar o numero de episódios;
end
salvar parâmetros da rede neural;
```

4.11 Jogos Selecionados Para Teste

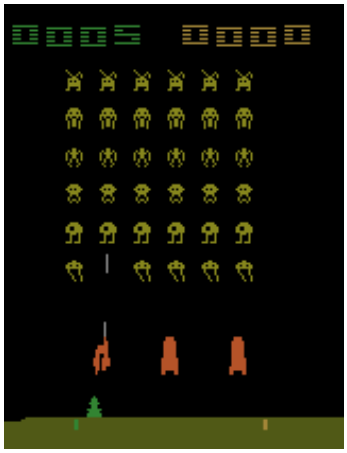
Utilizando a Biblioteca Gym, foram selecionados com base em suas características quatro jogos distintos, sendo eles respectivamente Space Invaders, Breakout, Demon Attack e Kung Fu Master. A principal características dos jogos selecionados foi a padronização das informações fornecidas pelo ambiente de emulação, onde em cada um dos jogos foi possível obter a imagem da tela, pontuação atual e a informação correspondente ao *game over*.

4.11.1 Space Invaders

O jogo Space Invaders foi o primeiro jogo a ser selecionado por este trabalho, por se tratar de um jogo com poucos comandos, mas com grau de dificuldade elevado e amplamente conhecido. A Figura 13 apresenta uma imagem da tela obtida durante a fase de treinamento do agente.

Trata-se de um jogo no qual o jogador controla uma nave e deve destruir os inimigos

Figura 13 – Tela do jogo Space Invaders



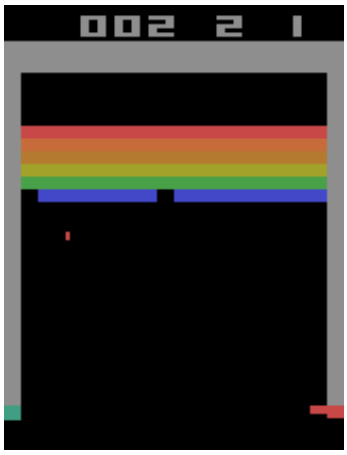
Fonte: Elaborado pelo autor.

que se encontram na parte superior da tela, para isto, na biblioteca Gym, o jogador conta com a lista de comandos *FIRE*, *RIGHT*, *LEFT*, *RIGHTFIRE* e *LEFTFIRE*.

4.11.2 Breakout

A seleção do jogo Breakout, foi baseada na escolha do jogo Space Invaders. Foi selecionado um jogo com a quantidade de comandos similar, entretanto com proposta completamente diferente, possibilitando assim uma comparação entre os desempenhos obtidos em cada um dos jogos. Na Figura 14 encontra-se uma captura de tela obtida durante a fase de treinamento do agente.

Figura 14 – Tela do jogo Breakout



Fonte: Elaborado pelo autor.

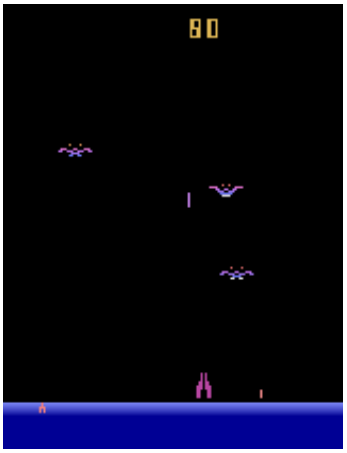
O jogo Breakout trata-se de um clássico de *arcade*, no qual o jogador controla o

movimento de uma plataforma na parte inferior da tela, e deve impedir que a bola caia. A bola interage rebatendo nas paredes e destruindo os blocos que se encontram na parte superior da tela. Neste jogo, o jogador tem a sua disposição os seguintes comandos na biblioteca Gym: *FIRE*, *RIGHT* e *LEFT*.

4.11.3 Demon Attack

A seleção do Jogo Demon Attack, foi seguindo o principio de que ele é um jogo similar ao jogo Space Invaders, tanto no objetivo do jogo, quanto nos comandos disponíveis ao jogador.

Figura 15 – Tela do jogo Demon Attack



Fonte: Elaborado pelo autor.

O Demon Attack é um jogo que lembra em muito seu concorrente, Space Invaders, pois trata-se de um jogo no qual o jogador controla uma nave localizada na parte inferior da tela, e deve combater os adversários encontrados na parte superior. Na biblioteca Gym, os comandos disponíveis ao jogados são: *FIRE*, *RIGHT*, *LEFT*, *RIGHTFIRE* e *LEFTFIRE*.

