



UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Instituto de Ciência e Tecnologia
Câmpus de Sorocaba

NATAN RAZERA PINTO

**DESENVOLVIMENTO DE UMA VIRTUAL TEST BED E DIGITAL TWIN PARA
TREINAMENTO DE VEÍCULOS AUTÔNOMOS**

Sorocaba

2025

NATAN RAZERA PINTO

**DESENVOLVIMENTO DE UMA VIRTUAL TEST BED E DIGITAL TWIN PARA
TREINAMENTO DE VEÍCULOS AUTÔNOMOS**

Trabalho de Conclusão de Curso apresentado à
Universidade Estadual Paulista (UNESP),
Instituto de Ciência e Tecnologia, Sorocaba,
como parte dos requisitos para obtenção do
grau de Bacharel em Engenharia de Controle e
Automação.

Orientador: Prof. Dr. Leopoldo André Dutra
Lusquino Filho

Sorocaba

2025

P659d	Pinto, Natan Razera Desenvolvimento de uma Virtual Test Bed e Digital Twin para treinamento de veículos autônomos / Natan Razera Pinto. -- Sorocaba, 2025 62 p. : tabs., fotos Trabalho de conclusão de curso (Bacharelado - Engenharia de Controle e Automação) - Universidade Estadual Paulista (UNESP), Instituto de Ciência e Tecnologia, Sorocaba Orientador: Leopoldo André Dutra Lusquino Filho 1. Aprendizagem profunda (Aprendizado do computador). 2. Veículos autônomos. 3. Simulação por computador. I. Título.
-------	---

NATAN RAZERA PINTO

**DESENVOLVIMENTO DE UMA VIRTUAL TEST BED E DIGITAL TWIN PARA
TREINAMENTO DE VEÍCULOS AUTÔNOMOS**

Trabalho de Conclusão de Curso apresentado ao Instituto de Ciência e Tecnologia de Sorocaba, Universidade Estadual Paulista (UNESP), como parte dos requisitos para obtenção do grau de Bacharel(a) em Engenharia de Controle e Automação.

Data da defesa: 24/04/2025

BANCA EXAMINADORA:

Prof. Dr. Leopoldo André Dutra Lusquino Filho
UNESP – Instituto de Ciência e Tecnologia – Campus de Sorocaba

Prof. Dr. Felix Mauricio Escalante Ortega
UNESP – Instituto de Ciência e Tecnologia – Campus de Sorocaba

Prof. Dr. Luis Armando De Oro Arenas
UNESP – Instituto de Ciência e Tecnologia – Campus de Sorocaba

AGRADECIMENTOS

Agradeço, em primeiro lugar, ao meu Senhor Jesus, que todos os dias tem derramado Sua misericórdia e graça sobre a minha vida. Ele tem me guiado e guardado tanto nos momentos felizes quanto nos momentos difíceis.

Sou imensamente grato aos meus pais, Moacir e Claudia, um homem e uma mulher de fé, que me ensinaram a trilhar o caminho correto e me corrigiram quando necessário. Muito obrigado por sempre me incentivarem a seguir meus sonhos e por estarem ao meu lado em todos os momentos.

Estendo meus agradecimentos aos queridos amigos que fiz ao longo desta jornada universitária: Luigi Marson Grandi, Samuel Rodrigues Fortes, Guilherme Moyses Ginack, Alan Tonon e Dimitri Margutti Cezar. Obrigado por todos os momentos de amizade, apoio e companheirismo.

Agradeço especialmente ao professor Dr. Leopoldo André Dutra Lusquino Filho, pela orientação dedicada neste Trabalho de Conclusão de Curso, sempre me guiando na direção certa. Sou grato também pelas aulas ministradas ao longo do curso de Engenharia de Controle e Automação, que foram de grande valor para minha formação.

Por fim, agradeço a todos os professores e funcionários da UNESP Sorocaba, pelo trabalho exemplar e pelo comprometimento com um ensino de excelência.

RESUMO

A crescente demanda por soluções de mobilidade urbana impulsionou o desenvolvimento de veículos autônomos como alternativa eficiente e segura. Contudo, os altos custos, riscos e tempo envolvidos em testes físicos limitam o avanço dessa tecnologia. Este trabalho propõe a criação de uma Plataforma de Testes Virtual (*Virtual Test Bed*) integrada a um Gêmeo Digital (*Digital Twin*), com o objetivo de treinar agentes de Inteligência Artificial para direção autônoma utilizando Aprendizado por Reforço Profundo (*Deep Reinforcement Learning*) e visualizar a evolução do treinamento. A abordagem adotada combina o motor gráfico Unity com a arquitetura Ator-Crítico (*Actor-Critic*) com o algoritmo de Otimização de Política Proximal (*Proximal Policy Optimization – PPO*), integrando a extensão Unity ML-Agents Toolkit e a biblioteca PyTorch, para o treinamento dos agentes com visualização da evolução das variáveis de treinamento no TensorBoard. O ambiente foi validado por meio da simulação de duas pistas com diferentes níveis de dificuldade e até 40 agentes simultâneos, com desempenho avaliado por métricas como recompensa cumulativa, entropia e tempo de episódio. Os resultados demonstraram aprendizado eficiente e estável dos agentes, com políticas cada vez mais otimizadas. A principal contribuição desta pesquisa é a entrega de uma plataforma acessível e reproduzível para treinamento e validação de agentes autônomos, reduzindo significativamente os riscos de testes físicos e acelerando o desenvolvimento de sistemas inteligentes aplicáveis à mobilidade urbana.

Palavras-chave: Aprendizado por Reforço Profundo; Veículos Autônomos; Simulação Virtual; Unity.

ABSTRACT

The growing demand for urban mobility solutions has driven the development of autonomous vehicles as an efficient and safe alternative. However, the high costs, risks, and time associated with physical testing limit the advancement of this technology. This work proposes the creation of a Virtual Test Bed integrated with a Digital Twin, aimed at training Artificial Intelligence agents for autonomous driving using Deep Reinforcement Learning and visualizing the training progress. The adopted approach combines the Unity game engine with the Actor-Critic architecture and the Proximal Policy Optimization algorithm, integrating the Unity ML-Agents Toolkit extension and the PyTorch library for agent training, with training variable visualization through TensorBoard. The environment was validated through the simulation of two tracks with different levels of difficulty and up to 40 simultaneous agents, with performance evaluated by metrics such as cumulative reward, entropy, and episode length. The results demonstrated efficient and stable agent learning, with increasingly optimized policies. The main contribution of this research is the delivery of an accessible and reproducible platform for training and validating autonomous agents, significantly reducing the risks of physical testing and accelerating the development of intelligent systems applicable to urban mobility.

Keywords: Deep Reinforcement learning; Autonomous Vehicles; Virtual Simulation; Unity.

LISTA DE ILUSTRAÇÕES

Figura 1 – Ciclo de interação entre um agente e o ambiente no <i>Reinforcement Learning</i>	21
Figura 2 – Arquitetura <i>Actor-Critic</i> no <i>Reinforcement Learning</i>	23
Figura 3 – Digital Twin para monitoramento de emissões de gases do efeito estufa	26
Figura 4 – Editor Unity	29
Figura 5 – Lista de Objetos em cena e Inspetor da Unity	29
Figura 6 – Modelo do carro utilizado no software Blender	30
Figura 7 – Implementação da pista na cena	31
Figura 8 – Configuração do componente Wheel Collider utilizado nos veículos	31
Figura 9 – Raycast utilizado como sensor de distância do veículo	32
Figura 10 – Ambiente virtual em Python para o treinamento	33
Figura 11 – Arquivo YAML de configuração de hiperparâmetros	35
Figura 12 – Dashboard do TensorBoard	36
Figura 13 – Diagrama de Blocos da abordagem proposta	37
Figura 14 – Main Menu	40
Figura 15 – Tela de escolha de pista e quantidade de agentes no treinamento	40
Figura 16 – Track 0	41
Figura 17 – Track 1	41
Figura 18 – Visualização web do Digital Twin do Tensorboard	42
Figura 19 – Ambiente final de treino e simulação implementado	43
Figura 20 – Recompensas cumulativas com 40 agentes na Pista 0	44
Figura 21 – Recompensas cumulativas com 40 agentes na Pista 1	44
Figura 22 – Histograma das recompensas cumulativas com 40 agentes na Pista 0	45
Figura 23 – Histograma das recompensas cumulativas com 40 agentes na Pista 1	45
Figura 24 – Evolução da entropia com 40 agentes na Pista 0	46
Figura 25 – Evolução da entropia com 40 agentes na Pista 1	46
Figura 26 – Evolução do tempo de episódio com 40 agentes na Pista 0	47
Figura 27 – Evolução do tempo de episódio com 40 agentes na Pista 1	47
Figura 28 – Evolução da perda de política com 40 agentes na Pista 0	48
Figura 29 – Evolução da perda de política com 40 agentes na Pista 1	48
Figura 30 – Evolução da perda de valor com 40 agentes na Pista 0	49
Figura 31 – Evolução da perda de valor com 40 agentes na Pista 1	49

Figura 32 – Comparação entre valores de gamma no teste com 40 agentes na Pista 0	50
Figura 33 – Gráfico de Entropia do teste com 40 agentes na Pista 0 na variação de gamma	50
Figura 34 – Histograma de recompensa cumulativa do teste com 40 agentes na Pista 0 e baixo gamma	51
Figura 35 – Comparação entre valores de gamma no teste com 40 agentes na Pista 1	51
Figura 36 – Histograma de recompensa cumulativa do teste com 40 agentes na Pista 1 e baixo valor de gamma	52
Figura 37 – Gráfico de Entropia do teste com 40 agentes na Pista 1 na variação de gamma	52
Figura 38 – Gráfico de evolução da recompensa cumulativa na pista 0 – Teste de Quantidade de agentes	53
Figura 39 – Gráfico de evolução da recompensa cumulativa na pista 1 – Teste de Quantidade de agentes	53

LISTA DE TABELAS

Tabela 1 – Componentes da função de recompensa na abordagem aplicada.....	34
Tabela 2 – Hiperparâmetros utilizados na simulação.....	38

LISTA DE ABREVIATURAS E SIGLAS

IoT	Internet of Things
IoV	Internet of Vehicles
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
ML	Machine Learning
PPO	Proximal Policy Optimization
DT	Digital Twin
DQN	Deep Q-Network
LSTM	Long Short-Term Memory
YAML	Yet Another Markup Language
API	Application Programming Interface
GPU	Graphics Processing Unit
ONNX	Open Neural Network Exchange
ABM	Agent-Based Modeling
MDP	Markov Decision Process

LISTA DE SÍMBOLOS

λ	Lambda (Fator de Decaimento)
γ	Gamma (Fator de Desconto)
β	Beta (Parâmetro de Entropia)
α	Alpha (Taxa de Aprendizado)
π	Política

SUMÁRIO

1.	INTRODUÇÃO	12
2	REVISÃO BIBLIOGRÁFICA	16
2.1	Simulação de Veículos	17
2.2	Aprendizado por reforço	19
2.3	Gêmeos Digitais	24
2.4	Motores Gráficos	26
3	ABORDAGEM PROPOSTA	28
3.1	Materiais e Métodos	28
3.2	Setup Experimental	38
4	RESULTADOS E DISCUSSÕES	40
4.1	Ambiente de Simulação	40
4.2	Validação do Treinamento	43
4.3	Discussão dos Resultados Apresentados	54
5	CONCLUSÃO	55
	REFERÊNCIAS	57
	GLOSSÁRIO	61

1. INTRODUÇÃO

Poluição, congestionamento de veículos e acidentes de trânsito estão entre os principais desafios enfrentados pelos centros urbanos contemporâneos, especialmente no contexto da mobilidade. Esses problemas, somados a outros, como a distribuição de energia elétrica e o tratamento de esgoto, são objeto de estudo de diversas organizações que buscam soluções eficazes para os entraves característicos das cidades modernas. Uma das abordagens que vem ganhando destaque no meio acadêmico e é considerada um dos principais caminhos para superar tais desafios e promover a sustentabilidade urbana é o modelo de Cidades Inteligentes (*Smart Cities*) (RIZZON et al., 2017).

As *Smart Cities* são ambientes urbanos que utilizam Tecnologias da Informação e Comunicação (TIC) para integrar diferentes sistemas, com o objetivo de melhorar a qualidade de vida dos cidadãos e impulsionar o desenvolvimento econômico por meio da otimização de recursos e da promoção da sustentabilidade (ISMAGILOVA et al., 2019). Essas cidades fazem uso de redes de sensores, análise de dados e conectividade avançada para monitorar variáveis como tráfego, consumo de energia e distribuição de serviços públicos. Dessa forma, possibilitam decisões mais eficientes por parte dos gestores públicos e incentivam a colaboração entre governo, iniciativa privada e sociedade civil, tornando os centros urbanos mais responsivos e adaptáveis (ODENDAAL, 2003).

No escopo desse modelo de cidade inteligente, destaca-se o conceito de *Smart Mobility*, que se refere à aplicação de TIC na organização dos sistemas de transporte, por meio de tecnologias da Internet das Coisas (*Internet of Things* – IoT) voltadas à comunicação entre veículos e sistemas de controle. Essa abordagem busca reduzir congestionamentos e atrasos por meio da coleta de dados em tempo real das vias urbanas e do controle automatizado e inteligente de redes semafóricas e veículos autônomos (FAISAL et al., 2019). A conectividade entre veículos e sistemas, denominada Internet dos Veículos (*Internet of Vehicles* – IoV) por Ismagilova et al. (2019), constitui a base para o desenvolvimento de um tráfego mais seguro, eficiente e coordenado.

Entre as tecnologias que integram esse ecossistema conectado, destacam-se os Veículos Autônomos (VA). Esses veículos inteligentes não apenas realizam o transporte de passageiros de um ponto a outro, mas também são projetados para oferecer segurança, conforto e conectividade. Suas principais capacidades incluem reconhecimento do ambiente, navegação autônoma, tomada de decisões em tempo real, obediência às leis de trânsito e atenção ao bem-estar dos passageiros (PAREKH et al., 2022).

O interesse por Veículos Autônomos aumentou significativamente nos últimos anos, especialmente a partir de 2018, como destacado por Parekh et al. (2022), impulsionado pelos avanços em Aprendizado de Máquina (*Machine Learning* – ML) e pela eletrificação veicular. Apesar do crescimento recente, o conceito remonta a 1939, com sua primeira exibição realizada pela empresa *General Motors*. As pesquisas e os primeiros desenvolvimentos desse conceito, iniciaram-se na década de 1950, em parceria com o *Radio Corporation of America Sarnoff Laboratory* (SHLADOVER, 2018). O progresso da área foi limitado por décadas, devido às restrições computacionais da época, até 2004, quando a Agência de Projetos de Pesquisa Avançada de Defesa (*Defense Advanced Research Projects Agency* – DARPA) promoveu desafios que resultaram em veículos capazes de navegar de forma autônoma em desertos (FAISAL et al., 2019). Desde então, os Veículos Autônomos vêm ganhando espaço em ambientes urbanos, com potencial para causar impactos positivos relevantes na mobilidade e na segurança no trânsito (YURTSEVER et al., 2020).

A arquitetura dos sistemas de controle de veículos autônomos pode ser dividida em duas abordagens principais: sistemas modulares e sistemas *end-to-end*. Os sistemas modulares são compostos por componentes independentes que conectam os dados de entrada sensoriais às saídas atuadoras (MCALLISTER et al., 2017). Nessa abordagem, o processamento ocorre de forma sequencial: o módulo de percepção capta e interpreta o ambiente; os dados são encaminhados ao planejador, que determina a melhor ação a ser tomada; e, por fim, o controlador executa essa ação. Essa estrutura oferece flexibilidade, facilita a adição de novas funcionalidades e permite a depuração de erros por módulo, visto que cada etapa possui um algoritmo especializado. No entanto, apresenta desvantagens como a complexidade na integração dos módulos e a possibilidade de propagação de erros ao longo da cadeia. Essas limitações foram apontadas como fatores contribuintes no primeiro acidente fatal envolvendo um veículo autônomo (YURTSEVER et al., 2020), o que evidenciou os desafios dessa abordagem em ambientes urbanos dinâmicos.

A arquitetura mais adotada atualmente é a *end-to-end*, também conhecida como percepção direta. Nessa abordagem, os dados sensoriais são conectados diretamente a um único modelo de rede neural, responsável por processar todas as informações e gerar os comandos de controle, diferentemente da abordagem modular, em que cada tarefa é atribuída a um algoritmo específico (YURTSEVER et al., 2020). Duas estratégias principais compõem essa arquitetura: Aprendizado Profundo Supervisionado Direto (*Direct Supervised Deep Learning*) e Aprendizado por Reforço Profundo (*Deep Reinforcement Learning*).

Dentre essas abordagens, apenas o *Direct Supervised Deep Learning* permite o treinamento em ambiente *offline*, utilizando imagens e vídeos de condutores humanos acompanhados dos respectivos comandos de direção. Apesar de ser uma alternativa viável, apresenta baixo desempenho na generalização. Já o *Deep Reinforcement Learning* tem demonstrado maior eficácia, proporcionando uma direção otimizada, com testes bem-sucedidos realizados em ambientes reais (YURTSEVER et al., 2020). Nessa abordagem, o agente é treinado com base em tentativa e erro, aprendendo por meio de recompensas definidas previamente. Contudo, esse método exige o uso contínuo do veículo durante o treinamento, o que pode acarretar riscos de acidentes ou danos ao sistema.

Considerando o alto custo e o tempo significativo envolvidos no desenvolvimento de Veículos Autônomos, uma alternativa amplamente adotada é a utilização de Ambientes Virtuais de Simulação, conhecidos como *Virtual Test Beds*. Esses ambientes permitem a integração de diferentes modelos, como modelos dinâmicos e de inteligência artificial, reunindo-os em um único espaço capaz de simular e visualizar os resultados dessas interações (GÖKDERE et al., 2002).

Os *Virtual Test Beds* podem atuar não apenas como simuladores, mas também como réplicas digitais completas ou parciais de sistemas físicos, refletindo as alterações que ocorrem no mundo real. Esse tipo de aplicação é conhecido como *Digital Twin*, definido como uma cópia digital integrada a um sistema físico, na qual todas as ações e variações do sistema real são diretamente refletidas em sua contraparte digital (BHATTI; MOHAN; SINGH, 2021). O *Digital Twin* funciona de forma bidirecional, podendo ser utilizado tanto para o monitoramento de variáveis físicas do sistema, por meio de sensores instalados no componente físico, quanto para a análise e predição de dados. Essas características se alinham diretamente aos objetivos da *Smart Mobility*, sendo aplicáveis à visualização e análise de dados veiculares, como emissão de gases, consumo de combustível e distância percorrida. Tal abordagem pode reduzir custos com testes de estrada e manutenção, especialmente no contexto dos veículos autônomos.

