Eficácia da Otimização de Parâmetros do Proximal Policy Optimization para Agentes em um Jogo Digital: Um Estudo Comparativo

Cristhian S. Minoves[1], **André R. da Cruz**[1]

[1]Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Belo Horizonte, MG – Brasil

cminoves26@gmail.com, dacruz@cefetmg.br

Abstract. *The role of autonomous non-human characters has become crucial with the growing demand for immersive environments in digital games. However, optimizing the configuration of these intelligent agents presents a complex challenge for developers, given the intrinsic nature of their models and the vast number of parameters involved. This work aims to compare the effectiveness of two parameter tuning heuristics: one based on Bayesian optimization via Gaussian Processes and the other on an iterated race procedure using the iRace method. Both heuristics were applied to the parameter tuning of the Proximal Policy Optimization (PPO) technique, a neural network-based approach, aimed at training an agent to play Push the Block. To this end, a computational experiment was conducted. After tuning, the optimized parameter sets, along with the default configuration, were tested over an extended time horizon. The results indicated that the tuning performed by iRace outperformed the other approaches, providing a parameter set that significantly improved the agent's effectiveness.*
Keywords: *Parameter tuning, Digital Games, Proximal Policy Optimization.*

Resumo. *O papel dos personagens autônomos não humanos tem se tornado fundamental com a crescente demanda por ambientes imersivos em jogos digitais. No entanto, a otimização da configuração desses agentes inteligentes apresenta um desafio complexo para os desenvolvedores, dada a intrínseca natureza de seus modelos e o vasto número de parâmetros envolvidos. Este trabalho se propõe a comparar a eficácia de duas heurísticas de sintonia de parâmetros: uma baseada em otimização Bayesiana via Processos Gaussianos e outra em um procedimento de corrida iterado utilizando o método iRace. Ambas as heurísticas foram aplicadas à sintonia de parâmetros da técnica Proximal Policy Optimization (PPO), uma abordagem baseada em redes neurais, visando o treinamento de um agente para jogar Push The Block. Para isso, um experimento computacional foi conduzido. Após a sintonização, os conjuntos de parâmetros otimizados, juntamente com a configuração padrão, foram testados em um horizonte de tempo estendido. Os resultados obtidos indicaram que a sintonia realizada pelo iRace superou as demais abordagens, fornecendo um conjunto de parâmetros que aprimorou significativamente a eficácia do agente.*
Palavras-chave*: Sintonia de parâmetros, Jogos Digitais, Proximal Policy Optimization.*

1. Introdução

A crescente complexidade dos jogos digitais tem proporcionado ambientes cada vez mais imersivos. É cada vez mais comum encontrar títulos com agentes que mimetizam personagens não jogáveis (NPCs). Esses agentes empregam rotinas de Inteligência Artificial (IA) para interagir de forma dinâmica com o ambiente e o jogador [Kishimoto 2004]. Algumas características dos agentes de IA incluem: (i) Percepção: aquisição de dados de entrada, seja diretamente (e.g., distância a um alvo) ou indiretamente (e.g., visão por câmera virtual); (ii) Objetivo: propósito intrínseco que guia suas ações, com recompensas associadas à sua consecução; (iii) Tomada de Decisão e Ação: realização de ações com base em observações e uma política pré-estabelecida, intimamente ligada ao objetivo (e.g., um NPC companheiro que ajuda o protagonista cercado por inimigos); e (iv) Aprendizado e Adaptação: análise dos resultados das interações e ações para determinar se a conduta pode ser aprimorada, baseando-se em recompensas (e.g., um NPC muda sua estratégia ao perceber que confrontar inimigos diretamente é ineficaz) [Roa et al. 2008].

Para que um agente execute sua tarefa de forma eficaz, faz-se necessário determinar as melhores configurações dos parâmetros comportamentais. Esse processo envolve um treinamento computacional que requer a otimização de parâmetros para máxima eficiência. Esse procedimento, denominado sintonia de hiperparâmetros [Pellicer 2020], realiza testes sequenciais para ajustar os valores com base em indicadores de qualidade para melhorar o desempenho do agente.

