IA Player de Games

Guilherme M. Tesch
Engenharia de Computação
Fundação Hermínio Ometto (FHO)
Araras, SP, Brasil
guilherme.tesch@alunos.fho.edu.br
RA: 113800

Samuel A. L. Lucatelli
Engenharia de Computação
Fundação Hermínio Ometto (FHO)
Araras, SP, Brasil
salucatelli@alunos.fho.edu.br
RA: 113114

I. Introdução

Os jogos eletrônicos são amplamente utilizados como aplicação base de pesquisas para o aprendizado por reforço. Esta opção traz um retorno visual intuitivo da aprendizagem após cada tentativa e esclarece as ações que geram recompensas ou punições. O ambiente de jogo escolhido foi Super Mario World, um dos jogos mais icônicos da plataforma Super Nintendo. A escolha se baseia no fato de que o jogo oferece uma dinâmica de desafios simples, mas suficientes para testar a capacidade do agente em aprender comportamentos complexos, como navegação e resolução de obstáculos.

O seguinte projeto tem como objetivo o desenvolvimento de um agente de aprendizado por reforço para jogar Super Mario World utilizando o Deep Q-Network (DQN), com o intuito de explorar e aplicar conceitos fundamentais da área, como redes neurais profundas, aprendizado por reforço e algoritmos de otimização. Em relação com os estudos desenvolvidos em aula, temos o objetivo específico de explorar conceitos de IA no desenvolvimento do seguinte projeto.

II. Trabalhos Relacionados

Diversos estudos anteriores abordaram a aplicação de métodos de aprendizado por reforço profundo em jogos, especialmente com arquiteturas de Deep Q-Network (DQN), fornecendo base teórica e metodológica para o nosso projeto.

Um trabalho relevante é o TCC de Jonatan Amaral da Silva (2019), que desenvolve um agente para jogar Super Mario utilizando técnicas de aprendizado por reforço. No estudo, o autor emprega a extração de dados do jogo via emulador para gerar estados e utiliza recompensas baseadas no progresso do personagem, o que é conceitualmente muito semelhante ao nosso método de coleta de estado simbólico por meio do BizHawk e scripts Lua. Este TCC demonstra ainda os desafios de sincronização entre o ambiente do jogo e o agente, reforçando a importância de um pipeline robusto de comunicação entre emulador e modelo de IA.

Além disso, o trabalho seminal de Mnih et al. (2013) intitulado "Playing Atari with Deep Reinforcement Learning" introduziu o uso de DQN com experiência de replay e de uma rede alvo (target network), componentes centrais do nosso agente. A arquitetura original do DQN proposta por Mnih et al. é a inspiração para o nosso agente, adaptada para estados simbólicos ao invés de imagens.

Outros autores exploraram variações do DQN em ambientes semelhantes: por exemplo, Moreno-Vera (2019) propôs uma versão com rede recorrente ("Deep Recurrent Double Q-Learning") para jogos de Atari, argumentando que as redes LSTM podem capturar dependências temporais mais longas. Embora não utilizemos redes recorrentes em nosso agente, esse trabalho reforça a ideia de que diferentes arquiteturas de rede podem trazer ganhos significativos dependendo da estrutura de observação do estado — algo que pode ser uma extensão futura para nosso projeto.

Já Jain (2023), no estudo "Ramario: Experimental Approach to Reptile Algorithm – Reinforcement Learning for Mario", explorou não apenas DQN, mas também algoritmos de meta-aprendizagem (como o Reptile) e PPO aplicados ao Mario. Este trabalho mostra a viabilidade de treinar agentes em ambientes de jogos clássicos usando estados que não são necessariamente visuais (por exemplo, RAM ou variáveis internas), o que reforça a nossa escolha de utilizar variáveis simbólicas extraídas do emulador como entrada para a rede neural.

Em síntese, os trabalhos relacionados destacam tanto a fidelidade do DQN como agente de tomada de decisão em jogos, quanto a flexibilidade de diferentes representações de estado (visuais, memória interna, input simbólico). Nossa abordagem se insere nesse contexto, combinando a robustez do DQN com uma representação simbólica leve, implementada via BizHawk e scripts Lua, permitindo aprendizado eficiente e sincronizado sem depender exclusivamente de processamento de imagens.

III. Metodologia

Para este projeto, foi selecionada a abordagem de Aprendizado por Reforço, mais especificamente o algoritmo Deep Q-Network (DQN). Essa escolha se justifica por tratar-se de um problema de tomada de decisão sequencial em um ambiente dinâmico, o jogo Super Mario World. O agente aprende interagindo com o ambiente, recebendo recompensas positivas ao progredir na fase e penalizações quando comete erros (como morrer ou perder vida).

