UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

FERNANDO CÉSAR LISBOA MENDES JUNIOR

MODELAGEM DE AGENTES INTELIGENTES USANDO APRENDIZADO POR REFORÇO EM AMBIENTES UNITY

SÃO LUÍS
2025

FERNANDO CÉSAR LISBOA MENDES JUNIOR

MODELAGEM DE AGENTES INTELIGENTES USANDO APRENDIZADO POR REFORÇO EM AMBIENTES UNITY

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

SÃO LUÍS
2025

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Lisboa Mendes Junior, Fernando César.
MODELAGEM DE AGENTES INTELIGENTES USANDO APRENDIZADO POR REFORÇO EM AMBIENTES UNITY / Fernando César Lisboa Mendes Junior. - 2025.
56 f.

Orientador(a): Tiago Bonini Borchartt.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luis, 2025.

1. Aprendizado Por Reforço. 2. Proximal Policy Optimization. 3. Unity. 4. Jogos Digitais. I. Bonini Borchartt, Tiago. II. Título.

FERNANDO CÉSAR LISBOA MENDES JUNIOR

MODELAGEM DE AGENTES INTELIGENTES USANDO APRENDIZADO POR REFORÇO EM AMBIENTES UNITY

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Trabalho apresentado em SÃO LUÍS, 05 de junho de 2025:

Prof. Dr. Tiago Bonini Borchartt
Orientador

Prof. Dr. Mario Antonio Meireles Teixeira
Examinador

Prof. Dr. Darlan Bruno Pontes Quintanilha
Examinador

SÃO LUÍS

2025

Agradecimentos

A Deus, por ser a fonte da minha existência, por me fortalecer diante das adversidades da vida e por sempre me inspirar na busca pelo conhecimento e pela sabedoria.

Ao meu pai, em especial, que, mesmo sem ter tido acesso à escolarização, reconheceu a importância da educação e se dedicou para que eu pudesse ter uma formação que me conduzisse à universidade. Com o esforço de seu trabalho árduo como operário e, posteriormente, como mecânico, enfrentou o desafio de criar e educar sozinho dois filhos.

À minha mãe, que, apesar dos desafios da vida e da distância, sempre incentivou minha jornada educacional.

À minha irmã, Fernanda, minha melhor amiga, que sempre esteve ao meu lado nos momentos mais difíceis, oferecendo seu amor e cuidado. Sua presença foi essencial, sendo, muitas vezes, minha maior conselheira e contribuindo significativamente para minha formação intelectual e humana.

Ao meu orientador, Professor Doutor Tiago Bonini Borchartt, pela orientação neste trabalho e, sobretudo, por despertar em mim, através de sua dedicação ao ensino, a curiosidade e o interesse pelo conhecimento explorado nesta pesquisa.

Aos docentes do Departamento de Informática, por proporcionarem oportunidades valiosas aos discentes ao longo da formação acadêmica.

Aos meus amigos de graduação, cuja companhia tornou essa jornada mais leve e enriquecedora.

E a todos que, de alguma forma, contribuíram para a realização deste estudo, expresso meus mais sinceros agradecimentos.

RESUMO

Este trabalho tem como objetivo compreender o processo de aplicação dos algoritmos de aprendizado por reforço por meio da implementação em ambientes da Unity. Como metodologia, optou-se por uma pesquisa do tipo exploratória e abordagem quantitativa, desenvolvida em três etapas: análise da literatura associada aos algoritmos de aprendizado por reforço; implementação do algoritmo de aprendizado por reforço PPO (*Proximal Policy Optimization*) em ambientes da Unity; desenvolvimento e avaliação da eficácia e desempenho dos agentes em cenários específicos. Para tanto, aplicou-se AR em diferentes tarefas, visando obter a melhoria do desempenho dos agentes em ambientes complexos e dinâmicos. Foram construídos diversos cenários com características e mecânicas comumente utilizadas em jogos digitais. Como resultado deste estudo, verificou-se, por meio da análise da taxa de sucesso, que todos os agentes foram capazes de realizar suas tarefas com grande precisão, com uma taxa mínima de sucesso de 87,4% em 1000 tentativas, mesmo em cenários com desafios significativos. Isso comprova a eficácia do algoritmo utilizado e mostra a capacidade dos agentes de generalizar estratégias e alcançar seus objetivos em diferentes situações.

Palavras-chave: Aprendizado por Reforço; Proximal Policy Optimization; Unity; Jogos Digitais.

ABSTRACT

This work aims to understand the process of applying reinforcement learning algorithms through implementation in Unity environments. As a methodology, an exploratory research approach with a quantitative perspective was chosen, developed in three stages: analysis of the literature related to reinforcement learning algorithms; implementation of the PPO (Proximal Policy Optimization) reinforcement learning algorithm in Unity environments; and development and evaluation of the effectiveness and performance of agents in specific scenarios. To achieve this, reinforcement learning was applied to different tasks, aiming to improve agent performance in complex and dynamic environments. Various scenarios were constructed with characteristics and mechanics commonly used in digital games. As a result of this study, it was found-through success rate analysis-that all agents were able to perform their tasks with high accuracy, achieving a minimum success rate of 87.4% in 1,000 attempts, even in scenarios with significant challenges. This confirms the effectiveness of the algorithm used and demonstrates the agents' ability to generalize strategies and achieve their objectives in various situations.

Keywords: Reinforcement Learning, Proximal Policy Optimization; Unity; Digital Games.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AR	*Aprendizado por Reforço*
DQN	*Deep Q-Network*
IA	*Inteligência Artificial*
MDP	*Markov Decision Process*
TRPO	*Trust Region Policy Optimization*
PPO	*Proximal Policy Optimization*
A3C	*Asynchronous Advantage Actor Critic*
KL	*Kullback-Leibler*

Sumário

1 INTRODUÇÃO

O aprendizado por reforço (AR) é uma técnica de *Machine Learning* que se destaca como uma importante ferramenta na área da inteligência computacional. As inovações habilitadas são vastas e estão em constante evolução. À medida que novas técnicas e algoritmos são desenvolvidos, expandem-se as possibilidades de aplicação do AR em uma variedade de campos. Nos últimos anos, foi amplamente utilizado em áreas como robótica, processamento de linguagem natural e jogos digitais (JANG et al., 2019; AL-HAMADANI et al., 2024).

A teoria subjacente ao AR encontra suas bases em investigações psicológicas e neurocientíficas, muitas vezes inspiradas pela observação do comportamento animal, que ajudam a responder como os agentes podem aprimorar o controle de um ambiente. Por exemplo, a recompensa concedida ao agente pode ser comparada à liberação de dopamina no cérebro, uma substância química intimamente ligada aos processos de compensação e prazer (BROWN; JENKINS, 1968; ADEL; GRIFFITH, 2021; MNIH et al., 2015).

Em 2009, o *Institute of Electrical And Engineers Symposium on Computational Intelligence and Games* realizou a *Mario AI Competion*, onde o AR foi utilizado pela primeira vez para jogar vídeo games. Alguns anos depois, em 2013, pesquisadores propuseram um novo algoritmo que consistia em uma rede neural convolucional combinada com um algoritmo de AR conhecido como *Deep Q-learning* (DQN). Com o passar dos anos, novos pesquisadores apresentaram formas de aperfeiçoar o modelo (LIN et al., 2019).

Atualmente, a integração de técnicas de Inteligência Artificial em jogos é uma prática consolidada, sendo respaldada por um amplo cenário de eventos acadêmicos e publicações especializadas voltadas exclusivamente para esse domínio de pesquisa (JUSTESEN et al., 2020). Paralelamente, a percepção de que os videogames representam ambientes ideais para testes de métodos de inteligência artificial em contextos complexos e dinâmicos ganhou notoriedade nos últimos anos em toda a comunidade de IA (TORRADO et al., 2018).

Essas abordagens são eficazes para identificar áreas de interesse dos jogadores, aprimorar a qualidade e dinâmica da produção, diminuir os custos globais do produto e definir estratégias de investimento futuro. Além disso, elas desempenham um papel significativo ao ajudar os desenvolvedores na implementação de iniciativas voltadas para a retenção do maior número possível de jogadores (CHEN et al., 2018).

Em vista do que foi abordado, este trabalho tem como objetivo geral: demonstrar e avaliar o processo de aplicação de aprendizado por reforço por meio da implementação em jogos da Unity. Foram elaborados três objetivos específicos: i) analisar a literatura associada aos algoritmos de aprendizado por reforço; ii) Desenvolver múltiplos agentes inteligentes em ambientes tridimensionais utilizando a plataforma Unity, adotando o algoritmo Proximal Policy Optimization (PPO) como método de treinamento; e iii) Avaliar o desempenho dos agentes modelados por meio de métricas quantitativas.

A fim de atender aos objetivos propostos, aplicou-se AR em diferentes tarefas, visando obter a melhoria do desempenho dos agentes em ambientes complexos e dinâmicos. Para isso, foram construídos diversos cenários com características e mecânicas comumente utilizadas em jogos digitais. A escolha da Unity como plataforma de implementação propiciou a experimentação e validação dos agentes modelados.

A decisão de abordar o desenvolvimento de jogos e o aprendizado por reforço justifica-se pela importância de contribuir para o conhecimento nesse campo emergente. Ao aprofundar a compreensão sobre a capacidade dos algoritmos de aprendizado por reforço e como eles podem aprimorar o desenvolvimento de jogos, busca-se averiguar o potencial dessa abordagem e suas implicações para o futuro do entretenimento digital.

Esta pesquisa foi conduzida por meio da abordagem quantitativa, que se constitui como um método de investigação que emprega técnicas estatísticas, partindo da premissa de que qualquer fenômeno pode ser mensurado numericamente. Nesse sentido, esse tipo de procedimento converte informações em dados numéricos, possibilitando sua classificação e análise. Seus resultados são úteis na orientação do planejamento de ações e geram conclusões passíveis de generalização. Dessa forma, a aplicação da abordagem quantitativa é apropriada para este estudo, pois há a necessidade de quantificar ou mensurar os dados obtidos (BOTELHO; CRUZ, 2013).

Com base nos objetivos definidos para o escopo deste trabalho, realizou-se uma pesquisa exploratória, cujo propósito é promover uma compreensão mais aprofundada do problema, buscando torná-lo mais explícito. Pode-se afirmar que o principal enfoque dessas investigações é aprimorar ideias. Seu planejamento é, portanto, altamente flexível, permitindo a consideração de uma variedade de aspectos relacionados ao fenômeno estudado (GIL et al., 2002).

Tendo em vista que pesquisas exploratórias envolvem levantamento bibliográfico, este estudo pretende analisar a literatura associada aos algoritmos de aprendizado por reforço, com o propósito de estabelecer uma base teórica sólida que respalde a compreensão dos seus princípios fundamentais. Isso proporcionou uma estrutura teórica robusta para a aplicação prática em ambientes Unity. Os procedimentos técnicos utilizados se constituem como experimentais, que envolvem a identificação de um objeto de estudo, a seleção de variáveis que possam influenciá-lo e a definição de estratégias para controlar e observar os efeitos causados pela variável no objeto. Portanto, trata-se de uma abordagem em que o pesquisador desempenhou um papel ativo, atuando como agente participante, em vez de ser um mero observador passivo (GIL et al., 2002).

Por fim, resta destacar a organização deste trabalho, que está estruturado em três capítulos. O primeiro capítulo, denominado "Fundamentos do Aprendizado por Reforço", apresenta a fundamentação teórica necessária para embasar o estudo. No segundo capítulo, "Metodologia", são detalhados os procedimentos adotados e as ferramentas utilizadas ao longo do desenvolvimento dos agentes. Finalmente, o terceiro capítulo, intitulado "Análise e Discussão de Resultados", discute os resultados obtidos e suas implicações.

Assim, este estudo busca respaldar a aplicabilidade e a efetividade dos algoritmos, considerando métricas como taxa de aprendizado, convergência e desempenho em tarefas específicas nos ambientes desenvolvidos na Unity. A validação desses algoritmos contribui para a consolidação de conhecimentos e a tomada de decisões informadas na escolha de abordagens de AR para aplicações futuras.

2 FUNDAMENTOS DO APRENDIZADO POR REFORÇO

Diferentemente de outras abordagens de aprendizado de máquina que dependem de grandes conjuntos de dados históricos, o AR permite que um agente aprenda por meio da interação direta com o ambiente, dispensando a necessidade de tabelas de dados preexistentes (KESIM; OZARSLAN, 2012; LUONG et al., 2019). Dessa forma, o Aprendizado por Reforço constitui uma abordagem distinta do aprendizado supervisionado e não supervisionado, pois o agente aprende interagindo com o ambiente e recebendo feedback por meio de um sinal de recompensa (JANG et al., 2019).

Essa capacidade de autonomia é essencial em contextos onde a adaptabilidade e a tomada de decisões em tempo real são necessárias. Desse modo, os agentes podem explorar e experimentar diferentes ações, recebendo *feedback* imediato sobre a eficácia de suas decisões. Com base nas ações tomadas, eles aprendem a otimizar seu comportamento para buscar uma solução adequada, utilizando as funções para encontrar uma política ótima, mesmo em ambientes complexos e dinâmicos (BAO; GONG; YANG, 2023; JANG et al., 2019; DULAC-ARNOLD et al., 2021).

No AR, um agente aprende interagindo com um ambiente E através do tempo t. Em cada momento t o agente seleciona uma ação a de acordo com um conjunto de ações possíveis seguindo uma política π. Ao realizar uma ação, o ambiente retorna uma recompensa r que pode ser negativa ou positiva dependendo da qualidade da ação realizada (MNIH et al., 2016; LI, 2017).

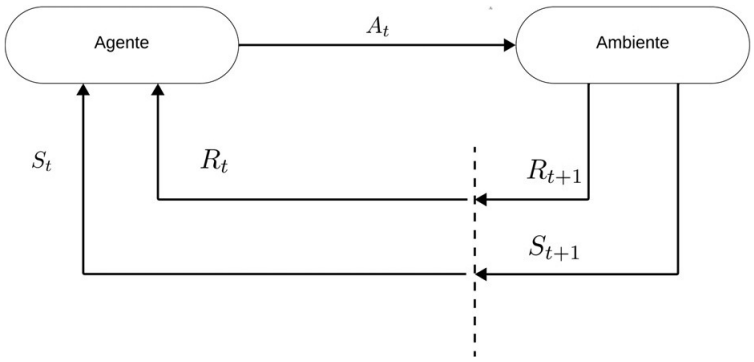
Contudo, ao aplicar efetivamente o AR em cenários que se assemelham à complexidade do mundo real, os agentes enfrentam um desafio significativo. Isso ocorre devido à necessidade de extrair representações eficazes do ambiente a partir de entradas sensoriais complexas e, em seguida, utilizar essas representações para generalizar experiências anteriores às novas situações (SUTTON, 2018; KORMUSHEV; CALINON; CALDWELL, 2013; DULAC-ARNOLD et al., 2021).

