

UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL PICCOLOMINI DE LIMA

**USO DE INTELIGÊNCIA ARTIFICIAL EM JOGOS DIGITAIS
TRIDIMENSIONAIS: APLICAÇÃO DE ALGORITMOS DE
MACHINE LEARNING PARA MOVIMENTAÇÃO DE AGENTES**

Bauru, São Paulo, Brasil
2025

RAFAEL PICCOLOMINI DE LIMA

**USO DE INTELIGÊNCIA ARTIFICIAL EM JOGOS DIGITAIS
TRIDIMENSIONAIS: APLICAÇÃO DE ALGORITMOS DE
MACHINE LEARNING PARA MOVIMENTAÇÃO DE AGENTES**

Trabalho de Conclusão de Curso do Curso
de Bacharelado em Ciência da Computação
da Universidade Estadual Paulista “Júlio
de Mesquita Filho”, Faculdade de Ciências,
Campus de Bauru.

Orientadora: Profa. Dra. Juliana da Costa
Feitosa

Bauru, São Paulo, Brasil

2025

L732u

Lima, Rafael Piccolomini

Uso de inteligência artificial em jogos digitais tridimensionais :
Aplicação de algoritmos de machine learning para movimentação de
agentes / Rafael Piccolomini Lima. -- Bauru, 2025

31 p. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da
Computação) - Universidade Estadual Paulista (UNESP), Faculdade
de Ciências, Bauru

Orientadora: Juliana da Costa Feitosa

1. Inteligência artificial. 2. Jogos eletrônicos. 3. Machine learning.
I. Título.

Resumo

O uso de Inteligência Artificial em ambientes virtuais tem ampliado as possibilidades de criação de agentes autônomos, mas ainda apresenta desafios quanto à definição de estratégias de treinamento que garantam aprendizado consistente e generalizável em cenários tridimensionais. Este trabalho investiga o uso de técnicas de Inteligência Artificial (IA) e Aprendizado de Máquina (*Machine Learning*) para treinar um agente em ambiente 3D no Unity, construído como uma pista com obstáculos e sensores de percepção. Duas abordagens foram implementadas: aprendizado por reforço (PPO via ML-Agents) e algoritmo genético controlando ações de locomoção. A avaliação considerou indicadores como taxa de sucesso, *waypoints alcançados*, distância ao objetivo, colisões e passos por episódio. Observou-se que ajustes finos em recompensas, hiperparâmetros e sensores impactam fortemente a estabilidade do aprendizado e a capacidade de generalização. Como contribuição, o trabalho apresenta um panorama prático das escolhas de projeto que mais influenciam o treinamento de agentes com aprendizado por reforço e algoritmos genéticos em cenários 3D, oferecendo diretrizes úteis para configurações mais estáveis e comportamentos mais consistentes no desenvolvimento de jogos 3D utilizando a engine Unity.

Palavras-chave: *machine learning*; inteligência artificial; desenvolvimento de jogos; jogos digitais

Abstract

The use of Artificial Intelligence in virtual environments has expanded the possibilities for creating autonomous agents but still poses challenges regarding the definition of training strategies that ensure consistent and generalizable learning in three-dimensional scenarios. This work investigates the use of Artificial Intelligence (AI) and Machine Learning (ML) techniques to train an agent in a 3D Unity environment, designed as a track with obstacles and perception sensors. Two approaches were implemented: reinforcement learning (PPO via ML-Agents) and a genetic algorithm controlling locomotion actions. The evaluation considered indicators such as success rate, waypoints reached, distance to the goal, collisions, and steps per episode. It was observed that fine adjustments in rewards, hyperparameters, and sensors strongly impact the stability of learning and the ability to generalize. As a contribution, this work presents a practical overview of the design choices that most influence the training of agents with reinforcement learning and genetic algorithms in 3D scenarios, providing useful guidelines for achieving more stable configurations and more consistent behaviors in the development of 3D games using the Unity engine.

Keywords: machine learning; artificial intelligence; game development; digital games

Lista de figuras

Figura 1 – Estrutura que compunha o mapa base	18
Figura 2 – Gráfico referente à primeira etapa de treinamento	19
Figura 3 – Gráfico referente ao treinamento após a primeira etapa de ajustes	21
Figura 4 – Gráfico referente ao treinamento com aprendizado por currículo	22
Figura 5 – Estrutura do mapa de treinamento do algoritmo genético	24
Figura 6 – Gráfico referente à convergência entre as gerações 40 e 50	25
Figura 7 – Imagem do circuito com a adição dos <i>waypoints</i>	25
Figura 8 – Imagem do agente que completou 5 <i>waypoints</i>	26

Lista de tabelas

Tabela 1 – Tempo necessário para cada configuração atingir aproximadamente 500 000 <i>steps</i>	20
Tabela 2 – Comparação dos treinamentos considerando estabilidade, desvio padrão e tempo para convergência	23

Lista de abreviaturas e siglas

ML	<i>Machine Learning</i>
IA	<i>Inteligência Artificial</i>
RL	<i>Reinforcement Learning</i>
GA	<i>Genetic Algorithm</i>
PPO	<i>Proximal Policy Optimization</i>