O desenvolvimento será realizado em um ambiente virtual Python 3.12.10, com suporte à principal biblioteca de aprendizado de máquina, o PyTorch (criação e treinamento da rede neural responsável por interpretar os estados do jogo

e tomar ações). Além do ambiente python, também será usado o BizHawk, um emulador para executar uma ROM de Super Mario World. O BizHawk possui suporte a scripts na linguagem Lua internos, que serão utilizados para comunicação do emulador com a rede neural.

O projeto será executado em uma máquina com processador AMD Ryzen 7, e GPU NVIDIA GTX. O pipeline de execução será:

Captura do estado atual do jogo;

Passagem do estado pela rede neural;

Execução da ação escolhida;

Cálculo da recompensa e armazenamento da experiência no replay buffer;

Atualização periódica da rede neural.

A modelagem do sistema foi elaborada com base na arquitetura clássica de aprendizado por reforço, composta por dois principais elementos: o agente inteligente e o ambiente (o jogo Super Mario World). O agente interage continuamente com o ambiente, observando o estado atual, executando ações e recebendo recompensas, com o objetivo de maximizar a soma total de recompensas ao longo do tempo.

A. *Estrutura do Agente*

O agente foi modelado utilizando o algoritmo Deep Q-Network (DQN), que combina Q-Learning com uma rede neural profunda responsável por aproximar a função ação-valor Q(s,a) Q(s, a) Q(s,a). No estado atual do projeto, diferentemente de abordagens clássicas baseadas em imagens do jogo (frames de vídeo), a representação do estado foi simplificada para um conjunto de variáveis simbólicas extraídas diretamente da memória do emulador via script Lua. Essa decisão reduz drasticamente a complexidade computacional e permite um aprendizado mais rápido nas etapas iniciais do desenvolvimento.

A rede neural recebe como entrada um vetor de atributos contendo:

posição horizontal do Mario (marioX)

posição vertical (marioY)

pontuação atual (score)

quantidade de vidas restantes

indicador binário de morte (0 = vivo, 1 = morto)

A arquitetura da rede é composta inteiramente por camadas densas (fully-connected):

Entrada: vetor com 5 características representando o estado atual.

Camadas ocultas: duas ou três camadas densas com funções de ativação ReLU, responsáveis por aprender relações não lineares entre o estado do jogo e o impacto esperado de cada ação.

Saída: vetor com os valores Q estimados para cada ação possível do agente (ex.: mover à direita, pular, correr, ficar parado, etc.).

Durante o treinamento, o agente segue uma política e-greedy, começando com alto nível de exploração (e = 1.0) e reduzindo gradualmente até e= 0.1, favorecendo ações mais lucrativas conforme o aprendizado progride.

B. *Estrutura do Ambiente*

O ambiente é fornecido pelo emulador BizHawk, que executa a ROM do Super Mario World e se comunica com o agente por meio de scripts Lua. O script extrai o estado atual da memória do jogo, salva essas informações em arquivo CSV e aplica no jogo as ações escolhidas pelo agente.

A função de recompensa considera:

Progresso horizontal do personagem (avanço em marioX $\rightarrow$ recompensa positiva)

Penalização por morte ou queda (recompensa negativa alta)

Bônus por completar a fase (recompensa positiva adicional)

IV. Fluxo de Aprendizado

O processo segue as etapas típicas do DQN:

O script Lua captura o estado atual do jogo e salva em game_state.csv.

O agente em Python lê o estado, processa na rede neural e escolhe uma ação.

A ação é escrita em action.csv.

O Lua lê a ação e a aplica no BizHawk.

O ambiente retorna um novo estado e uma recompensa.

A experiência (s, a, r, s) é armazenada no replay buffer.

Periodicamente, minibatches são usados para treinar a rede neural.

A target network é atualizada a cada N passos para aumentar a estabilidade.

V. Resultados

Após a realização de diversos experimentos, foi desenvolvido um ecossistema funcional capaz de executar simultaneamente um script em Python — responsável pela inferência da rede neural gerada com o framework PyTorch — e o emulador BizHawk, configurado para carregar um script em Lua. Esse script atua na comunicação entre o ambiente de jogo e a inteligência artificial, enviando os estados coletados ao modelo e recebendo suas ações previstas. A troca de informações foi implementada por meio de dois arquivos no formato .csv, garantindo simplicidade, compatibilidade e fácil depuração durante o desenvolvimento.

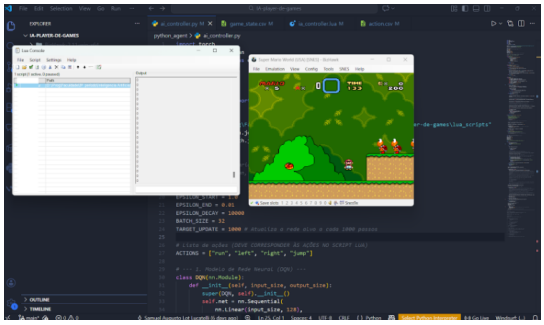

Fig. 1: Exemplo do Funcionamento.