Diante dos desafios apresentados, este trabalho propõe o desenvolvimento de uma *Virtual Test Bed* para o treinamento de Veículos Autônomos utilizando a abordagem de maior desempenho atualmente disponível, *Deep Reinforcement Learning*. O trabalho visa criar um ambiente acessível e de baixo custo para o desenvolvimento de modelos de *Deep Learning* voltadas a veículos autônomos, mitigando os riscos e os altos investimentos associados a testes em ambientes físicos. O projeto será implementado a partir de uma *game engine*, que servirá como base para a construção da pista e da simulação do comportamento veicular. Adicionalmente, será integrada uma estrutura de *Digital Twin*, permitindo a análise das

variáveis do treinamento por meio de um painel visual, contribuindo para a compreensão do desempenho do agente e para o aprimoramento contínuo dos parâmetros de treino utilizados. Dessa forma, o trabalho busca unir conceitos de *Smart Mobility* e inteligência artificial, oferecendo uma ferramenta modular, reprodutível e alinhada com os avanços recentes da área de veículos autônomos.

Este trabalho está organizado em cinco seções principais: Introdução, Revisão Bibliográfica, Abordagem Proposta, Resultados e Discussão e Conclusão. Na seção de *Revisão Bibliográfica*, serão apresentados os conceitos teóricos e o estado da arte das tecnologias utilizadas ao longo do projeto. A seção *Abordagem Proposta* descreve a metodologia adotada, incluindo as ferramentas, bibliotecas, configurações e estratégias de desenvolvimento empregadas. Em *Resultados e Discussão*, são analisados os dados obtidos a partir da validação do ambiente e do processo de treinamento dos agentes. Por fim, a seção *Conclusão* apresenta as considerações finais, destacando os principais resultados alcançados e sugerindo possíveis direções para trabalhos futuros.

2. REVISÃO BIBLIOGRÁFICA

O desenvolvimento de plataformas virtuais para simulação, teste e validação de veículos autônomos tem se consolidado como uma das principais estratégias para acelerar o avanço do desenvolvimento de veículos autônomos, reduzindo os custos e riscos associados à experimentação em ambientes físicos. Diversos trabalhos na literatura exploram esse conceito por meio da criação de *Virtual Test Beds* com diferentes propósitos e níveis de complexidade. Entre eles, destaca-se o *framework* proposto por Elmquist et al. (2021), que apresenta uma plataforma de simulação baseada na *Virtual Test Beds Chrono*, para teste de sensores e algoritmos de controle. O trabalho foca na simulação realista de sensores de posicionamento e de captura de imagens, e foi desenvolvido com foco em aplicações em robótica e veículos autônomos, promovendo a geração de dados sintéticos com fidelidade física para o treinamento em aplicações de ML.

Outra contribuição relevante é apresentada por Sivayazi e Mannayee (2024), que propuseram a aplicação de um algoritmo de *Deep Q-Learning* para navegação de veículos autônomos com capacidade de adaptação do agente a novos desafios, utilizando a *Virtual Test Beds CARLA*. A simulação no CARLA foi realizada com um ambiente urbano 3D com obstáculos móveis e gráficos realistas para fins de validação segura. O trabalho destaca o potencial do aprendizado por *Deep Learning* no controle de sistemas autônomos, que pode gerar sistemas que praticam comandos de direção eficientes sem a necessidade de planejamento prévio de rotas fixas.

Complementando as abordagens feitas em *Deep Q-Learning* por Sivayazi e Mannayee (2024), o trabalho de Pérez-Gil et al. (2022) propõe a comparação entre dois tipos de modelos *Deep Learning* aplicados à condução autônoma no simulador CARLA. Uma das contribuições relevantes do estudo está na integração eficiente entre os modelos de aprendizado por reforço desenvolvidos em Python e o ambiente simulado em CARLA, permitindo a coleta das variáveis como posição, aceleração e desvios de trajetória. Essa integração proporciona um ciclo completo de percepção, decisão e atuação, essencial para a validação de algoritmos de controle baseados em redes neurais. Assim como os trabalhos de Sivayazi e Mannayee (2024) e Elmquist et al. (2021), este estudo reforça o uso de ambientes de simulação realistas como plataformas robustas para desenvolvimento, teste e validação de sistemas autônomos em contextos urbanos complexos.

O restante dessa seção apresenta os principais fundamentos teóricos e trabalhos relacionados a esses temas, que sustentam o desenvolvimento do projeto proposta nesse

trabalho. Os temas abordados abrangem tanto os conceitos técnicos que fundamentam a simulação e o controle de veículos autônomos, quanto as tecnologias utilizadas para sua implementação e validação em ambientes virtuais.

2.1 Simulação de Veículos

Nos cenários complexos de tráfego urbano, a simulação virtual surge como uma alternativa segura, escalável e de baixo custo para o desenvolvimento e validação de soluções voltadas à *Smart Mobility*. Diversas técnicas e modelos têm sido utilizados para representar o comportamento do tráfego e dos veículos em diferentes escalas e níveis de fidelidade. As abordagens de simulação podem ser classificadas de acordo com a representação dos agentes simulados. Os modelos macroscópicos tratam o tráfego como um fluxo contínuo, semelhante ao comportamento de fluidos, sendo úteis para análises agregadas de redes de transporte, estudos de capacidade viária e avaliação de políticas públicas. Por outro lado, os modelos microsimulados representam os veículos individualmente, permitindo a observação de interações entre motoristas, tomada de decisões e variações comportamentais. Essas duas abordagens, macro e micro, se complementam e são amplamente aplicadas em projetos de *Smart Mobility*.

No contexto das simulações macroscópicas, uma das principais bases teóricas é a Teoria de Fluxo de Tráfego, um campo consolidado da engenharia de transportes que se fundamenta em três variáveis principais: fluxo (número de veículos que passam por um ponto em determinado intervalo de tempo), densidade veicular (quantidade de veículos por unidade de comprimento da via) e velocidade média. A relação entre essas variáveis dá origem aos diagramas fundamentais de tráfego, que permitem a análise de regimes distintos como tráfego livre, instável e congestionado (SHI; LI, 2021). Essa teoria é amplamente utilizada em sistemas inteligentes de transporte, não apenas em rodovias, mas também em ferrovias, tráfego aéreo e vias navegáveis, considerando as particularidades de cada meio, como velocidade, dimensões dos veículos e organização espacial (CHEN et al., 2024).

Além das aplicações em análises estatísticas e modelos agregados, a Teoria de Fluxo de Tráfego também tem sido incorporada a ambientes de simulação urbana, inclusive em modelos baseados em *Deep Learning*, voltados à previsão de tráfego em redes urbanas complexas (CHEN et al., 2024). Nessas simulações, a teoria permite estudar os efeitos de tecnologias como controle adaptativo de cruzamento e tomada de decisão automática, características dos veículos

autônomos que impactam diretamente o espaçamento entre veículos e a estabilidade do fluxo viário (SHI; LI, 2021).

No campo da microssimulação, onde o foco está nas decisões e interações individuais entre veículos e o ambiente, destacam-se duas abordagens principais: a modelagem baseada em agentes (*Agent-Based Modeling* – ABM) e os modelos de dinâmica veicular. A ABM representa sistemas complexos a partir do comportamento individual de entidades autônomas, os agentes, que interagem entre si e com o ambiente de forma adaptativa. Essa abordagem é eficaz para simular cenários de mobilidade heterogênea e descentralizada. Tang et al. (2024) aplicaram essa técnica para avaliar o impacto de diferentes estratégias de estacionamento adotadas por veículos autônomos na cidade de Hangzhou, China. No modelo, os veículos tomavam decisões descentralizadas, como retornar ao ponto de partida, circular continuamente ou utilizar estacionamentos remotos, com base em um modelo de escolha discreta que considera custos operacionais e tarifas. De forma complementar, Li et al. (2024) utilizaram a plataforma MATSim para simular a integração de veículos autônomos compartilhados com sistemas do tipo *Park-and-Ride* (pontos de estacionamento nas regiões periféricas da cidade que incentivam o uso de modais sustentáveis) como alternativa ao uso de veículos privados no centro urbano de Bruxelas.

Por sua vez, os modelos de dinâmica veicular têm como objetivo representar com alta fidelidade os aspectos físicos e mecânicos do movimento dos veículos, sendo amplamente utilizados quando é necessário que o comportamento simulado se aproxime do real. Esses modelos incluem a simulação detalhada de subsistemas como suspensão, pneus, direção, transmissão e a interação entre veículo e terreno, permitindo a análise de forças atuantes durante aceleração, frenagem e manobras. Diferentemente da ABM, que abstrai a física para focar no comportamento estratégico dos agentes, os simuladores dinâmicos priorizam a precisão física. No trabalho de Elmquist et al. (2021), a plataforma Chrono foi utilizada como base para a simulação de veículos autônomos, integrando um motor de dinâmica física capaz de representar corpos rígidos e flexíveis, terrenos deformáveis e contatos complexos. A estrutura modular do Chrono, permite construir modelos baseados em parâmetros reais dos veículos, enquanto se simula sensores utilizados em veículos autônomos com ruído, latência e distorções físicas. Essa abordagem permite uma análise mais precisa de algoritmos de controle e percepção, reduzindo a lacuna entre simulação e realidade.

2.2 Aprendizado por Reforço

O Aprendizado por Reforço (*Reinforcement Learning* – RL) é uma estratégia de aprendizado de máquina baseada na interação sequencial entre um agente e um ambiente. Nesse paradigma, o agente aprende um comportamento ideal por meio de tentativa e erro, ajustando suas ações com base nas recompensas obtidas por interações anteriores. Esse processo de aprendizagem é inspirado na forma como humanos e animais desenvolvem habilidades por meio da experiência (SHAKYA; PILLAI; CHAKRABARTY, 2023).

A dinâmica do RL é formalizada por meio de um Processo de Decisão de Markov (*Markov Decision Process* – MDP), que fornece a base matemático para modelar problemas de decisão sequenciais em ambientes estocásticos. Um MDP é definido por uma quádrupla (X, U, P, R) , em que:

- X : Conjunto de Estados Possíveis do ambiente (como coordenadas num *grid*);
- U : Conjunto de Ações Possíveis que o agente pode executar (como virar à esquerda ou à direita);
- $P(x' | x, u)$: Função de Transição de Estados, que representa a probabilidade de transitar do estado x para o estado x' , dado que a ação u foi executada;
- $R(x, u)$: Função de Recompensa, que define o valor esperado da recompensa ao executar a ação u no estado x .

Durante a simulação, a cada instante de tempo k , o agente observa o estado atual x_k , executa uma ação u_k , e recebe uma recompensa imediata r_{k+1} . Ao final de um episódio, definido por um número finito de passos K , o ambiente é reiniciado. A recompensa total acumulada pelo agente nesse episódio é dada pela soma das recompensas obtidas ao longo do tempo, como expressa a equação (1):

$$G_k = r_{k+1} + r_{k+2} + r_{k+3} + \dots + r_K \quad (1)$$

O objetivo do agente é aprender uma política de tomada de decisão $\pi(u|x)$, que maximize a recompensa esperada G_k ao longo do tempo. A política define a probabilidade de executar uma ação u dado um estado x . O valor esperado do retorno, considerando que o agente segue a política π , é representado pela função valor de estado $V_\pi(x)$, conforme a equação (2) (SUTTON; BARTO, 2015):

$$V_\pi(x) = E_\pi[G_k | x_k = x] \quad (2)$$

Essa expectativa pode ser ajustada por um fator de desconto $\gamma \in [0,1]$, que pondera a importância das recompensas futuras. Quando γ se aproxima de 0, o agente tende a valorizar recompensas imediatas, adotando um comportamento imediatista, enquanto valores próximos de 1 incentivam decisões que consideram benefícios de longo prazo. Incorporando o fator de desconto, a equação (2) torna-se:

$$V_{\pi}(x) = E_{\pi} \left\{ \sum_{n=0}^{\infty} \gamma^n r_{k+n+1} | x_k = x \right\}, \forall x \in X \quad (3)$$

Além disso, é possível calcular o valor esperado da recompensa para um par estado-ação, utilizando a função de valor de estado-ação $Q_{\pi}(x, u)$, que mede o retorno esperado a partir do estado x , realizando a ação u e seguindo a política π posteriormente, como visto na equação (4).

$$Q_{\pi}(x, u) = E_{\pi} \left\{ \sum_{n=0}^{\infty} \gamma^n r_{k+n+1} | x_k = x, u_k = u \right\}, \forall x \in X \text{ e } \forall u \in X \quad (4)$$

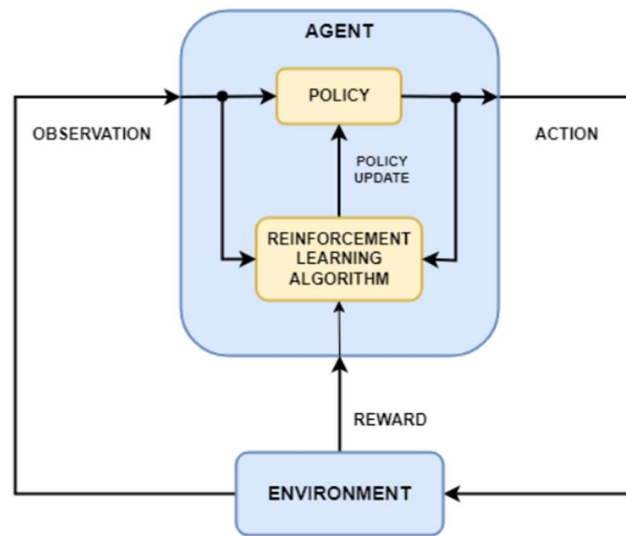
Essas funções são estimadas com base nas interações do agente com o ambiente. Em MDPs finitos, uma vez estimadas $Q^*(x, u)$, e $V^*(x)$ pelas equações de Bellman, é possível obter a política mais otimizada π^* , que maximiza o valor esperado em cada estado, como se pode observar na equação (5) calculada por Sutton e Barto (2015).

$$\pi^*(x) = \operatorname{argmax}_u Q^*(x, u) \quad (5)$$

Durante o processo de treinamento, tanto o fator de desconto γ quanto o grau de entropia da política, podem ser ajustados dinamicamente, influenciando o comportamento do agente ao longo do tempo. Assim, o agente é capaz de aprimorar continuamente suas estratégias, com base no retorno acumulado obtido durante os episódios de interação (SHAKYA; PILLAI; CHAKRABARTY, 2023).

A Figura 1 ilustra o ciclo de interação entre o agente e o ambiente durante o processo de aprendizado por reforço. A cada passo de tempo, o agente observa o ambiente, toma uma decisão com base na política atual, realiza uma ação, recebe uma recompensa e, ao final do episódio, a política é atualizada com base nas recompensas obtidas.

Figura 1 – Ciclo de interação entre um agente e o ambiente no *Reinforcement Learning*



Fonte: MATHWORKS (2025)

Esse tipo de aprendizado tem sido amplamente empregado em tarefas de controle, jogos, navegação e robótica, especialmente em contextos em que não há supervisão explícita. Dentre as estratégias de aplicação de RL, diversas pesquisas apontam o Aprendizado por Reforço Profundo (*Deep Reinforcement Learning* - DRL) como uma das abordagens mais promissoras para controle de sistemas autônomos (YURTSEVER et al., 2020; SHAKYA; PILLAI; CHAKRABARTY, 2023).

O *Deep Reinforcement Learning* (DRL) é uma extensão do Aprendizado por Reforço tradicional (RL) que incorpora técnicas de *Deep Learning* para lidar com espaços de estado e ação de alta dimensionalidade. Sua principal vantagem está na capacidade de utilizar redes neurais profundas (DNNs) como aproximadores de função para modelar políticas e funções de valor em ambientes complexos, nos quais métodos tabulares ou lineares se tornam inviáveis (SHAKYA; PILLAI; CHAKRABARTY, 2023).

As primeiras aplicações bem-sucedidas do DRL em problemas complexos ocorreram no domínio dos jogos. Um dos casos mais emblemáticos foi o algoritmo TD-Gammon, desenvolvido por Gerald Tesauro, no qual o agente jogava contra si mesmo, sem qualquer conhecimento prévio das regras ou estratégias humanas, ajustando os pesos da rede com base nos resultados das partidas. Versões avançadas desse algoritmo chegaram a competir em igualdade com campeões mundiais de Gamão, influenciando até mesmo a forma como especialistas humanos interpretavam determinadas aberturas e estratégias (TESAURO, 1995).

Durante o desenvolvimento da arquitetura de DRL, foi se criando diferentes abordagens de otimização, como busca por uma melhor função de valor estado ação (*Value-based*), foco na melhoria da otimização da política (*policy-based*) ou uma mistura dos dois (*Actor-critic*). As abordagens *value-based* em Aprendizado por Reforço concentram-se na estimação da função de valor de estado-ação $Q(x, u)$, descrita na equação (4). Nessa abordagem, o foco é otimizar a função de valor ao invés de modelar diretamente uma política, pois a política otimizada é obtida indiretamente por meio da ação que maximiza essa função. Essa estratégia é particularmente eficiente em ambientes com espaços de ação discretos, permitindo a escolha da melhor ação sem a necessidade de aprender uma política parametrizada.

No contexto do DRL, uma das contribuições mais relevantes foi o desenvolvimento do *Deep Q-Network* (DQN) pela equipe do Google DeepMind, descrito por Mnih et al. (2015). O DQN utilizou redes neurais convolucionais para estimar valores Q a partir de imagens brutas, tornando possível treinar agentes em ambientes visuais complexos como os jogos do Atari 2600.

Uma técnica fundamental dentro dessa abordagem é o *Q-Learning*, cujo objetivo é aprender a função de valor ótima $Q^*(x, u)$ sem necessidade de modelar uma política explicitamente. No artigo de Mnih et al. (2015), a atualização da função Q é feita por meio da minimização do erro quadrático temporal, conforme a função de perda $L_i(\theta_i)$, descrita na equação (6):

$$L_i(\theta_i) = E_{x,u,r,x'} \left[\left(y_i^{\text{DQN}} - Q(x, u; \theta_i) \right)^2 \right] \quad (6)$$

O valor alvo y_i^{DQN} , é descrito por (7).

$$y_i^{\text{DQN}} = r + \gamma \max_{u'} Q(x', u'; \theta_i^-) \quad (7)$$

No DQN, a função Q é aproximada por uma rede neural convolucional profunda, treinada com amostras de uma memória de *replay* e estabilizada por meio de uma rede-alvo separada. Essa formulação permitiu treinar agentes que superaram o desempenho humano em diversos jogos do Atari (MNIH et al., 2015).