2.1 Processo de decisão de Markov

Quando o problema de aprendizado por reforço satisfaz a propriedade de Markov, ou seja, quando o futuro depende apenas da ação e do estado atual, ele pode ser formulado como um Processo de Decisão de Markov (MDP) (KUNCZIK, 2022). Um MDP é um modelo matemático que descreve como um agente interage com o ambiente dinâmico ao longo de uma sequência de etapas. No MDP, o agente utiliza o conceito de função valor, uma medida que avalia a utilidade esperada de estar em um determinado estado e tomar uma determinada ação. A função valor está intrinsecamente ligada à equação de Bellman, uma formulação que expressa a relação entre a função valor de um estado e suas possíveis ações futuras (JANG et al., 2019; KORMUSHEV; CALINON; CALDWELL, 2013; KUNCZIK, 2022).

A figura 1 mostra o funcionamento de um agente dentro de um ambiente markoviano. Nesse ambiente, o agente observa o estado atual, escolhe uma ação com base em sua política, recebe uma recompensa do ambiente e transita para um novo estado.

Figura 1 – Processo de Decisão de Markov.



Fonte: Acervo do Autor.

O AR utiliza o processo de decisão de Markov e a função valor para encontrar uma equação de Bellman, que representa a base matemática para avaliar as ações futuras do agente em relação ao ambiente. Para alcançar um aprendizado eficaz, é de extrema importância escolher um algoritmo adequado a fim de resolver a equação de Bellman, pois isso influenciará diretamente em seu desempenho na tarefa de aprendizado (JANG et al., 2019; GOTTWALD et al., 2021; SUTTON, 2018).

A distinção entre tarefas simples e complexas no AR frequentemente se relaciona com a facilidade de modelar recompensas que incentivem o algoritmo a aprimorar sua política. Em tarefas simples, as recompensas podem ser definidas com parâmetros claros e diretos, permitindo que o algoritmo aprenda de forma eficiente e rápida (DULAC-ARNOLD et al., 2021; KWON; KIM; KWON, 2024). Entretanto, em tarefas complexas, a definição de recompensas se torna significativamente mais desafiadora. Essas tarefas, geralmente, envolvem múltiplas etapas, condições dinâmicas e objetivos variados, o que dificulta a criação de um sistema de recompensas que guie o algoritmo de forma eficaz (GAO et al., 2024; ANDERSEN; GOODWIN; GRANMO, 2018).

A recompensa imediata recebida pelo agente em um dado estado s, após realizar uma entre as possíveis ações naquele momento e transitar para o próximo estado s', é dada por:

$$R(s, a, s') \tag{2.1}$$

Onde S é o conjunto de todos os estados observáveis pelo agente; e A é o conjunto de todas as possíveis ações que o agente pode executar em um dado estado.

Outro elemento crucial é o fator de desconto, representado pela letra grega γ, que desempenha um papel fundamental na modelagem da tomada de decisões do agente ao longo

do tempo (DEWANTO; GALLAGHER, 2022). O objetivo é permitir que o agente considere não apenas as recompensas imediatas, mas também as consequências a longo prazo de suas ações. Ele é necessário para garantir que o agente tome decisões que não maximizem somente as recompensas imediatas, mas também otimizem as recompensas acumuladas ao longo do tempo. Ao levar em conta as recompensas futuras de forma ponderada, o fator de desconto ajuda a evitar que o agente se comporte de maneira impulsiva e $\gamma \in (0, 1)$ (PITIS, 2019).

Em um dado estado, o agente precisa realizar uma ação entre todas as disponíveis, cujo processo de decisão é feito com base em uma política. A política π mapeia cada estado s para uma distribuição de probabilidades sobre as ações a. Em outras palavras, ela é responsável por determinar qual ação o agente deve escolher quando se encontrar em um determinado estado(THéATE; ERNST, 2023; ZHU et al., 2024). Portanto, a probabilidade do agente escolher uma ação é dada por:

$$\pi\left(a|s\right) = P\left(A = a|S = s\right) \tag{2.2}$$

Onde $pi\left(a|s\right)$ representa a probabilidade da ação a ser selecionada quando o agente está no estado s.

Para que o agente calcule a recompensa futura, é importante considerar todas as ações realizadas ao seguir uma determinada política. Logo, torna-se fundamental avaliar a política utilizada, medindo o retorno esperado a partir de um estado (AHILAN, 2023). Como solução, utiliza-se a função valor, que corresponde à soma de todas as recompensas esperadas ao seguir uma política específica a partir do estado atual e é definida por:

$$V_{\pi}\left(s\right) = \mathbb{E}_{\pi}\left[R_{t+1} + \gamma V_{\pi}\left(S_{t+1}\right)|S_t = s\right] \tag{2.3}$$

Onde $V_{\pi}\left(s\right)$ representa a função do estado s sob a política π; $\mathbb{E}_{\pi}$ denota a expectativa de acordo com a política π; R_{t+1} é a recompensa recebida no tempo $t + 1$; e S_{t+1} é o próximo estado.

A função valor representa a expectativa de valor associada a um estado. É calculada como a soma das recompensas que serão recebidas após a realização de ações conforme a política atual (ADEL; GRIFFITH, 2021). A equação de Bellman, que descreve a relação entre a função valor do estado atual e o valor do próximo estado, é definida por:

$$V_{\pi}\left(s\right) = \sum_{a} \pi\left(a|s\right) \sum_{s'} P\left(s'|s, a\right)\left[R\left(s, a, s'\right) + \gamma V_{\pi}\left(s'\right)\right] \tag{2.4}$$

Onde $P(s' \mid |s, a)$ é a probabilidade de transição do estado s para o estado s' após realizar a ação a; e $V_{\pi}(s)$ é a função valor do próximo estado s' sob a política π.

Uma política ótima é aquela que maximiza a recompensa total acumulada ao longo do tempo. Em termos matemáticos, uma política π^* é considerada ótima se para todos os estados s, a função valor $V_\pi(s)$ é maior ou igual à função valor de qualquer outra política π, ou seja, ${V_\pi}^*(s) \geq V_\pi(s)$ para todos os estados s (ADEL; GRIFFITH, 2021; KUNCZIK, 2022). Portanto, a equação de Bellman para a função valor ótima, conhecida como a equação de Bellman ótima é definida por:

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right] \tag{2.5}$$

Para resolver a equação de Bellman, é necessário encontrar uma forma eficiente de calcular os valores ótimos para cada estado e ação. Uma das abordagens mais tradicionais e amplamente utilizadas para essa tarefa é o *Q-learning*, que permite ao agente aprender a política ótima por meio da interação direta com o ambiente.

2.2 Q-learning

Proposto por Watkins em 1989, o *Q-learning* é um algoritmo de *reinforcement learning* livre de modelo que se destaca por sua capacidade de aprendizado sem a necessidade de um modelo explícito do ambiente. Esse algoritmo é uma poderosa ferramenta capaz de resolver a equação de Bellman por meio de programação dinâmica. A principal ideia por trás do *Q-learning* é aprender o valor Q, que representa a previsão do retorno associado a cada ação a pertencente ao conjunto de ações A em cada estado s (DAYAN; WATKINS, 1992; RUMMERY; NIRANJAN, 1994).

O valor $Q\,(s,a)$ indica a qualidade de executar a ação a no estado s, levando em conta as recompensas futuras esperadas. Essa previsão é atualizada iterativamente à medida que o agente explora o ambiente, ajustando-se com base nas novas informações obtidas (RUMMERY; NIRANJAN, 1994). A equação de atualização do *Q-learning* é fundamental para este processo e é dada por:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{2.6}$$

Onde $Q\,(s,a)$ é o valor atual estimado para a ação a no estado s; α é a taxa de aprendizado; e $max_{a'}Q(s',a')$ é o valor Q máximo para o próximo estado s' sobre todas as ações a', indicando o valor ótimo esperado do próximo estado.

O *Q-learning* se baseia em uma abordagem iterativa e exploratória. Inicialmente, o agente escolhe ações de maneira aleatória para explorar o ambiente e acumular conhecimento sobre os estados e as recompensas associadas. À medida que o agente acumula mais experiência, ele começa a escolher ações que maximizam o valor Q, equilibrando a exploração de novas

ações e de ações conhecidas que proporcionam altas recompensas (DAYAN; WATKINS, 1992; RUMMERY; NIRANJAN, 1994).

Devido à sua aplicação simples e aos bons resultados obtidos em ambientes discretos, o *Q-learning* se tornou a base de muitos algoritmos de aprendizado por reforço. No entanto, seu desempenho é limitado em problemas mais complexos com grande número de estados e ações. Os dados das recompensas são armazenados em *Q-Tables*, e realizar tal procedimento para cada par estado-ação exige uma quantidade significativa de memória (JANG et al., 2019).

Outro problema encontrado é a possibilidade de sofrer de superestimações devido à natureza da atualização de seus valores Q, que pode introduzir um viés otimista. Isso ocorre durante o processo de maximização usado para escolher a ação com o maior valor Q estimado. A máxima operação gananciosa $max_{a'}Q(s', a')$ tende a superestimar a recompensa esperada, efeito que ocorre porque o processo de maximização não leva em conta a incerteza ou erro nas estimativas Q. Normalmente, mesmo que as estimativas Q estejam distribuídas em torno do valor real, a seleção do valor máximo tende a selecionar a média das estimativas para cima (HASSELT, 2010; LAN et al., 2020).

Para superar essa limitação, novos modelos foram desenvolvidos, como o *Deep Q-Network* (DQN), que combina as propriedades do *Q-learning* com uma rede neural profunda, utilizada como um aproximador de função para a função Q. Esse modelo dispõe da capacidade das redes neurais profundas para generalizar a partir de grandes volumes de dados e para representar funções complexas, permitindo que o agente aprenda políticas eficazes mesmo em ambientes complexos (JANG et al., 2019; TAN; YAN; GUAN, 2017).

2.3 Deep Q-Network

Desenvolvido por pesquisadores da DeepMind, o *Deep Q-Network* (DQN) é um algoritmo de aprendizado por reforço que combina *Q-learning* com redes neurais profundas, com o objetivo de solucionar os problemas de escalabilidade que o *Q-learning* tradicional enfrentava em ambientes com grandes espaços de estados e ações (MNIH et al., 2015).

No *Q-learning* tradicional, a função Q é armazenada em tabelas (*Q-Tables*), o que se torna impraticável em ambientes complexos. No DQN, a função Q é aproximada usando uma rede neural profunda cujos parâmetros são representados por θ (FAN et al., 2020). A equação de atualização para o DQN, incorporando os pesos da rede, é dada por:

$$Q(s, a; \theta) \leftarrow Q(s, a; \theta) + \alpha \left[R + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right] \tag{2.7}$$

Onde $Q(s, a; \theta)$ é o valor Q estimado pela rede neural com parâmetros θ para o estado s e a ação a; $\max_{a'} Q(s', a'; \theta^-)$ é o valor Q máximo estimado para o próximo estado s' usando uma rede neural alvo θ^-; e θ^- são os parâmetros da rede neural alvo, que são periodicamente

atualizados para uma cópia dos parâmetros θ da rede neural principal, ajudando a estabilizar o aprendizado.

Os parâmetros da rede neural principal θ são atualizados através da minimização do erro entre o valor Q estimado e um valor-alvo y, definido como:

$$y = R + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{2.8}$$

Esse valor-alvo representa a melhor estimativa do retorno futuro esperado, utilizando a rede neural alvo, cujos parâmetros θ^- são periodicamente sincronizados com a rede principal.

A função de perda $L(\theta)$ é então definida como:

$$L(\theta) = \mathbb{E}\left[(y - Q(s, a; \theta))^2\right] \tag{2.9}$$

Esse erro é minimizado usando técnicas de otimização baseada em gradiente, como o gradiente descendente estocástico.

Outra importante característica do DQN é o uso de um buffer de experiência (*experience replay*). Durante o treinamento, as transições de estado, ação, recompensa e próximo estado são armazenadas em um buffer. As amostras são então aleatoriamente extraídas do buffer para treinar a rede neural. Essa técnica reduz a correlação entre experiências consecutivas e melhora a eficiência do aprendizado, permitindo que a rede neural se beneficie de uma maior diversidade de exemplos de treinamento e evitando o sobreajuste a sequências específicas de estados e ações (MNIH et al., 2015; SEWAK; SEWAK, 2019).

A rede neural alvo θ^-, cujos parâmetros são copiados periodicamente da rede neural principal θ, mantém os valores Q estimados fixos por um certo número de atualizações da rede principal. Essa abordagem estabiliza o treinamento, evitando que a rede neural principal persiga constantemente um valor Q em movimento, o que poderia levar a oscilações e divergências durante o aprendizado (ZHANG et al., 2023; WANG; UEDA, 2021).

O *Deep Q-Network* (DQN) trouxe avanços significativos para o AR, especialmente ao lidar com grandes espaços de estados e ações. No entanto, o DQN também apresenta algumas limitações e problemas que impactam seu desempenho e eficácia em certos cenários. Alguns exemplos de adversidades encontradas são: a) instabilidades durante o treinamento; b) superestimação do valor Q na operação $\max_{a'} Q(s', a'; \theta^-)$, que pode ser influenciado por ruídos ou erros nas estimativas; e c) grande ineficiência em espaços de ações contínuos (FAN et al., 2020; HASSELT; GUEZ; SILVER, 2016; LI et al., 2024).

Novos métodos foram desenvolvidos para solucionar esses problemas. Melhorias como: o *Double DQN*, que utiliza duas redes para reduzir a superestimação; *Dueling DQN*, que decompõe a estimativa de valores Q em valor do estado e vantagem da ação, melhorando a eficiência do

aprendizado; e o *Asynchronous Advantage Actor Critic* (BABAEIZADEH et al., 2016; MNIH et al., 2016).

2.4 Gradiente de Política

Métodos de gradiente de política são uma classe importante de algoritmos de aprendizado por reforço, que se concentram em melhorar diretamente a política aprendida. Esses métodos trabalham computando um estimador do gradiente da política e, em seguida, aplicando esse estimador em um algoritmo de gradiente ascendente estocástico para atualizar os parâmetros da política (SCHULMAN et al., 2017).

$$\hat{g} = \hat{\mathbb{E}}t \left[\nabla\theta \log \pi_\theta \left(a_t \mid s_t \right) \hat{A}_t \right] \tag{2.10}$$

Nesse contexto,π_θ representa a política estocástica parametrizada por θ, onde a política define a probabilidade de selecionar a ação a_t dado o estado s_t. O termo $\hat{A}_t$ é um estimador da função vantagem no tempo t, que mede a diferença entre a recompensa recebida ao tomar a ação a_t no estado s_t e a recompensa esperada seguindo a política π_θ.

A expectativa $\hat{\mathbb{E}}t \left[\nabla\theta \log \pi_\theta \left(a_t \mid s_t \right) \hat{A}_t \right]$ representa a média empírica calculada a partir de um conjunto de amostras obtidas através da interação com o ambiente. Isso significa que o estimador do gradiente é calculado sobre várias trajetórias de experiência, capturando a variabilidade inerente ao ambiente estocástico (SCHULMAN et al., 2017).