Apesar do funcionamento adequado dessa arquitetura híbrida, não foi possível estabelecer um *loop* completo de treinamento. Especificamente, ocorreram falhas ao tentar reiniciar a fase automaticamente ao término do nível jogado.

Como consequência, o personagem permanece no menu inicial aguardando uma ação do jogador, impedindo a continuidade da interação. Paralelamente, a rede neural entra em estado ocioso, permanecendo à espera de novos estados vindos do ambiente, o que inviabiliza o processo contínuo de coleta de experiências e atualização dos parâmetros do modelo.

Essas limitações impactaram diretamente a implementação de um ciclo de aprendizado por reforço completo, uma vez que o agente depende da capacidade de reiniciar o ambiente inúmeras vezes para acumular episodicamente recompensas, estados e ações. Assim, apesar da comunicação entre Python, BizHawk e Lua funcionar de forma correta, a impossibilidade de reinicialização automática comprometeu a execução integral do treinamento.

VI. Conclusões

O desenvolvimento deste projeto permitiu analisar, na prática, os principais desafios envolvidos na integração entre algoritmos de aprendizado por reforço e ambientes de jogos clássicos executados em emuladores. Foi possível construir um ecossistema funcional no qual uma rede neural, implementada em Python utilizando o framework PyTorch, comunica-se em tempo real com o emulador BizHawk por meio de scripts em Lua. Essa arquitetura possibilitou a captura contínua dos estados do jogo e o envio das ações selecionadas pelo agente, estabelecendo uma base adequada para experimentações com o algoritmo Deep Q-Network (DQN).

Apesar desses avanços, algumas limitações impediram a execução de um ciclo de treinamento completo. O principal desafio identificado esteve relacionado ao controle programático do fluxo do jogo, especialmente nos momentos em que o personagem morria ou concluía a fase. Não foi possível reiniciar o nível de forma confiável por meio do script Lua, fazendo com que o agente permanecesse preso no menu inicial após o término da fase. Como consequência, a rede neural deixava de receber novos estados, interrompendo a coleta de experiências necessária para o processo de aprendizado.

Mesmo diante dessas restrições, o projeto demonstrou a viabilidade da proposta e evidenciou a complexidade de aplicar aprendizado por reforço em ambientes reais e não estruturados, como jogos clássicos de console. Ademais, estabeleceu um alicerce importante para trabalhos futuros, que podem incluir aprimoramentos no controle do emulador via Lua, ajustes na função de recompensa e o desenvolvimento de mecanismos mais robustos de sincronização entre o agente e o ambiente.

Conclui-se, portanto, que o projeto representou um avanço significativo na aplicação prática de técnicas modernas de inteligência artificial para a criação de agentes autônomos em jogos, destacando tanto o potencial quanto os desafios envolvidos. Superadas as limitações técnicas identificadas, espera-se que trabalhos futuros permitam a execução completa do ciclo de treinamento e a análise quantitativa da evolução do desempenho do agente ao longo dos episódios.

Referências

J. A. da Silva, "Aprendizado por reforço no ambiente de jogos," Trabalho de Conclusão de Curso, Univ. de Caxias do Sul, 2019. [Online]. Available: https://repositorio.ucs.br/xmlui/bitstream/handle/11338/13649/TCC%20Jonatan%20Amaral%20da%20Silva.pdf

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint*, arXiv:1312.5602, 2013. [Online]. Available: https://arxiv.org/pdf/1312.05602

F. Moreno-Vera, "Performing Deep Recurrent Double Q-Learning for Atari Games," *arXiv preprint*, arXiv:1908.06040, 2019. [Online]. Available: https://arxiv.org/pdf/1908.06040

S. Jain, "RAMario: Experimental Approach to Reptile Algorithm – Reinforcement Learning for Mario," *arXiv preprint*, arXiv:2305.09655, 2023. [Online]. Available: https://arxiv.org/abs/2305.09655

W. Celes, L. H. Figueiredo and R. Ierusalimschy, "A Linguagem Lua e suas Aplicações em Jogos," in *WJogos 2004 – Workshop de Jogos*, Salvador, 2004. [Online]. Available: https://www.lua.org/doc/wjogos04.pdf

A. Paszke and M. Towers, "Reinforcement Learning (DQN) Tutorial," *PyTorch Tutorials*, 2017 (updated 2025). [Online]. Available: https://docs.pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

TASVideos, "BizHawk Lua Functions Documentation," 2025. [Online]. Available: https://tasvideos.org/Bizhawk/LuaFunctions

R. Ierusalimschy, L. H. Figueiredo and W. Celes, *Lua 5.1 Reference Manual*, Rio de Janeiro: PUC-Rio, 2014. [Online]. Available: https://www.lua.org/manual/5.1/pt/manual.html