Sumário

1	INTRODUÇÃO	9
1.1	Problema	10
1.2	Justificativa	10
1.3	Objetivos	10
1.3.1	Objetivo Geral	10
1.3.2	Objetivos Específicos	11
2	FUNDAMENTAÇÃO TEÓRICA	12
2.1	Jogos Digitais e Ambientes Tridimensionais	12
2.2	Machine Learning em Jogos Digitais	13
3	MATERIAIS E MÉTODOS	14
3.1	Ambiente e Ferramentas	14
3.2	Configuração do Cenário e Elementos do Jogo	14
3.3	Configuração do Aprendizado por Reforço	15
3.4	Configuração do Algoritmo Genético	15
3.4.1	Rede Neural	16
3.4.2	Avaliação	16
3.4.3	Jogo (laço de interação com o ambiente)	16
3.4.4	Recombinação	17
3.4.5	Mutação	17
3.4.6	Orquestrador	17
4	TREINAMENTO DOS AGENTES	18
4.1	Treinamento utilizando Aprendizado por Reforço	18
4.1.1	Processo de treinamento: dificuldades e ajustes	19
4.1.2	Conclusão Parcial	22
4.2	Treinamento utilizando Algoritmo Genético	23
4.2.1	Processo de treinamento: dificuldades e ajustes	23
4.2.2	Conclusão Parcial	26
5	CONCLUSÃO	28
	REFERÊNCIAS	30

1 Introdução

O uso de Inteligência Artificial (IA) e *Machine Learning* (ML) em jogos digitais tem se tornado um campo de grande relevância na Computação, especialmente pela capacidade dessas técnicas em permitir que agentes aprendam comportamentos de forma autônoma (MNIH et al., 2015). Com o avanço das GPUs e dos motores gráficos, como o Unity, tornou-se possível aplicar algoritmos de aprendizado por reforço (*Reinforcement Learning*), redes neurais e métodos evolutivos diretamente em ambientes tridimensionais, ampliando o realismo e a complexidade das simulações (JULIANI et al., 2018).

Contudo, a aplicação prática desses métodos ainda enfrenta grandes desafios. Pequenas alterações nos parâmetros de treinamento, como taxas de aprendizado ou recompensas, podem gerar comportamentos imprevisíveis e instabilidades (HENDERSON et al., 2019). Assim, o processo de treinar um agente torna-se não apenas técnico, mas também experimental, exigindo ajustes constantes e interpretação cuidadosa dos resultados (HENDERSON et al., 2019).

A criação de agentes com comportamentos realistas é fundamental para a imersão dos jogos, mas o desenvolvimento desses agentes requer experimentação extensiva. Durante este trabalho, foram observadas dificuldades recorrentes, como falhas de comunicação entre Unity e Python e a convergência prematura de modelos genéticos. Essas dificuldades foram o foco da pesquisa, bem como compreender as limitações e desafios do treinamento de agentes autônomos (ZHANG; BALLAS; PINEAU, 2018).

O problema de pesquisa, portanto, consiste em investigar como desenvolver agentes inteligentes capazes de aprender locomoção e desvio de obstáculos de forma autônoma, considerando a instabilidade e sensibilidade dos algoritmos de aprendizado em ambientes tridimensionais. O objetivo geral é analisar e documentar as dificuldades do processo de treinamento, identificando os fatores que influenciam o aprendizado e a estabilidade dos agentes.

Para isso, este trabalho adotou uma metodologia experimental e exploratória. Primeiramente, foi feita uma revisão bibliográfica sobre IA aplicada a jogos e técnicas de aprendizado. Em seguida, desenvolveu-se um ambiente tridimensional no Unity, no qual agentes foram treinados com duas abordagens: algoritmo genético e aprendizado por reforço (STANLEY; MIIKKULAINEN, 2002). O foco da análise concentrou-se na observação dos comportamentos emergentes e das falhas recorrentes durante o processo.

1.1 Problema

Um dos principais fatores que ditam a percepção de qualidade em um jogo é o quão crível esse jogo se faz parecer frente ao mundo real, gerando assim uma imersão maior aos jogadores (AZOUBEL et al., 2016).

No entanto, a complexidade de desenvolver agentes que se movam de maneira fluida, adaptável e reativa ao contexto do jogo continua sendo uma barreira técnica significativa, se intensificando em jogos em que essa característica se faz necessária, como por exemplo, em jogos de mundo aberto e simulação (ONTANON et al., 2015).

Este trabalho busca investigar o processo de treinamento de modelos de ML, como RL e redes neurais e analisar os principais obstáculos e dificuldades presentes no refinamento do treinamento de cada agente, visando a otimização da locomoção e velocidade de reação de agentes a estímulos externos e obstáculos (MÜLLER; BITTENCOURT, 2021).

1.2 Justificativa

A capacidade de criar agentes com comportamentos complexos e adaptáveis é um desafio técnico significativo, mas essencial para o desenvolvimento de jogos que proporcionem uma experiência verdadeiramente envolvente (TUPE et al., 2016). Ao investigar e comparar o processo de treinamento de agentes em ambientes 3D, e identificar as sensibilidades e dificuldades de levantar os melhores parâmetros, recompensas e regras, a fim de que o treinamento ocorra de forma otimizada, este trabalho busca contribuir para o avanço da IA em jogos e para a melhoria da experiência do jogador (UPDYKE et al., 2021). Além disso, as técnicas desenvolvidas neste estudo podem ter aplicações em outras áreas, como simulação, robótica e treinamento virtual, demonstrando o potencial de impacto desta pesquisa além do contexto dos jogos digitais (HALEEM, 2023).