Implementações que utilizam diferenciação automática facilitam esse processo construindo uma função objetivo cuja derivada corresponde ao estimador do gradiente da política. Isso é vantajoso porque permite a utilização de frameworks de aprendizado profundo para automatizar o cálculo de gradientes (SCHULMAN et al., 2017). A função objetivo típica usada é:

$$L^{PG}(\theta) = \hat{\mathbb{E}}t \left[\log \pi\theta \left(a_t \mid s_t \right) \hat{A}_t \right] \tag{2.11}$$

Ao diferenciar essa função objetivo em relação aos parâmetros θ, obtém-se o estimador do gradiente $\hat{g}$. Esse gradiente é então usado para ajustar os parâmetros da política, buscando melhorar o desempenho ao longo do tempo.

Os métodos de gradiente de política são particularmente úteis em problemas onde a política precisa ser melhorada continuamente com base em novas experiências. Tais métodos são fundamentais para algoritmos que utilizam políticas estocásticas, como o TRPO e o PPO, pois permitem garantir estabilidade nas atualizações de política e maximizar a performance de forma eficiente em ambientes de AR (HU, 2023).

2.5 Asynchronous Advantage Actor Critic (A3C)

O *Asynchronous Advantage Actor-Critic* (A3C), proposto pela DeepMind, é um algoritmo robusto que surge como alternativa às limitações enfrentadas por métodos tradicionais como o *Deep Q-Network* (DQN). Esses desafios incluem a escalabilidade em ambientes com grandes espaços de estados e ações, além de problemas de instabilidade e superestimação. O A3C combina técnicas baseadas em política e valor, juntamente com um treinamento assíncrono, para melhorar a eficiência e estabilidade durante o aprendizado (MNIH et al., 2016).

O A3C utiliza uma abordagem híbrida que integra redes neurais profundas para aprender tanto a política quanto a função de valor. A arquitetura é composta por duas redes neurais principais: a rede *actor* e a rede *critic* (MNIH et al., 2016; MIROWSKI et al., 2016).

A rede *actor* parametriza a política $\pi\left(a_t \mid s_t; \theta_\pi\right)$, que define a probabilidade de selecionar uma ação a_t em um estado s_t. Essa rede é atualizada com base no gradiente da política ponderado pela vantagem, o que promove a melhoria contínua da política de decisão (MNIH et al., 2015). A rede *critic* estima a função valor-estado $V(s_t; \theta_V)$, que representa a expectativa de recompensa futura a partir do estado s. Esta função é utilizada para calcular a função de vantagem, que mede o quanto uma ação específica é melhor do que a média das ações tomadas no estado (MNIH et al., 2016; SEWAK, 2019). A função de valor pode ser também expressa em termos de uma função de valor de ação $Q(s, a)$:

$$Q(s_t, a_t) = V(s_t; \theta_V) + A(s_t, a_t; \theta, \theta_V) \tag{2.12}$$

A função vantagem $A(s_t, a_t; \theta, \theta_v)$ mede o quanto a ação a_t no estado s_t é melhor em comparação ao valor esperado do estado s_t, ela ajuda a reduzir a variância nas atualizações da política, resultando em um aprendizado mais estável (MNIH et al., 2016; SEWAK, 2019). A vantagem é calculada usando tanto os parâmetros da política θ quanto os parâmetros do crítico θ_v e é definida por:

$$A(s_t, a_t; \theta, \theta_V) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_V) - V(s_t; \theta_V) \tag{2.13}$$

Onde $\sum_{i=0}^{k-1} \gamma^i r_{t+i}$ representa a soma das recompensas recebidas a partir do estado s_t ao longo de k passos executados, $\gamma^k V(s_{t+k}; \theta_V)$ é a estimativa do valor do estados s_{t+k} fornecida pela rede crítica parametrizada por θ_v e $V(s_t; \theta_V)$ é o valor estimado do estado atual s_t.

A política e a função de valor são atualizadas a cada t_{max} passos ou quando um estado terminal é alcançado. A atualização realizada pelo A3C pode ser expressa pela seguinte equação:

$$\nabla_\theta \log \pi(a_t|s_t; \theta) A(s_t, a_t; \theta, \theta_V) \tag{2.14}$$

onde $\pi(a_t \mid s_t; \theta_\pi)$ representa a política do agente, que define a probabilidade de selecionar a ação a_t no estado s_t, parametrizada por θ_π. O gradiente da política $\nabla_{\theta_\pi} \log \pi(a_t \mid s_t; \theta_\pi)$ indica a direção em que os parâmetros θ_π devem ser ajustados para aumentar a probabilidade da ação a_t no estado s_t, promovendo uma melhoria na política de decisão.

Uma característica fundamental do A3C é o treinamento assíncrono. Diferentemente dos métodos tradicionais que atualizam os parâmetros de rede de maneira síncrona, o A3C permite que múltiplos agentes explorem diferentes partes do espaço de estados simultaneamente e de forma independente. Esses agentes coletam experiências e atualizam os parâmetros da rede global de forma assíncrona, oferecendo várias vantagens: a) a execução assíncrona reduz a correlação entre as experiências consecutivas, levando a um aprendizado mais robusto; b) utilizar exploração com múltiplos agentes resulta em uma cobertura mais ampla do espaço de estados, acelerando o processo de aprendizado; e c) a agregação assíncrona de gradientes ajuda a estabilizar as atualizações de parâmetros, mitigando problemas de divergência e instabilidade (MNIH et al., 2016; BABAEIZADEH et al., 2016).

Apesar de ser mais eficiente, o A3C ainda enfrenta alguns desafios. A implementação do A3C é mais complexa do que métodos mais simples, como o DQN. Durante sua execução é necessário coordenar múltiplos agentes que operam de forma assíncrona, interagindo com cópias independentes do ambiente. Essa complexidade pode aumentar significativamente o tempo de desenvolvimento. Erros na sincronização dos agentes ou na gestão de seus estados podem levar a comportamentos inesperados ou a um desempenho indesejado (MNIH et al., 2016; SEWAK, 2019).

Por possuir uma natureza assíncrona, os gradientes calculados pelos diferentes agentes podem ser bastante ruidosos. Esse ruído pode resultar em atualizações menos precisas dos parâmetros da rede, afetando a estabilidade e a taxa de convergência do algoritmo. Em comparação com métodos síncronos, onde os gradientes são calculados de maneira mais coordenada, o A3C pode enfrentar dificuldades adicionais para alcançar uma solução estável (CHEN et al., 2021; SHEN et al., 2023).

Sua natureza assíncrona pode introduzir desafios adicionais na convergência do algoritmo. A presença de gradientes ruidosos e a falta de coordenação entre os agentes podem resultar em oscilações no desempenho durante o treinamento. Técnicas de regularização e ajustes frequentes de hiperparâmetros são muitas vezes necessários para mitigar esses problemas e garantir uma convergência estável (BABAEIZADEH et al., 2016; SHEN et al., 2023) .

2.6 Trust Region Policy Optimization

Desenvolvido por John Schulman e outros colaboradores em 2015, o *Trust Region Policy Optimization* (TRPO) é um algoritmo robusto projetado para resolver problemas comuns

enfrentados por algoritmos de aprendizado por reforço baseados em gradientes, como oscilações instáveis e grandes quedas de desempenho durante o treinamento (SCHULMAN et al., 2015).

O TRPO busca maximizar a expectativa da vantagem ao mesmo tempo em que mantém a nova política dentro de uma "região de confiança" em relação à política antiga. Para isso, o algoritmo utiliza uma função objetivo de surrogado, maximizada sob uma restrição que limita o tamanho da atualização da política com base na divergência Kullback-Leibler (KL) (SCHULMAN et al., 2015; WANG et al., 2019).

$$\max_{\theta} \mathbb{E} s \sim p\theta_{\text{old}}, a \sim \pi_{\theta_{\text{old}}} \left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} \hat{A}_{\theta_{\text{old}}}(s, a) \right] \tag{2.15}$$

sujeito à restrição:

$$\mathbb{E} s \sim p\theta_{\text{old}} \left[D_{KL} \left(\pi_{\theta_{\text{old}}}(\cdot \mid s) \parallel \pi_\theta(\cdot \mid s) \right) \right] \leq \delta \tag{2.16}$$

Onde π_θ é a nova política; $\pi_{\theta_{old}}$ é a política antiga; $\hat{A}_{\theta_{\text{old}}}(s, a)$, a é a vantagem estimada com base na política antiga, que mede quão boa é a ação a em comparação com o valor esperado de ações no estado s sob a política $\pi_{\theta_{old}}$; $D_{KL}(\pi_{\theta_{\text{old}}}(\cdot\, s) \parallel \pi_\theta(\cdot \mid s))$ é a divergência de KL entre as políticas nova e antiga; e δ é um parâmetro que controla o limite de divergência permitido (SCHULMAN et al., 2015; LIU et al., 2019).

A teoria subjacente ao TRPO sugere a utilização de uma penalidade em vez de uma restrição, ou seja, resolvendo o problema de otimização irrestrita para algum coeficiente β. Isso ocorre porque um certo objetivo substituto, que calcula o KL máximo sobre os estados em vez da média, estabelece um limite inferior pessimista no desempenho da política π. O TRPO opta por uma restrição rígida em vez de uma penalidade por conta da dificuldade de escolher um único valor de β que funcione bem em diferentes situações (SCHULMAN et al., 2017; LIU et al., 2019; KUBA et al., 2021) .

$$\max_{\theta} \mathbb{E} s \sim p\theta_{\text{old}}, a \sim \pi_{\theta_{\text{old}}} \left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} \hat{A}_{\theta_{\text{old}}}(s, a) - \beta D_{KL} \left(\pi_{\theta_{\text{old}}}(\cdot \mid s) \parallel \pi_\theta(\cdot \mid s) \right) \right] \tag{2.17}$$

Essa equação objetiva penalizada representa o objetivo substituto que busca maximizar a expectativa da vantagem, penalizando a divergência KL, com β sendo o coeficiente de penalidade.

A região de confiança no TRPO refere-se ao espaço dentro do qual as atualizações de política são permitidas, de modo a garantir que uma nova política não se afaste excessivamente de uma política antiga. A ideia é que, ao restringir a magnitude das mudanças nas distribuições de probabilidade que são encontradas através da divergência KL, o TRPO mantenha a nova política dentro de uma região onde a performance é estável e previsível (SCHULMAN et al., 2015; KUBA et al., 2021).

A divergência de KL é uma medida de dissimilaridade entre duas distribuições de probabilidade. Ela quantifica o quanto uma distribuição de probabilidade Q diverge de uma segunda distribuição de probabilidade P. No contexto do TRPO a restrição de divergência KL funciona como um limitador que impede que grandes atualizações causem deterioração no desempenho do agente. Isso é crucial em ambientes complexos onde grandes saltos na política podem levar a comportamentos indesejados ou mesmo catastróficos (SCHULMAN et al., 2015; KUBA et al., 2021).

Entre as vantagens do TRPO estão: a) a estabilidade proporcionada pelas atualizações controladas, visto que a restrição KL assegura que cada passo de atualização seja pequeno o suficiente para evitar grandes oscilações e quedas de desempenho; b) oferece garantias teóricas de que a nova política não se desviará significativamente da política anterior, o que é uma propriedade desejável para garantir a convergência e a estabilidade do algoritmo (SHANI; EFRONI; MANNOR, 2020).

Entretanto, implementar o TRPO é uma tarefa mais complexa quando comparado com algoritmos mais simples como o DQN. Um dos motivos é a necessidade de um entendimento profundo de otimização de políticas, da divergência KL e de técnicas avançadas de programação. Calcular a divergência KL e resolver problemas de otimização complexos pode tornar o TRPO computacionalmente intensivo. Isso pode resultar em tempos de treinamento mais longos, especialmente em ambientes grandes e complexos (ENGSTROM et al., 2020; SCHULMAN et al., 2017).

2.7 Proximal Policy Optimization

O *Proximal Policy Optimization* (PPO) foi desenvolvido pela OpenAI como uma alternativa ao *Trust Region Policy Optimization* (TRPO), com o objetivo de simplificar sua implementação e torná-lo mais escalável. Embora o PPO compartilhe muitos dos princípios fundamentais do TRPO, ele minimiza algumas de suas complexidades, mantendo uma performance robusta em uma variedade de tarefas. Projetado para otimizar políticas estocásticas em ambientes de aprendizado por reforço, o PPO se destaca por sua simplicidade conceitual e implementação relativamente direta, ao mesmo tempo em que oferece desempenho competitivo (SCHULMAN et al., 2017; QUEENEY; PASCHALIDIS; CASSANDRAS, 2021).

A abordagem principal do PPO é iterativamente atualizar a política de forma a melhorar progressivamente o desempenho do agente, mantendo as mudanças dentro de uma região controlada. Diferente do TRPO, que impõe uma restrição rígida na divergência de Kullback-Leibler (KL), o PPO utiliza um mecanismo de "*clipping*", que limita a magnitude das atualizações da política (SCHULMAN et al., 2017). Isso evita grandes mudanças nos parâmetros da política, garantindo estabilidade durante o treinamento. A função objetivo do PPO é definida como:

$$L_{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{2.18}$$

onde θ refere-se aos parâmetros da política; $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ é o rácio entre as probabilidades de ação sob a nova política e a antiga política (também chamado de *probability ratio*); $\hat{A}_t$ é um estimador de vantagem; e ε é um hiperparâmetro que controla a operação de "*clipping*".

O termo *clipping* é uma das inovações centrais do PPO. Ele impede que as atualizações da política se afastem excessivamente da política anterior, limitando a magnitude das mudanças em $r_t(\theta)$. Isso é crucial para manter a estabilidade durante o treinamento e evitar oscilações abruptas que poderiam comprometer o desempenho do agente (SCHULMAN et al., 2017). O clipping suaviza as atualizações e impede mudanças bruscas nos parâmetros da política, o que melhora a eficiência da otimização.

Além do *clipping*, algumas variantes do PPO podem incluir uma penalidade de divergência KL para restringir a diferença entre a nova política e a anterior. Embora essa abordagem possa ser útil, o *clipping* é mais amplamente adotado por ser mais simples de ajustar e geralmente mais eficiente em termos computacionais. A divergência KL mede a diferença entre duas distribuições de probabilidade e, no contexto do PPO, seu uso ajuda a garantir que as atualizações não causem uma grande mudança na política, o que pode resultar em comportamento instável do agente (SCHULMAN et al., 2017).

O PPO oferece várias vantagens, incluindo uma implementação relativamente simples, estabilidade de treinamento e desempenho robusto em uma ampla gama de ambientes de AR, como jogos, simulações de controle de robôs e outros cenários desafiadores. Além disso, o PPO tende a produzir políticas eficazes e bem-generalizadas, o que permite que o agente lide com diversas tarefas de forma eficiente. A simplicidade e a eficácia do PPO o tornaram um dos algoritmos de AR mais populares em uso atualmente (YU et al., 2022; LARSEN et al., 2021).