A sintonia iterativamente busca uma configuração ideal de um modelo, ajustando valores paramétricos ao otimizar um objetivo específico. Isso pode envolver a maximização de um resultado desejado ou a minimização de um erro [Toal et al. 2008]. A seleção dos parâmetros a serem ajustados, a metodologia de otimização e as métricas de desempenho são cruciais para o sucesso desse processo, e é fundamental em diversas áreas para o desenvolvimento de modelos e sistemas precisos e eficientes [Probst et al. 2019]. A realização inadequada desse processo pode resultar não convergência, sobreajuste ou subajuste, comprometendo o desempenho do agente.

Este trabalho propõe a sintonia de parâmetros do Proximal Policy Optimization (PPO) [Schulman et al. 2017], uma técnica baseada em redes neurais para o treinamento de um agente no jogo *Push The Block*, um benchmark ML-Agents da Unity [Juliani et al. 2020]. Para isso, duas heurísticas de sintonização são empregadas: uma baseada em Otimização Bayesiana (OB) via Processos Gaussianos [Rana et al. 2017] e outra em um procedimento de corrida iterado utilizando o método iRace [López-Ibáñez et al. 2016]. Durante o ajuste, maximiza-se a recompensa média obtida pelo agente treinado. Após a sintonia, um experimento de validação é conduzido, avaliando os parâmetros otimizados pela OB e iRace, juntamente com a configuração padrão do ML-Agents, em 60 execuções em um horizonte de tempo estendido. As amostras de recompensas médias foram analisadas estatisticamente para determinar qual sintonizador resultou no melhor conjunto de parâmetros do PPO para o treinamento do agente.

O restante deste trabalho está organizado da seguinte forma: a Seção 2 apresenta uma revisão bibliográfica; a Seção 3 descreve a metodologia utilizada; a Seção 4 exibe e discute os resultados; por fim, a Seção 5 conclui o trabalho e sugere futuras pesquisas.

2. Trabalhos Relacionados

Diversos trabalhos sintonizaram os parâmetros de agentes para aprimorar o comportamento deles. Por exemplo, [Patel et al. 2011] focou na técnica de aprendizado por reforço Q-learning para desenvolver comportamentos dinâmicos e inteligentes. Os autores argumentam que a IA tradicional em jogos, baseada em rotinas pré-programadas, é limitada e consome muito tempo. Eles propuseram o Q-learning como uma alternativa eficiente e online para melhorar o comportamento dos bots. Também foi investigado como os bots poderiam aprender comportamentos básicos e como o conhecimento adquirido em modelos abstratos poderia ser transferido para ambientes de jogo mais detalhados.

O trabalho [Adil et al. 2017] propôs o treinamento de um agente competitivo para o jogo Doom utilizando uma combinação de Rede Neural Convolucional e Q-learning. Neste, ilustrou-se como o agente pode tomar decisões baseadas nos pixels brutos da tela, simulando a percepção humana. O agente foi avaliado no cenário básico pela plataforma VizDoom e os resultados experimentais mostraram que ele superou jogadores humanos médios e agentes de IA internos do jogo, mesmo com um conjunto limitado de ações.

Em [Lucas et al. 2019], os autores utilizaram uma versão do jogo Planet Wars como um ambiente para avaliar agentes de IA baseados em planejamento estatístico e na otimização de hiperparâmetros em cenários ruidosos. O estudo principal buscou otimizar um agente de jogo, comparando o desempenho do N-Tuple Bandit Evolutionary Algorithm (que se mostrou superior no estudo) com outros otimizadores, como o SMAC.

O artigo [Lai et al. 2019] elaborou um ambiente de tiro em terceira pessoa no Unity ML-Agents para treinar agentes utilizando aprendizagem por reforço profundo (PPO e Actor-Critic). Demonstrou-se que, através do DRL visual, os agentes aprendem habilidades complexas como buscar inimigos e suprimentos, superando o comportamento de IAs baseadas em máquinas de estado. Concluiu-se que o Unity ML-Agents é uma plataforma adequada para DRL, permitindo o desenvolvimento de agentes melhores.

Em [Savid et al. 2023] explorou-se o uso da Unity ML-Agents Toolkit para treinar agentes de carro em um ambiente de simulação, visando a navegação autônoma em pistas de corrida e a evasão de obstáculos, utilizando algoritmos de aprendizado por reforço. Os autores compararam o desempenho de diferentes algoritmos e configurações, identificando o PPO como a abordagem mais eficaz. Adicionalmente, foi investigado o uso de clonagem comportamental como pré-treinamento para o algoritmo PPO, com o objetivo de aprimorar a capacidade dos agentes de evitar obstáculos de forma mais eficiente e demonstrar um desempenho similar ao humano.