Já as abordagens *policy-based* buscam otimizar diretamente uma política parametrizada $\pi_\theta(u | x)$, evitando a necessidade de estimar uma função de valor como etapa intermediária.

Essa política é representada por uma rede neural com parâmetros θ , ajustada com base na expectativa de retorno. A função objetivo a ser maximizada é apresentada na equação (8):

$$J(\theta) = E_{\pi_{\theta}}[G] \quad (8)$$

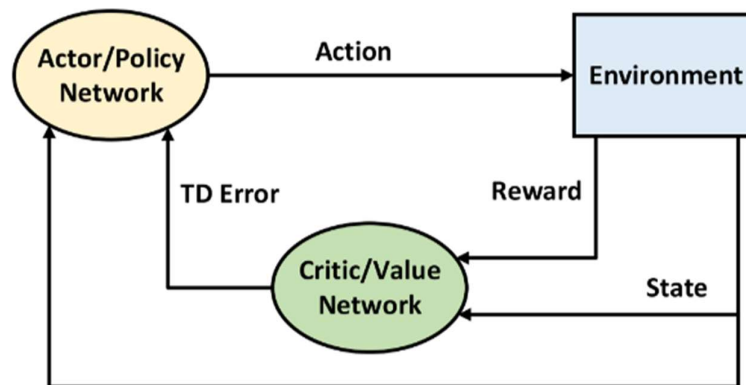
O gradiente dessa função, conhecido como o teorema do gradiente de política, é estimado conforme a equação (9):

$$\nabla_{\theta} J(\theta) = E_{\pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(u|x) \cdot Q^{\pi}(x, u)] \quad (9)$$

A equação (9), indica que a política deve ser ajustada para aumentar a probabilidade das ações que resultaram em maiores retornos. Uma das principais vantagens das abordagens *policy-based* é sua aplicabilidade a espaços de ação contínuos, em que algoritmos como *Q-Learning* se tornam limitados. No entanto, esses métodos tendem a apresentar alta variância nas estimativas de gradiente, o que pode comprometer a estabilidade do treinamento.

Para mitigar esse problema, surgem os métodos *Actor-Critic*, que combinam as abordagens *value-based* e *policy-based*. A arquitetura *Actor-Critic* utiliza dois módulos distintos: o ator, responsável por representar a política $\pi_{\theta}(u | x)$, e o crítico, que estima a função de valor, auxiliando o ator na atualização da política. Essa estrutura é ilustrada na Figura 2, como uma extensão do ciclo de interação padrão mostrado na Figura 1.

Figura 2 – Arquitetura *Actor-Critic* no *Reinforcement Learning*



Fonte: SHAKYA; PILLAI; CHAKRABARTY (2023)

Dentro dessa arquitetura, o algoritmo *Proximal Policy Optimization* (PPO), proposto por Schulman et al. (2017) da equipe da OpenAI, representa o estado da arte no contexto do DRL. O PPO realiza atualizações de gradiente baseadas em trajetórias coletadas previamente,

otimizando uma função objetivo com recorte (*clipping*) que evita grandes alterações na política, limitando o desvio em relação à política anterior. A razão de probabilidade entre as políticas nova e antiga é descrita na equação (10):

$$r_t(\theta) = \frac{\pi_\theta(u_t | x_t)}{\pi_{\theta_{\text{antiga}}}(u_t | x_t)} \quad (10)$$

A função de vantagem é utilizada para medir o ganho relativo da ação escolhida em relação à média esperada para o estado. Sua definição formal é dada pela equação (11):

$$A^\pi(x, u) = Q^\pi(x, u) - V^\pi(x) \quad (11)$$

A função de perda principal do PPO é definida na equação (12), em que $\widehat{A}_t$ representa a estimativa da vantagem no tempo t , e ϵ é um hiperparâmetro que controla a margem permitida de atualização, como descrito na equação (12).

$$L^{\text{CLIP}}(\theta) = \widehat{E}_t[\min(r_t(\theta)\widehat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\widehat{A}_t)] \quad (11)$$

O uso do operador mínimo assegura que apenas atualizações benéficas sejam mantidas, enquanto grandes mudanças são penalizadas, mantendo a estabilidade do processo de treinamento. A eficiência, simplicidade e robustez do PPO o tornaram um dos algoritmos mais utilizados em aplicações com ações contínuas, como locomoção robótica e veículos autônomos.

2.3 Gêmeos Digitais

O conceito de *Digital Twin* (DT) refere-se a uma representação virtual dinâmica de um sistema físico, conectada a ele de forma bidirecional e sincronizada, com a finalidade de simular, monitorar, analisar e prever o comportamento ao longo do tempo de seu correspondente no mundo real. Essa representação digital permite que decisões sejam tomadas com base em dados atualizados do mundo físico, além de possibilitar o teste de diferentes cenários sem riscos ao sistema real com simulações preditivas. Segundo Singh et al. (2021), o DT deve possuir três elementos essenciais:

1. Modelo digital capaz de representar com fidelidade o sistema físico;

2. Dados provenientes de sensores instalados no sistema real;
3. Conexão contínua que possibilite trocas de informação entre os domínios físico e virtual.

O conceito foi inicialmente proposto por Michael Grieves, em 2002, no contexto do curso de *Product Lifecycle Management* na Universidade de Michigan, sob o nome de *Mirrored Spaces Model*. A formalização do termo *Digital Twin*, no entanto, ocorreu apenas em 2010, em um relatório técnico da NASA, no qual a tecnologia foi aplicada para o monitoramento estrutural de naves aeroespaciais. A partir desse marco, o avanço da IoT e do poder computacional possibilitou a adoção do conceito em setores como manufatura, construção civil, energia e transportes (SINGH et al., 2021). Conforme Liu et al. (2021), a evolução da tecnologia pode ser dividida em três fases: formação (2003–2011), incubação (2011–2014) e crescimento (a partir de 2015), com destaque para seu papel estratégico na Indústria 4.0.

Os *Digital Twins* podem ser classificados de acordo com sua complexidade, propósito ou escopo de aplicação. Em uma perspectiva evolutiva, distinguem-se três conceitos correlatos:

- **Digital Model:** Representação digital sem conexão direta com o ativo físico
- **Digital Shadow:** Fluxo de dados unidirecional do físico para o digital
- **Digital Twin:** Comunicação bidirecional entre os domínios virtual e real

Em relação à finalidade, os DTs podem ser classificados como descritivos (voltados ao monitoramento e visualização), preditivos (com capacidade de antecipar falhas e eventos) ou prescritivos (capazes de recomendar ou automatizar ações com base em simulações). Também é comum a classificação segundo a granularidade, variando do nível de componente ao de sistema de sistemas, como no caso de redes de veículos conectados (SINGH et al., 2021; TAO et al., 2022).

Um bom exemplo do uso de *Digital Twins* como ferramenta de monitoramento e visualização de dados pode ser encontrado no trabalho de Alva et al. (2024). Nesse estudo, foi desenvolvido o DT representado na Figura 3, com o objetivo de apoiar políticas públicas de descarbonização do setor residencial na cidade de Londres. O sistema apresenta um painel interativo que integra dados reais de consumo energético, características físicas dos edifícios, informações geoespaciais e projeções climáticas. A partir desses dados, o modelo é capaz de simular o desempenho energético de edificações residenciais antigas e avaliar diferentes estratégias de reformas, como aplicação de isolamento térmico, substituição de sistemas de aquecimento e implementação de fontes renováveis de energia, como a solar.

Figura 3 – Digital Twin para monitoramento de emissões de gases do efeito estufa



Fonte: ALVA et al. (2024)

Na indústria automotiva, o uso de DTs tem ganhado relevância com o crescimento dos veículos elétricos e autônomos. A complexidade desses veículos que integram sistemas mecânicos, elétricos, eletrônicos e de controle, exige soluções avançadas para previsão de falhas, otimização de desempenho e garantia de segurança operacional. Nesse contexto, os DTs têm sido aplicados em diversas áreas: controle de navegação autônoma, sistemas avançados de assistência ao condutor, monitoramento de saúde veicular, gerenciamento de baterias e arquiteturas inteligentes de carregamento (BHATTI; MOHAN; SINGH, 2021; WANG et al., 2022).

Essas aplicações são particularmente promissoras em veículos elétricos inteligentes, permitindo o teste seguro de funcionalidades, a personalização de componentes e a implementação de estratégias de manutenção preditiva. Segundo Bhatti, Mohan e Singh (2021), os DTs viabilizam a integração de subsistemas com diferentes níveis de fidelidade, fornecendo uma visão holística e contínua de todo o ciclo de vida do veículo, desde o projeto até o descarte.

2.4 Motores Gráficos

Motores gráficos são plataformas de desenvolvimento que integram, em um único ambiente, ferramentas para criação de aplicações interativas, como jogos e simulações. Essas plataformas oferecem suporte à renderização gráfica 2D/3D, simulação física, processamento de áudio, inteligência artificial e scripts personalizados, possibilitando a criação de experiências

visuais realistas e imersivas. Embora tenham origem na indústria de entretenimento, esses motores têm se mostrado cada vez mais relevantes em projetos de engenharia, pesquisa e desenvolvimento de sistemas ciberfísicos, especialmente quando combinados com técnicas de inteligência artificial e aprendizado por reforço.

Uma das funcionalidades mais importantes desses motores é a simulação de física, que permite a modelagem de interações complexas entre objetos, como colisões, atrito, gravidade e dinâmica veicular. Essas capacidades tornam os motores gráficos ferramentas poderosas e acessíveis para a criação de *virtual test beds*, com grande aplicabilidade em projetos de veículos autônomos, *smart mobility* e gêmeos digitais.

Dentre os motores gráficos mais utilizados, destacam-se o Unity e a Unreal Engine. O Unity é amplamente adotado em projetos acadêmicos e industriais devido à sua flexibilidade, suporte a múltiplas plataformas, capacidade de integração com sensores virtuais e compatibilidade com ferramentas externas. A Unreal Engine, por sua vez, é reconhecida por seu alto desempenho gráfico e por oferecer recursos avançados para renderização fotorrealista, sendo preferida em aplicações que exigem alta fidelidade visual. Informações técnicas e recursos das duas plataformas estão disponíveis em seus respectivos sites institucionais: Unity Technologies (2025) e Epic Games (2025).

No contexto de simulações veiculares, o Unity tem se destacado pela sua capacidade de integração com ferramentas como o simulador de tráfego SUMO e o simulador de redes OMNeT++, possibilitando ambientes híbridos que reproduzem com fidelidade as interações entre tráfego urbano, comunicação veicular e controle dinâmico de veículos. No estudo de Zhao et al. (2023), foi desenvolvido um ambiente de simulação ciberfísico combinando Unity, SUMO e OMNeT++, capaz de simular, em realidade virtual, a operação de veículos autônomos em faixas dedicadas, otimizando o tráfego e a segurança viária. De forma semelhante, Liao et al. (2021) propuseram uma plataforma integrada Unity-SUMO para testar estratégias de fusão em rampas em cenários de tráfego misto, com veículos convencionais e autônomos, utilizando sensores virtuais e modelos de controle realistas.

3. ABORDAGEM PROPOSTA

Este trabalho propõe o desenvolvimento de uma *Virtual Test Bed* para simulação e treinamento de veículos autônomos em ambientes virtuais realistas, com base em técnicas de *Deep Reinforcement Learning* (DRL). A abordagem adotada utiliza a arquitetura *Actor-Critic*, com o algoritmo de otimização Proximal Policy Optimization (PPO), reconhecido por seu bom desempenho em ambientes contínuos de alta dimensionalidade, promovendo estabilidade no aprendizado e convergência eficiente.

A simulação foi implementada na Unity, uma *game engine* amplamente utilizada tanto no setor de jogos quanto em aplicações de engenharia, devido à sua acessibilidade, flexibilidade e capacidade de renderização 3D. O ambiente foi composto por duas pistas de simulação com obstáculos e pontos de progresso (*checkpoints*), onde os agentes (veículos autônomos simulados) realizam tarefas de navegação e tomada de decisão.

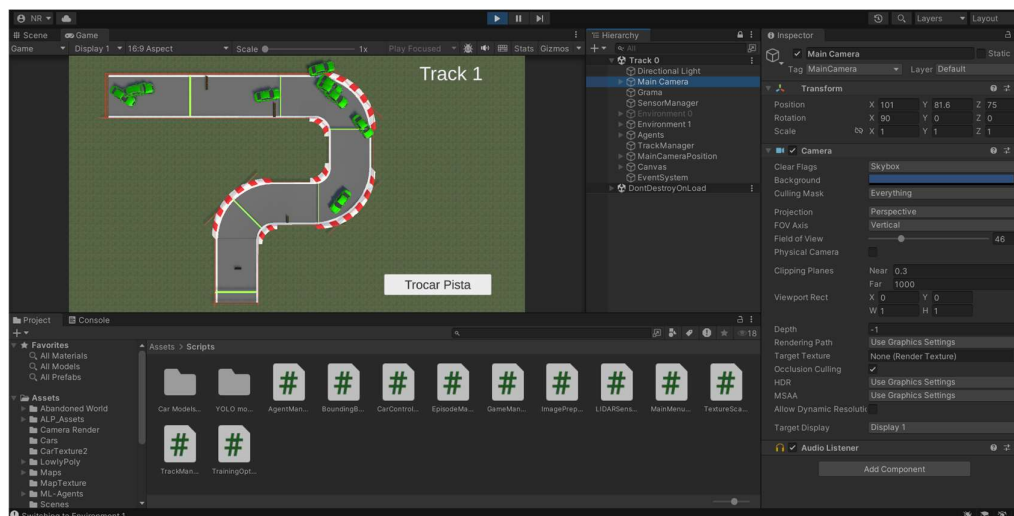
O treinamento dos agentes foi realizado por meio da extensão Unity ML-Agents Toolkit, que permite a integração de algoritmos de aprendizado por reforço diretamente no ambiente da Unity. Essa ferramenta proporciona a comunicação entre a Unity e algoritmos implementados em Python, utilizando bibliotecas como PyTorch para o treinamento da rede neural responsável pela política e pelo crítico. Como parte do sistema, foi implementada também uma estrutura de *Digital Twin* do PyTorch, o Tensorboard. Nesse *Digital Twin*, o ambiente virtual feito na Unity substitui o ambiente físico característico do *Digital Twin*, com objetivo de trazer a redução de custos nesse tipo de treinamento. O ambiente virtual gera os dados e estes são coletados durante o treinamento e organizados para fins de análise, controle e visualização.

3.1 Materiais e Métodos

A plataforma Unity utiliza a linguagem de programação C# e uma arquitetura baseada em objetos, permitindo a composição de ambientes por meio da inserção e configuração de elementos tridimensionais nas chamadas cenas (*scenes*). Cada cena representa um conjunto de objetos, componentes e scripts que são carregados e renderizados no momento da execução, otimizando o desempenho do sistema.

No editor da Unity (Figura 4), os objetos inseridos na cena são organizados na janela Hierarchy, que define suas relações parentais e estrutura hierárquica. As propriedades de cada objeto são exibidas no painel Inspector, onde podem ser configurados parâmetros como transformações espaciais, componentes de física e scripts de controle.

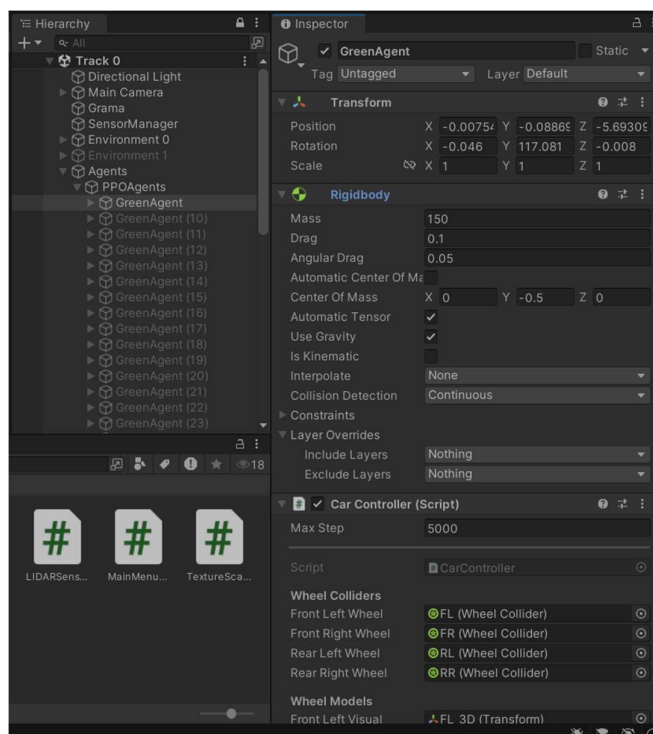
Figura 4 – Editor Unity



Fonte: Autoria própria.

Na Figura 5, observa-se um exemplo de agente configurado na Unity. As propriedades incluem o componente *Transform*, que define posição, rotação e escala; o *Rigidbody*, responsável pela simulação de física e colisões do objeto; e o script responsável pelo comportamento e comunicação com o sistema de aprendizado por reforço.

Figura 5 – Lista de Objetos em cena e Inspetor da Unity



Fonte: Autoria própria.

Para a construção do ambiente do projeto, foram utilizados modelos 3D gratuitos e sem restrição de direitos autorais, obtidos na plataforma Sketchfab (SKETCHFAB, 2025). O modelo da pista foi criado por Quarks Studios (QUARKS STUDIOS, 2020), com topologia ajustada para compor o traçado das pistas simuladas. Já o modelo do veículo, um Lexus GS 350 2011, foi desenvolvido por 3DShowroom (3DSHOWROOM, 2017) e adaptado para uso na Unity.

Os modelos foram inicialmente importados para o software Blender, onde foram realizados ajustes nas escalas, aplicação de texturas e exportação para o formato FBX, compatível com a Unity. A Figura 6 apresenta o modelo do carro no Blender, já preparado para exportação.