1.3 Objetivos

1.3.1 Objetivo Geral

Realizar o treinamento de agentes controlados por modelos de ML utilizando a ferramenta Unity em um ambiente 3D, identificar as principais dificuldades e limitações em executar o treinamento de cada técnica avaliada e documentar a evolução de cada modelo, enfatizando como as regras de parametrização e de recompensa impactam o desempenho do treinamento como um todo.

1.3.2 Objetivos Específicos

Os objetivos específicos para o presente trabalho são apresentados a seguir:

- Desenvolver os ambientes de treinamento;
- Desenvolver o algoritmo de avaliação do agente para o modelo de aprendizado por reforço;
- Desenvolver as regras e a jogabilidade aplicadas ao jogo, otimizadas para o modelo de aprendizado por reforço;
- Executar o treinamento utilizando o modelo de aprendizado por reforço e documentar os resultados;
- Desenvolver e estruturar a rede neural, o processo de evolução e a avaliação para o algoritmo genético;
- Desenvolver o algoritmo de movimentação e recompensa do agente para o algoritmo genético;
- Desenvolver as regras e a jogabilidade aplicadas ao jogo, otimizadas para o algoritmo genético;
- Executar o treinamento utilizando o algoritmo genético e documentar os resultados; e
- Efetuar a análise da evolução de cada treinamento, catalogar os principais obstáculos e dificuldades enfrentadas em cada caso.

2 Fundamentação Teórica

Neste capítulo, apresentam-se os fundamentos teóricos que sustentam o desenvolvimento do ambiente 3D em Unity e o treinamento dos agentes deste trabalho. São revisados conceitos de IA e ML aplicados a jogos digitais com destaque para aprendizado por reforço e algoritmos genéticos, bem como noções de observações e ações do agente e recursos do Unity ML-Agents.

2.1 Jogos Digitais e Ambientes Tridimensionais

Os jogos digitais tridimensionais vem cada vez mais se sobressaindo não apenas para entretenimento, sendo utilizados também para educação, pesquisa e simulação (CONNOLLY et al., 2012). A utilização da visualização em três dimensões promove uma maior fidelização à realidade, abrindo um leque muito maior de possibilidades de aplicação dos jogos. Porém ao passo que a simulação se torna mais real, também aumenta a complexidade do ambiente em que o jogo é desenvolvido, adicionando mais possibilidades de navegação (movimento em eixos adicionais, colisões, perspectiva, física), e fazendo com que aumente a dificuldade de adaptação de agentes automatizados.

Ambientes 3D impõem exigências adicionais aos agentes, tais como modelos de interação física, cálculos de colisão, obstáculos volumétricos e dinâmicas espaciais que não existem em 2D. Isso requer que os algoritmos manejem estados de maior dimensão, sensoriamento mais complexo e capacidade de generalização para diferentes alinhamentos espaciais.

Devido a esse grau maior de complexidade, surgiu o interesse e a procura pela utilização de algoritmos de ML no treinamento desses agentes automatizados, pois devido a sua capacidade de lidar com grandes volumes de dados e observações, se tornam uma opção forte para resolver o dilema de como lidar com a complexidade dos ambientes 3D (MNIH et al., 2015).

Além disso, a engine Unity se estabeleceu como uma plataforma robusta e amplamente adotada para desenvolvimento de jogos 3D, com recursos gráficos, de física e extensibilidade por meio de scripts. Muitos projetos acadêmicos e industriais escolhem o Unity por sua flexibilidade e por permitir integração com sistemas de IA externos. A introdução do Unity ML-Agents Toolkit transformou o Unity não apenas em uma plataforma de visualização, mas também em ambiente de simulação para treinar agentes inteligentes (JULIANI et al., 2018).

2.2 Machine Learning em Jogos Digitais

A aplicação de ML em jogos decorre da possibilidade de criar agentes capazes de aprender com a experiência e adaptar comportamentos, sem depender unicamente de scripts fixos ou regras codificadas manualmente. Em jogos digitais, isso permite que agentes evoluam com base no ambiente e no *feedback*, ajustem suas estratégias e respondam de modo mais fluido e imprevisível ao jogador.

Dentre as técnicas utilizadas para tal treinamento destacam-se o aprendizado por reforço, um modelo clássico de treinamento que foca em uma melhoria contínua do desempenho enquanto o agente joga o jogo em tempo real, ajustando sua política a partir de recompensas e penalidades (equilibrando exploração e resultado, conforme o desenho da função de recompensa). Outro método frequentemente utilizado é o aprendizado por algoritmo genético, cujo agente é submetido a uma evolução geracional que aprimora o desempenho ao colocá-lo em grandes populações, avaliando cada indivíduo por uma métrica de *fitness*, que representa o quão bem o agente cumpre o objetivo da tarefa, geralmente combinando fatores como distância ao objetivo, *waypoints* alcançados, colisões e eficiência nos passos. A partir dessa avaliação, selecionam-se os melhores de cada geração e aplicam-se recombinações (*crossover*) e pequenas mutações, sendo assim capaz de multiplicá-los e selecioná-los novamente, garantindo que o indivíduo final reúna as características necessárias para obter o melhor desempenho no jogo (STANLEY; MIIKKULAINEN, 2002).