Já o [Liu et al. 2025] introduziu um arcabouço de treinamento para agentes baseados em Grandes Modelos de Linguagem (LLMs) no ML-Agents, superando a dependência da engenharia manual de prompts em engenharia de Machine Learning (ML) autônoma. Um agente LLM aprendeu por meio de experimentação interativa usando aprendizado por reforço online, incorporando ajuste fino enriquecido por exploração, RL passo a passo eficiente e um módulo de recompensa específico para ML. Notavelmente, o agente treinado com apenas 7B parâmetros superou um modelo de 671B, demonstrando melhorias contínuas e excelente generalização entre tarefas, o que avança a engenharia de ML autônoma de automação baseada em regras para um aprendizado dinâmico e guiado pela experiência.

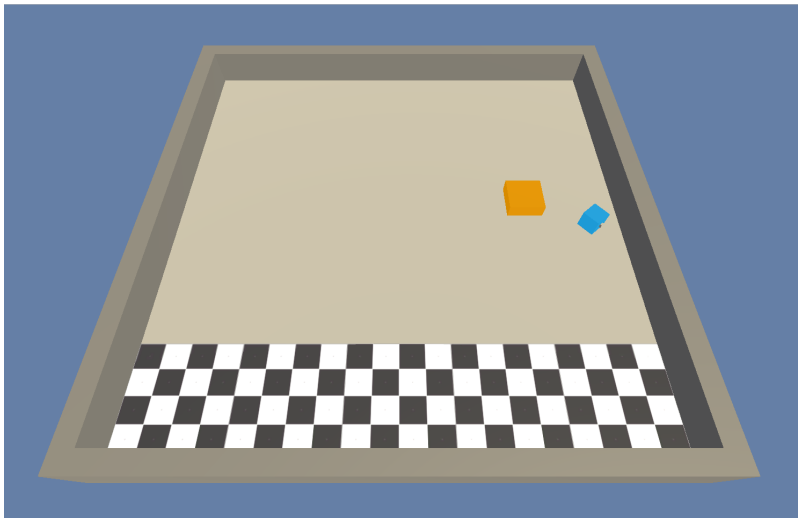

Figura 1. Execução do jogo *Push the Block*

3. Metodologia

Apresenta-se aqui a metodologia adotada. As subseções: 3.1 elucida o *ML-Agents Toolkit*; 3.2 descreve o PPO, a heurística baseada em redes neurais que treina o agente para jogar *Push The Block*; 3.3 aborda a sintonia de parâmetros; 3.4 apresenta o modelo do problema de ajuste de parâmetros; em seguida, 3.5 delineia as heurísticas para solucionar o problema de metaotimização; e, por fim, 3.6 detalha o planejamento experimental.

3.1. *Ml-Agent Toolkit*

O *Unity Machine Learning Agents Toolkit* (ML-Agents) é um ambiente de código aberto desenvolvido pela Unity Technologies que possibilita a criação e o treinamento de agentes inteligentes em diversos ambientes virtuais. Essa ferramenta se destaca por sua integração com o motor de jogos Unity, oferecendo um ambiente rico e flexível para a experimentação e o desenvolvimento de algoritmos de aprendizado de máquina [Lanham 2018].

O ML-Agents possui mais de 17 ambientes, dentre os quais o *Push the Block* (Figura 1) foi escolhido. Neste jogo, o objetivo do agente azul é de levar um cubo laranja partindo de uma localização aleatória até a área quadriculada de chegada, que se alterna entre as laterais da área a cada execução. A fim de realizar o treinamento do agente, deve-se escolher uma técnica de IA, sendo a técnica escolhida neste trabalho o Proximal Policy Optimization (PPO).

3.2. *Proximal Policy Optimization*

Desenvolvido pela OpenAI, o PPO [Schulman et al. 2017] é um algoritmo eficiente de aprendizado por reforço utilizado para treinar agentes em diversas tarefas. O PPO se destaca por sua consistência, estabilidade e capacidade de gerar políticas de alta qualidade.

A principal característica do PPO é a atualização gradual da política do agente através de um mecanismo de *clipping*, que limita a razão entre a nova e a antiga política. Essa abordagem conservadora impede que o agente explore regiões do espaço de políticas que podem ser instáveis ou subótimas [Zhuang et al. 2023].