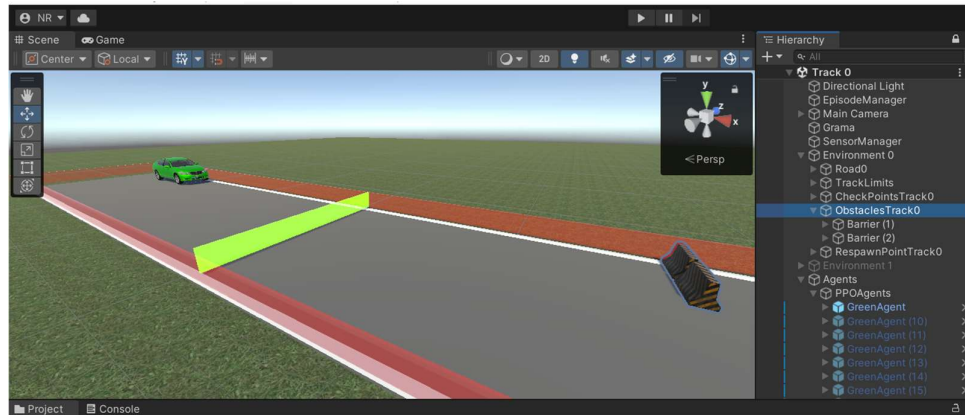
Figura 6 – Modelo do carro utilizado no software Blender



Fonte: Autoria própria.

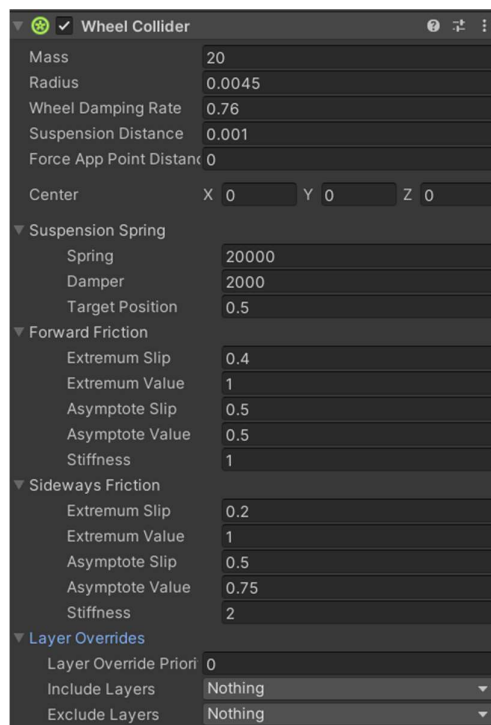
Com os modelos devidamente preparados, foi realizada a montagem do ambiente de simulação na Unity, incluindo a construção da pista, a inserção de obstáculos e a definição dos pontos de progresso (*checkpoints*), utilizados para guiar o treinamento do agente e determinar as recompensas. Os obstáculos implementados consistem em barreiras físicas, representadas por blocos de concreto, e barreiras virtuais, que delimitam as extremidades da pista. Conforme ilustrado na Figura 7, as barras vermelhas representam os limites virtuais de pista, enquanto as verdes indicam os checkpoints. Além disso, foi implementado um sistema de respawn point aleatório numa área antes do primeiro checkpoint, com objetivo trazer aumento nos dados (*data augmentation*) no treinamento.

Figura 7 – Implementação da pista na cena



Fonte: Autoria própria.

Após a inserção do modelo do veículo, foram implementados os comandos de dirigibilidade e os sensores responsáveis pela percepção do ambiente. As ações de saída do agente foram configuradas como variáveis contínuas, associadas à direção e aceleração. A simulação do comportamento do veículo foi realizada por meio do componente *Wheel Collider*, aplicado individualmente a cada roda. Esse componente simula o comportamento físico da suspensão, tração e atrito das rodas com o solo, com base no motor de física PhysX da Unity.

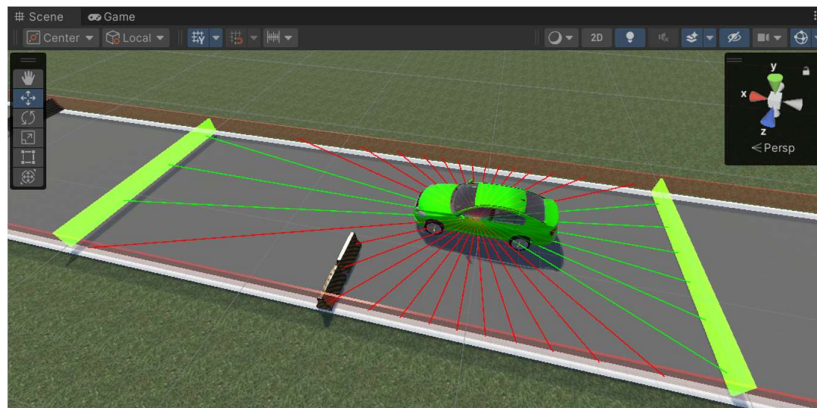
Figura 8 – Configuração do componente *Wheel Collider* utilizado nos veículos

Fonte: Autoria própria.

Para simular a percepção do ambiente pelo veículo, foi implementado um sensor baseado em *raycast*, que emite raios em múltiplas direções a partir do veículo, simulando o funcionamento de sensores LiDAR. Esses sensores, amplamente utilizados em veículos autônomos, operam por meio da emissão de pulsos de luz (geralmente infravermelha), que retornam ao sensor ao colidirem com objetos, permitindo calcular distâncias com alta precisão (YURTSEVER et al., 2020).

Na Unity, o sensor foi implementado de forma simplificada por meio de um *raycast* horizontal, que emite diversos raios a partir do centro do veículo, conforme apresentado na Figura 9. Esses raios retornam a distância até o objeto detectado, além de identificar a etiqueta TAG atribuída ao objeto atingido. As TAGs são rótulos de classificação utilizados para identificar elementos na cena e orientar as decisões do agente. Foram utilizadas as seguintes TAGs: "*fora da pista*", "*obstáculo*", "*pista*", "*chão*", "*carros*", "*objetos*" e "*checkpoint*". Adicionalmente, foram implementados quatro *raycasts* verticais posicionados nas rodas, com a função de identificar a quantidade de rodas que permanecem dentro dos limites da pista, permitindo refinar o sistema de recompensas com base no comportamento do veículo.

Figura 9 – *Raycast* utilizado como sensor de distância do veículo

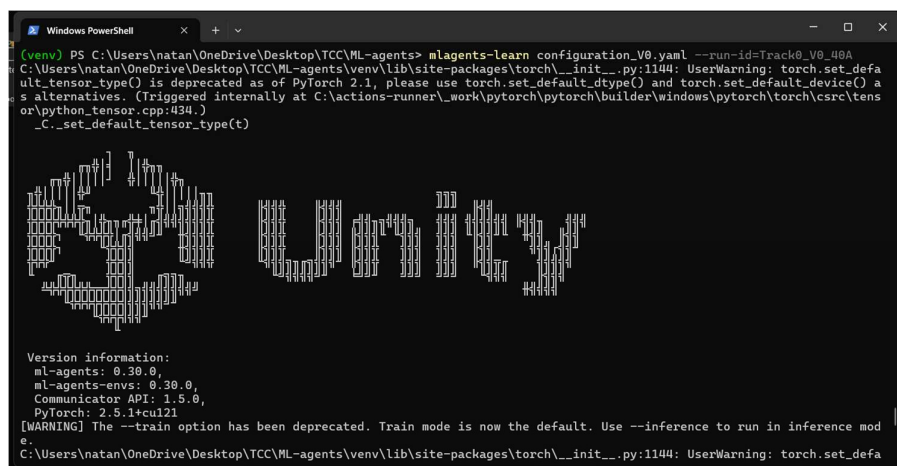


Fonte: Autoria própria.

A integração entre o ambiente simulado e o treinamento do agente foi realizada por meio da extensão Unity ML-Agents Toolkit, que permite a comunicação entre a *game engine* e algoritmos de aprendizado por reforço desenvolvidos em Python. Essa extensão é composta por dois módulos principais: o Python API, responsável pelo treinamento externo do modelo, e o ambiente Unity, que realiza a simulação e envia as observações e recompensas ao script Python. A troca de informações entre os dois módulos é feita por meio de um protocolo baseado em gRPC, garantindo sincronização eficiente durante os episódios de treinamento.

Para a execução do treinamento, foi criado um ambiente virtual em Python por meio do terminal (Command Prompt ou PowerShell). A partir desse ambiente, os comandos da extensão Unity ML-Agents foram utilizados para iniciar e gerenciar o processo de treinamento, conforme ilustrado na Figura 10.

Figura 10 – Ambiente virtual em Python para o treinamento



```

(venv) PS C:\Users\natan\OneDrive\Desktop\TCC\ML-agents> mlagents-learn configuration_V0.yaml --run-id=Track0_V0_40A
C:\Users\natan\OneDrive\Desktop\TCC\ML-agents\venv\Lib\site-packages\torch\__init__.py:1144: UserWarning: torch.set_default_tensor_type() is deprecated as of PyTorch 2.1, please use torch.set_default_dtype() and torch.set_default_device() as alternatives. (Triggered internally at C:\actions-runner\work\pytorch\pytorch\builder\windows\pytorch\torch\csrc\tensor_python\tensor.cpp:494.)
  _C._set_default_tensor_type(t)

Version information:
ml-agents: 0.30.0,
ml-agents-envs: 0.30.0,
Communicator API: 1.5.0,
PyTorch: 2.5.1+cu121
[WARNING] The --train option has been deprecated. Train mode is now the default. Use --inference to run in inference mode.
C:\Users\natan\OneDrive\Desktop\TCC\ML-agents\venv\Lib\site-packages\torch\__init__.py:1144: UserWarning: torch.set_default_tensor_type() is deprecated as of PyTorch 2.1, please use torch.set_default_dtype() and torch.set_default_device() as alternatives. (Triggered internally at C:\actions-runner\work\pytorch\pytorch\builder\windows\pytorch\torch\csrc\tensor_python\tensor.cpp:494.)
  _C._set_default_tensor_type(t)

```

Fonte: Autoria própria.

O treinamento do agente foi conduzido utilizando a biblioteca PyTorch, que oferece suporte à criação e otimização de redes neurais profundas. Durante o aprendizado, foram registradas métricas como recompensas acumuladas, entropia da política, perdas das funções de valor e política, entre outras. Essas métricas foram monitoradas durante o treinamento por meio da ferramenta TensorBoard, conforme apresentado anteriormente na Figura 3, permitindo o ajuste dinâmico dos hiperparâmetros e a avaliação do desempenho do agente.

Para acelerar o processo de treinamento, foi utilizado o CUDA Toolkit, com suporte a uma GPU NVIDIA compatível. A tecnologia CUDA permite que os cálculos tensoriais do PyTorch sejam distribuídos nos núcleos da GPU, reduzindo significativamente o tempo de treinamento em comparação com a execução em CPU. Essa aceleração foi fundamental para viabilizar múltiplas iterações e ajustes no modelo em um intervalo de tempo viável. Além disso, foi utilizado um ambiente de academia, em que vários agentes são inseridos na simulação e cada um passa por seu próprio episódio e pode tomar suas próprias decisões, mas que ao mesmo tempo as experiências adquiridas por cada agente são utilizadas para otimizar um único algoritmo DLR, dessa forma, agilizando também o treinamento.

Ao final do treinamento, o modelo otimizado foi exportado no formato ONNX (*Open Neural Network Exchange*), compatível com a Unity. Para executar o modelo treinado

diretamente no ambiente Unity, foi utilizada a extensão Unity Barracuda, uma biblioteca de inferência neural que permite rodar redes neurais ONNX de forma eficiente dentro da *engine*, sem necessidade de dependências externas como Python.

O design da função de recompensa implementada no agente seguiu os princípios discutidos por Knox et al. (2023), que enfatizam a importância de recompensas alinhadas com os objetivos do comportamento desejado e alertam sobre os riscos da modelagem de recompensas (*reward shaping*) mal planejada.

A Tabela 1 apresenta um resumo dos principais componentes da função de recompensa e seus respectivos valores, determinados por meio de uma abordagem heurística, considerando análises experimentais do comportamento dos agentes durante o treinamento.

Tabela 1 – Componentes da função de recompensa na abordagem aplicada

Ação	Condição de Aplicação	Recompensa / Penalidade
Passar por um <i>checkpoint</i>	Ao passar pelos checkpoints na ordem correta	+0,3
Bônus por tempo entre <i>checkpoint</i>	Proporcional à velocidade entre dois <i>checkpoints</i>	+0,1 a +1,0
Alinhamento do carro com o próximo <i>checkpoint</i>	Direção do carro alinhada ao próximo <i>checkpoint</i> e avanço em direção ao próximo <i>checkpoint</i>	Até +0,01 por passo
Finalização do percurso	Ao passar pelo último <i>checkpoint</i>	+10,00 + encerramento do episódio
Rodas fora da pista	Penalidade contínua proporcional a quantidade de rodas fora da pista (1 a 3 rodas)	-0,1 por roda
Todas as rodas fora da pista	As quatro rodas do veículo fora da pista	-0,5 + encerramento do episódio
Imobilidade	Após 1 segundo com velocidade baixíssima	-0,01 por segundo
Colisão com obstáculo	Colisão com qualquer obstáculo na pista	-7,0 + encerramento do episódio

Fonte: Autoria própria.

As recompensas são contabilizadas no fim de cada episódio, seja ele forçado após o número de passos passar o limite pré-estabelecido ou após alcançar as recompensas ou penalizações que forçam o fim dele. Esse conjunto de recompensas foi concebido com base em boas práticas discutidas no artigo, como a inclusão de múltiplos atributos relevantes que se deseja incentivar (progresso e eficiência) e a retirada de recompensas que favoreçam atalhos ou comportamentos não generalizáveis. A estratégia adotada procurou balancear o aprendizado eficiente com a preservação da integridade comportamental esperada de um veículo autônomo em ambiente urbano simulado.

Os hiperparâmetros utilizados foram configurados com base nas recomendações da documentação oficial do ML-Agents e ajustados heurísticamente ao longo do desenvolvimento, por meio de análise iterativa dos resultados dos treinamentos. Os parâmetros são definidos em um arquivo YAML, como ilustrado na Figura 11, gerado automaticamente ao iniciar um treinamento.

Figura 11 – Arquivo YAML de configuração de hiperparâmetros

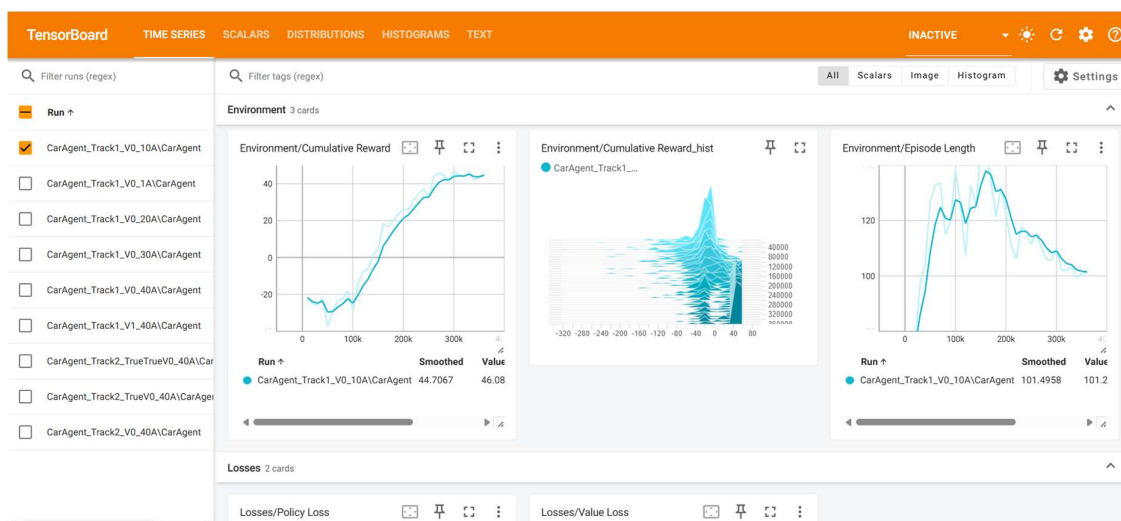
```

! configuration_V0.yaml X
C:\> Users > natan > OneDrive > Desktop > TCC > ML-agents > ! configuration_V0.yaml
1  default_settings: null
2  behaviors:
3    CarAgent:
4      trainer_type: ppo
5      hyperparameters:
6        batch_size: 2048
7        buffer_size: 20480
8        learning_rate: 0.0007
9        beta: 0.01
10       epsilon: 0.2
11       lambda: 0.9
12       num_epoch: 13
13       shared_critic: false
14       learning_rate_schedule: linear
15       beta_schedule: linear
16       epsilon_schedule: linear
17     network_settings:
18       normalize: true
19       hidden_units: 256
20       num_layers: 3
21       vis_encode_type: simple
22       memory: null
23     goal_conditioning_type: hyper
24     deterministic: false
25     reward_signals:
26       extrinsic:
27         gamma: 0.9
28         strength: 1.0
29         network_settings:
30           normalize: false
31           hidden_units: 128
32           num_layers: 2
33           vis_encode_type: simple
34           memory: { sequence_length: 64, memory_size: 128 }
35           goal_conditioning_type: hyper
36           deterministic: false
37     init_path: null
  
```

Fonte: Autoria própria.

O monitoramento do desempenho dos agentes durante o treinamento foi realizado por meio do TensorBoard, uma ferramenta que permite visualizar métricas fundamentais como recompensa acumulada, entropia da política e estimativas de valor da rede crítica. Nesse *Digital Twin*, os dados do ambiente virtual são coletados durante o treinamento, organizados e apresentados para fins de análise, controle e visualização do processo de aprendizado. Para acessar o DT, foi necessário iniciar um servidor local na pasta onde os resultados dos treinamentos eram salvos. Dessa forma, o painel poderia ser acessado por meio de um navegador web, conforme ilustrado na Figura 12.

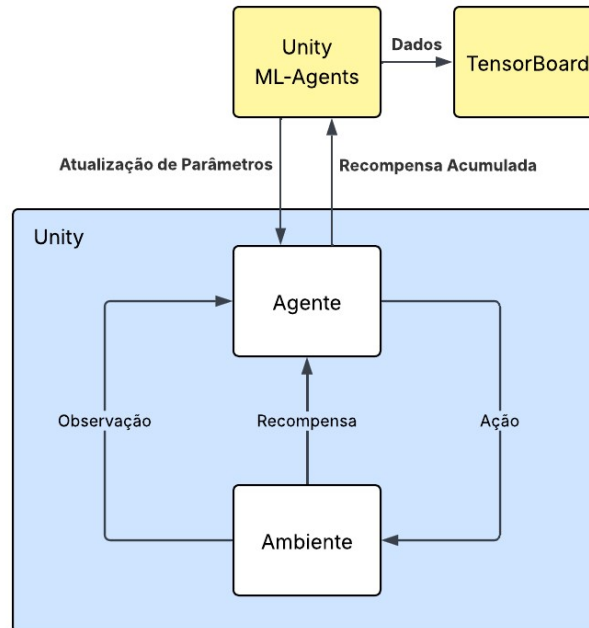
Figura 12 – Dashboard do TensorBoard



Fonte: Autoria própria.