Para este trabalho utilizou-se a variante PPO (*Proximal Policy Optimization*) como algoritmo de aprendizado por reforço, caracterizado por atualizar a política de forma iterativa a partir das trajetórias coletadas pelo agente durante os episódios. O método opera comparando a nova política com a política anterior por meio de uma razão de probabilidades, controlando a magnitude dessas atualizações para evitar mudanças abruptas no comportamento aprendido. Esse processo é realizado a partir de um gradiente otimizado com um termo de *clipping* como mostrado na equação 2.1, responsável por restringir atualizações que ultrapassem um limite definido e garantindo que a política seja ajustada apenas dentro de uma faixa segura (SCHULMAN et al., 2017).

$$\mathcal{L}^{\text{CLIP}}(\theta) = E_t \left[\min \left(r_t(\theta) A_t, \text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) A_t \right) \right], \quad r_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}. \quad (2.1)$$

A vantagem de usar ML em jogos está em permitir que agentes descubram estratégias não óbvias, adaptáveis e não previstas pelo desenvolvedor, comportamento este que enriquece a interação, a imprevisibilidade e a robustez do jogo.

3 Materiais e Métodos

Esta seção visa descrever os materiais, ferramentas e procedimentos utilizados no desenvolvimento do projeto, detalhando desde a configuração do ambiente de treinamento até os métodos empregados na implementação e avaliação dos agentes.

3.1 Ambiente e Ferramentas

Os treinamentos foram conduzidos na engine Unity 6, utilizando o pacote ML-Agents Toolkit (versão 3.0.0) para integrar o ambiente tridimensional desenvolvido com os módulos de IA externos. O cenário de testes foi estruturado como uma pista de corrida com obstáculos, contendo paredes, curvas e objetos que exigiam desvio por parte do agente. A comunicação entre Unity e Python ocorreu por meio do ML-Agents, utilizando scripts em C# no Unity e o módulo *mlagents_envs* no Python, permitindo o registro de métricas como recompensa média e desvio-padrão, além da gravação de vídeos dos episódios para análise qualitativa do comportamento dos agentes durante o treinamento.

3.2 Configuração do Cenário e Elementos do Jogo

O mapa base foi construído em formato quadrado, delimitado por quatro paredes de contenção rotuladas como Wall, definindo a área de navegação do agente. No centro do cenário foi inserido o objeto Goal, representado por uma esfera que serviu como objetivo do treinamento, tendo sua posição aleatorizada a cada episódio para evitar um ambiente estático e estimular a capacidade de generalização. O agente, por sua vez, foi modelado como um cubo de dimensões compatíveis com o cenário, acompanhado por duas esferas menores posicionadas na parte frontal, responsáveis por indicar sua orientação e direção de visualização.

O agente realiza observações contínuas do ambiente a partir de distâncias calculadas com vetores de posição e sensores de detecção (*Ray Perception Sensors*), permitindo interpretar obstáculos e orientar suas decisões de movimento. A base de observações e ações foi mantida idêntica para os métodos de aprendizado por reforço e algoritmo genético, garantindo uma comparação consistente entre os comportamentos obtidos. As ações disponíveis ao agente incluem mover-se para frente ou para trás e girar à esquerda ou à direita, compondo o conjunto fundamental de locomoção no cenário tridimensional.

3.3 Configuração do Aprendizado por Reforço

O modelo de treinamento utilizado foi o aprendizado por reforço na sua variação PPO (*Proximal Policy Optimization*), devido a oferecer um equilíbrio entre estabilidade de treinamento, eficiência amostral e simplicidade de implementação como demonstrado por (SCHULMAN et al., 2017). As regras de recompensa e penalização foram definidas para direcionar o comportamento do agente de forma equilibrada, incentivando o alcance do objetivo e penalizando ações indesejadas. O agente recebe uma recompensa de +1.0 ao atingir o *Goal*, enquanto é penalizado em -0.0001 por passo executado, estimulando-o a completar o percurso no menor tempo possível. Colisões com as paredes (*Wall*) geram penalizações mais severas -0.3 no impacto inicial e -0.003 por passo mantendo contato. O episódio é finalizado ao atingir o *Goal* ou ao ultrapassar o limite de 3 000 passos sem sucesso.

3.4 Configuração do Algoritmo Genético

Para complementar o processo de aprendizado e explorar uma estratégia alternativa de otimização, empregou-se um algoritmo genético com a finalidade específica de selecionar e ajustar os pesos da rede neural utilizada pelo agente. Nessa abordagem, cada indivíduo da população representa um conjunto completo de pesos da rede, codificados de forma direta. A avaliação desses indivíduos é realizada com base no desempenho do agente no ambiente, permitindo que a métrica de fitness reflita o quão eficaz aquela configuração de pesos foi para executar a tarefa proposta. A partir dessa avaliação, o algoritmo genético aplica operadores de seleção, recombinação e mutação para gerar novas combinações de pesos, promovendo uma busca evolutiva por configurações mais eficientes. Assim, o algoritmo genético não atua sobre parâmetros de alto nível do treinamento, mas diretamente sobre os valores internos da rede neural.