O PPO funciona em um ciclo iterativo que coleta dados do ambiente, computa a função-objetivo e atualiza a política para maximizar a diferença entre as recompensas real e estimada. As vantagens são (i) estabilidade: o *clipping* garante que as atualizações da política sejam suaves, evitando oscilações e divergências; (ii) eficiência: o PPO converge rapidamente para soluções de alta qualidade; (iii) facilidade de implementação: o algoritmo é relativamente simples de implementar e pode ser aplicado a uma ampla variedade de problemas; e (iv) robustez: o PPO é menos sensível à configuração dos hiperparâmetros em comparação com outros algoritmos [Unity ML-Agents 2024].

3.3. Sintonia de parâmetros

A sintonia de parâmetros é o processo de ajustar as configurações de um modelo ou algoritmo para otimizar seu desempenho em uma tarefa específica. Em um nível acima de abstração, a sintonia de hiperparâmetros é um processo iterativo que sintoniza os parâmetros do procedimento que define os parâmetros de um outro modelo [Bardenet et al. 2013]. Diferentemente dos parâmetros do modelo, que são ajustados durante o treinamento com base nos dados, os hiperparâmetros são definidos antes do treinamento e afetam diretamente como o modelo aprende. Escolhas adequadas de hiperparâmetros podem melhorar significativamente o desempenho de um modelo, enquanto escolhas inadequadas podem levar a resultados subótimos, como sobreajuste ou subajuste.

Além disso, a sintonia de hiperparâmetros é especialmente relevante em algoritmos complexos como redes neurais profundas e métodos de aprendizado por reforço, nos quais o número de hiperparâmetros pode ser grande e suas interações podem ser não triviais. Por exemplo, no PPO, ajustar corretamente a taxa de aprendizado, o fator de *clipping* e o número de épocas de atualização é crucial para garantir a estabilidade do treinamento e maximizar a recompensa média que o agente obtém. Esses ajustes garantem que o agente possa explorar e refinar de forma equilibrada, convergindo para uma política ótima.

3.4. Modelo de otimização

O modelo adotado (Equação 1) possui vetor de decisão $\mathbf{x} = [x_1, \ldots, x_8]$ com 8 hiperparâmetros no espaço viável X do PPO a serem sintonizados, para serem utilizados no treinamento do agente no *Push the Block*. A função-objetivo do tipo caixa preta, $f(\mathbf{x})$ retorna a recompensa média do agente após um treinamento com 30 épocas de observação.

$$\max_{\mathbf{x}} \quad f(\mathbf{x}), \quad \text{sujeito a:} \quad \mathbf{x} \in X \tag{1}$$

As variáveis de decisão com os intervalos são *Max Steps*, $x_1 \in \{5{\cdot}10^5, \ldots, 1{\cdot}10^7\}$, indica o número de passos dados no treinamento; *Batch Size*, $x_2 \in \{512, \ldots 5120\}$, que é o número de experiências em cada iteração do gradiente descendente; *Buffer Size*, $x_3 \in \{2048, \ldots, 409600\}$, determina as experiências de coleta feitas antes de atualizar a política; *Beta*, $x_4 \in [1 \cdot 10^{-4}, \ldots, 1 \cdot 10^{-2}]$, regula a entropia do treino para controlar a exploração e evitar mínimos locais; *Numbers of Layers*, $x_5 \in \{1, 2, 3\}$, é a quantidade de camadas na rede neural; *Hidden Units*, $x_6 \in \{32, \ldots 512\}$, são as unidades dentro das camadas; *Summary Frequency*, $x_7 \in \{1000, \ldots, 3000\}$, é o número de experiências mínimas para a geração de estatísticas de treinamento; e *Time Horizon*, $x_8 \in \{32, \ldots, 2048\}$, é o número de passos executados por cada agente ao executar antes de coletar as experiências para o *buffer*, e interage intimamente com o *Buffer Size*.

3.5. Heurísticas adotadas

A sintonia dos hiperparâmetros do PPO foi realizada aqui pelas heurísticas Otimização Bayesiana via processos gaussianos (OB) e pelo procedimento de corrida iterado iRace.