2.8 Aprendizado por reforço em jogos

O AR tem sido amplamente aplicado em diversos tipos de tarefas, abrangendo áreas como pesquisa operacional, otimização baseada em simulação, computação evolutiva e jogos. A combinação de *Deep Learning* com métodos de AR tem levado a resultados de sucesso notáveis (ANDERSEN; GOODWIN; GRANMO, 2018).

2.8.1 Atari 2600

Os agentes desenvolvidos pela DeepMind, como os baseados em DQN, alcançaram desempenho humano ou super-humano em uma ampla variedade de jogos do Atari 2600, aprendendo a partir de pixels brutos e recompensas do jogo. Esses agentes, treinados ao longo de milhões de iterações, demonstraram capacidade de generalização em diversos ambientes,

evidenciando a habilidade de dominar jogos específicos, bem como sua adaptabilidade e capacidade de aprendizado em ambientes de jogo complexos e variados. O treinamento desses agentes envolveu a exploração de estratégias ótimas em condições dinâmicas e desafiadoras, ressaltando o potencial do AR na resolução de problemas em tempo real e de alta complexidade (LI, 2017; JUSTESEN et al., 2020; MNIH, 2013).

2.8.2 AlphaGo

O AlphaGo superou os melhores jogadores humanos no jogo de Go, considerado um dos jogos de tabuleiro mais complexos do mundo devido à sua vasta gama de possibilidades de movimentos e profundidade estratégica. Utilizando o AR em combinação com redes neurais profundas, o AlphaGo conseguiu aprender estratégias avançadas e imprevisíveis. Seu treinamento consistiu em duas etapas principais: inicialmente, o sistema foi treinado em um extenso conjunto de partidas de jogadores humanos, permitindo-lhe adquirir padrões básicos e estratégias fundamentais. Em seguida, o AlphaGo engajou-se em milhões de partidas contra versões de si mesmo, refinando suas habilidades e desenvolvendo novas estratégias por meio de auto-aprendizado (LI, 2017; TORRADO et al., 2018; SILVER et al., 2016).

Essa abordagem inovadora culminou na vitória contra o campeão mundial Lee Sedol em 2016, onde o AlphaGo venceu por 4 partidas a 1. Este resultado evidenciou sua capacidade de dominar um jogo considerado inatingível para máquinas e representou um avanço significativo no campo da inteligência artificial, demonstrando o potencial das técnicas de AR e redes neurais profundas na resolução de problemas altamente complexos e abstratos (LI, 2017; TORRADO et al., 2018; SILVER et al., 2016).

2.8.3 AlphaZero

O AlphaZero, um outro notável desenvolvimento da DeepMind, aplicou o AR para aprender a jogar xadrez, shogi e Go a partir do zero, sem possuir qualquer conhecimento prévio específico dos jogos. Seu processo de treinamento consistiu em um extenso conjunto de partidas, no qual o sistema jogou contra si mesmo milhões de vezes, explorando e refinando estratégias por meio do auto-aprendizado (LI, 2017; ANDERSEN; GOODWIN; GRANMO, 2018).

Essa abordagem permitiu que o AlphaZero adquirisse um profundo entendimento das complexidades e particularidades de cada jogo, desenvolvendo habilidades que, eventualmente, superaram os melhores motores de xadrez, como Stockfish (TORRADO et al., 2018; ANDERSEN; GOODWIN; GRANMO, 2018).

2.8.4 Dota 2

A OpenAI empregou o AR para competir no jogo multijogador online Dota 2, conhecido por sua complexidade e dinamismo. Em 2019, o OpenAI Five alcançou um marco notável ao derrotar uma equipe de jogadores profissionais, demonstrando habilidades avançadas em

coordenação, estratégia e tomada de decisão em tempo real. A abordagem do OpenAI Five envolveu um extenso processo de treinamento, no qual o sistema foi submetido a milhares de partidas contra si mesmo e adversários controlados por inteligência artificial, refinando suas táticas e estratégias ao longo do tempo (ANDERSEN; GOODWIN; GRANMO, 2018).

A vitória sobre jogadores profissionais ressaltou a eficácia do aprendizado por reforço em lidar com ambientes complexos e imprevisíveis, bem como sua aplicabilidade em cenários competitivos de alto nível, a exemplo de jogos multiplayer online de grande escala (ANDERSEN; GOODWIN; GRANMO, 2018).

2.9 Frameworks de RL

Existem vários frameworks de AR que oferecem ambientes ricos e diversificados para o desenvolvimento e treinamento de agentes de inteligência artificial. Essas plataformas representam uma peça fundamental no avanço da pesquisa em AR, proporcionando as ferramentas necessárias para explorar, experimentar e aprimorar algoritmos em uma variedade de cenários desafiadores (JUSTESEN et al., 2020).

2.9.1 OpenAI Gym

Desenvolvido pela OpenAI, o OpenAI Gym é uma das bibliotecas mais utilizadas e bem estabelecidas para o desenvolvimento e teste de algoritmos de AR em ambientes de jogos. Essa biblioteca oferece uma ampla variedade de ambientes de aprendizado, que vão desde jogos Atari clássicos até simuladores de robótica avançada. O OpenAI Gym possui interface simples e consistente, que facilita o desenvolvimento, o treinamento e a avaliação de agentes de AR (LI, 2017; JUSTESEN et al., 2020; ANDERSEN; GOODWIN; GRANMO, 2018).

Com uma vasta coleção de ambientes, o OpenAI Gym abrange uma ampla gama de complexidades e desafios, permitindo que os usuários explorem e experimentem algoritmos de AR em uma variedade de contextos e aplicações. Esta diversidade de ambientes oferece aos pesquisadores e desenvolvedores a oportunidade de testar suas soluções em cenários realistas e desafiadores (LI, 2017; JUSTESEN et al., 2020; ANDERSEN; GOODWIN; GRANMO, 2018).

2.9.2 Project Malmo

Criado pela Microsoft, o Project Malmo é uma plataforma de pesquisa que utiliza o popular jogo Minecraft como ambiente de treinamento para agentes de AR. Esta plataforma oferece uma interface flexível e poderosa para controlar e interagir com o ambiente do Minecraft, permitindo a condução de experimentos em uma ampla variedade de cenários (JUSTESEN et al., 2020; ANDERSEN; GOODWIN; GRANMO, 2018).

Com o Project Malmo, é possível definir objetivos específicos, criar recompensas personalizadas e modificar dinamicamente o ambiente do jogo para testar e aprimorar algoritmos

de AR. Além disso, possui suporte a colaboração entre equipes de pesquisa, facilitando o compartilhamento de agentes treinados e a replicação de experimentos em diferentes contextos (JUSTESEN et al., 2020; ANDERSEN; GOODWIN; GRANMO, 2018).

2.9.3 PySC2

PySC2, da DeepMind, é uma biblioteca que disponibiliza uma interface Python para interagir com o jogo StarCraft II. Além de facilitar o desenvolvimento e o treinamento de agentes de AR, o PySC2 oferece recursos avançados para análise e visualização de dados do jogo. Projetado para lidar com ambientes complexos e desafiadores, esse framework amplia, consideravelmente, as possibilidades de pesquisa em jogos de estratégia em tempo real (VINYALS et al., 2017).

Com o PySC2 é possível personalizar e ajustar os ambientes de treinamento conforme necessidades específicas, experimentar diferentes estratégias e avaliar a eficácia de algoritmos de AR em um dos jogos mais complexos e dinâmicos disponíveis. Essa ferramenta tem sido fundamental para impulsionar a pesquisa em inteligência artificial aplicada a jogos de estratégia em tempo real, resultando em avanços significativos na compreensão e no desenvolvimento de agentes inteligentes capazes de competir em ambientes altamente competitivos (VINYALS et al., 2017).

2.9.4 Unity ML-Agents

O Unity ML-Agents, uma avançada estrutura de AR, é integrado diretamente à engine de jogos Unity. Essa ferramenta robusta possibilita o treinamento de agentes de AR em ambientes de jogos criados na Unity, abrindo vastas oportunidades para o desenvolvimento de soluções de inteligência artificial em jogos e simulações interativas. O Unity ML-Agents oferece uma variedade de recursos e funcionalidades que simplificam o processo de treinamento de agentes inteligentes (JULIANI, 2018).

É possível personalizar e ajustar o ambiente do jogo de acordo com as particularidades necessárias, experimentar diversas estratégias de AR e avaliar a eficácia dos algoritmos implementados. O framework disponibiliza ferramentas de análise e visualização que ajudam a compreender melhor o comportamento dos agentes e a otimizar o processo de treinamento (JULIANI, 2018).

3 METODOLOGIA

Neste capítulo, serão apresentados os procedimentos adotados para o desenvolvimento e treinamento dos agentes, detalhando as etapas do processo, os ambientes utilizados e as técnicas de aprendizado por reforço implementadas. A descrição dos métodos empregados visa proporcionar uma visão clara de como os experimentos foram conduzidos, desde a configuração dos cenários até o ajuste das políticas de treinamento.

3.1 Ferramentas Utilizadas

Neste tópico, serão apresentadas as ferramentas utilizadas na execução deste trabalho, incluindo os softwares e bibliotecas essenciais para o desenvolvimento e treinamento dos agentes. Cada ferramenta será discutida em termos de suas funcionalidades, destacando como contribuem para a criação dos ambientes e implementação dos algoritmos de aprendizado por reforço.

3.1.1 Unity

Para o desenvolvimento dos agentes jogadores, foi fundamental utilizar uma plataforma robusta e flexível, capaz de oferecer suporte à criação e desenvolvimento dos jogos e à implementação de sistemas de simulação complexos. O Unity se destaca como um motor de desenvolvimento de jogos amplamente reconhecido e adotado na indústria.

Além de facilitar a criação de jogos, o Unity oferece uma vasta gama de ferramentas que possibilitam a modelagem e simulação de agentes inteligentes em ambientes tridimensionais. Entre suas funcionalidades, destacam-se a gestão de física, a renderização gráfica avançada e a capacidade de integração com bibliotecas externas, como o ML-Agents, que desempenha um papel crucial neste trabalho.

3.1.2 ML-Agents

O Unity ML-Agents, como mencionado anteriormente, é um framework de AR integrado à Unity, que permite o treinamento de agentes em ambientes de jogos criados na plataforma. Com sua ampla variedade de recursos, essa ferramenta facilita a personalização dos ambientes, a experimentação de diversas estratégias de aprendizado e a avaliação dos algoritmos implementados.

3.1.3 C#

A linguagem C# foi utilizada dentro do Unity, sendo necessária para a criação e manipulação dos componentes do jogo, como o controle de física, animações, e a lógica dos agentes. Com o C#, foi possível desenvolver *scripts* que governam o comportamento dos agentes no ambiente de jogo, permitindo uma integração com as ferramentas nativas do Unity.

3.1.4 Python 3.9

A versão de Python utilizada foi a 3.9, necessária para aplicação dos algoritmos de AR; configuração e execução dos processos de treinamento, controle e interação entre o ambiente simulado e os agentes; e gerenciamento dos dados gerados. A escolha do Python se deve à sua compatibilidade com frameworks de aprendizado de máquina, como TensorFlow e PyTorch, que são necessários para o ML-Agents, facilitando a implementação de modelos complexos de aprendizado.

3.2 Ambientes Desenvolvidos

Neste trabalho foram desenvolvidos quatro ambientes na Unity, cada um projetado para explorar e demonstrar as capacidades do AR em contextos variados. Eles foram concebidos com base em mecânicas comumente utilizadas em jogos, como corrida, combate, estacionamento e furtividade, buscando replicar desafios reais que exigem tomadas de decisão complexas por parte dos agentes.

O foco principal foi dado na criação de cenários que apresentassem um alto nível de desafio e destacassem as diferentes possibilidades do AR. Esses cenários foram desenhados para testar a eficácia dos algoritmos de aprendizado em situações que envolvem controle preciso e estratégia a longo prazo. Cada ambiente foi modelado para permitir que os agentes interajam de maneira significativa com os elementos do jogo, desenvolvendo habilidades específicas para atingir seus objetivos.

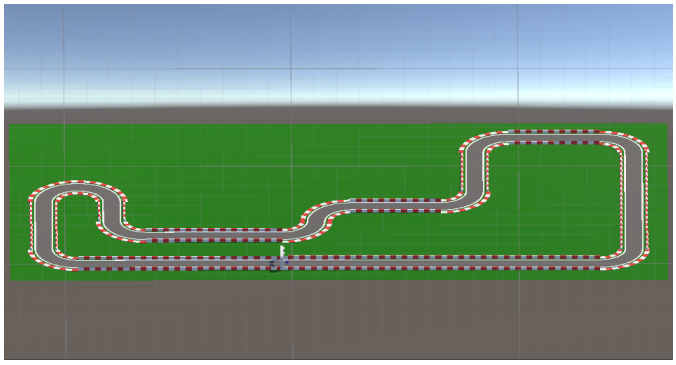
Dessa forma, os ambientes criados servem como uma plataforma robusta para a experimentação e avaliação das técnicas de AR, demonstrando o potencial dessa abordagem em uma variedade de aplicações dentro do desenvolvimento de jogos e simulações.

3.2.1 *Turbo Racing*

O *Turbo Racing* é um ambiente de jogo de corrida tradicional. Seus elementos principais incluem: a estrada, que marca o percurso a ser percorrido pelos carros; barreiras, que limitam a passagem dos veículos; e um objeto utilizado para detectar sempre que uma volta é concluída. Esses elementos são fundamentais para definir os desafios e as restrições do ambiente e são mostrados na Figura 2.

Todos os objetos no *Turbo Racing* possuem *Colliders*, que permitem a detecção de colisões entre os veículos e os elementos da pista. Os objetos foram configurados com *Tags* apropriadas, que ajudam na identificação rápida e precisa durante o treinamento do agente. Essas *Tags* serão utilizadas para orientar as decisões do agente, especialmente na identificação de limites da pista e em obstáculos que devem ser evitados.

Figura 2 – Ambiente Turbo Racing.

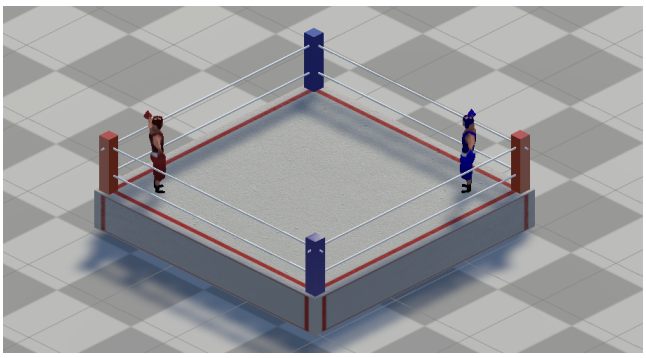


Fonte: Acervo do Autor.

3.2.2 *Fighting Warrior*

O *Fighting Warrior* é um ambiente de jogo de luta com movimentação tridimensional. O cenário é composto, principalmente, por um modelo de ringue, onde o jogador e seu oponente realizam seu embate. Além do ringue, o ambiente conta com detectores de colisão (*Colliders*) nos limites do espaço, que impedem que os lutadores ultrapassem a área de combate como é mostrado na Figura 3.

Figura 3 – Ambiente Fighting Warrior.