A Figura 13 apresenta o diagrama de blocos da estrutura da abordagem desenvolvida neste trabalho. Conforme já mencionado, dentro do Unity ocorre a interação entre o ambiente 3D e o agente, mediada pelas observações que o agente realiza do ambiente. Com base nessas observações, o agente, por meio de sua política, gera ações que, ao serem executadas no ambiente, resultam em recompensas. Essas recompensas são então utilizadas pela extensão Unity ML-Agents para atualizar os parâmetros do agente. Além disso, os dados gerados durante o treinamento podem ser visualizados em tempo real por meio do TensorBoard.

Figura 13 – Diagrama de Blocos da abordagem proposta



Fonte: Autoria própria.

O agente realiza a coleta de observações por meio de sensores simulados e propriedades físicas do veículo. As observações incluem leituras do sensor LiDAR, que fornecem distâncias em múltiplas direções ao redor do carro, informações sobre a velocidade linear e angular do corpo rígido (Rigidbody), ângulos de direção das rodas dianteiras e, adicionalmente, vetores de direção e distância até o próximo checkpoint. Esses dados são normalizados e enviados à política de decisão, representando a percepção do agente sobre o ambiente e servindo como base para o processo de inferência das ações.

Com base nas observações coletadas, a política do agente retorna um conjunto de três ações contínuas, que são responsáveis por controlar o comportamento dinâmico do veículo no ambiente simulado. As ações consistem em um valor de esterço do volante, um valor de aceleração e um valor de frenagem. Essas saídas são aplicadas diretamente nos componentes físicos do modelo.

3.2 Setup Experimental

Os experimentos de treinamento e simulação foram realizados em um computador pessoal com as seguintes especificações de hardware:

- **Processador:** AMD Ryzen 7 5800H com Radeon Graphics @ 3.20 GHz
- **Memória RAM:** 16 GB DDR4
- **Placa de Vídeo (GPU):** NVIDIA GeForce RTX 3060 Laptop GPU com 6 GB de VRAM
- **Sistema Operacional:** Windows 11 Home (64 bits)

Durante o treinamento, o ambiente Unity foi executado em modo *headless* (sem renderização gráfica) com escala de tempo (*time scale*) de 20, permitindo uma simulação em tempo acelerado e eficiente. A configuração dos hiperparâmetros utilizados na simulação está detalhada na Tabela 2, em que para os valores de épsilon, beta e taxa de aprendizado (*learning rate*) foram utilizados com uma técnica de decaimento linear para incentivar a exploração no começo do treinamento e convergir nas fases finais.

A rede neural utilizada pelo agente foi configurada com três camadas ocultas contendo 256 unidades cada, com normalização das entradas, permitindo uma aprendizagem estável e progressiva em cenários contínuos com múltiplas variáveis de entrada.

Além disso, foi incorporada uma memória recorrente com tamanho de sequência (*sequence length*) de 64 e tamanho de memória (*memory size*) de 128, que fornece ao agente a capacidade de lembrar informações de passos anteriores ao longo do episódio. Essa memória é implementada por meio de uma camada LSTM (*Long Short-Term Memory* – Memória de Curto Longo Prazo), permitindo que o agente reconheça padrões temporais e tome decisões baseadas em sequências de observações, o que é particularmente útil em tarefas de navegação contínua como a condução veicular.

Tabela 2 – Hiperparâmetros utilizados na simulação

Hiperparâmetro	Valor
<i>Batch_size</i>	2048
<i>buffer_size</i>	20480
<i>learning_rate</i>	0,0007
<i>epsilon</i>	0,2
<i>gamma</i>	0,9

num_epoch	13
lambda	0,9
beta	0,01

Fonte: Autoria própria.

Os treinamentos foram realizados com uma quantidade máxima de quarenta agentes simultâneos para evitar prejuízo na performance da simulação, além de um máximo de um milhão de passos por treino. Essa configuração foi determinante para o desempenho do agente, assegurando um processo de aprendizado progressivo, estável e alinhado aos objetivos da simulação proposta.

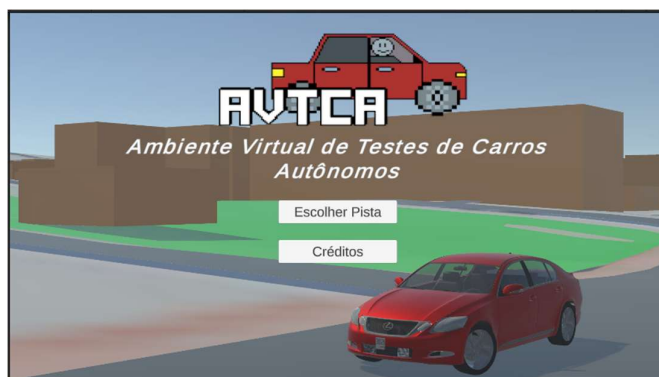
4. RESULTADOS E DISCUSSÕES

Nesta seção, são apresentados os principais resultados obtidos com a implementação do ambiente de simulação e treinamento de veículos autônomos, desenvolvido na plataforma Unity com técnicas de Aprendizado por Reforço Profundo (DRL). A discussão é organizada de acordo com os componentes que compõem o projeto: ambiente de simulação e validação do treinamento.

4.1 Ambiente de Simulação

A interface inicial da aplicação (*Main Menu*) permite ao usuário selecionar a pista desejada e o número de agentes ativos durante o treino, conforme ilustrado nas Figuras 14 e 15, respectivamente. Essa funcionalidade é essencial para a organização dos testes e permite configurar diferentes cenários de simulação com base na complexidade da pista e no número de agentes treinados em paralelo.

Figura 14 – *Main Menu*



Fonte: Autoria própria.

Figura 15 – Tela de escolha de pista e quantidade de agentes no treinamento

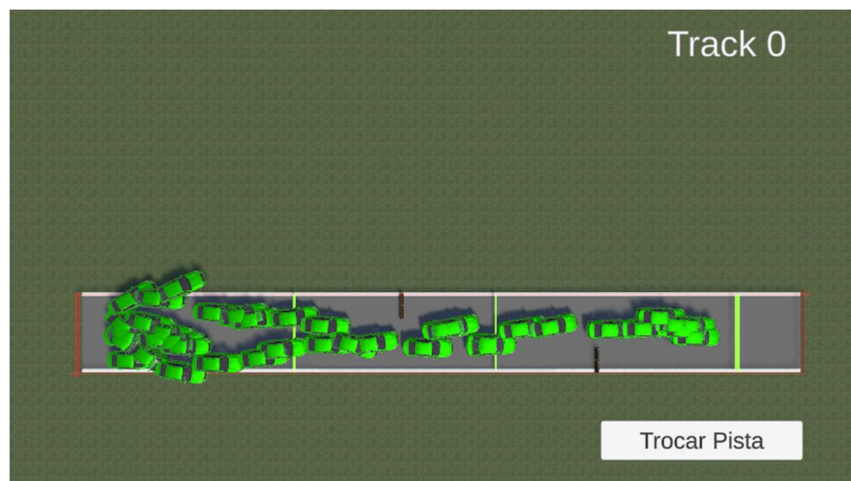


Fonte: Autoria própria.

Ao clicar no botão “Escolher Pista” (Figura 14), o usuário é redirecionado para a tela de configuração de simulação (Figura 15), onde pode ajustar, por meio de um controle deslizante (*slider*), a quantidade de agentes que serão instanciados durante o treinamento. O valor numérico exibido é utilizado na configuração da cena ao selecionar a pista.

As duas pistas foram projetadas com diferentes níveis de dificuldade. A Pista 0 (*Track 0*) apresenta um trajeto mais simples, composto por uma reta curta com três checkpoints e alguns obstáculos distribuídos ao longo do percurso (Figura 16). Já a Pista 1 (*Track 1*) é mais desafiadora, contendo uma sequência de curvas fechadas, maior densidade de obstáculos e seis checkpoints ao longo do trajeto (figura 17).

Figura 16 – *Track 0*



Fonte: Autoria própria.

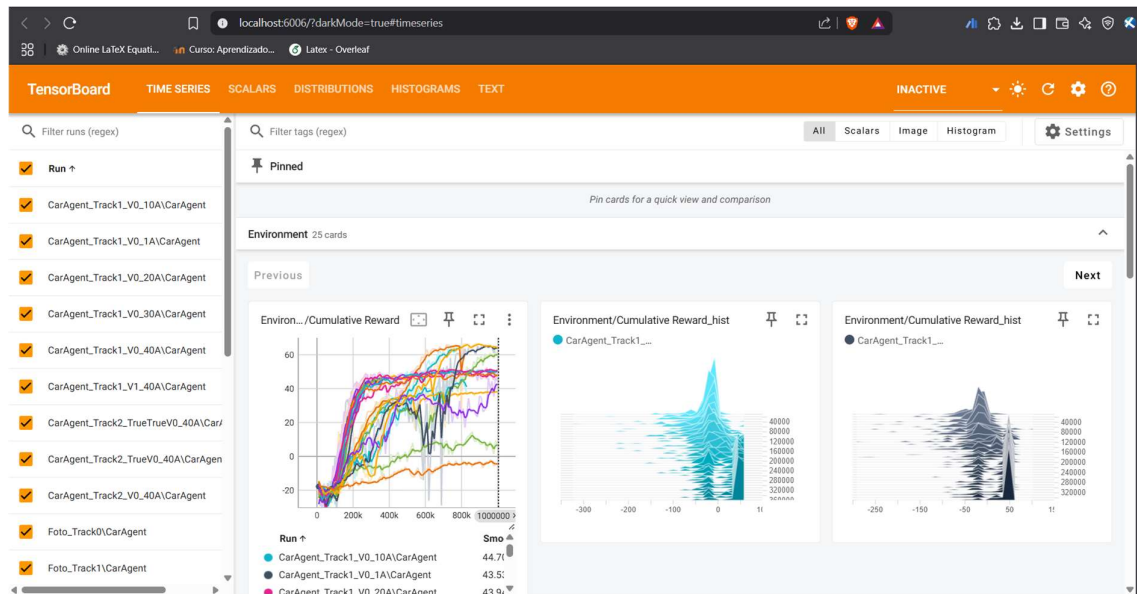
Figura 17 – *Track 1*



Fonte: Autoria própria.

Juntamente com o ambiente de treinamento, o *Digital Twin* do TensorBoard foi implementado corretamente utilizando a biblioteca PyTorch e um servidor local, conforme ilustrado na Figura 18. Nessa interface web, é possível acompanhar a evolução de variáveis relevantes do treinamento, como métricas do ambiente (*environment*), perdas (*losses*) e parâmetros da política (*policy*).

Figura 18 – Visualização web do *Digital Twin* do Tensorboard

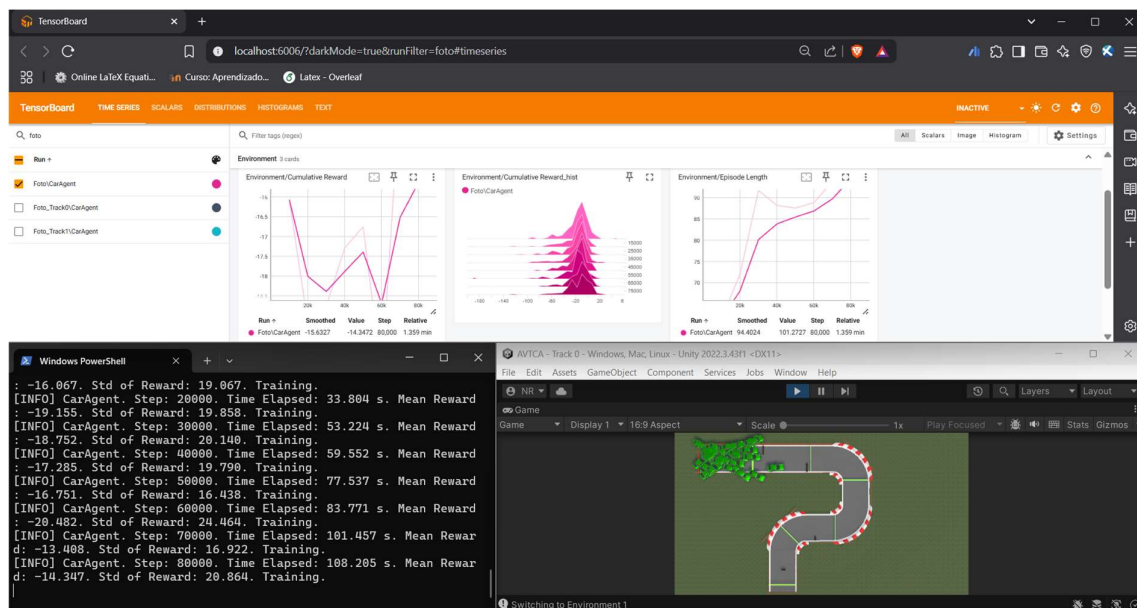


Fonte: Autoria própria.

Na Figura 18, observa-se a geração automática dos gráficos de treino utilizados para testes e validação do ambiente. A página do TensorBoard é criada a partir da pasta local em que o servidor foi iniciado, permitindo ao usuário visualizar resultados de múltiplas sessões de treino armazenadas em uma mesma estrutura de diretórios ou filtradas por experiências específicas.

A Figura 19 apresenta o ambiente final implementado, onde, na parte superior, está o TensorBoard exibindo os gráficos de desempenho, e na parte inferior, o terminal de execução com os scripts em Python (Unity ML-Agents), juntamente com a simulação do treinamento no ambiente Unity. O funcionamento do ambiente pode ser visualizado através do vídeo disponível em: https://youtu.be/CbJABsToDt8?si=KrLd-MscVf_G-INc.

Figura 19 – Ambiente final de treino e simulação implementado



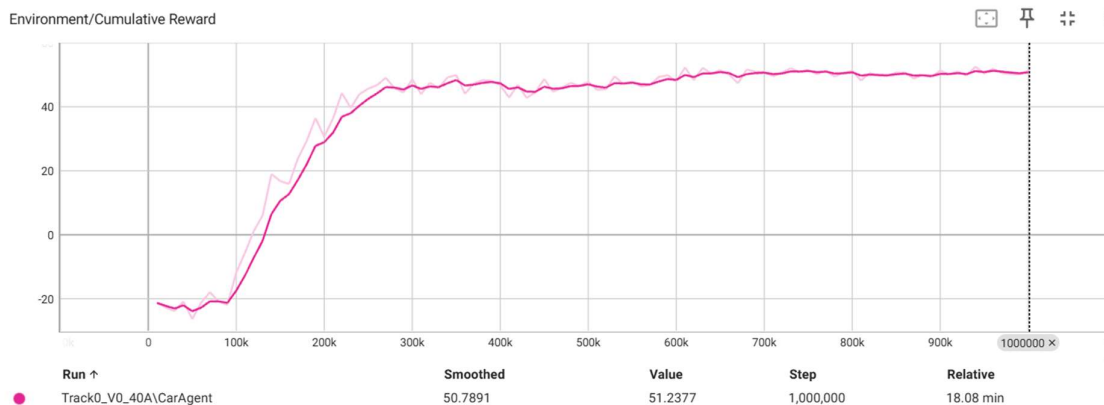
Fonte: Autoria própria.

4.2 Validação do Treinamento

Para validação do ambiente simulado e da arquitetura de treinamento, foi realizada uma sessão de treino com base nos hiperparâmetros apresentados na Tabela 2, utilizando 40 agentes ativos simultaneamente. Ao longo do processo, foi possível observar a evolução do comportamento dos agentes, que passaram por uma fase inicial de exploração até convergirem para estratégias mais eficientes. Essa evolução foi acompanhada por meio dos gráficos gerados pelo TensorBoard.

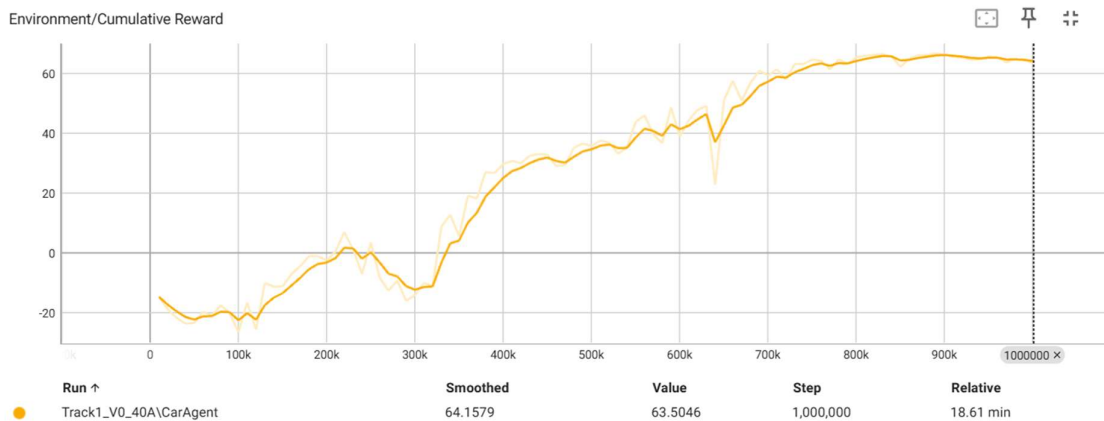
A Figura 20 mostra a evolução da recompensa cumulativa dos agentes treinados na Pista 0, enquanto a Figura 21 apresenta o mesmo indicador para a Pista 1. Em ambas as simulações, observa-se um crescimento contínuo das recompensas acumuladas à medida que o número total de passos aumenta, evidenciando o aprendizado progressivo. Como esperado, a convergência foi mais rápida na Pista 0, devido à sua menor complexidade. Já a Pista 1 apresentou maior oscilação, reflexo dos desafios adicionais impostos pela pista.

Figura 20 – Recompensas cumulativas com 40 agentes na Pista 0



Fonte: Autoria própria.