3.5.1. Otimização Bayesiana

A OB é um método robusto para a otimização de funções complexas, multimodais e/ou custosas e/ou cuja forma analítica é desconhecida. Ela permite modelar a incerteza inerente ao processo de otimização, tornando-a ideal para problemas do tipo caixa preta. A OB reside na construção de um modelo da função-objetivo, utilizando processos gaussianos, que fornece uma descrição probabilística para capturar a incerteza sobre a verdadeira forma. A cada avaliação da função, refina-se o modelo incorporando novas informações sobre a estimativa. A seleção dos pontos a serem avaliados é guiada por uma função de aquisição, que equilibra a exploração de regiões desconhecidas do espaço com a exploração de regiões promissoras [Wang et al. 2023].

As aplicações da OB abrangem áreas como o ajuste de hiperparâmetros em modelos de aprendizado de máquina, o design de experimentos e a otimização de sistemas robóticos. A vantagem se dá pela eficiência obtida com menos avaliações e pelo equilíbrio da exploração do espaço. Devido à flexibilidade, a OB se mostra como um método adequado para resolver problemas de otimização em diversos contextos [Wang et al. 2023].

3.5.2. iRace

O iRace é um método estatístico e computacional que sintoniza os parâmetros com o objetivo de maximizar o desempenho em um conjunto de problemas. Ele explora o espaço de parâmetros de forma eficiente e encontra as configurações promissoras [López-Ibáñez et al. 2016].

Os passos do iRace são [López-Ibáñez et al. 2016]: (i) Inicialização: Define-se um conjunto inicial de configurações de hiperparâmetros. Cada configuração é avaliada utilizando um conjunto de validação; (ii) Avaliação: Calcula-se o desempenho de cada configuração e seleciona-se um subconjunto baseado em critérios de desempenho; (iii) Eliminação: As configurações com desempenho inferior são descartadas. Este processo reduz o número de configurações a serem avaliadas em iterações subsequentes; e (iv) Iteração: O processo de avaliação e eliminação é repetido várias vezes. Em cada iteração, o número de configurações é reduzido, e as melhores configurações são refinadas.

3.6. Planejamento Experimental

O experimento planejado é dividido nas etapas: (i) sintonia dos parâmetros do PPO via OB e iRace; e (ii) comparação da eficácia do agente treinado pelo PPO com os parâmetros sintonizados pelas duas heurísticas e pelo conjunto padrão da ML-Agents.

A OB e o iRace são executados, com configuração padrão, uma única vez na sintonia dos parâmetros e avaliam 100 funções-objetivo para tentar solucionar a Equação 1. Na avaliação de um conjunto de parâmetros, considera-se 30 iterações do *Push the Block*, com 100 episódios cada, para se extrair a recompensa média. O episódio é a

Tabela 1. Hiperparâmetros sintonizados

Parâmetro	Padrão	OB	iRace
Max Steps	50.000	7.964.397	6.599.964
Batch Size	128	225	1.339
Buffer Size	2.048	667.991	273.311
Beta	0,0100	0,0098	0,0089
Hidden Units	256	188	42
Summary Frequency	2.000	1.600	1.251
Time Horizon	64	1.163	1.506
Numbers of Layers	2	3	1

unidade de medida do treinamento, que acaba caso o agente conclua ou atinja o tempo limite da tentativa.

Após a sintonia, realizou-se a validação executando o PPO para treinar o agente para jogar *Push the Block* com o conjunto de parâmetros padrão do ML-Agents, e com aqueles provenientes da otimização via OB e iRace. Para a validação, foram realizadas 60 iterações do *Push the Block* com 100 episódios cada. As amostras de valores de recompensa de cada tratamento (Padrão, OB, iRace) foram extraídas e comparadas, a fim de estabelecer qual conjunto de parâmetros induz os maiores valores. Tais dados foram submetidos aos testes não paramétricos de Friedman e Nemenyi [Derrac et al. 2011] para testar a hipótese nula H_0, com significância $\alpha = 0{,}05$, que afirma que as distribuições das amostras de recompensa média dos três tratamentos são iguais. Se o valor-p do teste de Friedman for menor que α, haverá a rejeição de H_0, indicando diferença estatística entre algum par. Daí o teste de Nemenyi indicará qual(is) par(es) são estatisticamente distintos.

4. Resultados

Após executar a sintonia, foram obtidos os valores de hiperparâmetros na Tabela 1, juntamente com os da ML-Agents. Assim, realizou-se a validação treinando, independentemente, o agente usando os hiperparâmetros determinados. Em seguida, foram extraídos os valores de recompensa média de cada uma das execuções. A Figura 2 ilustra as amostras geradas pelo agente induzido por cada conjunto de parâmetros por meio da combinação de gráficos de dispersão, caixa e violino. A Tabela 2 apresenta estatísticas extraídas das amostras.