4.11.4 Kung Fu Master

A escolha do jogo Kung Fu Master foi baseada no fato de que ele é completamente diferente dos demais jogos selecionados, apresentando uma perspectiva diferente, por contar com adversários localizados nas laterais da tela, além de possuir quase três vezes a quantidade de comandos que os demais jogos.

O jogo Kung Fu Master apresenta um ambiente no qual o jogador controla um personagem localizado no centro da parte inferior da tela. Os adversários surgem a partir das laterais da tela e o objetivo do jogo é derrotar todos os adversários em um sistema de níveis. Para isto, o jogador conta com as ações: *UP*, *RIGHT*, *LEFT*, *DOWN*, *DOWNRIGHT*,

Figura 16 – Tela do jogo Kung Fu Master

Fonte: Elaborado pelo autor.

DOWNLEFT, *RIGHTFIRE*, *LEFTFIRE*, *DOWNFIRE*, *UPRIGHTFIRE*, *UPLEFTFIRE*, *DOWNRIGHTFIRE* e *DOWNLEFTFIRE*.

4.12 Considerações do Capítulo

No decorrer deste capítulo foram apresentadas as tecnologias utilizadas no desenvolvimento do protótipo do agente e do agente final, juntamente com a parte essencial do funcionamento do agente. Por fim, foram apresentados os jogos selecionados, e seus respectivos motivos de seleção

5 RESULTADOS

Neste capítulo são apresentados os resultados obtidos com os testes do agente desenvolvido neste trabalho. O computador utilizado para os testes possui um processador i5, 5200U com 2.20 GHz e 8GB de memória RAM DDR3L. Considerando que o agente se comporta de maneira randômica no início do treinamento, e ao final escolhe com maior acurácia suas ações, as pontuações iniciais e finais obtidas em cada um dos jogos podem ser encontradas na Tabela 1.

Tabela 1 – Comparação das pontuações entre o agente e jogadores humanos.

Jogo	Pontuação Inicial	Pontuação Final	Jogador Humano
SpaceInvaders	75	580	1200
Breakout	2	24	26
DemonAttack	100	640	1440
KungFuMaster	400	7500	3400

Fonte: Elaborado pelo autor.

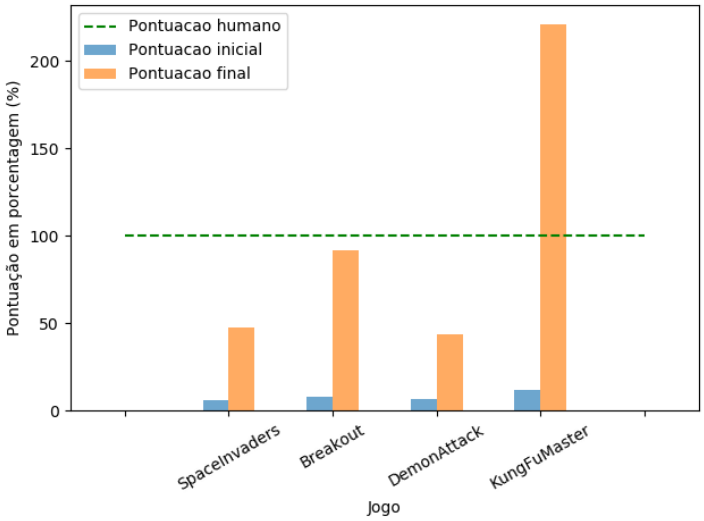
Os dados apresentados na Tabela 1 foram obtidos ao submeter o agente a um treinamento de 1000 episódios em cada um dos jogos, com o tamanho do vetor de memória definido para 80000 posições, onde cada posição ocupa o espaço de 88 bytes, o que levou aproximadamente 20 horas de treinamento para cada um deles. Para comparar adequadamente o desempenho em cada um dos jogos, a Figura 17 apresenta o ganho de desempenho percentual em cada um dos jogos, onde a pontuação humana é representada pelo valor de 100% e as pontuações iniciais e finais são representadas pelo melhoria percentual em relação a pontuação humana.