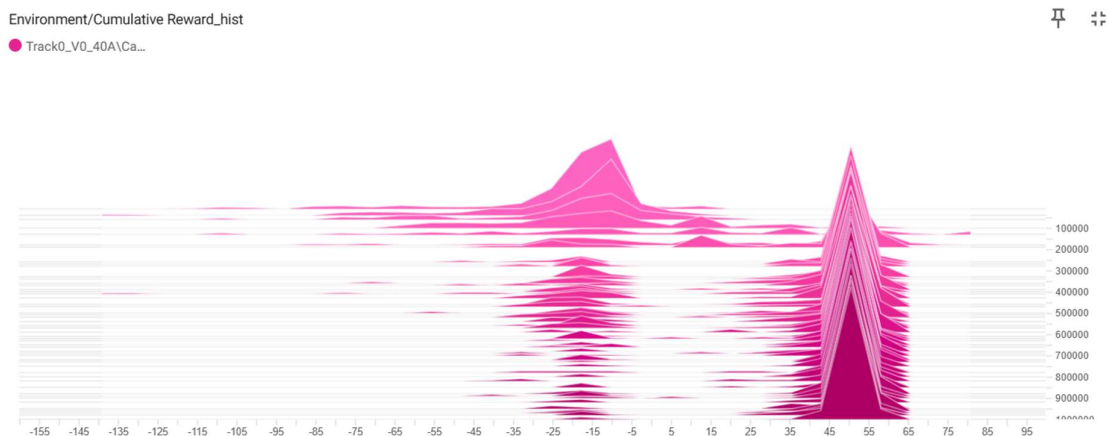
Figura 21 – Recompensas cumulativas com 40 agentes na Pista 1



Fonte: Autoria própria.

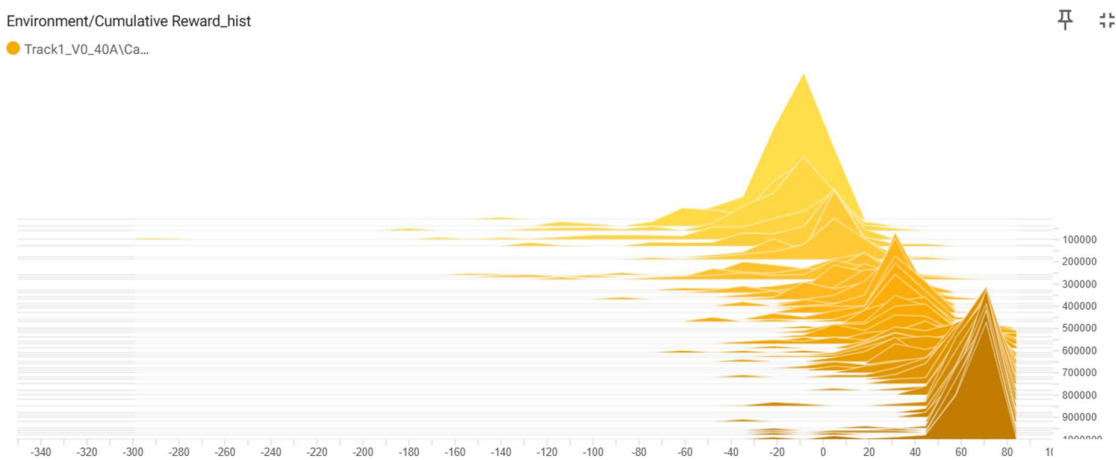
As Figuras 22 e 23 apresentam histogramas da distribuição das recompensas cumulativas, registrados a cada 100 mil passos. Esses gráficos permitem observar a evolução da variabilidade das estratégias ao longo do tempo. Nas fases iniciais, as recompensas são mais dispersas, indicando a etapa de exploração. Com o avanço do treinamento, há uma concentração crescente em faixas de maior retorno, sinalizando convergência para políticas mais eficazes. A convergência na Pista 0 ocorre por volta de 200 mil passos, enquanto na Pista 1 apenas após aproximadamente 1 milhão de passos, conforme também observado nas Figuras 20 e 21.

Figura 22 – Histograma das recompensas cumulativas com 40 agentes na Pista 0



Fonte: Autoria própria.

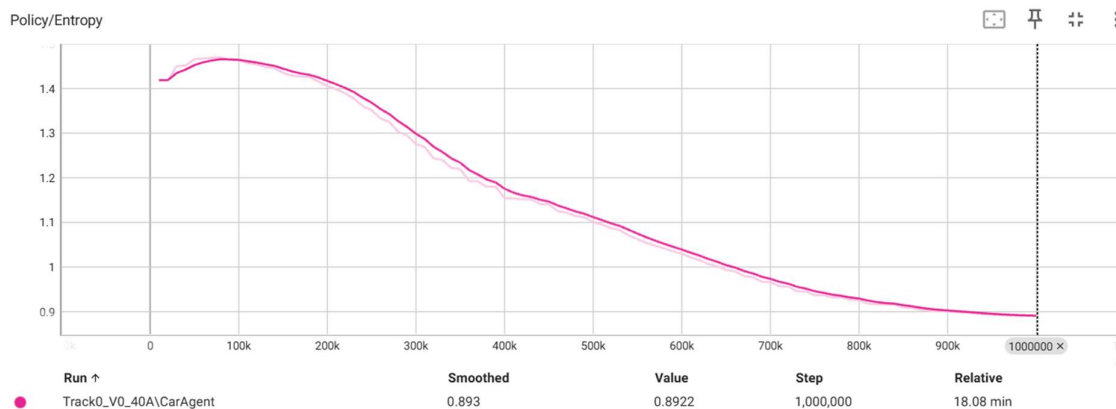
Figura 23 – Histograma das recompensas cumulativas com 40 agentes na Pista 1



Fonte: Autoria própria.

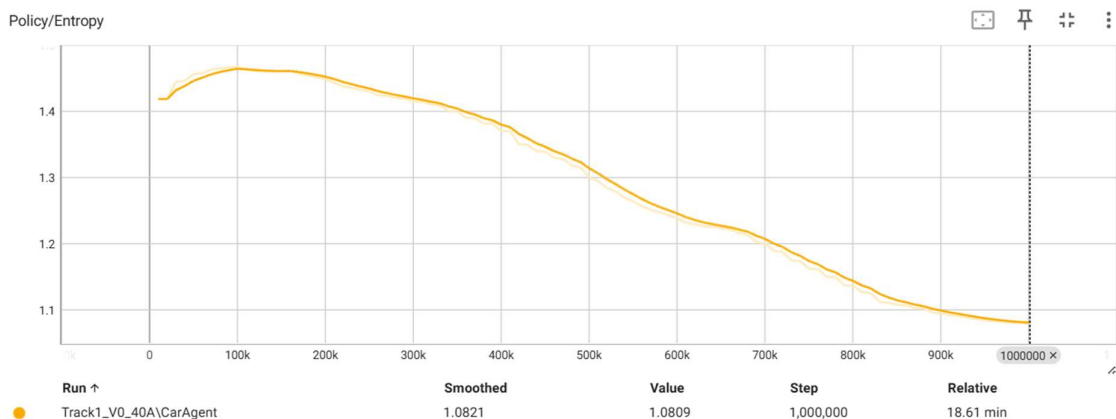
Outra métrica importante é a entropia da política, que representa o grau de aleatoriedade nas decisões do agente. As Figuras 24 e 25 mostram que, em ambas as pistas, há um padrão de decaimento progressivo da entropia, indicando uma transição natural do comportamento exploratório para um comportamento mais determinístico, à medida que a política se estabiliza. Inicialmente, a entropia média está acima de 1,4, refletindo escolhas estocásticas. Na Pista 0, a entropia final converge para aproximadamente 0,9, enquanto na Pista 1, permanece mais elevada (cerca de 1,1), condizente com a maior complexidade e variabilidade do ambiente.

Figura 24 – Evolução da entropia com 40 agentes na Pista 0



Fonte: Autoria própria.

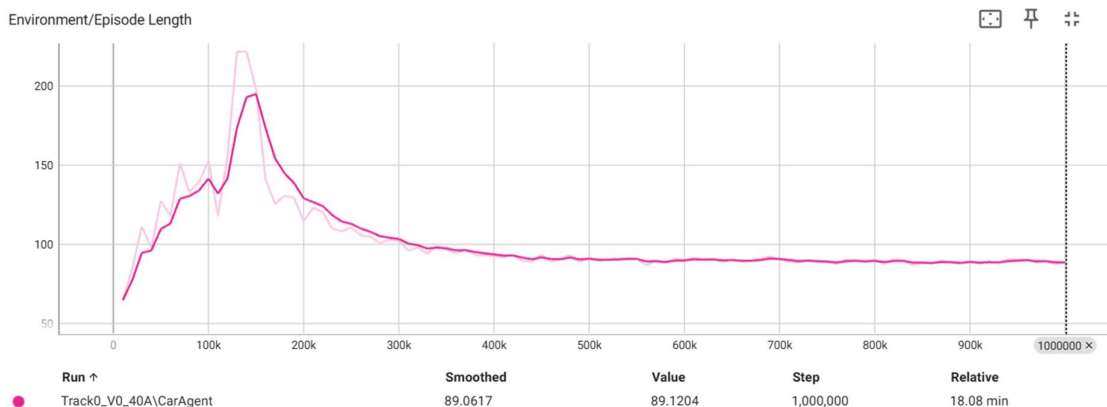
Figura 25 – Evolução da entropia com 40 agentes na Pista 1



Fonte: Autoria própria.

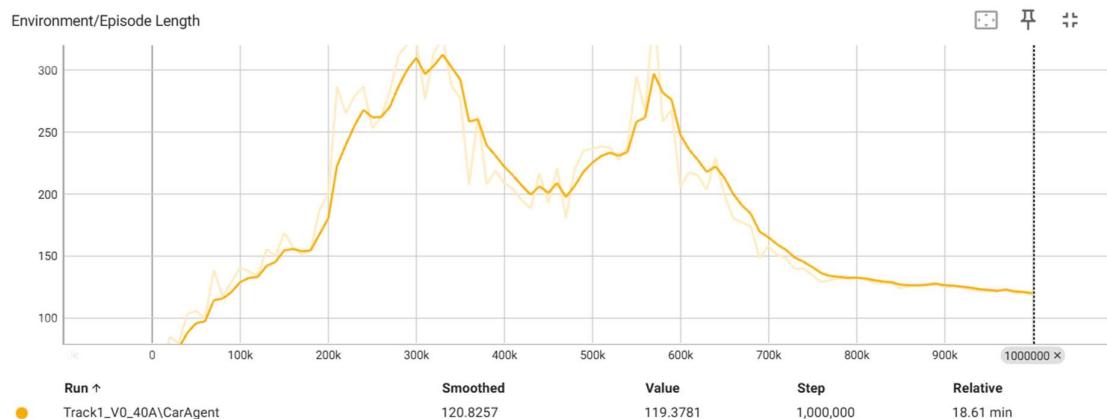
Outro indicador relevante para avaliar o aprendizado dos agentes é a duração dos episódios ao longo do tempo, ilustrada nas Figuras 26 e 27. No início do processo, os episódios são mais longos, indicando que os agentes permanecem mais tempo na pista explorando o ambiente. Com o avanço do treinamento e a descoberta de trajetórias mais otimizadas, o tempo médio de episódio diminui. Esse comportamento, aliado ao aumento da recompensa acumulada, sugere que os agentes passaram a completar o trajeto de forma mais eficiente. Os picos de duração observados estão associados a momentos em que os agentes realizam manobras repetitivas (como avançar e recuar) ao tentar evitar penalizações ou obstáculos.

Figura 26 – Evolução do tempo de episódio com 40 agentes na Pista 0



Fonte: Autoria própria.

Figura 27 – Evolução do tempo de episódio com 40 agentes na Pista 1

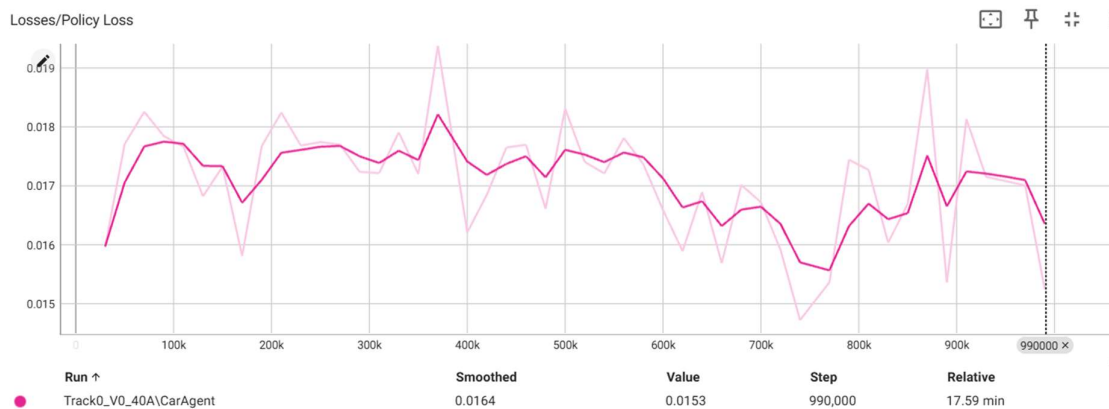


Fonte: Autoria própria.

Além da análise comportamental, também foram avaliadas métricas relacionadas ao processo de otimização das redes neurais. Os gráficos de perda de política (Policy Loss) e perda de valor (Value Loss), apresentados nas Figuras 28 a 31, permitem avaliar a estabilidade do algoritmo de aprendizado por reforço profundo (PPO).

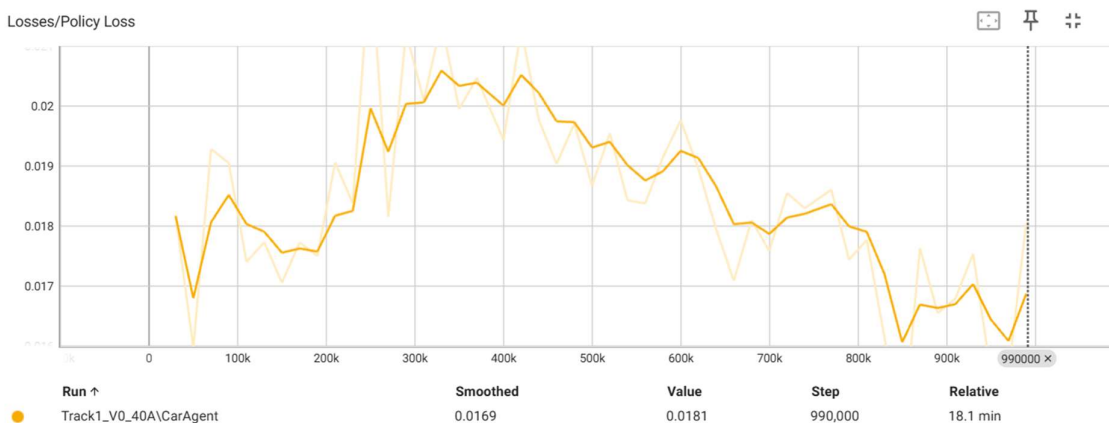
A Figura 28 mostra que a perda de política na Pista 0 permaneceu estável ao longo do treinamento, com valores médios em torno de 0,017. Já na Pista 1 (Figura 29), os valores são ligeiramente superiores, entre 0,017 e 0,020, com maior variação. Isso é esperado, pois a política precisa lidar com situações mais complexas, exigindo maior flexibilidade e adaptação.

Figura 28 – Evolução da perda de política com 40 agentes na Pista 0



Fonte: Autoria própria.

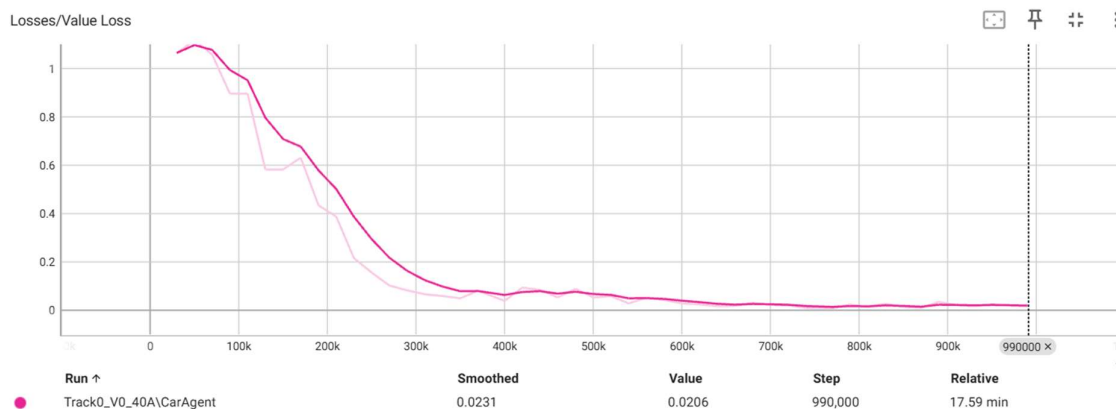
Figura 29 – Evolução da perda de política com 40 agentes na Pista 1



Fonte: Autoria própria.

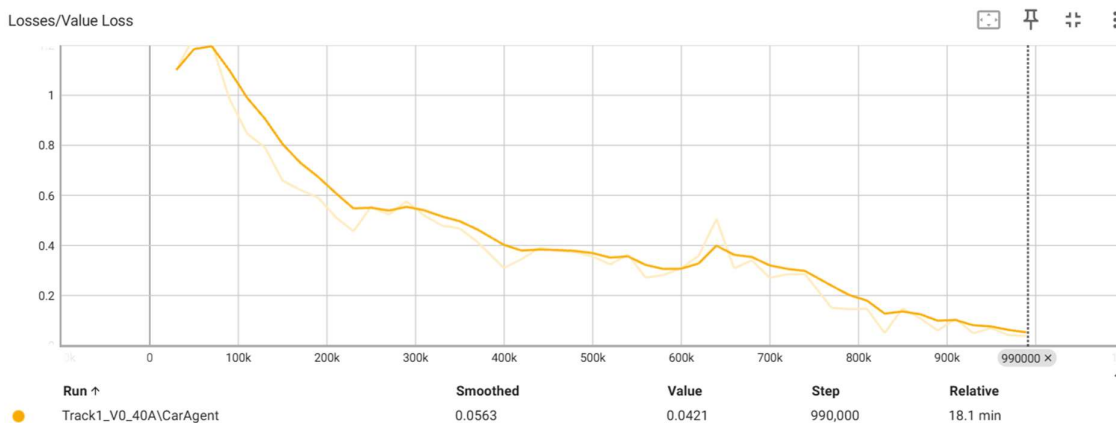
Na Pista 0, a perda de valor (Figura 30) apresenta rápida convergência nos primeiros 300 mil passos, saindo de valores superiores a 1,0 e estabilizando-se abaixo de 0,03 até o fim do treinamento. Esse comportamento indica que a rede crítica aprendeu a estimar com precisão os retornos esperados. Na Pista 1 (Figura 31), a convergência é mais lenta e instável, com flutuações até cerca de 900 mil passos. A perda estabiliza-se em torno de 0,05, sugerindo maior dificuldade da rede para prever retornos em um ambiente mais desafiador.