Nota-se que a distribuição das recompensas induzidas pelas configurações Padrão e OB são parecidas, e que apenas o terceiro quartil dessas se diferencia um pouco, com uma pequena vantagem para a primeira. Já a amostra cujos parâmetros foram sintonizados iRace evidencia uma clara superioridade em relação às demais.

Este resultado é corroborado pelos testes não paramétricos. Primeiramente, o teste de Friedman foi aplicado e retornou um valor-p de aproximadamente $1{,}26 \cdot 10^{-19}$. Esse valor é muito menor que o nível de significância $\alpha = 0{,}05$ e indica que há uma diferença estatística entre algum par de tratamentos. Ao aplicar o teste de Nemenyi obteve-se os valores-p da Tabela 3. A tabela evidencia que o agente treinado com PPO sintonizado pelo iRace se diferenciou dos demais. Por outro lado, não se detectou diferença entre a sintonia via OB e os parâmetros padrão do ML-Agents.

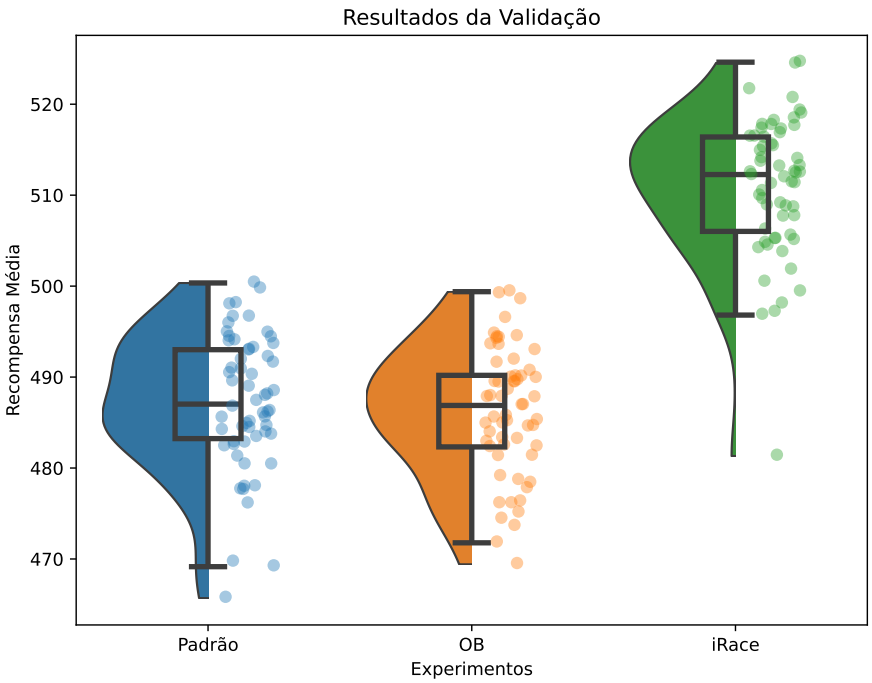

Figura 2. Recompensas dos agentes por sintonizador de parâmetros

Tabela 2. Estatísticas das amostras de recompensas

	Padrão	**OB**	**iRace**
Mínimo	469,1525	471,7764	496,8155
1º Quartil	483,2341	482,3222	506,0171
Mediana	487,0277	486,8803	512,2669
3º Quartil	493,0112	490,1907	516,3977
Máximo	500,3476	499,3889	524,6220

Tabela 3. Valores-p do teste de Nemenyi

Heurísticas	**OB**	**iRace**
Padrão	0,9	0,001
OB	–	0,001

5. Conclusões e Trabalhos Futuros

Este trabalho apresentou uma comparação da capacidade de sintonia das heurísticas OB e iRace para configurar os parâmetros do PPO, que treina um agente que joga *Push the Block*. Tais sintonias foram comparadas com a parametrização padrão do ML-Agents. Após a sintonia, realizou-se a validação com cada conjunto de parâmetros otimizados e com o conjunto padrão para analisar em qual situação o agente acumula mais recompensas. Pelas análises estatísticas, conclui-se que o iRace gerou hiperparâmetros melhores, enquanto não foi observado diferenças entre aqueles gerados pela OB e o conjunto padrão.