Fonte: Acervo do Autor.

Esses elementos fornecem o espaço necessário para a simulação de combates e garantem que os agentes operem dentro das regras do ambiente. O ringue e os detectores de colisão são úteis para determinar o limite de movimentos dos agentes e nas interações físicas durante as lutas.

3.2.3 *Parking Club*

O *Parking Club* é um ambiente que simula um estacionamento. O cenário é composto por diversos carros que já estão estacionados, todos com propriedades físicas que simulam colisões e movimento, utilizando *Colliders* e *Rigidbodies* para garantir interações realistas como é mostrado na Figura 4. Além disso, o ambiente possui uma vaga vazia destinada ao jogador.

Figura 4 – Ambiente Parking Club.

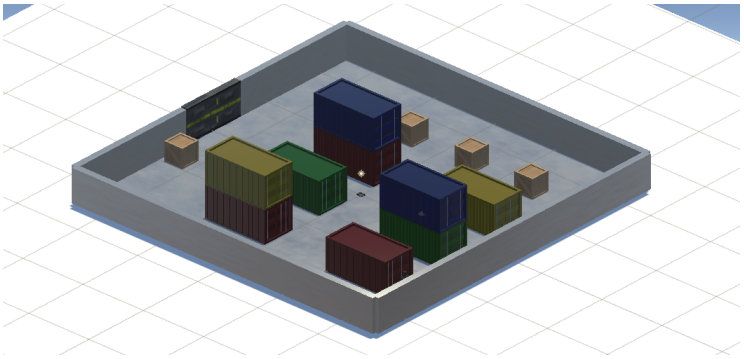
Fonte: Acervo do Autor.

O objetivo principal do jogador é estacionar o carro na vaga sem colidir com os outros veículos e na orientação correta. A configuração do cenário desafia o agente a realizar manobras precisas, o que é essencial para o treinamento de habilidades de estacionamento controlado.

3.2.4 *Sneaking Scape*

O *Sneaking Scape* é o cenário mais complexo desenvolvido, foi projetado para simular mecânicas de furtividade. Os elementos estáticos do ambiente consistem em caixas e contêineres, que servem como obstáculos e cobertura para o jogador conforme mostrado na Figura 5.

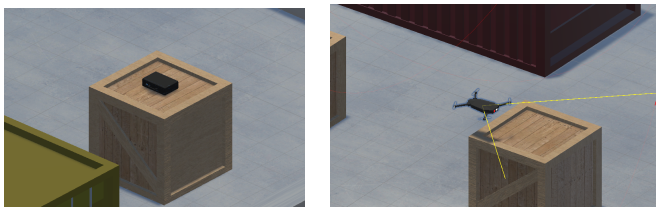
Figura 5 – Ambiente Sneaking Scape.



Fonte: Acervo do Autor.

O objetivo principal do jogador é coletar uma maleta e, em seguida, sair do local através de uma porta. No entanto, o ambiente é patrulhado por drones que se movimentam constantemente. Para ter sucesso, o agente deve evitar ser visto pelos drones, utilizando as caixas e contêineres para se esconder e planejar seus movimentos com cuidado como mostrado na Figura 6.

Figura 6 – Maleta e Drone.



(a) (b)

Fonte: acervo do autor.

3.3 Agentes Modelados

Neste tópico, serão apresentados inicialmente os componentes comuns do ML-Agents, Em seguida, serão descritas as propriedades particulares de cada agente desenvolvido, destacando suas características específicas, comportamentos esperados e como cada um deles foi configurado para realizar suas respectivas tarefas.

3.3.1 Componentes ML-Agents

Os componentes comuns do ML-Agents são fundamentais para o treinamento e controle no ambiente de aprendizado por reforço na Unity. Eles são utilizados em todos os agentes deste trabalho e proporcionam a estrutura básica para o desenvolvimento de comportamentos inteligentes e adaptativos dos agentes em seus respectivos ambientes.

3.3.1.1 *Behavior Parameters*

Esse componente tem como função ajustar as características de interação com o ambiente e como o agente aprende a partir dessas interações. Ele define os parâmetros do comportamento dos agentes, o tipo de espaço de ações, as dimensões das observações e o nome do comportamento. Também permite configurar o uso de políticas de aprendizado, seja para treinamento ou inferência, especificando qual modelo de AR o agente deve utilizar.

3.3.1.2 *Agent*

O *Agent* é o componente central que define o comportamento do agente dentro do ambiente. Ele gerencia a coleta de observações, a execução de ações e o recebimento de recompensas. É responsável por integrar todas as partes do aprendizado, tornando-se o núcleo do treinamento. Dentro desse *script*, métodos como *OnEpisodeBegin*, *CollectObservations*, *OnActionReceived* e *Heuristic* são implementados para personalizar o comportamento do agente durante o treinamento.

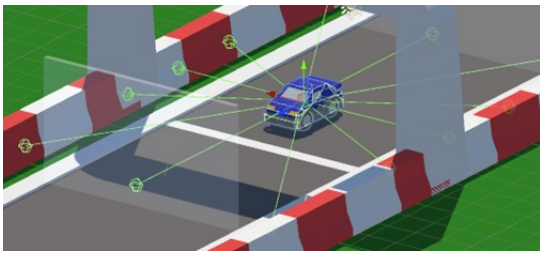
3.3.1.3 *Decision Requester*

O *Decision Requester* é um componente que automatiza a solicitação de decisões do agente em intervalos fixos, definidos pelo número de passos. Com ele é possível especificar a frequência com que o agente deve tomar decisões, sem a necessidade de solicitar manualmente uma decisão a cada frame. É útil para sincronizar as decisões com a física do ambiente e para otimizar o processo de treinamento.

3.3.1.4 *Ray Perception Sensor 3D*

O *Ray Perception Sensor3D*, mostrado na figura 7, é um sensor que permite ao agente perceber o ambiente ao seu redor usando raios (*raycasts*). Esse componente simula a visão do agente, detectando objetos dentro de um alcance especificado e coletando informações como a distância até os obstáculos, os nomes dos objetos detectados e as direções relativas. As observações geradas pelo *Ray Perception Sensor3D* são essenciais para que o agente possa tomar decisões informadas, baseadas em sua percepção do ambiente.

Figura 7 – Ray Perception Sensor 3D.



Fonte: Acervo do Autor.

No total, quatro agentes foram desenvolvidos, cada um com propostas que envolvem mecânicas amplamente utilizadas em diferentes gêneros de jogos. Esses agentes incluem o controle de veículos em contextos de corrida e estacionamento, combates em jogos de luta com movimentação 3D e cenários que exploram mecânicas de furtividade (*stealth*), com cada agente adaptado para suas respectivas tarefas e ambientes.

3.3.2 Componentes Específicos

Os componentes específicos de cada agente possuem uma configuração própria, adaptada ao seu ambiente e objetivos, o que inclui ajustes em sensores, funções de recompensa, e mecanismos de interação. Eles foram implementados nos cenários desenvolvidos neste trabalho a fim de otimizar o desempenho, sendo importantes para que cada agente alcançasse um comportamento desejado, possibilitando um comportamento coerente com as dinâmicas e os desafios de cada tarefa.

3.3.2.1 *Racer Agent*

O *Racer Agent* é um agente treinado para executar tarefas no *Turbo Racing*. Sua função principal é controlar o carro e completar o percurso no menor tempo possível, evitando colisões com os limites da pista. Além dos componentes comuns do ML-Agents, o *Racer Agent* conta com os seguintes componentes específicos: i) *Rigidbody*, um componente da Unity que possibilita que o carro seja afetado pelas propriedades físicas do ambiente, como gravidade, colisões e interações dinâmicas, garantindo que o comportamento do veículo seja realista; e ii) Controlador do Carro, um *script* personalizado que permite ao agente executar ações como acelerar, frear e virar à esquerda ou à direita. Esse controlador é fundamental para que o agente possa reagir de maneira adequada às observações feitas durante o percurso, ajustando suas ações de acordo com as necessidades da corrida.

3.3.2.2 *Boxer Agent*

O *Boxer Agent* é um agente projetado para executar tarefas de combate no *Fighting Warrior*. Além dos componentes comuns do ML-Agents, o *Boxer Agent* conta com os seguintes componentes específicos: i) *Rigidbody*, componente de física da Unity que permite que o agente seja afetado pelas leis da física do ambiente, como colisões e forças, garantindo um comportamento realista durante os movimentos de combate; ii) *Animation Manager*, esse componente controla as animações dos ataques realizados pelo agente após cada ação, sincronizando as animações com os movimentos e garantindo que as transições de ataque ocorram de maneira fluida; iii) *Player Collision*, é responsável pela gestão das colisões no ambiente, identifica qual parte do corpo do agente entrou em contato com o oponente e determina onde o agente sofreu um golpe. O *Boxer Agent* possui *hitboxes* separadas para os braços, tronco e cabeça, o que permite um controle detalhado sobre o impacto dos golpes e as áreas atingidas; iv) *Boxer*, é o componente que gerencia as propriedades vitais do agente, como quantidade de vida, estamina e velocidade. Ele mantém o controle sobre o status geral do agente, influenciando seu desempenho no combate.

3.3.2.3 *Parking Agent*

O *Parking Agent* possui componentes semelhantes aos do *Racer Agent*, porém com algumas particularidades que o tornam mais adequado para a simulação do mundo real em um contexto de estacionamento. As propriedades físicas do agente foram ajustadas para refletir com maior precisão o comportamento de um carro real, especialmente durante manobras de estacionamento. Também inclui uma função de freio de mão (*Handbrake*) implementada no seu controlador, que permite ao carro permanecer estacionado de maneira estável após concluir a tarefa. Esse detalhe é fundamental para o ambiente de estacionamento, onde o agente precisa garantir que o carro pare completamente e permaneça na posição correta.

Assim como o *Racer Agent*, utiliza o *Ray Perception Sensor 3D* para detectar obstáculos no ambiente. No entanto, uma característica exclusiva do *Parking Agent* é o uso do *Camera*

Sensor, um sensor disponibilizado pelo ML-Agents que permite ao agente receber observações visuais, capturando imagens do ambiente para auxiliar no treinamento. Isso possibilita que o agente utilize dados visuais, além das observações tradicionais, para tomar decisões com base na percepção visual do ambiente.

3.3.2.4 *Stealth Agent*

O *Stealth Agent* realiza tarefas furtivas no *Sneaking Enviroment*. Assim como outros objetos físicos da Unity, ele possui um componente *Rigidbody*, que aplica as propriedades físicas do ambiente ao agente, garantindo interações realistas com o mundo ao seu redor. Além disso, o *Stealth Agent* conta com um componente chamado *Player*, responsável por gerenciar a movimentação global do agente.

Embora seu objetivo seja realizar uma tarefa complexa de furtividade, envolvendo a coleta de itens e a evasão de drones, ele é o agente mais simples em termos de configuração dentro do contexto da Unity. O design enxuto do *Stealth Agent* reflete a necessidade de foco na estratégia de movimentação e evasão, sem a necessidade de controles complexos ou sensores adicionais.

3.3.3 Espaço de Ação

O espaço de ação representa o conjunto de todas as ações que um agente pode realizar em um ambiente de aprendizado por reforço. Ele pode ser definido como discreto ou contínuo, dependendo da natureza da tarefa. No espaço de ação discreto, o agente escolhe entre um número finito de ações pré-definidas, como mover-se para frente, virar ou saltar. Já no espaço de ação contínuo, as ações são representadas por valores reais, permitindo decisões mais finas e graduais, como ajustar a velocidade ou a direção de um movimento com precisão. Em ambientes como o Unity ML-Agents, essa distinção é fundamental para configurar corretamente o comportamento do agente e garantir que suas ações sejam compatíveis com a dinâmica do ambiente.

Nos agentes *Racing Agent* e *Parking Agent*, é necessário definir um espaço de ação compatível com a lógica de controle de um veículo. As ações desses agentes incluem acelerar, frear, virar à direita, virar à esquerda e, no caso do ParkingAgent, utilizar o freio de mão. O tipo de espaço de ação deve permitir que o agente execute essas ações de forma gradual, pois é essencial controlar valores reais de aceleração, frenagem e movimentação do volante com precisão. Portanto, nesses casos, trata-se de um espaço de ação contínuo.

Por outro lado, os demais agentes executam apenas ações predefinidas. O *Boxing Agent* possui quatro ações de movimentação, uma ação de defesa e outras cinco animações de ataque, totalizando dez ações distintas. Já o *Stealth Agent* conta apenas com ações de movimentação, somando quatro ações disponíveis. Como esses agentes operam com um conjunto finito e enumerável de ações, seu espaço de ação é do tipo discreto. Nesse tipo de espaço, o agente

seleciona uma entre várias opções fixas, o que é adequado para tarefas em que não há necessidade de controle contínuo ou graduado das ações.

3.4 Algoritmo de Aprendizado por Reforço

O algoritmo escolhido para o treinamento dos agentes neste trabalho foi o *Proximal Policy Optimization*. O PPO é um dos algoritmos mais utilizados em AR devido à sua capacidade de fornecer estabilidade e eficiência mesmo em ambientes complexos (YU et al., 2022; LARSEN et al., 2021).

Ao contrário de métodos tradicionais, que podem causar grandes variações nas políticas aprendidas, o PPO mantém as atualizações dentro de um limite controlado, evitando mudanças drásticas que possam prejudicar o processo de aprendizado. Isso é feito através de um mecanismo de limitação das mudanças na política, o que torna o treinamento mais estável (YU et al., 2022; LARSEN et al., 2021).

Por possuir a capacidade de equilibrar desempenho, facilidade de implementação e possuir ótima performance em ambientes tridimensionais complexos, como os simulados neste trabalho, ele se mostrou adequado para a diversidade de tarefas envolvidas, Desafios que envolvem desde controle preciso de veículos em corridas até mecânicas de combate e furtividade (YU et al., 2022; LARSEN et al., 2021).

3.5 Configurações de Treinamento

Parte da complexidade em modelar agentes inteligentes reside na definição precisa das recompensas que guiam seu comportamento. As tarefas onde o AR obteve grande sucesso apresentam um detalhe em comum: todas possuem uma função de recompensa bem definida. No entanto, tarefas de controle contínuo apresentam um desafio maior na modelagem dessas recompensas (IBRAHIM et al., 2024; ABOUELAZM; MICHEL; ZOELLNER, 2024).

As recompensas desempenham um papel fundamental no AR, pois são elas que incentivam os agentes a realizar ações específicas em busca da maximização dos resultados. Entretanto, a eficácia dessas recompensas vai além da simples maximização; elas precisam ser cuidadosamente projetadas para garantir que, ao buscar maximizar sua recompensa, o agente também resolva a tarefa proposta de maneira eficiente (IBRAHIM et al., 2024; ABOUELAZM; MICHEL; ZOELLNER, 2024).