Figura 30 – Evolução da perda de valor com 40 agentes na Pista 0



Fonte: Autoria própria.

Figura 31 – Evolução da perda de valor com 40 agentes na Pista 1



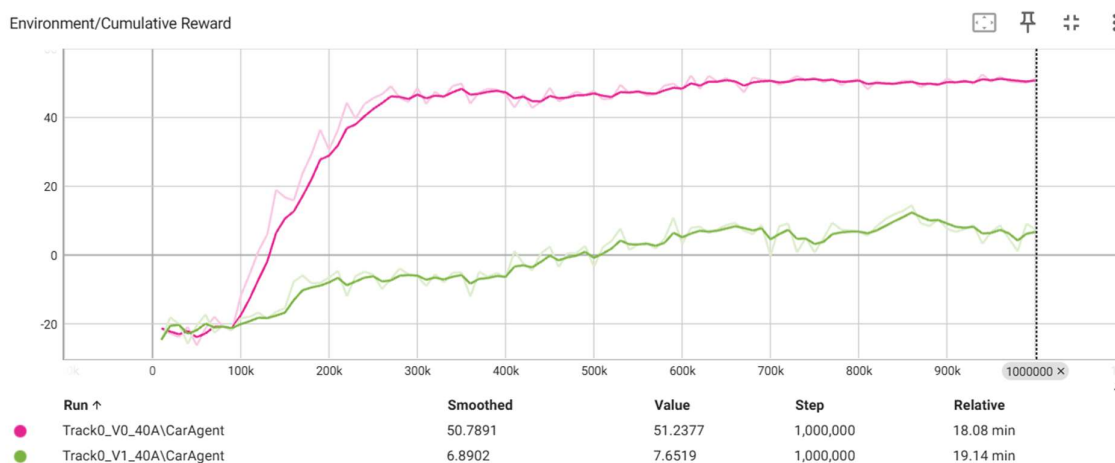
Fonte: Autoria própria.

Foram conduzidos dois experimentos adicionais com o objetivo de validar a robustez do processo de treinamento por meio da variação de hiperparâmetros críticos: o fator de desconto e a quantidade de agentes simultâneos.

O primeiro experimento consistiu na alteração do valor de gamma, o fator de desconto responsável por ponderar a importância de recompensas futuras. Para este teste, foram escolhidos dois valores extremos: $\gamma = 0,9$, utilizado nos experimentos anteriores, e $\gamma = 0,1$, representando um comportamento imediatista. A Figura 32 apresenta os resultados na Pista 0. Observa-se que, com $\gamma = 0,9$ (linha rosa), a rede foi capaz de aprender uma política eficiente, convergindo para altos valores de recompensa e completando o trajeto com sucesso. Já com $\gamma = 0,1$ (linha verde), o agente apresentou dificuldade em generalizar comportamentos eficazes,

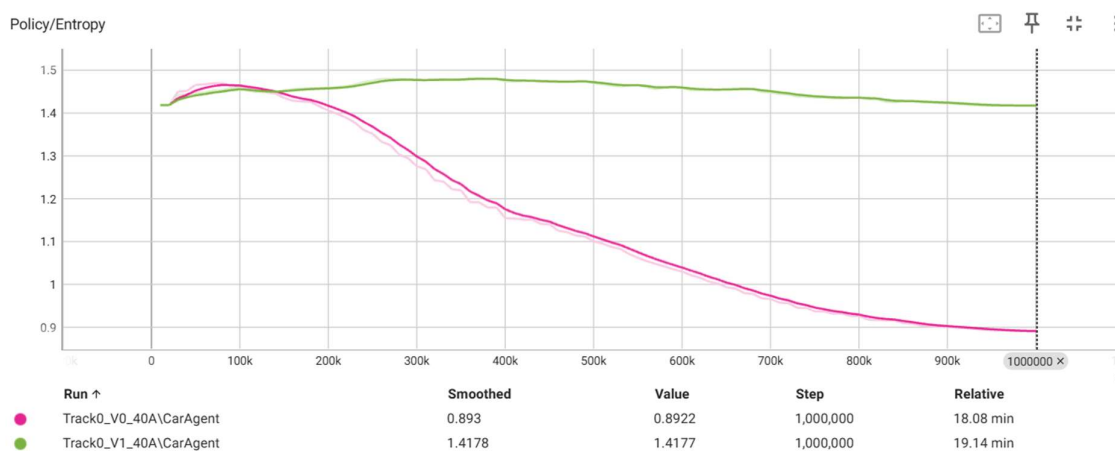
demonstrando baixa estabilidade e alta variância no aprendizado. A entropia da política (Figura 33) permaneceu praticamente constante com $\gamma = 0,1$, o que indica uma política pouco adaptativa e com pouca exploração, contrariando o comportamento esperado em treinos estáveis com PPO. Essa limitação também é refletida na distribuição de recompensas (Figura 34), com predominância de episódios de baixo desempenho.

Figura 32 – Comparação entre valores de gamma no teste com 40 agentes na Pista 0



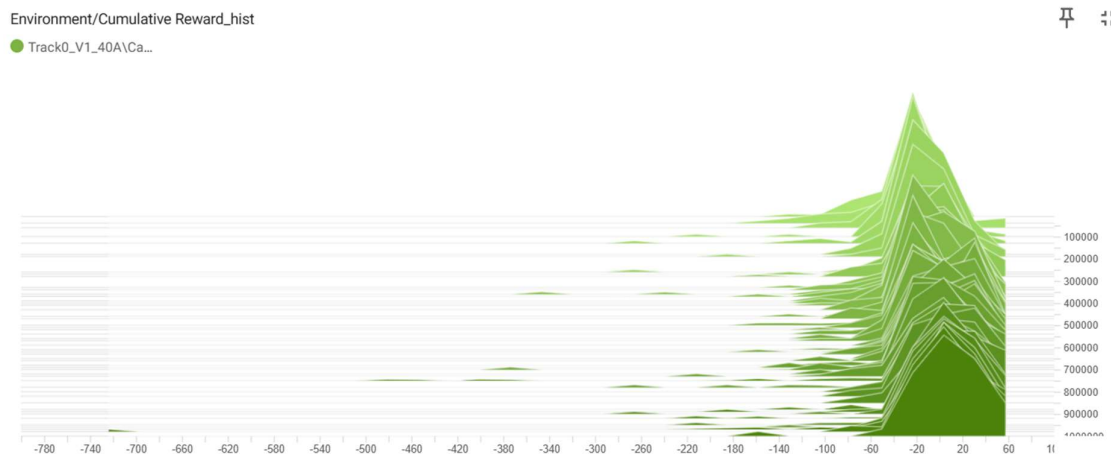
Fonte: Autoria própria.

Figura 33 – Gráfico de Entropia do teste com 40 agentes na Pista 0 na variação de gamma



Fonte: Autoria própria.

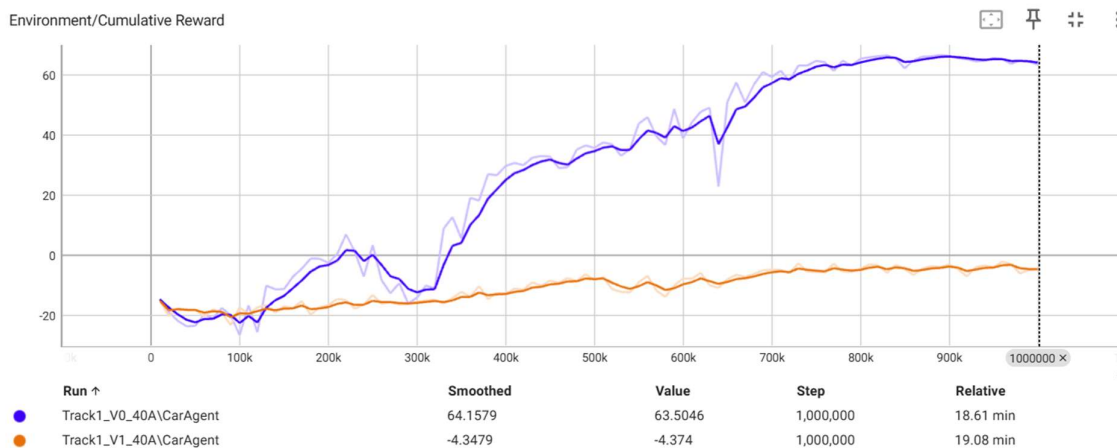
Figura 34 – Histograma de recompensa cumulativa do teste com 40 agentes na Pista 0 e baixo valor de gamma



Fonte: Autoria própria.

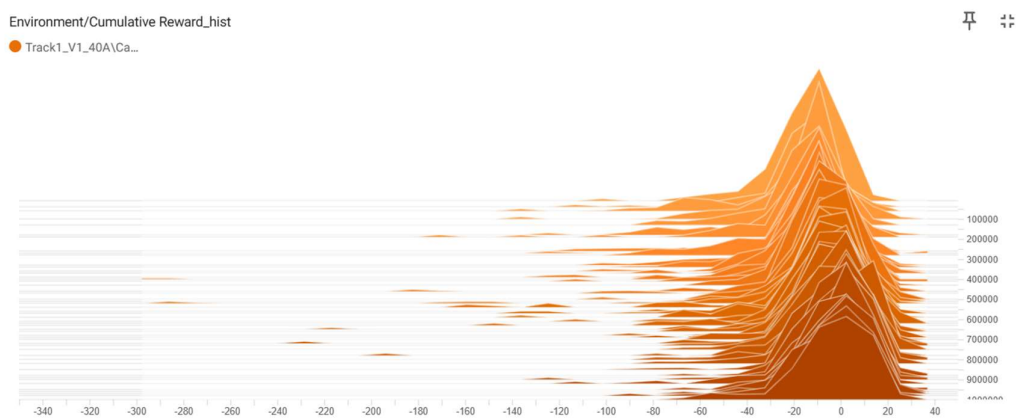
O mesmo experimento foi repetido na Pista 1, de maior complexidade, e os resultados (Figuras 35 a 37) mantiveram o padrão. Embora o ambiente seja mais desafiador, os agentes com $\gamma = 0,9$ (linha azul) demonstraram melhor desempenho, tanto em termos de convergência da recompensa quanto em comportamento de exploração. Por outro lado, os agentes com $\gamma = 0,1$ (linha laranja) mantiveram elevada variância, baixa entropia e baixa capacidade de finalização do trajeto. Isso reforça que, em ambientes de alta dimensionalidade e complexidade, é essencial que o agente atribua importância às recompensas futuras para alcançar bom desempenho.

Figura 35 – Comparação entre valores de gamma no teste com 40 agentes na Pista 1



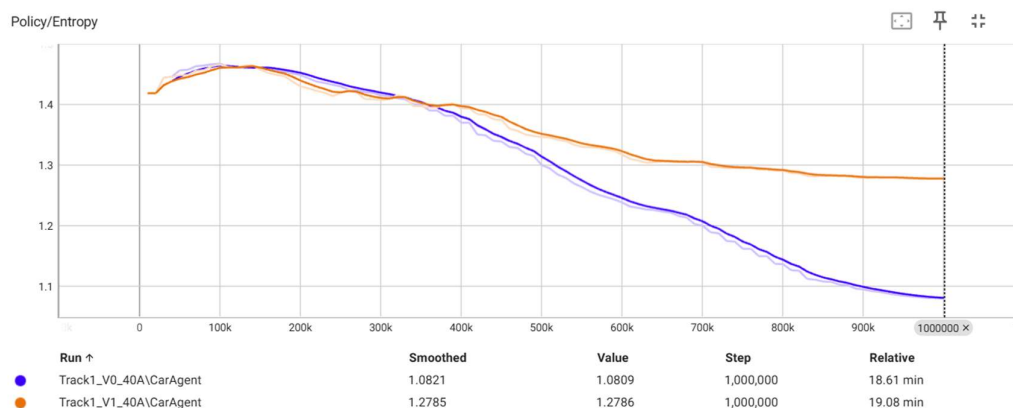
Fonte: Autoria própria.

Figura 36 – Histograma de recompensa cumulativa do teste com 40 agentes na Pista 1 e baixo valor de gamma



Fonte: Autoria própria.

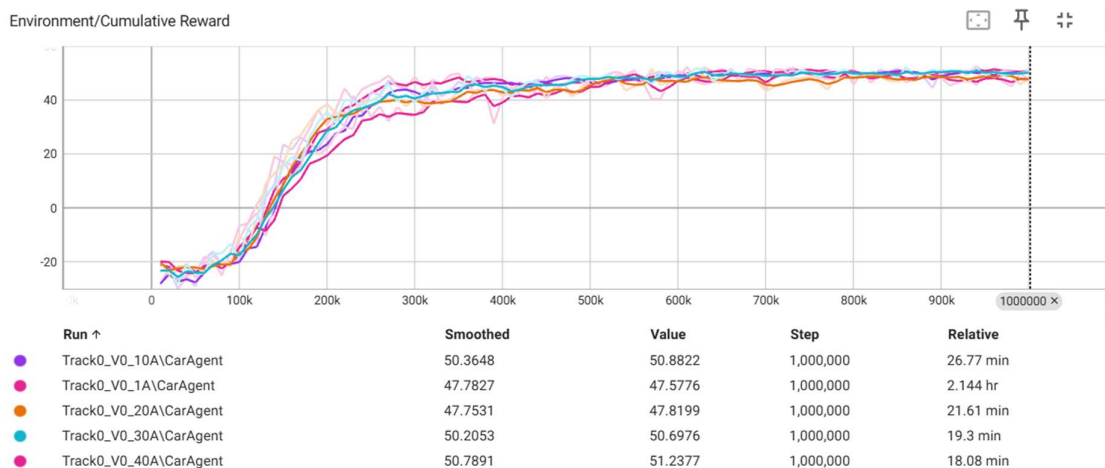
Figura 37 – Gráfico de Entropia do teste com 40 agentes na Pista 1 na variação de gamma



Fonte: Autoria própria

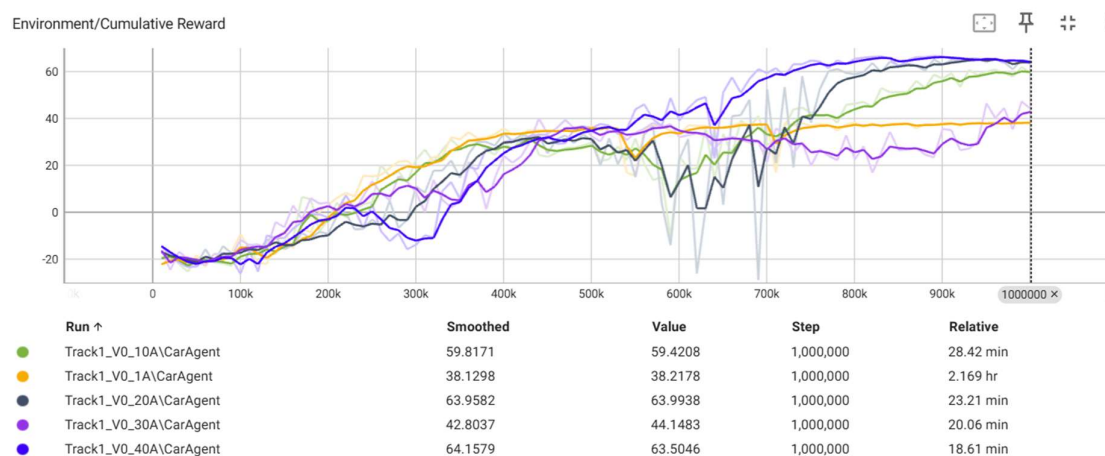
O segundo experimento avaliou o impacto da quantidade de agentes simultâneos no treinamento. Foram testadas cinco configurações: 1, 10, 20, 30 e 40 agentes. Os resultados para a Pista 0 estão apresentados na Figura 38 e para a Pista 1 na Figura 39. Na Pista 0, notou-se que o número de agentes influenciou pouco na qualidade do aprendizado, pois, as curvas de recompensa apresentaram formas e valores finais muito semelhantes. No entanto, a principal diferença observada foi no tempo total de treinamento. Com apenas 1 agente, o tempo necessário para completar 1 milhão de passos foi de aproximadamente 2 horas e 10 minutos. Com 10 agentes, esse tempo reduziu para cerca de 27 minutos. A redução continuou de forma significativa até atingir 18 minutos com 40 agentes, o número máximo testado.

Figura 38 – Gráfico de evolução da recompensa cumulativa na pista 0 – Teste de Quantidade de agentes



Fonte: Autoria própria.

Figura 39 – Gráfico de evolução da recompensa cumulativa na pista 1 – Teste de Quantidade de agentes



Fonte: Autoria própria.

Na Pista 1, o comportamento foi semelhante em termos de tempo, mas, devido à sua maior complexidade, a quantidade de agentes influenciou também na qualidade do aprendizado. Com apenas um agente, o desempenho foi insatisfatório, a rede não conseguiu aprender uma política capaz de completar o trajeto. Já com um maior número de agentes, a diversidade de experiências coletadas durante o treinamento promoveu melhor exploração dos possíveis estados e ações, o que resultou em políticas mais eficazes e estáveis. Esse resultado demonstra a importância do paralelismo no treinamento por reforço profundo, especialmente em ambientes complexos.

4.3 Discussão dos Resultados Apresentados

Através dos gráficos extraídos do TensorBoard, observou-se uma tendência de convergência da recompensa cumulativa, redução da perda da função de valor e estabilidade no tempo médio de cada episódio, especialmente nos cenários com menor complexidade (Pista 0), confirmando o aprendizado eficaz por parte dos agentes.

Comparado ao obtido por Pérez-Gil et al. (2022), que desenvolveu a aplicação no simulador CARLA, os resultados obtidos nesse trabalho apresentaram menor estabilidade e desempenho do que os modelos aplicados por eles. Apesar disso, os resultados obtidos apresentam tendências similares quanto à estabilidade e ao comportamento aprendido, sugerindo que a escolha do algoritmo de PPO foi apropriada à arquitetura aplicada de controle contínuo em ambiente de alta dimensionalidade.

Sivayazi e Mannayee (2024) empregaram uma variante *Double* DQN para navegação em ambientes com obstáculos dinâmicos. O enfoque do estudo foi na evasão ativa de obstáculos e uso de diferentes arquiteturas neurais. Apesar das diferenças no escopo e no algoritmo de controle, ambos os trabalhos evidenciam a importância do refinamento da função de recompensa e da arquitetura da rede para estabilidade no treinamento. Além disso, destaca-se que neste trabalho os obstáculos são estáticos, o que representa uma limitação em relação ao cenário dinâmico abordado por Sivayazi e Mannayee (2024).