A configuração do algoritmo genético seguiu um conjunto de regras de *fitness* voltadas a incentivar o progresso do agente em direção ao objetivo e penalizar comportamentos indesejados. O agente recebe +1.0 ao atingir o *Goal* e uma recompensa proporcional à aproximação do objetivo entre o passo anterior e o atual, enquanto o afastamento resulta em penalização proporcional. Colisões com as paredes (*Wall*) acarretam uma penalidade forte de -1.0 , e cada passo executado adiciona uma penalização leve de -0.0005 , estimulando trajetórias mais curtas e eficientes.

Os parâmetros típicos utilizados no algoritmo genético foram definidos de forma a equilibrar diversidade populacional e estabilidade no processo de evolução. A população variou entre 50 e 100 indivíduos, com taxa de mutação de 0.05 e probabilidade de crossover de 0.7, valores ajustados empiricamente para manter a variabilidade sem comprometer a convergência. Foi aplicado o elitismo de aproximadamente 10%, garantindo que os melhores

indivíduos fossem preservados a cada geração. O critério de parada considerou a estagnação do desempenho ou colisões com objetos rotulados como *Wall*.

A estrutura do algoritmo genético foi dividida em seis módulos que atuam de forma integrada ao longo do processo evolutivo, são eles: rede neural; avaliação; jogo; recombinação; mutação; e Orquestração.

3.4.1 Rede Neural

O módulo de rede neural define a política que transforma observações do ambiente em comandos de ação. Adotou-se uma arquitetura perceptron multicamadas com oito entradas (correspondentes às observações enviadas pelo Unity), doze neurônios na camada oculta e duas saídas contínuas. Inicialmente utilizaram-se ativações sigmóides na camada oculta e na saída como exemplificado na equação 3.1; em iterações posteriores, testou-se ReLU na camada oculta e *tanh* na saída para produzir ações naturalmente no intervalo -1 a 1. O processamento é estritamente direto (*feed-forward*), sem memória recorrente, o que simplifica a avaliação em grande volume. Para o algoritmo genético, os pesos e vieses são tratados como um vetor de parâmetros, permitindo sua cópia, combinação e perturbação controlada ao longo das gerações.

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad z = \mathbf{w}^\top \mathbf{x} + b. \quad (3.1)$$

3.4.2 Avaliação

A avaliação opera a ponte entre a população de candidatos e o ambiente de simulação. Para cada indivíduo, o procedimento padroniza a inicialização da cena, injeta os parâmetros da rede no agente e executa um episódio até critério de término (sucesso, colisão ou tempo limite). Durante a execução, são coletadas as observações, produzidas as ações pela rede e acumulado o retorno conforme a função de *fitness* (progresso rumo ao objetivo, penalidades por colisão, custo por passo). Ao final, os indivíduos recebem uma pontuação comparável e são ordenados segundo desempenho. A consistência de tempo de episódio, *seed* e regras de término é preservada para reduzir variação entre avaliações.

3.4.3 Jogo (laço de interação com o ambiente)

Este módulo coordena a interação contínua entre agente e cenário. Em cada passo de decisão, as observações do ambiente (direção ao *waypoint*, velocidade, sinalização de solo e proximidade de obstáculos) são agregadas e encaminhadas à política, e o vetor de ações resultante é aplicado ao controlador de movimento do agente no Unity. As regras de recompensa e penalização são computadas de acordo com aproximação ao objetivo, colisões, afastamento desnecessário e custo temporal, enquanto eventos de término de episódio são

disparados quando metas são alcançadas ou quando não há progresso significativo. Esse laço também é responsável por reinicializar episódios e registrar métricas operacionais utilizadas na análise posterior.

3.4.4 Recombinação

Neste módulo é realizada a produção de descendentes a partir de dois indivíduos selecionados. Os parâmetros da rede (pesos e vieses) são combinados camada a camada por esquemas como recombinação segmentada ou mistura aritmética, de modo a preservar estruturas úteis enquanto se introduz variação. A operação respeita a forma e a escala dos tensores para evitar incompatibilidades e manter a estabilidade numérica. Com isso, características comportamentais promissoras tendem a ser herdadas pelos descendentes, ao mesmo tempo em que se reduz a dependência de mutações amplas para explorar o espaço de soluções.

3.4.5 Mutação

A mutação introduz diversidade controlada nos descendentes e nos indivíduos não elitistas. Emprega-se perturbação aditiva de pequena magnitude sobre os parâmetros (ruído gaussiano com desvio padrão reduzido), aplicada de forma probabilística por elemento ou por bloco, evitando alterações destrutivas em soluções já competitivas. Em fases iniciais, a intensidade é maior para favorecer exploração; com o avanço das gerações, a taxa e a amplitude são reduzidas para consolidar soluções.

3.4.6 Orquestrador

O orquestrador integra todas as etapas do processo evolutivo. A cada geração, conduz a avaliação da população, executa a seleção (mantendo uma fração elitista), aplica recombinação e mutação para formar a nova geração e registra métricas de acompanhamento (melhor *fitness*, mediana, desvio, checkpoints). São empregados critérios de parada por número de gerações ou por estagnação de desempenho, além da manutenção de reprodutibilidade por controle de sementes.