Dentre as pontuações apresentadas, a pontuação do jogador humano presente na Tabela 1 foi obtida por meio de jogadores casuais que permitiram a coleta de suas pontuações e utilização das mesmas neste trabalho. O valor presente na tabela, é a média da pontuação obtida pelos jogadores em cada um dos jogos selecionados. Todas as pontuações de jogadores humanos, foram obtidas através do emulador MAME.

A Figura 17 mostra um padrão interessante, ao considerar os critérios de seleção de cada jogo, observando a pontuação do agente nos jogos Space Invaders e Demon Attack, os quais possuem propostas similares. Tratam-se de jogos nos quais o jogador controla uma nave e deve destruir os adversários encontrados na parte superior da tela. Nestes jogos, se compararmos a pontuação obtida pelo agente ao de um humano, veremos que ambos os jogos atingiram uma porcentagem similar nas pontuações. No jogo Breakout no qual a proposta é diferente dos jogos anteriores, o agente quase atingiu a pontuação humana. Por fim, no jogo Kung Fu Master, o jogo selecionado por quase não compartilhar semelhanças com os demais jogos, a pontuação atingida pelo agente foi superior a de um jogador humano.

As pontuações obtidas em todos os jogos deste trabalho, poderiam ser comparadas

Figura 17 – Comparação das pontuações obtidas



Fonte: Elaborado pelo autor.

com as obtidas pelos autores Mnih et al. (2015). Entretanto, diferente do presente trabalho, o trabalho do grupo DeepMind não possui um limitador de memória baixo, o que faria com que o presente trabalho necessita-se de uma máquina com aproximadamente 80GB de memória RAM, para conseguir armazenar a devida quantidade de estados e recompensas vivenciados pelo agente. Além do tamanho da memória, seria necessário aumentar a quantidade de episódios na fase de treinamento, o que acarretaria em um gasto de tempo consideravelmente maior.

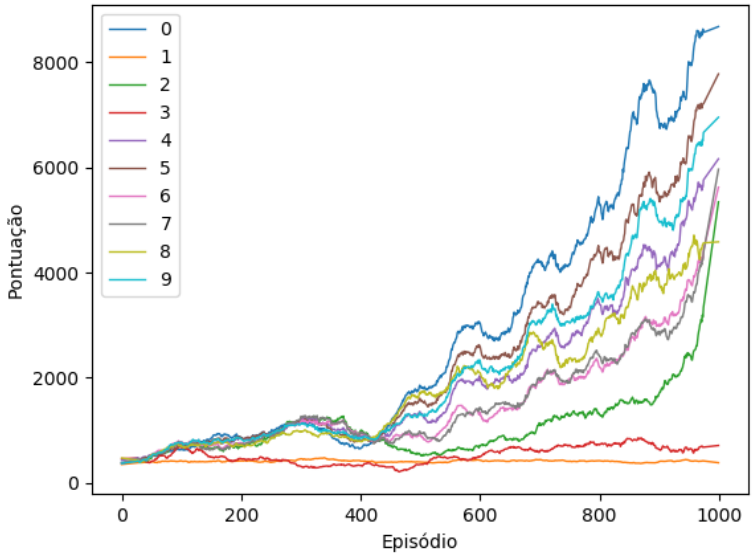
Segundo as informações apresentadas na Tabela 1, o agente foi capaz de determinar quais ações deveria tomar em cada um dos jogos. A Figura 17 mostra que em todos os jogos testados, as pontuações obtidas ao final do treinamento melhoraram significativamente em relação ao inicio do treinamento. Mesmo não tendo obtido a mesma pontuação de um humano na maioria dos jogos testados, o agente conseguiu inferir informações sobre o ambiente no qual se encontra.

Para medir a evolução do agente, foi selecionado o jogo Kung Fu Master, por se tratar do jogo mais complexo dos jogos testados. Assim como pode ser visto na Figura 18, em oito execuções as pontuações obtidas pelo agente encontram-se em uma região similar. O que proporciona esta característica é que um dos pontos chave do jogo Kung Fu Master. O jogo consiste em atacar os adversários vindos das laterais da tela, mas golpes desferidos na direção errada não afetam o adversário.

Em duas das execuções da fase de treinamento, o agente falhou em perceber que não poderia desferir golpes estando de costas para o adversário, nas demais execuções, quando esta característica foi notada pelo agente, as pontuações foram apresentando

avanço em cada episódio sequente. Outra característica presente neste jogo, é que para terminal uma fase, é necessário derrotar um chefe, mas o agente desenvolvido, ficar no inicio do jogo, atacando adversários e obtendo pontos, sem efetivamente avançar no jogo, o que permite a obtenção de pontuações elevadas, sem a necessidade de enfrentar desafios maiores no decorrer da partida atual.