A precisão na modelagem das funções de recompensa é crucial para o sucesso do aprendizado e para que o agente alcance seus objetivos de forma satisfatória. Essas funções de recompensa variam de acordo com a natureza da tarefa a ser executada por cada agente, o que torna o processo de design ainda mais desafiador. A seguir, serão descritas as funções de recompensa específicas para cada um dos agentes modelados neste trabalho, destacando como

essas recompensas influenciam o comportamento e o sucesso de cada agente (IBRAHIM et al., 2024; ABOUELAZM; MICHEL; ZOELLNER, 2024).

3.5.1 *Racer Agent*

O *Racer Agent* recebe recompensas de acordo com o progresso feito ao longo da pista. A cada movimento na direção correta e mantendo a orientação ideal, o agente é recompensado. Completar a volta rapidamente e sem colisões gera uma recompensa adicional. Penalidades são aplicadas sempre que o agente colide com as barreiras ou sai da pista.

3.5.2 *Boxer Agent*

Para o *Boxer Agent*, as recompensas são baseadas em infligir dano ao oponente e evitar receber golpes. O agente é recompensado ao acertar o oponente, com valores maiores atribuídos a golpes críticos, como aqueles na cabeça. Por outro lado, penalidades são aplicadas se o agente sofrer dano, especialmente em áreas vulneráveis, como a cabeça. Além disso, uma pequena recompensa é atribuída por manter distância dos limites da arena, com o objetivo de incentivá-lo a evitar ficar encurralado.

3.5.3 *Parking Agent*

O *Parking Agent* recebe recompensas à medida que se aproxima da vaga de estacionamento correta, estaciona sem colidir e na orientação adequada. As recompensas são concedidas progressivamente conforme o agente se posiciona corretamente na vaga. Penalidades são aplicadas por colisões com obstáculos e por tentativas de estacionamento incorretas. Uma penalidade por tempo foi incluída com o objetivo de incentivar o agente a reduzir o tempo gasto na tarefa.

3.5.4 *Stealth Agent*

O *Stealth Agent* é recompensado por evitar ser detectado pelos drones, coletar a maleta e, em seguida, extrair do mapa. Cada movimento furtivo bem-sucedido aumenta sua recompensa, enquanto a detecção pelos drones ou a tentativa de deixar o local da missão antes de coletar a maleta resulta em penalidades. O agente também é recompensado por realizar a tarefa no menor tempo possível.

As Tabelas 1 e 2, apresentadas a seguir, ilustram a forma como as recompensas e punições foram estruturadas nos ambientes de treinamento. Foi cuidadosamente planejada para guiar o agente no processo de aprendizado, incentivando comportamentos desejáveis e desencorajando ações indesejadas. Os valores atribuídos a cada ação desempenham um papel crucial na convergência do aprendizado e na obtenção de uma política eficaz.

Tabela 1 – Recompensas

Racer Agent	**Parking Agent**	**Boxer Agent**	**Boxer Agent**
Manter orientação	Manter orientação	Acertar ataques	Coletar Maleta
Coletar checkpoints	Encontrar vaga disponível	Defender ataque	Extrair do mapa
Permanecer na pista	Estacionar	Nocautear adversário	Tempo de extração

Tabela 2 – Punições

Racer Agent	**Parking Agent**	**Boxer Agent**	**Boxer Agent**
Colidir com obstáculos	Colidir com carros	Receber ataques	Ser detectado
Sentido errado	Colidir com obstáculos	Não atacar	Tempo gasto
Tempo de corrida	Tempo gasto	Ser nocauteado	Distância dos objetivos

3.6 Métricas de Avaliação

Para avaliar o desempenho dos agentes ao longo do treinamento, foram utilizadas métricas extraídas diretamente do *TensorBoard*, uma ferramenta de visualização amplamente utilizada no AR. A principal métrica considerada foi o número de passos realizados pelo agente, que reflete o tempo necessário para concluir uma tarefa ou atingir um objetivo específico dentro do ambiente.

Além do número de passos, outras métricas disponibilizadas pelo *TensorBoard* foram monitoradas, como a recompensa acumulada ao longo do tempo, a taxa de sucesso em completar as tarefas e a estabilidade do agente em manter um comportamento desejado. Essas métricas ajudam a identificar áreas que podem necessitar de ajustes no processo de treinamento.

3.6.1 *Cumulative Reward*

Cumulative Reward indica a recompensa total acumulada pelo agente ao longo dos episódios. Essa métrica é útil para avaliar o progresso geral do agente, visto que recompensas positivas significam que ele está tomando ações corretas. Ao observar a evolução da *Cumulative Reward* ao longo do tempo, é possível identificar se o agente está realmente aprendendo a tarefa ou se está preso em uma estratégia ineficiente. Um aumento contínuo nessa métrica sugere um bom aprendizado, enquanto estagnações ou quedas podem indicar a necessidade de ajustes na função de recompensa ou na política de treinamento.

3.6.2 *Value Loss*

O *Value Loss* representa a diferença entre a recompensa estimada pelo agente e a recompensa real recebida. Ele mede o quão bem o modelo está prevendo o valor esperado das recompensas futuras. Valores altos de *Value Loss* indicam que o modelo está com dificuldades para prever corretamente essas recompensas, o que pode prejudicar o processo de tomada de decisão. Idealmente, o *Value Loss* deve diminuir à medida que o agente aprende a tarefa e melhora suas previsões. A análise dessa métrica pode ajudar a identificar se o treinamento está

progredindo conforme esperado ou se ajustes são necessários, como modificar hiperparâmetros ou refinar a função de recompensa.

3.6.3 *Episode Length*

O *Episode Length* mede a duração dos episódios, ou seja, quantos passos o agente leva para completar a tarefa ou falhar. Episódios mais longos podem indicar que o agente está explorando o ambiente de forma ineficiente, enquanto episódios mais curtos, combinados com recompensas altas, sugerem que o agente está aprendendo a realizar a tarefa de forma mais rápida e eficiente. No caso do *Car Agent*, uma redução no número de passos, juntamente com um aumento nas recompensas acumuladas, seria um forte indicativo de que o agente está se aproximando da solução ótima.

3.6.4 Taxa de Vitória

A Taxa de Vitória mede a quantidade de vezes que o agente conseguiu superar a tarefa para a qual foi treinado. Essa métrica é particularmente útil para verificar se o agente está, de fato, atingindo seus objetivos dentro do ambiente. Uma alta taxa de vitória indica que o agente treinado está bem adaptado ao ambiente e conseguindo completar suas tarefas de forma consistente.

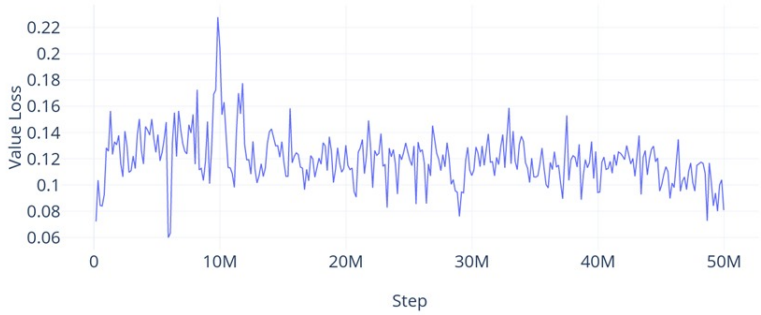
4 ANÁLISE E DISCUSSÃO DOS RESULTADOS:

Neste capítulo, serão apresentados os resultados obtidos durante o processo de treinamento dos agentes. As principais métricas coletadas ao longo do treinamento serão avaliadas em detalhes, permitindo uma análise do desempenho de cada agente em seus respectivos ambientes.

4.1 Resultados *Racer Agent*

Nas últimas etapas do treinamento, o *Racer Agent* apresentou um avanço significativo em sua capacidade de estimar corretamente as recompensas, mesmo com as oscilações causadas pela complexidade da tarefa, conforme ilustrado no gráfico apresentado na Figura 8.

Figura 8 – Racer Agent Value Loss.



Fonte: Acervo do Autor.

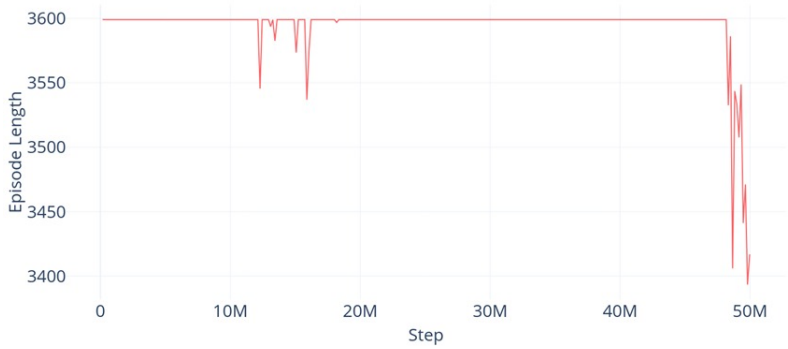
O modelo ajustou suas previsões de retorno futuro com mais precisão à medida que o treinamento progrediu. A redução consistente no *Value Loss* evidencia a eficácia do algoritmo na otimização do comportamento do agente, resultando em previsões mais ajustadas às dinâmicas do ambiente.

No ambiente de corrida, episódios longos são comuns, uma vez que o agente ainda está aprendendo a navegar pelo trajeto de maneira eficiente. Ao passo que o treinamento avança, o agente torna-se progressivamente mais rápido em concluir o percurso. Isso resulta em uma redução significativa nos valores do *Episode Length* nas etapas finais, conforme ilustrado no gráfico da Figura 9, pois o agente otimizou seu comportamento para completar o percurso de forma mais ágil e eficiente.

O aumento progressivo da recompensa acumulada ao longo do treinamento reforça a convergência satisfatória do modelo. O crescimento contínuo da *Cumulative Reward*, como mostrado no gráfico da Figura 10, indica que o agente está aprendendo a maximizar suas recompensas com eficácia.

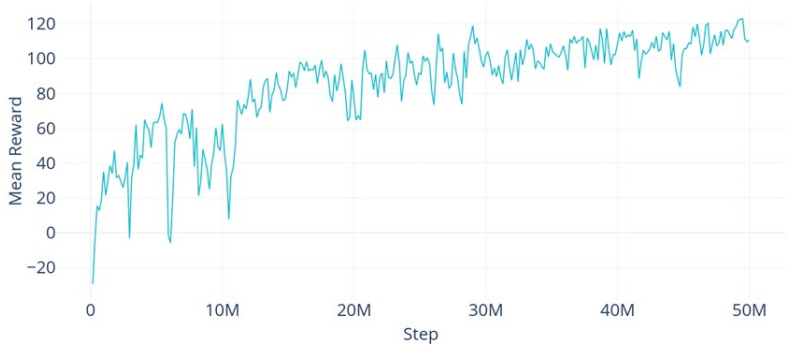
Esse resultado reflete uma adaptação crescente ao ambiente, permitindo que o agente atinja seu objetivo de forma mais consistente e bem-sucedida.

Figura 9 – Racer Agent Episode Length.



Fonte: Acervo do Autor.

Figura 10 – Racer Agent Mean Reward.

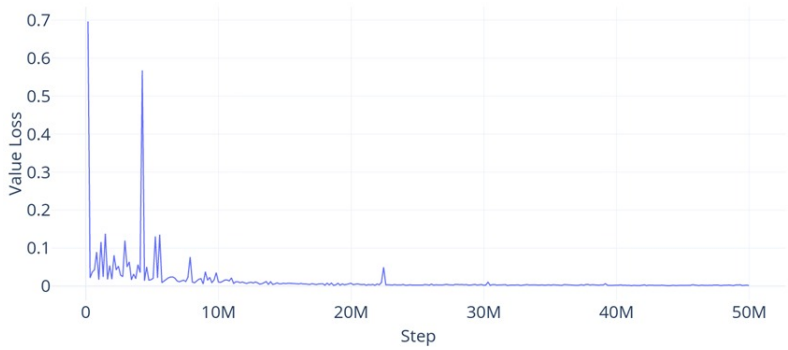


Fonte: Acervo do Autor.

4.2 Resultados *Boxer Agent*

No caso do Boxer Agent, o *Value Loss*, como mostrado no gráfico da Figura 11, apresentou uma regressão ao longo do treinamento.

Figura 11 – Boxer Agent Value Loss.

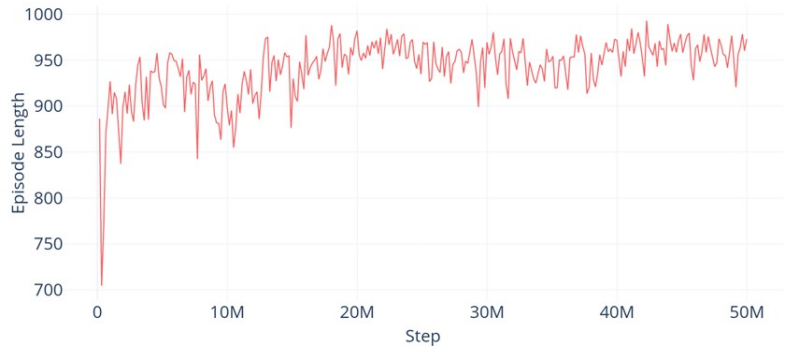


Fonte: Acervo do Autor.

Esse comportamento deve-se à menor complexidade do ambiente, que facilitou o refinamento das previsões. Com uma tarefa relativamente mais simples, o agente ajustou suas estimativas de recompensa de forma mais rápida e eficiente, resultando em um comportamento consistente e previsível.

Como o ambiente não impõe restrições de tempo para concluir a tarefa, o tempo entre os episódios não precisa diminuir necessariamente durante o treinamento. O agente não foi estimulado a minimizar o número de passos, o que explica os valores altos de *Episode Length*, conforme ilustrado no gráfico apresentado na Figura 12.

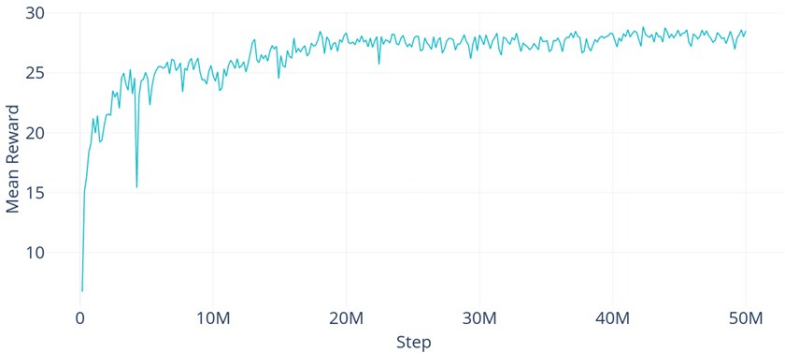
Figura 12 – Boxer Agent Episode Length.



Fonte: Acervo do Autor.

A recompensa acumulada cresce progressivamente e atinge estabilidade ao longo do treinamento. Devido à simplicidade do ambiente, o agente atinge a convergência rapidamente, por volta de 20 milhões de passos, e não apresenta ganhos adicionais significativos no desempenho a partir desse ponto, conforme exibido no gráfico da Figura 13.

Figura 13 – Boxer Agent Mean Reward.