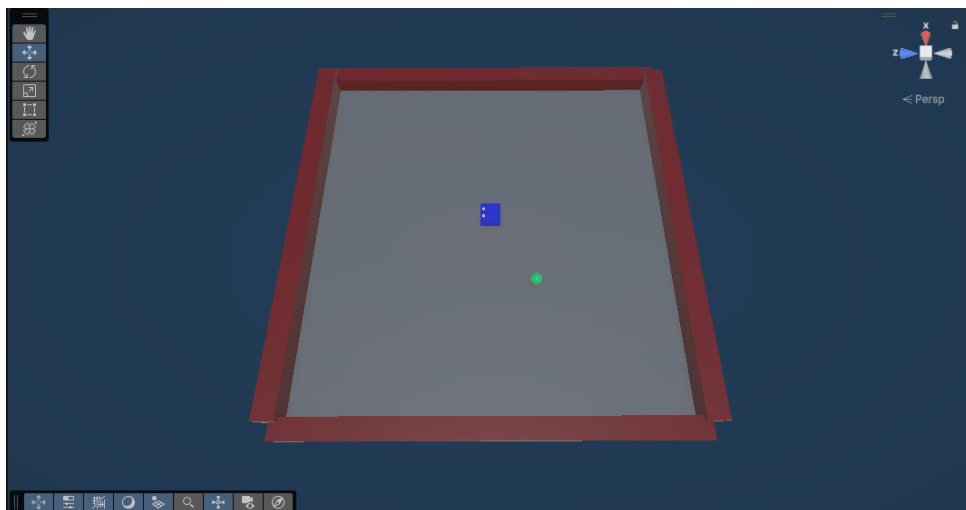
4 Treinamento dos Agentes

Esta seção descreve o treinamento dos agentes em ambiente 3D no Unity usando duas abordagens: aprendizado por reforço e algoritmo genético. O agente foi treinado para percorrer uma pista com obstáculos a partir de observações contínuas e ações de locomoção, com recompensas e penalidades no caso do aprendizado por reforço e função de *fitness* no algoritmo genético. Apresentam-se a configuração essencial do cenário e do agente, as métricas acompanhadas e os principais desafios e correções adotadas para obter comportamentos mais estáveis.

4.1 Treinamento utilizando Aprendizado por Reforço

O treinamento dos agentes utilizando o modelo de Aprendizado por Reforço se deu da seguinte forma: primeiramente foi configurado o ambiente Unity e para isso foi criado um mapa (Figura 1) que consiste em um quadrado demarcado por quatro paredes de contenção, todas devidamente nomeadas com o rótulo “Wall”; posteriormente foi inserido o objeto que seria o objetivo do treinamento, que consiste em uma esfera rotulada de “Goal”, que a cada iteração do treinamento tinha seu posicionamento aleatorizado entre 5 m e -5 m do centro do mapa para garantir que o ambiente não se mantivesse constante durante o treinamento.

Figura 1 – Estrutura que compunha o mapa base



Fonte: Elaborada pelo autor.

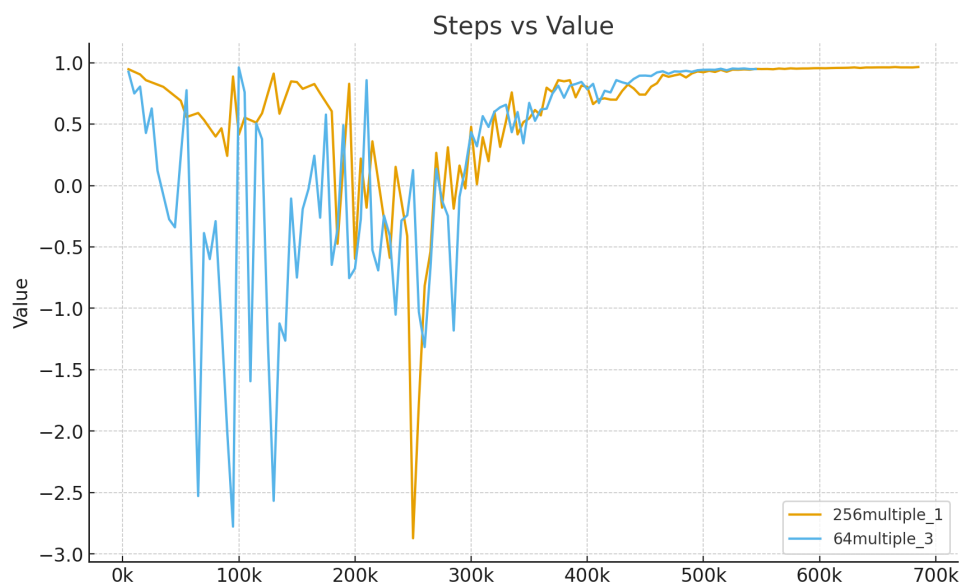
A definição das regras de treinamento foram feitas logo após a finalização do primeiro mapa de treinamento, sendo elas:

- Recompensa alta ao atingir o objetivo: caso o agente colidisse com o objeto que continha o rótulo Goal, essa recompensa inicialmente era de 1,0;
- Penalidade por tempo: cuja finalidade é incentivar o agente a finalizar a tarefa o mais rápido possível, a penalidade inicialmente era de 0,0001 por passo de treinamento;
- Penalidade maior: para caso o agente colidia com objetos rotulados como Wall, reduzindo-se gradualmente enquanto o contato era mantido. O valor inicial da penalização foi de 0.3 no momento do impacto e 0.003 por passo enquanto o agente permanecia em contato com o objeto; e
- Treinamento finalizado: em caso do agente interagir com o objetivo ou caso se passassem 3.000 passos dentro de um mesmo episódio sem que o objetivo fosse atingido, evitando assim episódios infinitos de treinamento.