Figura 18 – Evolução do agente no jogo Kung Fu Master



Fonte: Elaborado pelo autor.

6 CONSIDERAÇÕES FINAIS

Com base no levantamento do estado da arte, constatou-se que a combinação de técnicas de AM para a implementação de agentes destinados a GGP é algo factível. Nota-se também, na literatura, a constante aparição de agentes ligados a AR. Como visto nos resultados apresentados na seção 3.2, cada agente apresentado obteve resultados superiores aos de um jogador humano em algum dos jogos testados. Entretanto, mesmo estes agentes apresentavam situações nas quais não conseguiam compreender adequadamente os jogos e consequentemente não conseguiam obter resultados satisfatórios.

Durante o desenvolvimento do agente proposto, as pontuações foram salvas, proporcionando assim um modo de medir o crescimento ao longo do treinamento. O agente desenvolvido conseguiu jogar todos os jogos que possuíam um formato de entrada adequado, ou seja, forneciam informações como o *frame* atual, a pontuação obtida, e se a partida em execução já foi concluída.

No decorrer desenvolvimento do agente, notamos que a quantidade de partidas jogadas de cada jogo impacta diretamente em seu desempenho no mesmo. Entretanto, o fator que gerou maior impacto no desempenho dos jogos foi o tamanho da memória. Quanto maior a quantidade de eventos passados que o agente pode se recordar, mais cedo o agente atingia uma pontuação elevada no jogo. O problema desta abordagem é que, ao aumentar demasiadamente o tamanho da memória, passa-se a exigir demais do espaço utilizado na memória temporária do computador em que o agente está sendo executado. A otimização de memória permite a utilização do agente em ambientes que possuem uma parte considerável da sua capacidade destinada a execução dos jogos.

Pelo fato de que o agente foi desenvolvido com foco em jogar múltiplos jogos, com a habilidade de compreender os quais a que for submetido, o ambiente no qual ele se encontra poderá ser eventualmente alterado. Assim, para trabalhos futuros, propõe-se a adaptação do agente para utilizá-lo como parte de um jogo, como por exemplo um adversário (segundo jogador) ao invés do jogador principal. Dessa forma, o agente terá a oportunidade de aprender não apenas a sobrepujar os algoritmos dos agentes do jogo, mas o estilo do jogador humano com o qual ele estiver interagindo.

REFERÊNCIAS

ALPHASTAR. Alphastar: Mastering the real-time strategy game starcraft ii. **DeepMind**, 2019. Disponível em: <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>. Acesso em: 12.05.2019. Citado 2 vezes nas páginas 23 e 24.

BENGIO, Y. et al. Learning deep architectures for ai. **Foundations and trends® in Machine Learning**, Now Publishers, Inc., v. 2, n. 1, p. 1–127, 2009. Citado na página 30.

BOWLING, M. et al. Machine learning and games. **Machine Learning**, v. 63, n. 3, p. 211–215, Jun 2006. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/s10994-006-8919-x>. Citado na página 24.

BROWNE, C. et al. Foundations of Digital Archæoludology. **arXiv preprint arXiv:1905.13516**, 2019. Disponível em: <https://arxiv.org/pdf/1905.13516.pdf>. Acesso em: 12.05.2019. Citado na página 27.

DEEPMIND (Ed.). **DeepMind**. [S.l.], 2019. Disponível em: <https://deepmind.com/about/>. Acesso em: 10.05.2019. Citado na página 23.

DOCKHORN, A.; APELDOORN, D. Forward model approximation for general video game learning. In: **2018 IEEE Conference on Computational Intelligence and Games (CIG)**. [S.l.: s.n.], 2018. p. 1–8. ISSN 2325-4289. Citado 2 vezes nas páginas 27 e 39.

DORA, R. de O. **Uma biblioteca para a criação de jogadores virtuais em emuladores**. [S.l.], 2018. Disponível em: <http://dspace.unipampa.edu.br:8080/jspui/handle/riu/3699>. Acesso em: 10.04.2019. Citado na página 34.

FÜRNKRANZ, J. Machine learning in games: A survey. **Machines that learn to play games**, Huntington, NY: Nova Science Publishers, p. 11–59, 2001. Citado na página 29.