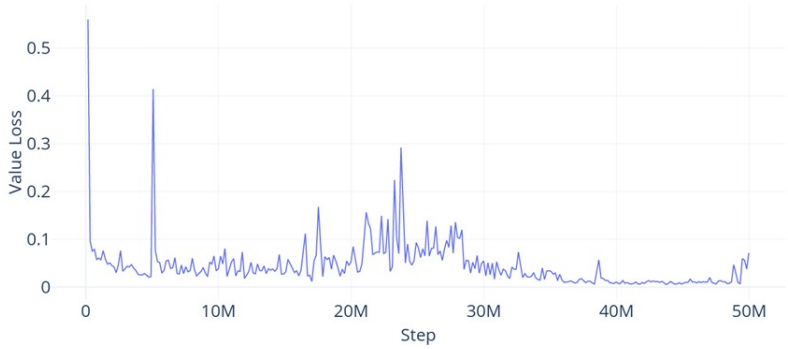


Fonte: Acervo do Autor.

4.3 Resultados *Parking Agent*

Durante o treinamento, o Parking Agent apresentou uma redução consistente no *Value Loss*, conforme ilustrado na Figura 14, indicando um progresso contínuo no refinamento das previsões.

Figura 14 – Parking Agent Value Loss.

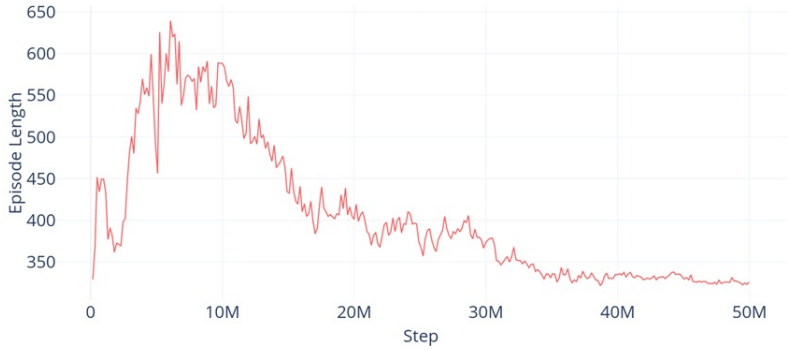


Fonte: Acervo do Autor.

Embora pequenas oscilações tenham sido observadas, essas variações são normais em processos de AR e não impactaram negativamente o desempenho geral do agente. A tendência de queda no *Value Loss* ao longo do tempo reflete a evolução positiva do agente.

O *Episode Length*, mostrado na Figura 15, diminuiu progressivamente à medida que o agente aprendeu a completar a tarefa de estacionar o veículo de forma mais eficiente, dentro do tempo estipulado.

Figura 15 – Parking Agent Episode Length.

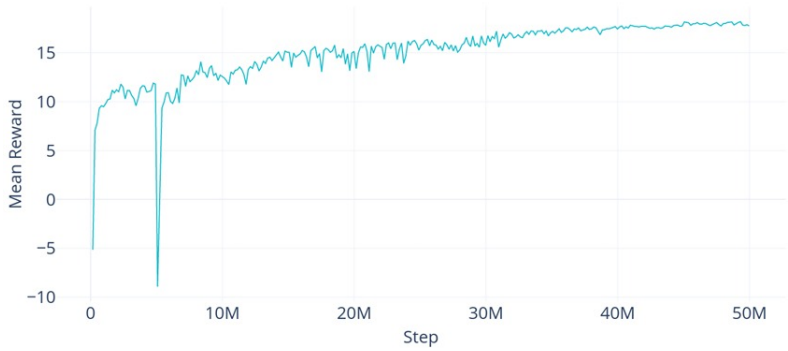


Fonte: Acervo do Autor.

Conforme o treinamento avançava, o agente passou a finalizar os episódios em menos passos, otimizando suas ações e reduzindo o tempo necessário para estacionar corretamente. Além disso, executar a tarefa o mais rápido possível contribui para aumentar a pontuação final do agente, reforçando seu desempenho dentro do ambiente.

A recompensa acumulada também aumentou de forma gradual durante o treinamento, sugerindo que o agente estava maximizando suas recompensas ao realizar a tarefa de maneira mais rápida e precisa. Esse aumento contínuo no *Cumulative Reward*, exibido no gráfico da Figura 16, indica que o agente convergiu para um comportamento otimizado, tornando-se cada vez mais eficaz em atingir o objetivo de estacionar o carro adequadamente no ambiente.

Figura 16 – Parking Agent Mean Reward.

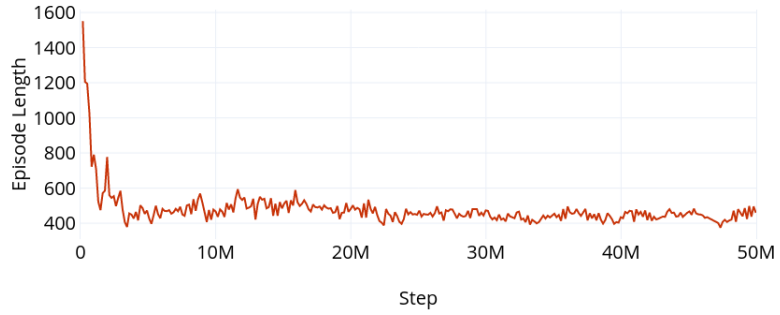


Fonte: Acervo do Autor.

4.4 Resultados *Stealth Agent*

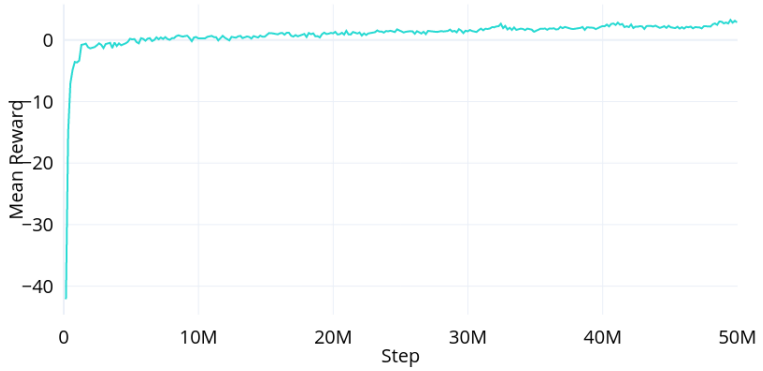
No *Stealth Agent*, é possível observar que o agente dimunui drasticamente o tamanho dos episódios e mantém os valores estáveis com o passar do tempo, como mostrado no gráfico da Figura 17. Mesmo realizando tarefas complexas em sequência, o agente consegue manter a estabilidade das recompensas acumuladas, conforme ilustrado no gráfico de *Mean Reward*, na Figura 18.

Figura 17 – Stealth Agent Episode Length.



Fonte: Acervo do Autor.

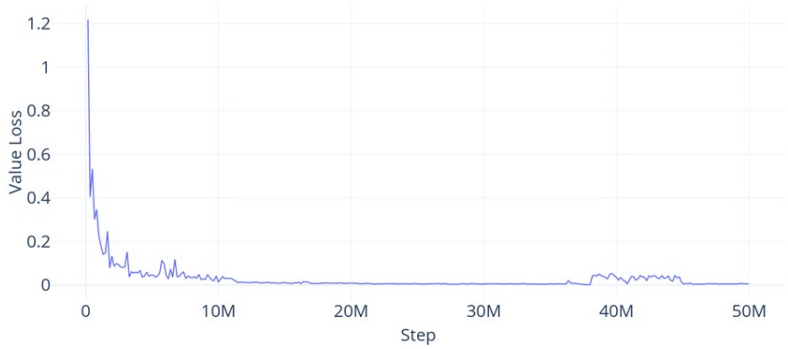
Figura 18 – Stealth Agent Mean Reward.



Fonte: Acervo do Autor.

O agente precisa realizar duas tarefas distintas dentro do ambiente. A primeira, "coletar a maleta", é aprendida inicialmente. No entanto, ao concluir essa etapa, o agente enfrenta um novo desafio, que é "extrair do local da missão". A pequena oscilação no *Value Loss*, como exibido na Figura 19, ocorre porque o modelo ainda está ajustando suas previsões para a segunda fase da tarefa, mas logo os valores se estabilizam.

Figura 19 – Stealth Agent Value Loss.



Fonte: Acervo do Autor.

À medida que o treinamento progride, o agente domina as tarefas de coleta e de extração, estabilizando suas métricas. A recompensa acumulada e o número de episódios indicam que o agente se adapta rapidamente à nova dinâmica do ambiente, mesmo precisando realizar diversas tarefas, como evitar ser detectado, coletar a maleta e retirar-se do local da missão.

4.5 Taxa de Vitória

A taxa de vitória reflete a quantidade de vezes que cada agente conseguiu concluir sua tarefa com sucesso dentro de seu respectivo ambiente. Para padronizar a avaliação, todos os agentes foram testados em 1000 tentativas, permitindo uma comparação direta do desempenho

nos cenários propostos. As respectivas taxas de vitória de cada agente, calculadas com base nesses testes, estão descritas na Tabela 3.

Tabela 3 – Taxa de Vitória

Agente	Acertos/Tentativas	Taxa de Vitória
Racer Agent	987/1000	98,7%
Boxer Agent	923/1000	92,3%
Parking Agent	968/1000	96,8%
Stealth Agent	874/1000	87,4%

Em todos os cenários testados, os agentes apresentaram desempenhos robustos, com uma taxa de vitória mínima de 87,4%. A precisão ao realizar suas respectivas tarefas foi notável, demonstrando a eficiência dos algoritmos e das políticas treinadas. Mesmo nos ambientes mais desafiadores, os agentes conseguiram aprender e generalizar bem as estratégias necessárias para alcançar os objetivos propostos.

Embora os resultados obtidos com a aplicação do algoritmo PPO tenham demonstrado alta taxa de sucesso e estabilidade nos ambientes propostos, é importante destacar algumas limitações observadas. O PPO tende a apresentar desempenho consistente em tarefas com recompensas bem definidas e feedbacks frequentes, como observado no *Parking Agent* e *BoxingAgent*. No entanto, em ambientes mais complexos e com múltiplas fases, o algoritmo pode enfrentar dificuldades para estabilizar o aprendizado. Apesar da robustez geral do PPO, ele pode ter desempenho sensível à estrutura da função de recompensa, à qualidade das observações sensoriais e à natureza sequencial das tarefas. Tais limitações indicam oportunidades para explorar variantes do algoritmo ou estratégias complementares que melhorem sua adaptabilidade.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresentou o desenvolvimento e treinamento de diversos agentes em ambientes complexos, utilizando algoritmos de aprendizado por reforço profundo. Cada agente foi submetido a cenários específicos e desafiadores, e os resultados obtidos evidenciaram a eficácia dos métodos empregados na otimização de seus comportamentos.

Os agentes mostraram a robustez de suas políticas treinadas em ambientes de maior complexidade, como o Parking Agent, que foi capaz de refinar progressivamente suas previsões e ações, tornando-se mais eficiente na tarefa de estacionar. O Stealth Agent, por sua vez, enfrentou um ambiente composto por duas fases distintas e, inicialmente, oscilou em seu desempenho. Contudo, o agente conseguiu estabilizar suas métricas à medida que aprendia a realizar a coleta e a extração de forma eficaz.

A análise da taxa de vitória demonstra que todos os agentes foram capazes de realizar suas tarefas com grande precisão, com uma taxa mínima de sucesso de 87,4% em 1000 tentativas, mesmo em cenários com desafios significativos. Isso comprova a eficácia dos algoritmos utilizados e mostra a capacidade dos agentes de generalizar estratégias e alcançar seus objetivos em diferentes situações.

Como perspectivas futuras, sugere-se expandir os cenários para incluir ambientes onde múltiplos agentes interajam de forma colaborativa ou competitiva, criando situações mais complexas que exijam coordenação, cooperação ou antagonismo entre eles. Essa abordagem poderia introduzir desafios adicionais, como a necessidade de os agentes aprenderem a dividir recursos ou superar oponentes, o que enriqueceria o processo de aprendizado.

Sobre PPO, propõe-se a adoção de técnicas que aprimorem sua estabilidade e eficiência durante o treinamento. Uma dessas estratégias é a utilização de clipping adaptativo, no qual o valor do hiperparâmetro é ajustado dinamicamente com base na divergência KL observada entre políticas sucessivas. Isso permite um controle mais refinado sobre o quanto a nova política pode se distanciar da anterior, evitando atualizações excessivamente conservadoras ou agressivas. Outra proposta é a normalização da vantagem estimada, o que consiste em padronizar os valores da função vantagem antes de serem usados na função objetivo. Essa abordagem reduz a variância dos gradientes, contribuindo para um aprendizado mais estável e eficiente.

Além disso, propõe-se a avaliação da capacidade de generalização dos agentes treinados em diferentes contextos, investigando sua habilidade de transferir conhecimento adquirido em um cenário para novos ambientes com regras ou layouts distintos. Outra direção promissora é a experimentação com algoritmos alternativos ao PPO, como o Soft Actor-Critic (SAC) ou o TD3, que oferecem vantagens em estabilidade e eficiência em tarefas com espaço de ação contínuo.

A integração dessas abordagens pode potencialmente acelerar o processo de treinamento e permitir que os agentes aprendam a colaborar ou competir em situações mais realistas. Espera-se

que os avanços apresentados neste trabalho contribuam para o desenvolvimento de agentes inteligentes e para o avanço de novas tecnologias relacionadas aos jogos digitais.

Referências

ABOUELAZM, A.; MICHEL, J.; ZOELLNER, J. M. A review of reward functions for reinforcement learning in the context of autonomous driving. *arXiv preprint arXiv:2405.01440*, 2024. Citado 2 vezes nas páginas 38 e 39.

ADEL, M.; GRIFFITH, L. C. The role of dopamine in associative learning in drosophila: an updated unified model. *Neuroscience Bulletin*, Springer, v. 37, n. 6, p. 831–852, 2021. Citado 3 vezes nas páginas 12, 17 e 18.

AHILAN, S. A succinct summary of reinforcement learning. *arXiv preprint arXiv:2301.01379*, 2023. Citado na página 17.

AL-HAMADANI, M. N. A.; FADHEL, M. A.; ALZUBAIDI, L.; HARANGI, B. Reinforcement learning algorithms and applications in healthcare and robotics: A comprehensive and systematic review. *Sensors*, v. 24, n. 8, 2024. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/24/8/2461>. Citado na página 12.

ANDERSEN, P.-A.; GOODWIN, M.; GRANMO, O.-C. Deep rts: A game environment for deep reinforcement learning in real-time strategy games. In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. [S.l.: s.n.], 2018. p. 1–8. Citado 5 vezes nas páginas 16, 26, 27, 28 e 29.

BABAEIZADEH, M.; FROSIO, I.; TYREE, S.; CLEMONS, J.; KAUTZ, J. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*, 2016. Citado 2 vezes nas páginas 21 e 23.