4.1.1 Processo de treinamento: dificuldades e ajustes

A primeira etapa de treinamento desempenhou de maneira satisfatória, e por isso, é possível observar que, independente da quantidade de agentes sendo treinados em paralelo, a convergência se mantém por volta dos 500 mil passos, variando apenas o tempo decorrido até o modelo apresentar convergência, como demonstrado na Figura 2 e evidenciado na Tabela 1.

Figura 2 – Gráfico referente à primeira etapa de treinamento



Fonte: Elaborada pelo autor.

Tabela 1 – Tempo necessário para cada configuração atingir aproximadamente 500 000 *steps*.

Treinamento	Step	Tempo até 500k (s)	Tempo até 500k (min)
256 em paralelo	499 998	176,12	2,94
64 em paralelo	499 997	219,19	3,65

Fonte: Elaborado pelo autor.

Uma vez que o agente mostrou-se eficiente em alcançar o objetivo com uma variação de 5 m, foi feita a alteração desse parâmetro para 15 m com a finalidade de generalizar o modelo em distâncias maiores. Após a alteração, o agente diminuiu seu desempenho consideravelmente, não apresentando convergência mesmo passando de 1 milhão de passos.

Devido a redução da eficiência, foi adicionado um novo sistema de pontuação por movimentação correta, que administrava uma pequena recompensa proporcional ao quanto o agente aproximou-se do objetivo entre o passo anterior e o atual. Essa adição mostrou-se assertiva, pois acelerou a evolução do treinamento, levando o modelo para um resultado satisfatório por volta de 1 milhão de passos.

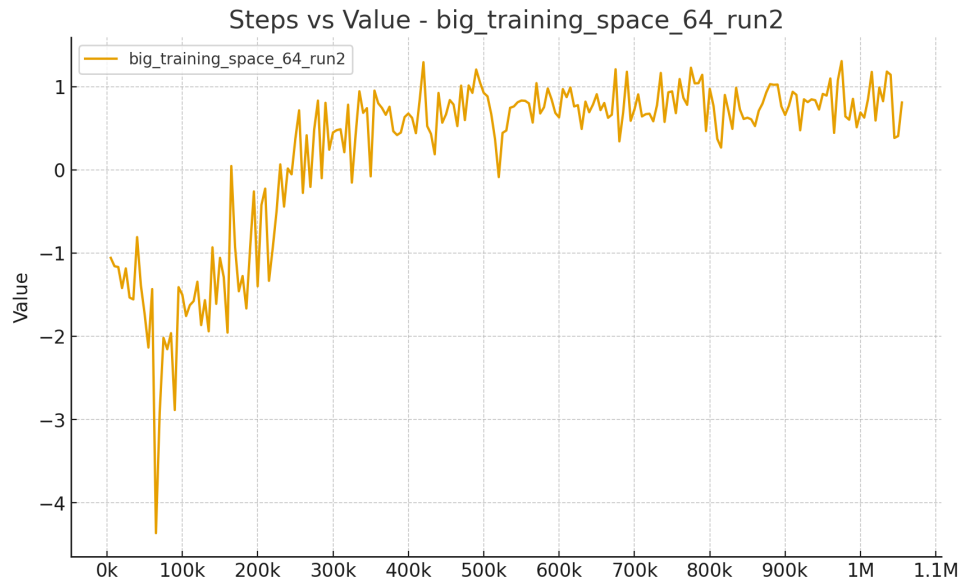
Porém, durante o treinamento foi percebido que em alguns casos, o agente havia entendido que, caso andasse em direção ao objetivo acumulando pontos de aproximação e passasse ao lado do objetivo ao invés de pegá-lo, poderia acumular infinitos pontos ao passo que a recompensa de aproximação era maior que o custo por passo para se manter jogando. Devido a essa interpretação inesperada do agente às regras de pontuação, foi necessário a implementação de uma penalidade por distanciamento do objetivo, assim o agente teria resultado negativo caso decidisse não finalizar o jogo colidindo com o objetivo.

Com a penalização implementada, o modelo mostrou-se melhor em cumprir o objetivo. Entretanto, novamente o agente se comportou de forma inconsistente, pois em alguns casos o modelo interpretou que seria mais vantajoso se movimentar em direção ao objetivo em espiral. Dessa forma, o agente receberia o reforço de pontuação por proximidade e não tomaria nenhuma penalidade, visto que o personagem não se distanciava do objetivo.

Para conter esse tipo de movimentação foi necessário a realização do aumento na penalização por tempo, pois dessa forma o ganho mínimo de aproximação seria menor que o custo por passo, garantindo assim que a pontuação se tornasse negativa. Após as alterações serem feitas, o agente mostrou-se eficiente em atingir o objetivo dentro do raio de 15 m, mostrando uma estabilidade nos resultados por volta de 600 mil passos, como exemplificado na Figura 3.