Foi adotado uma abordagem simplificada no uso de sensores LiDAR simulados, priorizando a integração com o Unity ML-Agents para validação comportamental dos agentes. Diferente do aplicado por Elmquist et al. (2021), cujo trabalho focou no desenvolvimento de uma estrutura robusta de simulação de sensores, com alto realismo físico de sensores, incluindo modelagens de ruído e latência. A simplificação dos sensores abordada nesse trabalho, embora suficiente para os objetivos de validação do comportamento aprendido, representa uma limitação no que diz respeito à fidelidade física dos dados sensoriais, fator crítico para aplicações com foco em transferência simulação-realidade.

Apesar dessas limitações, os resultados indicam que a Virtual Test Bed desenvolvida apresenta potencial para escalabilidade e extensões futuras, como a integração de sensores com ruído realístico, simulação de obstáculos dinâmicos e transferência do modelo para testes em hardware ou ambientes mais fotorrealistas.

5. CONCLUSÃO

Os desafios contemporâneos relacionados à mobilidade urbana têm despertado crescente interesse tanto da literatura quanto das políticas públicas, que buscam soluções integradas e sustentáveis. Nesse contexto, as *Smart Cities* surgem como um modelo promissor de integração tecnológica voltado à resolução de problemas sociais, especialmente os relacionados à mobilidade. A mobilidade urbana, um dos pilares centrais das *Smart Cities*, tem evoluído significativamente com o avanço das tecnologias embarcadas nos meios de transporte, como os veículos autônomos e a integração com sistemas IoT. Aliado ao progresso computacional, o uso de tecnologias como *Digital Twin* para monitoramento de status e predição de falhas, e de *Virtual Test Beds* para testes em ambientes simulados, tem aproximado ainda mais o conceito de smart mobility da sua aplicação prática.

Neste trabalho, foi proposto o desenvolvimento de um *Virtual Test Bed* integrado a uma estrutura de *Digital Twin* para o treinamento de veículos autônomos em ambientes simulados, utilizando a engine Unity combinada com o algoritmo de Aprendizado por Reforço Profundo *Proximal Policy Optimization* (PPO). O objetivo foi criar uma ferramenta acessível e modular para validar arquiteturas de controle autônomo baseadas em redes neurais, especialmente em cenários urbanos de diferentes níveis de complexidade. O agente foi treinado com sucesso em duas pistas com obstáculos e checkpoints distintos, utilizando sensores simulados (LiDAR) e uma função de recompensa cuidadosamente estruturada para incentivar comportamentos de direção rápida e precisa.

Os resultados obtidos por meio das análises no TensorBoard demonstraram a evolução consistente do agente ao longo do treinamento. Métricas como recompensa acumulada, entropia da política, tempo de episódio, perda da política e perda de valor evidenciaram um processo de aprendizado estável e eficaz. Testes complementares com variação do fator de desconto (gamma) e da quantidade de agentes ativos reforçaram a robustez da abordagem, destacando a importância de considerar recompensas futuras e de utilizar múltiplos agentes para promover maior diversidade de experiências durante o treinamento.

No cerne da proposta, este trabalho evidencia o potencial dos *Virtual Test Beds* como alternativa viável e segura para o desenvolvimento de tecnologias embarcadas em veículos autônomos. O uso de DRL em ambientes simulados não apenas acelera o processo de aprendizado sem riscos físicos, como também permite uma análise mais detalhada das variáveis de desempenho, especialmente quando integrado a uma estrutura de *Digital Twin*. Essa

abordagem contribui para a redução de custos operacionais, aumento da segurança e avanço de soluções voltadas à mobilidade inteligente.

Como trabalhos futuros, propõe-se a expansão do ambiente virtual com pistas maiores e a inclusão de obstáculos móveis, visando aumentar a complexidade do cenário. Também se sugere a implementação de novos algoritmos de aprendizado, como *Soft Actor-Critic* (SAC), além da aplicação de técnicas avançadas como *self-play* e *Adversarial Reinforcement Learning*, com o intuito de aprimorar ainda mais o desempenho dos agentes treinados.

REFERÊNCIAS

- 3DSHOWROOM. *Lexus GS 350 2011*. Sketchfab, 2017. Disponível em: <https://sketchfab.com/3d-models/lexus-gs-350-2011-a89d18432d5047549c4e8b9a66cb2d7d>. Acesso em: 5 abr. 2025.
- ALVA, Pradeep et al. **Mitigating operational greenhouse gas emissions in ageing residential buildings using an Urban Digital Twin dashboard**. *Energy and Buildings*, v. 322, p. 114681, 2024. Disponível em: <https://doi.org/10.1016/j.enbuild.2024.114681>. Acesso em: 7 abr. 2025.
- BHATTI, G.; MOHAN, H.; SINGH, R. R. **Towards the future of smart electric vehicles: Digital twin technology**. *Renewable and Sustainable Energy Reviews*, v. 141, p. 110801, 2021. Disponível em: <https://doi.org/10.1016/j.rser.2021.110801>. Acesso em: 26 mar. 2025.
- CHEN, Jian et al. **Traffic flow matrix-based graph neural network with attention mechanism for traffic flow prediction**. *Information Fusion*, v. 104, p. 102146, 2024. Disponível em: <https://doi.org/10.1016/j.inffus.2023.102146>. Acesso em: 4 abr. 2025.
- ELMQUIST, R.; PETERS, A.; XU, H.; LEVINE, S.; KOŠECKÁ, J. **A sensor simulation framework for training and testing robots and autonomous vehicles**. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION – ICRA, 2021, Xi'an, China. Piscataway, NJ: IEEE, 2021. p. 2247–2253. Disponível em: <https://doi.org/10.1109/ICRA48506.2021.9562056>. Acesso em: 3 abr. 2025.
- EPIC GAMES. *Unreal Engine*. Disponível em: <https://www.unrealengine.com/>. Acesso em: 5 abr. 2025.
- FAISAL, A.; YIGITCANLAR, T.; KAMRUZZAMAN, M.; CURRIE, G. **Understanding autonomous vehicles: a systematic literature review on capability, impact, planning and policy**. *Journal of Transport and Land Use*, v. 12, n. 1, p. 45–72, 2019. Disponível em: <https://doi.org/10.5198/jtlu.2019.1405>. Acesso em: 26 mar. 2025.
- GÖKDERE, L. U.; BENLYAZID, K.; DOUGAL, R. A.; SANTI, E.; BRICE, C. W. **A virtual prototype for a hybrid electric vehicle**. *Mechatronics*, v. 12, n. 4, p. 575–593, 2002. Disponível em: [https://doi.org/10.1016/S0957-4158\(01\)00009-5](https://doi.org/10.1016/S0957-4158(01)00009-5). Acesso em: 26 mar. 2025.
- ISMAGILOVA, E.; HUGHES, L.; DWIVEDI, Y. K.; RAMAN, K. R. **Smart cities: advances in research – an information systems perspective**. *International Journal of Information Management*, v. 47, p. 88–100, 2019. Disponível em: <https://doi.org/10.1016/j.ijinfomgt.2019.01.004>. Acesso em: 26 mar. 2025.
- KNOX, W. B. et al. **Reward (mis)design for autonomous driving**. *Artificial Intelligence*, v. 316, p. 103829, 2023. Disponível em: <https://doi.org/10.1016/j.artint.2022.103829>. Acesso em: 5 abr. 2025.
- LI, Jingjun; ROMBAUT, Evy; VANHAVERBEKE, Lieselot. **How far are we towards sustainable carfree cities combining shared autonomous vehicles with park-and-ride: an agent-based simulation assessment for Brussels**. *Computers, Environment and Urban*

Systems, v. 112, p. 102148, 2024. Disponível em: <https://doi.org/10.1016/j.compenvurbsys.2024.102148>. Acesso em: 4 abr. 2025.

LIAO, X. et al. **A game theory based ramp merging strategy for connected and automated vehicles in the mixed traffic: a Unity-SUMO integrated platform.** *arXiv*, arXiv:2101.11237, 2021. Disponível em: <https://arxiv.org/abs/2101.11237>. Acesso em: 5 abr. 2025.

LIU, Z. et al. **Digital twin modeling methodology based on the consistency of behavior and structure.** *Simulation Modelling Practice and Theory*, v. 112, p. 102086, 2021. Disponível em: <https://doi.org/10.1016/j.jmsy.2022.06.015>. Acesso em: 5 abr. 2025.

MATHWORKS. **What is reinforcement learning?** Disponível em: <https://www.mathworks.com/help/reinforcement-learning/ug/what-is-reinforcement-learning.html>. Acesso em: 3 abr. 2025.

MCALLISTER, R.; GAL, Y.; KENDALL, A.; VAN DER WILK, M.; SHAH, A.; CIPOLLA, R.; WELLER, A. **Concrete problems for autonomous vehicle safety: advantages of Bayesian deep learning.** In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE (IJCAI-17), 26., 2017, Melbourne. Melbourne: IJCAI, 2017. p. 1–9. Disponível em: <https://doi.org/10.24963/ijcai.2017/661>. Acesso em: 26 mar. 2025.

MNIH, V. et al. **Human-level control through deep reinforcement learning.** *Nature*, v. 518, p. 529–533, 2015. Disponível em: <https://doi.org/10.1038/nature14236>. Acesso em: 3 abr. 2025.

ODENDAAL, Nancy. **Information and communication technology and local governance: understanding the difference between cities in developed and emerging economies.** *Computers, Environment and Urban Systems*, v. 27, n. 6, p. 585-607, nov. 2003. Disponível em: [https://doi.org/10.1016/s0198-9715\(03\)00016-4](https://doi.org/10.1016/s0198-9715(03)00016-4). Acesso em: 27 mar. 2025.

PAREKH, D.; PODDAR, N.; RAJPURKAR, A.; CHAHAL, M.; KUMAR, N.; JOSHI, G. P.; CHO, W. **A review on autonomous vehicles: progress, methods and challenges.** *Electronics*, v. 11, n. 14, p. 2162, 2022. Disponível em: <https://doi.org/10.3390/electronics11142162>. Acesso em: 26 mar. 2025.

PÉREZ-GIL, Ó. et al. **Deep reinforcement learning based control for autonomous vehicles in CARLA.** *Multimedia Tools and Applications*, v. 81, p. 3553–3576, 2022. Disponível em: <https://doi.org/10.1007/s11042-021-11437-3>. Acesso em: 3 abr. 2025.

QUARKS STUDIOS. **Low poly race track.** Sketchfab, 2020. Disponível em: <https://sketchfab.com/3d-models/low-poly-race-track-b40628339fde4b2fbe41711edc7c7a93>. Acesso em: 5 abr. 2025.

RAZERA, Natan. **Virtual Test Bed for Autonomous Vehicle Training in Unity.** [S.l.]: YouTube, 2025. Disponível em: https://youtu.be/CbJABsToDt8?si=KrLd-MscVf_G-INc. Acesso em: 27 abr. 2025.

RIZZON, F.; BERTELLI, J.; MATTE, J.; GRAEBIN, R. E.; MACKE, J. **Smart City: um conceito em construção.** *Revista Metropolitana de Sustentabilidade*, São Paulo, v. 7, n. 3, p. 123–142, 2017. Disponível em: <https://revistaseletronicas.fmu.br/index.php/rms/article/view/1378>. Acesso em: 26 mar. 2025.

SCHULMAN, John et al. **Proximal policy optimization algorithms**. *arXiv*, arXiv:1707.06347, 2017. Disponível em: <https://arxiv.org/abs/1707.06347>. Acesso em: 5 abr. 2025.

SHAKYA, Ashish Kumar; PILLAI, Gopinatha; CHAKRABARTY, Sohom. **Reinforcement learning algorithms: A brief survey**. *Expert Systems with Applications*, v. 231, p. 120495, 2023. Disponível em: <https://doi.org/10.1016/j.eswa.2023.120495>. Acesso em: 5 abr. 2025.

SHI, Xiaowei; LI, Xiaopeng. **Constructing a fundamental diagram for traffic flow with automated vehicles: methodology and demonstration**. *Transportation Research Part B: Methodological*, v. 150, p. 279–292, 2021. Disponível em: <https://doi.org/10.1016/j.trb.2021.06.011>. Acesso em: 4 abr. 2025.

SHLADOVER, S. E. **Connected and automated vehicle systems: introduction and overview**. *Journal of Intelligent Transportation Systems*, v. 22, n. 3, p. 190–200, 2018. Disponível em: <https://doi.org/10.1080/15472450.2017.1336053>. Acesso em: 26 mar. 2025.

SINGH, M. et al. **Digital twin: origin to future**. *Applied System Innovation*, v. 4, n. 1, p. 36, 2021. Disponível em: <https://doi.org/10.3390/asi4010036>. Acesso em: 5 abr. 2025.

SIVAYAZI, K.; MANNAYEE, G. **Modeling and simulation of a double DQN algorithm for dynamic obstacle avoidance in autonomous vehicle navigation**. *e-Prime – Advances in Electrical Engineering, Electronics and Energy*, v. 8, 2024, 100581. Disponível em: <https://doi.org/10.1016/j.prime.2024.100581>. Acesso em: 3 abr. 2025.

SKETCHFAB. **Modelos 3D gratuitos e pagos para download**. Disponível em: <https://sketchfab.com/>. Acesso em: 5 abr. 2025.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: an introduction**. Draft 2nd ed. [S.l.], 2015. Disponível em: <https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf>. Acesso em: 3 abr. 2025.

TANG, Wei et al. **Assessment of future parking systems with autonomous vehicles through agent-based simulation: a case study of Hangzhou, China**. *Sustainable Cities and Society*, v. 100, p. 105016, 2024. Disponível em: <https://doi.org/10.1016/j.scs.2023.105016>. Acesso em: 4 abr. 2025.

TESAURO, G. **Temporal difference learning and TD-Gammon**. *Communications of the ACM*, v. 38, n. 3, p. 58–68, 1995. Disponível em: <https://doi.org/10.1145/203330.203343>. Acesso em: 3 abr. 2025.

UNITY TECHNOLOGIES. **Unity Real-Time Development Platform**. Disponível em: <https://unity.com/>. Acesso em: 5 abr. 2025.

WANG, Y. et al. **Digital Twin-Assisted Cooperative Driving at Non-Signalized Intersections Using Parallel Deep Reinforcement Learning**. *IEEE Transactions on Intelligent Transportation Systems*, v. 23, n. 6, p. 5567–5581, 2022. Disponível em: <https://doi.org/10.1109/TITS.2021.3103315>. Acesso em: 5 abr. 2025.

YURTSEVER, E.; LAMBERT, J.; CARBALLO, A.; TAKEDA, K. **A survey of autonomous driving: common practices and emerging technologies**. *IEEE Access*, v. 8, p. 58443–58469, 2020. Disponível em: <https://doi.org/10.1109/ACCESS.2020.2983149>. Acesso em: 26 mar. 2025.

ZHAO, X. et al. **Development of a cyber-physical-system perspective based simulation platform for optimizing connected automated vehicles dedicated lanes**. *Expert Systems with Applications*, v. 213, p. 118972, 2023. Disponível em: <https://doi.org/10.1016/j.eswa.2022.118972>. Acesso em: 5 abr. 2025.

GLOSSÁRIO

AGENTE: Entidade que executa ações em um ambiente com o objetivo de maximizar recompensas ao longo do tempo.

AMBIENTE: Sistema com o qual o agente interage, recebendo observações e recompensas a partir das ações realizadas.

APRENDIZADO POR REFORÇO: Área do aprendizado de máquina onde um agente aprende por meio de tentativa e erro, maximizando recompensas acumuladas.

APRENDIZADO POR REFORÇO PROFUNDO: Combinação de redes neurais profundas com algoritmos de aprendizado por reforço para lidar com ambientes complexos e de alta dimensão.

AÇÃO (ACTION): Decisão tomada pelo agente com base no estado atual do ambiente.

ENTROPIA DA POLÍTICA: Métrica que mede o grau de aleatoriedade das decisões do agente.

EPISÓDIO: Sequência de interações entre o agente e o ambiente com um estado inicial e um estado terminal.

ESTADO (STATE): Representação da situação atual do ambiente.

EXPLORAÇÃO (EXPLOITATION): Uso das melhores ações conhecidas para maximizar a recompensa com base no conhecimento atual.

EXPLORAÇÃO (EXPLORATION): Tentativa de experimentar novas ações para adquirir mais informações sobre o ambiente.

FATOR DE DESCONTO (γ / GAMMA): Valor entre 0 e 1 que define a importância das recompensas futuras em relação às imediatas.

FUNÇÃO DE VANTAGEM (ADVANTAGE FUNCTION): Mede o benefício adicional de executar uma ação específica em relação à média das ações naquele estado.

FUNÇÃO Q (Q-VALUE): Estimativa da recompensa esperada ao tomar uma ação em um estado específico e seguir a política depois.

FUNÇÃO VALOR (V): Estimativa do valor esperado de um estado, assumindo que o agente siga uma determinada política.

GENERALIZAÇÃO: Capacidade do agente de manter bom desempenho em estados ou ambientes não vistos durante o treinamento.

LSTM (LONG SHORT-TERM MEMORY): Tipo de rede neural recorrente usada para lidar com dependências temporais em sequências.

OBSERVAÇÃO (OBSERVATION): Informação que o agente recebe do ambiente para tomar decisões.

PASSO DE TEMPO (TIMESTEP): Unidade básica de tempo em Reinforcement Learning, representando uma iteração de ação e resposta do ambiente.

POLÍTICA (π / POLICY): Estratégia utilizada pelo agente para selecionar ações com base nos estados observados.

PPO (PROXIMAL POLICY OPTIMIZATION): Algoritmo de otimização de política robusto.

Q-LEARNING: Algoritmo baseado em valores que aprende a função Q diretamente através de atualizações iterativas.

RECOMPENSA (REWARD): Sinal numérico fornecido pelo ambiente após uma ação, indicando o desempenho do agente.

RECOMPENSA ACUMULADA (CUMULATIVE REWARD): Soma de todas as recompensas recebidas ao longo de um episódio.

REDE CRÍTICA (CRITIC): Rede Neural que estima o valor de um estado ou par estado-ação (função valor ou função Q).

REDES NEURAIS PROFUNDAS: Modelos com múltiplas camadas de neurônios capazes de extrair representações complexas dos dados.

REWARD SHAPING: Técnica para modificar a função de recompensa visando acelerar o aprendizado do agente.

TAXA DE APRENDIZADO (α / ALPHA): Parâmetro que controla a velocidade com que o agente atualiza seu conhecimento com base em novas experiências.

TEMPO DE EPISÓDIO: Duração (em etapas) de um episódio completo de interação agente-ambiente.