BAO, Y.; GONG, W.; YANG, K. A literature review of human–ai synergy in decision making: From the perspective of affordance actualization theory. *Systems*, v. 11, n. 9, 2023. ISSN 2079-8954. Disponível em: <https://www.mdpi.com/2079-8954/11/9/442>. Citado na página 15.

BOTELHO, J. M.; CRUZ, V. A. G. Metodologia científica. *São Paulo: Pierson Education do Brasil*, 2013. Citado na página 13.

BROWN, P.; JENKINS, H. Autoshaping of the pigeon's key peck. *Journal of the experimental analysis of behavior*, v. 11, p. 1–8, 02 1968. Citado na página 12.

CHEN, P. P.; GUITART, A.; RíO, A. F. del; PERIáñEZ, Customer lifetime value in video games using deep learning and parametric models. In: *2018 IEEE International Conference on Big Data (Big Data)*. [S.l.: s.n.], 2018. p. 2134–2140. Citado na página 12.

CHEN, T.; LIU, J.-Q.; LI, H.; WANG, S.-R.; NIU, W.-J.; TONG, E.-D.; CHANG, L.; CHEN, Q. A.; LI, G. Robustness assessment of asynchronous advantage actor-critic based on dynamic skewness and sparseness computation: A parallel computing view. *Journal of Computer Science and Technology*, Springer, v. 36, p. 1002–1021, 2021. Citado na página 23.

DAYAN, P.; WATKINS, C. Q-learning. *Machine learning*, v. 8, n. 3, p. 279–292, 1992. Citado 2 vezes nas páginas 18 e 19.

DEWANTO, V.; GALLAGHER, M. Examining average and discounted reward optimality criteria in reinforcement learning. In: SPRINGER. *Australasian Joint Conference on Artificial Intelligence*. [S.l.], 2022. p. 800–813. Citado na página 17.

DULAC-ARNOLD, G.; LEVINE, N.; MANKOWITZ, D. J.; LI, J.; PADURARU, C.; GOWAL, S.; HESTER, T. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, Springer, v. 110, n. 9, p. 2419–2468, 2021. Citado 2 vezes nas páginas 15 e 16.

ENGSTROM, L.; ILYAS, A.; SANTURKAR, S.; TSIPRAS, D.; JANOOS, F.; RUDOLPH, L.; MADRY, A. Implementation matters in deep rl: A case study on ppo and trpo. In: *International Conference on Learning Representations*. [s.n.], 2020. Disponível em: <https://openreview.net/forum?id=r1etN1rtPB>. Citado na página 25.

FAN, J.; WANG, Z.; XIE, Y.; YANG, Z. A theoretical analysis of deep q-learning. In: BAYEN, A. M.; JADBABAIE, A.; PAPPAS, G.; PARRILO, P. A.; RECHT, B.; TOMLIN, C.; ZEILINGER, M. (Ed.). *Proceedings of the 2nd Conference on Learning for Dynamics and Control*. PMLR, 2020. (Proceedings of Machine Learning Research, v. 120), p. 486–489. Disponível em: <https://proceedings.mlr.press/v120/yang20a.html>. Citado 2 vezes nas páginas 19 e 20.

GAO, X.; LIU, J.; WAN, B.; AN, L. Hierarchical reinforcement learning from demonstration via reachability-based reward shaping. *Neural Processing Letters*, Springer, v. 56, n. 3, p. 184, 2024. Citado na página 16.

GIL, A. C. et al. *Como elaborar projetos de pesquisa*. [S.l.]: atlas São Paulo, 2002. v. 4. Citado na página 13.

GOTTWALD, M.; GRONAUER, S.; SHEN, H.; DIEPOLD, K. Analysis and optimisation of bellman residual errors with neural function approximation. *arXiv preprint arXiv:2106.08774*, 2021. Citado na página 16.

HASSELT, H. Double q-learning. *Advances in neural information processing systems*, v. 23, 2010. Citado na página 19.

HASSELT, H. van; GUEZ, A.; SILVER, D. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 30, n. 1, Mar. 2016. Disponível em: <https://ojs.aaai.org/index.php/AAAI/article/view/10295>. Citado na página 20.

HU, M. Advanced policy gradient methods. In: *The Art of Reinforcement Learning: Fundamentals, Mathematics, and Implementations with Python*. [S.l.]: Springer, 2023. p. 205–220. Citado na página 21.

IBRAHIM, S.; MOSTAFA, M.; JNADI, A.; SALLOUM, H.; OSINENKO, P. Comprehensive overview of reward engineering and shaping in advancing reinforcement learning applications. *IEEE Access*, v. 12, p. 175473–175500, 2024. Citado 2 vezes nas páginas 38 e 39.

JANG, B.; KIM, M.; HARERIMANA, G.; KIM, J. W. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, v. 7, p. 133653–133667, 2019. Citado 4 vezes nas páginas 12, 15, 16 e 19.

JULIANI, A. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018. Citado na página 29.

JUSTESEN, N.; BONTRAGER, P.; TOGELIUS, J.; RISI, S. Deep learning for video game playing. *IEEE Transactions on Games*, v. 12, n. 1, p. 1–20, 2020. Citado 4 vezes nas páginas 12, 27, 28 e 29.

KESIM, M.; OZARSLAN, Y. Augmented reality in education: Current technologies and the potential for education. *Procedia - Social and Behavioral Sciences*, v. 47, p. 297–302, 2012. ISSN 1877-0428. Cyprus International Conference on Educational Research (CY-ICER-2012)North Cyprus, US08-10 February, 2012. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877042812023907>. Citado na página 15.

KORMUSHEV, P.; CALINON, S.; CALDWELL, D. G. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, v. 2, n. 3, p. 122–148, 2013. ISSN 2218-6581. Disponível em: <https://www.mdpi.com/2218-6581/2/3/122>. Citado na página 15.

KUBA, J. G.; CHEN, R.; WEN, M.; WEN, Y.; SUN, F.; WANG, J.; YANG, Y. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021. Citado 2 vezes nas páginas 24 e 25.

KUNCZIK, L. Reinforcement learning and bellman's principle of optimality. In: *Reinforcement Learning with Hybrid Quantum Approximation in the NISQ Context*. [S.l.]: Springer, 2022. p. 23–39. Citado 2 vezes nas páginas 15 e 18.

KWON, G.; KIM, B.; KWON, N. K. Reinforcement learning with task decomposition and task-specific reward system for automation of high-level tasks. *Biomimetics*, v. 9, n. 4, 2024. ISSN 2313-7673. Disponível em: <https://www.mdpi.com/2313-7673/9/4/196>. Citado na página 16.

LAN, Q.; PAN, Y.; FYSHE, A.; WHITE, M. Maxmin q-learning: Controlling the estimation bias of q-learning. *arXiv preprint arXiv:2002.06487*, 2020. Citado na página 19.

LARSEN, T. N.; TEIGEN, H. ; LAACHE, T.; VARAGNOLO, D.; RASHEED, A. Comparing deep reinforcement learning algorithms' ability to safely navigate challenging waters. *Frontiers in Robotics and AI*, v. 8, 2021. ISSN 2296-9144. Disponível em: <https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2021.738113>. Citado 2 vezes nas páginas 26 e 38.

LI, Y. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017. Citado 3 vezes nas páginas 15, 27 e 28.

LI, Z.; CHEN, X.; FU, J.; XIE, N.; ZHAO, T. Reducing q-value estimation bias via mutual estimation and softmax operation in madrl. *Algorithms*, v. 17, n. 1, 2024. ISSN 1999-4893. Disponível em: <https://www.mdpi.com/1999-4893/17/1/36>. Citado na página 20.

LIN, C.-J.; JHANG, J.-Y.; LIN, H.-Y.; LEE, C.-L.; YOUNG, K.-Y. Using a reinforcement q-learning-based deep neural network for playing video games. *Electronics*, v. 8, n. 10, 2019. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/8/10/1128>. Citado na página 12.

LIU, B.; CAI, Q.; YANG, Z.; WANG, Z. Neural trust region/proximal policy optimization attains globally optimal policy. In: WALLACH, H.; LAROCHELLE, H.; BEYGELZIMER, A.; ALCHé-BUC, F. d'; FOX, E.; GARNETT, R. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. v. 32. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2019/file/227e072d131ba77451d8f27ab9afdfb7-Paper.pdf>. Citado na página 24.

LUONG, N. C.; HOANG, D. T.; GONG, S.; NIYATO, D.; WANG, P.; LIANG, Y.-C.; KIM, D. I. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Communications Surveys Tutorials*, v. 21, n. 4, p. 3133–3174, 2019. Citado na página 15.

MIROWSKI, P.; PASCANU, R.; VIOLA, F.; SOYER, H.; BALLARD, A. J.; BANINO, A.; DENIL, M.; GOROSHIN, R.; SIFRE, L.; KAVUKCUOGLU, K. et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016. Citado na página 22.

MNIH, V. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. Citado na página 27.

MNIH, V.; BADIA, A. P.; MIRZA, M.; GRAVES, A.; HARLEY, T.; LILLICRAP, T. P.; SILVER, D.; KAVUKCUOGLU, K. Asynchronous methods for deep reinforcement learning. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. [S.l.]: JMLR.org, 2016. (ICML'16), p. 1928–1937. Citado 4 vezes nas páginas 15, 21, 22 e 23.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G. et al. Human-level control through deep reinforcement learning. *nature*, Nature Publishing Group UK London, v. 518, n. 7540, p. 529–533, 2015. Citado 4 vezes nas páginas 12, 19, 20 e 22.

PITIS, S. Rethinking the discount factor in reinforcement learning: A decision theoretic approach. *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 33, n. 01, p. 7949–7956, Jul. 2019. Disponível em: <https://ojs.aaai.org/index.php/AAAI/article/view/4795>. Citado na página 17.

QUEENEY, J.; PASCHALIDIS, Y.; CASSANDRAS, C. G. Generalized proximal policy optimization with sample reuse. In: RANZATO, M.; BEYGELZIMER, A.; DAUPHIN, Y.; LIANG, P.; VAUGHAN, J. W. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2021. v. 34, p. 11909–11919. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2021/file/63c4b1baf3b4460fa9936b1a20919bec-Paper.pdf>. Citado na página 25.

RUMMERY, G. A.; NIRANJAN, M. *On-line Q-learning using connectionist systems*. [S.l.]: University of Cambridge, Department of Engineering Cambridge, UK, 1994. v. 37. Citado 2 vezes nas páginas 18 e 19.

SCHULMAN, J.; LEVINE, S.; ABBEEL, P.; JORDAN, M.; MORITZ, P. Trust region policy optimization. In: BACH, F.; BLEI, D. (Ed.). *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France: PMLR, 2015. (Proceedings of Machine Learning Research, v. 37), p. 1889–1897. Disponível em: <https://proceedings.mlr.press/v37/schulman15.html>. Citado 2 vezes nas páginas 24 e 25.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. Citado 4 vezes nas páginas 21, 24, 25 e 26.

SEWAK, M. Actor-critic models and the a3c: The asynchronous advantage actor-critic model. *Deep reinforcement learning: frontiers of artificial intelligence*, Springer, p. 141–152, 2019. Citado 2 vezes nas páginas 22 e 23.

SEWAK, M.; SEWAK, M. Deep q network (dqn), double dqn, and dueling dqn: A step towards general artificial intelligence. *Deep reinforcement learning: frontiers of artificial intelligence*, Springer, p. 95–108, 2019. Citado na página 20.

SHANI, L.; EFRONI, Y.; MANNOR, S. Adaptive trust region policy optimization: Global convergence and faster rates for regularized mdps. *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 34, n. 04, p. 5668–5675, Apr. 2020. Disponível em: <https://ojs.aaai.org/index.php/AAAI/article/view/6021>. Citado na página 25.

SHEN, H.; ZHANG, K.; HONG, M.; CHEN, T. Towards understanding asynchronous advantage actor-critic: Convergence and linear speedup. *IEEE Transactions on Signal Processing*, v. 71, p. 2579–2594, 2023. Citado na página 23.

SILVER, D.; HUANG, A.; MADDISON, C. J.; GUEZ, A.; SIFRE, L.; DRIESSCHE, G. V. D.; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M. et al. Mastering the game of go with deep neural networks and tree search. *nature*, Nature Publishing Group, v. 529, n. 7587, p. 484–489, 2016. Citado na página 27.

SUTTON, R. S. Reinforcement learning: An introduction. *A Bradford Book*, 2018. Citado 2 vezes nas páginas 15 e 16.

TAN, F.; YAN, P.; GUAN, X. Deep reinforcement learning: From q-learning to deep q-learning. In: SPRINGER. *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14–18, 2017, Proceedings, Part IV 24*. [S.l.], 2017. p. 475–483. Citado na página 19.

THéATE, T.; ERNST, D. Risk-sensitive policy with distributional reinforcement learning. *Algorithms*, v. 16, n. 7, 2023. ISSN 1999-4893. Disponível em: <https://www.mdpi.com/1999-4893/16/7/325>. Citado na página 17.

TORRADO, R. R.; BONTRAGER, P.; TOGELIUS, J.; LIU, J.; PEREZ-LIEBANA, D. 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. Citado 2 vezes nas páginas 12 e 27.

VINYALS, O.; EWALDS, T.; BARTUNOV, S.; GEORGIEV, P.; VEZHNEVETS, A. S.; YEO, M.; MAKHZANI, A.; KÜTTLER, H.; AGAPIOU, J.; SCHRITTWIESER, J. et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017. Citado na página 29.

WANG, Y.; HE, H.; TAN, X.; GAN, Y. Trust region-guided proximal policy optimization. In: WALLACH, H.; LAROCHELLE, H.; BEYGELZIMER, A.; ALCHé-BUC, F. d'; FOX, E.; GARNETT, R. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. v. 32. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2019/file/a666587afda6e89aec274a3657558a27-Paper.pdf>. Citado na página 24.

WANG, Z. T.; UEDA, M. Convergent and efficient deep q network algorithm. *arXiv preprint arXiv:2106.15419*, 2021. Citado na página 20.

YU, C.; VELU, A.; VINITSKY, E.; GAO, J.; WANG, Y.; BAYEN, A.; WU, Y. The surprising effectiveness of ppo in cooperative multi-agent games. In: KOYEJO, S.; MOHAMED, S.; AGARWAL, A.; BELGRAVE, D.; CHO, K.; OH, A. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2022. v. 35, p. 24611–24624. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2022/file/9c1535a02f0ce079433344e14d910597-Paper-Datasets_and_Benchmarks.pdf>. Citado 2 vezes nas páginas 26 e 38.

ZHANG, Z.; ZOU, Y.; LAI, J.; XU, Q. M2dqn: A robust method for accelerating deep q-learning network. In: *Proceedings of the 2023 15th International Conference on Machine Learning and Computing*. New York, NY, USA: Association for Computing Machinery, 2023. (ICMLC '23), p. 116–120. ISBN 9781450398411. Disponível em: <https://doi.org/10.1145/3587716.3587735>. Citado na página 20.

ZHU, Y.; WANG, Z.; ZHU, Y.; CHEN, C.; ZHAO, D. Discretizing continuous action space with unimodal probability distributions for on-policy reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, p. 1–13, 2024. Citado na página 17.