GALWAY, L.; CHARLES, D.; BLACK, M. Machine learning in digital games: a survey. **Artificial Intelligence Review**, v. 29, n. 2, p. 123–161, Apr 2008. ISSN 1573-7462. Disponível em: <https://doi.org/10.1007/s10462-009-9112-y>. Citado na página 29.

GUO, X. et al. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In: GHAHRAMANI, Z. et al. (Ed.). **Advances in Neural Information Processing Systems 27**. Curran Associates, Inc., 2014. p. 3338–3346. Disponível em: <http://papers.nips.cc/paper/5421-deep-learning-for-real-time-atari-game-play-using-offline-monte-carlo-tree-search-planning.pdf>. Citado na página 38.

GYM (Ed.). **Gym Documentation**. [S.l.], 2019. Disponível em: <https://gym.openai.com/docs/>. Acesso em: 09.11.2019. Citado na página 34.

JEERIGE, A.; BEIN, D.; VERMA, A. Comparison of deep reinforcement learning approaches for intelligent game playing. In: **2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)**. [S.l.: s.n.], 2019. p. 0366–0371. Citado 2 vezes nas páginas 31 e 39.

JUSTESEN, N. et al. Deep learning for video game playing. **IEEE Transactions on Games**, p. 1–1, 2019. ISSN 2475-1502. Citado 2 vezes nas páginas 30 e 39.

KERAS (Ed.). **Keras Documentation**. [S.l.], 2019. Disponível em: <https://keras.io/>. Acesso em: 19.05.2019. Citado na página 35.

Li, Y. et al. Deep reinforcement learning for playing 2.5d fighting games. In: **2018 25th IEEE International Conference on Image Processing (ICIP)**. [S.l.: s.n.], 2018. p. 3778–3782. ISSN 2381-8549. Citado na página 39.

MAMEdev Team (Ed.). **MAME Documentation**. [S.l.], 2019. Disponível em: <https://docs.mamedev.org/_files/MAME.pdf>. Acesso em: 19.05.2019. Citado na página 28.

MILLINGTON, I.; FUNGE, J. **Artificial intelligence for games**. [S.l.]: CRC Press, 2009. Citado 3 vezes nas páginas 29, 30 e 32.

MNIH, V. et al. Asynchronous methods for deep reinforcement learning. In: **International conference on machine learning**. [S.l.: s.n.], 2016. p. 1928–1937. Citado na página 32.

MNIH, V. et al. Playing atari with deep reinforcement learning. **Computing Research Repository (CoRR)**, abs/1312.5602, 2013. Citado na página 37.

MNIH, V. et al. Human-level control through deep reinforcement learning. **Nature**, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. SN -, v. 518, p. 529 EP –, Feb 2015. Disponível em: <https://doi.org/10.1038/nature14236>. Citado 4 vezes nas páginas 24, 38, 44 e 54.

SERAFIM, P. B. S. et al. Towards playing a 3d first-person shooter game using a classification deep neural network architecture. In: **2017 19th Symposium on Virtual and Augmented Reality (SVR)**. [S.l.: s.n.], 2017. p. 120–126. Citado 2 vezes nas páginas 24 e 38.

SHENG, X.; THUENTE, D. Contextual decision making in general game playing. In: **2011 IEEE 23rd International Conference on Tools with Artificial Intelligence**. [S.l.: s.n.], 2011. p. 679–684. ISSN 2375-0197. Citado 2 vezes nas páginas 27 e 37.

SINGAL, H.; AGGARWAL, P.; DUTT, V. Modeling decisions in games using reinforcement learning. In: **2017 International Conference on Machine Learning and Data Science (MLDS)**. [S.l.: s.n.], 2017. p. 98–105. Citado na página 38.

TORRADO, R. R. et al. Deep reinforcement learning for general video game ai. In: **2018 IEEE Conference on Computational Intelligence and Games (CIG)**. [S.l.: s.n.], 2018. p. 1–8. ISSN 2325-4289. Citado na página 31.

Apêndices

APÊNDICE A – COMO UTILIZAR O AGENTE

1 - Clone o repositório pelo link:

```
$ git clone https://github.com/LucasAntunesdeAlmeida/OpenGGP.git
```

2 - Instale as dependencias:

```
$ pip install -r requirements.txt
```

3 - Para obter informações de como utilizar o agente:

```
$ python3 main.py -h
```

ÍNDICE