Como contribuição, este trabalho empregou e comparou heurísticas que otimizam o conjunto de parâmetros do PPO para realizar o treinamento de agentes de IA na área de jogos digitais. Dessa forma, foi possível melhorar o comportamento do agente dentro do ambiente de um jogo para aumentar a capacidade de acumular recompensas.

Para trabalhos futuros, pretende-se: (i) estudar outras heurísticas de sintonia, como o CMA-ES [Hansen et al. 2003]; (ii) empregar técnicas diferentes de IA, como SAC [Haarnoja et al. 2018], e comparar com os resultados obtidos pelo PPO; e (iii) realizar testes abordando outros jogos que compõem o *Ml-Agent*, pois como cada um dos jogos possuem finalidades e hiperparâmetros diferentes para serem sintonizados, pode ser produtivo analisar como essas diferenças podem impactar na sintonia.

Referências

Adil, K., Jiang, F., Liu, S., Grigorev, A., Gupta, B., and Rho, S. (2017). Training an agent for fps doom game using visual reinforcement learning and vizdoom. *International Journal of Advanced Computer Science and Applications*, 8(12).

Bardenet, R., Brendel, M., Kégl, B., and Sebag, M. (2013). Collaborative hyperparameter tuning. In *International conference on machine learning*, pages 199–207. PMLR.

Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18.

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., and Lange, D. (2020). Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

Kishimoto, A. (2004). Inteligência artificial em jogos eletrônicos. *Academic research about Artificial Intelligence for games*.

Lai, J., Chen, X.-l., and Zhang, X.-Z. (2019). Training an agent for third-person shooter game using unity ml-agents. In *International Conference on Artificial Intelligence and Computing Science. Hangzhou*, pages 317–332.

Lanham, M. (2018). *Learn Unity ML-Agents–Fundamentals of Unity Machine Learning: Incorporate new powerful ML algorithms such as Deep Reinforcement Learning for games*. Packt Publishing Ltd.

Liu, Z., Chai, J., Zhu, X., Tang, S., Ye, R., Zhang, B., Bai, L., and Chen, S. (2025). Ml-agent: Reinforcing llm agents for autonomous machine learning engineering. *arXiv preprint arXiv:2505.23723*.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Lucas, S. M., Liu, J., Bravi, I., Gaina, R. D., Woodward, J., Volz, V., and Perez-Liebana, D. (2019). Efficient evolutionary methods for game agent optimisation: Model-based is best. *arXiv preprint arXiv:1901.00723*.

Patel, P. G., Carver, N., and Rahimi, S. (2011). Tuning computer gaming agents using q-learning. In *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 581–588. IEEE.

Pellicer, L. F. A. O. (2020). *Otimização de hiperparâmetros de modelos machine learning com BarySearch.* PhD thesis, Universidade de São Paulo.

Probst, P., Wright, M. N., and Boulesteix, A.-L. (2019). Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: data mining and knowledge discovery*, 9(3):e1301.

Rana, S., Li, C., Gupta, S., Nguyen, V., and Venkatesh, S. (2017). High dimensional bayesian optimization with elastic gaussian process. In *International conference on machine learning*, pages 2883–2891. PMLR.

Roa, J., Gutiérrez, M., and Stegmayer, G. (2008). Faia: Framework para la enseñanza de agentes en ia. *IE Comunicaciones: Revista Iberoamericana de Informática Educativa*, (8):43–56.

Savid, Y., Mahmoudi, R., Maskeliūnas, R., and Damaševičius, R. (2023). Simulated autonomous driving using reinforcement learning: A comparative study on unity's ml-agents framework. *Information*, 14(5):290.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Toal, D. J., Bressloff, N. W., and Keane, A. J. (2008). Kriging hyperparameter tuning strategies. *AIAA journal*, 46(5):1240–1252.

Unity ML-Agents (2024). Training with proximal policy optimization. `https://github.com/miyamotok0105/unity-ml-agents/blob/master/docs/Training-PPO.md`. Online; acessado em 10/06/2025.

Wang, X., Jin, Y., Schmitt, S., and Olhofer, M. (2023). Recent advances in bayesian optimization. *ACM Computing Surveys*, 55(13s):1–36.

Zhuang, Z., Lei, K., Liu, J., Wang, D., and Guo, Y. (2023). Behavior proximal policy optimization. *arXiv preprint arXiv:2302.11312*.