Após o treinamento ser executado, o modelo treinado foi submetido a testes em ambientes mais complexos, compostos por maior variação de curvas e obstáculos, além

Figura 3 – Gráfico referente ao treinamento após a primeira etapa de ajustes



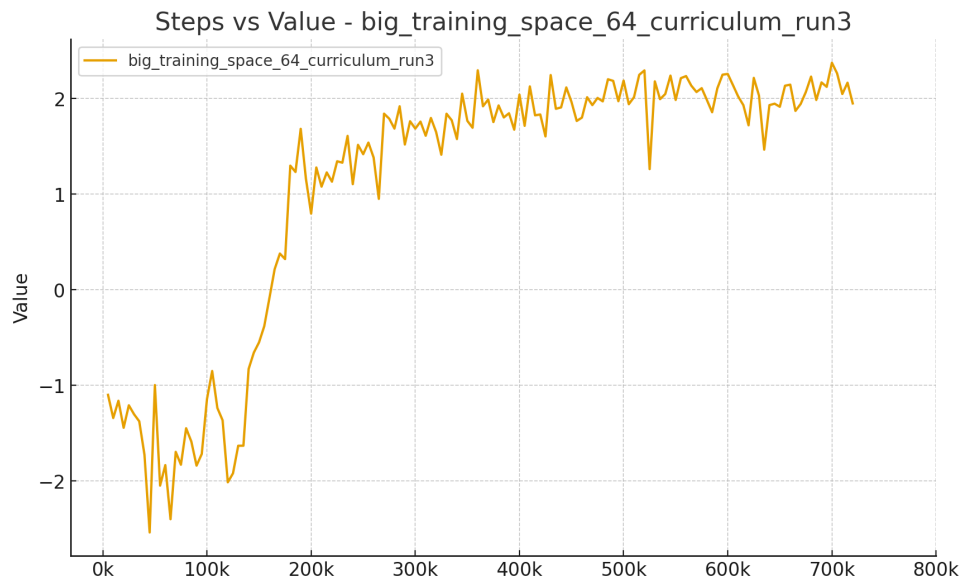
Fonte: Elaborada pelo autor.

de elevar o tamanho do mapa como um todo, fazendo com que o agente enfrentasse distâncias maiores que 15 m. Nesse novo desafio, o modelo teve um bom desempenho em atingir o objetivo quando o mesmo encontrava-se dentro da faixa de distância treinada, porém apresentando péssimos resultados quando o objetivo encontrava-se fora da faixa mencionada.

Análises foram feitas e foi constatado que o cenário era típico de um caso de *overfitting*, situação em que o agente aprende de forma excessiva as características específicas do ambiente de treinamento, tornando-se eficiente apenas naquele contexto e apresentando baixo desempenho em situações diferentes (ZHANG; BALLAS; PINEAU, 2018). Para contornar esse comportamento, foi elaborado um treinamento utilizando aprendizado por currículo, técnica que consiste em aumentar gradualmente a dificuldade durante o processo de treinamento, à medida que o modelo se mostra consistentemente bom em realizar a tarefa com os parâmetros definidos na etapa atual (BENGIO et al., 2009). Para isso foi configurado o seguinte currículo de treinamento de três etapas: na primeira etapa o objetivo teria variação em um raio de 5 m; uma vez que o modelo apresentasse consistentemente um resultado satisfatório, a distância seria aumentada para 15 m; e por último o raio seria definido para 25 m.

Finalmente, a técnica de aprendizado por currículo se mostrou bastante eficiente em aumentar a distância em que o agente era capaz de atingir o objetivo com consistência (Figura 4), apresentando uma estabilização nos resultados por volta de 500 mil passos.

Figura 4 – Gráfico referente ao treinamento com aprendizado por currículo



Fonte: Elaborada pelo autor.

4.1.2 Conclusão Parcial

Levando em conta os resultados apresentados e evidenciados na Tabela 2, o método de aprendizado por reforço tem como ponto principal positivo a sua fácil configuração técnica, uma vez que o ML-Agents Toolkit faz a integração entre Unity e Python de forma nativa. Essa integração reduz etapas operacionais, padroniza a comunicação entre ambiente e treinador e facilita a instrumentação inicial do cenário. Porém, quando se trata de performance de treinamento, o modelo se mostrou extremamente dependente de como as regras de recompensa e penalização são projetadas, fazendo com que necessite de uma extensa análise de testes e ajustes iterativos para que o treinamento apresente resultados consistentes e satisfatórios, assim como observado em (TERVO, 2022). Esse processo envolve calibrar cuidadosamente os sinais de reforço, definir condições de término adequadas e revisar continuamente métricas e registros para identificar comportamentos inesperados e correções necessárias. Embora o Unity disponibilize o ML-Agents Toolkit para o treinamento de agentes utilizando aprendizado por reforço, ainda é necessário conhecimento técnico profundo sobre análise de treinamentos e criação de jogos para se obter bons resultados utilizando a ferramenta. Em outras palavras, a qualidade do aprendizado depende diretamente de escolhas criteriosas no desenho do ambiente e das recompensas, bem como da capacidade de conduzir ciclos sucessivos de experimentação e refinamento até alcançar estabilidade e generalização aceitáveis.

Tabela 2 – Comparação dos treinamentos considerando estabilidade, desvio padrão e tempo para convergência

Treinamento	Recompensa	Desvio Padrão	Tempo (min)
Raio de 5m 64 em paralelo	0.9417	0.5540	3.65
Raio de 5m 256 em paralelo	0.9218	0.5093	2.94
Raio de 15m após ajustes	0.9267	0.3507	3.94
Raio de 25m utilizando currículo	2.1876	0.2428	4.25

4.2 Treinamento utilizando Algoritmo Genético

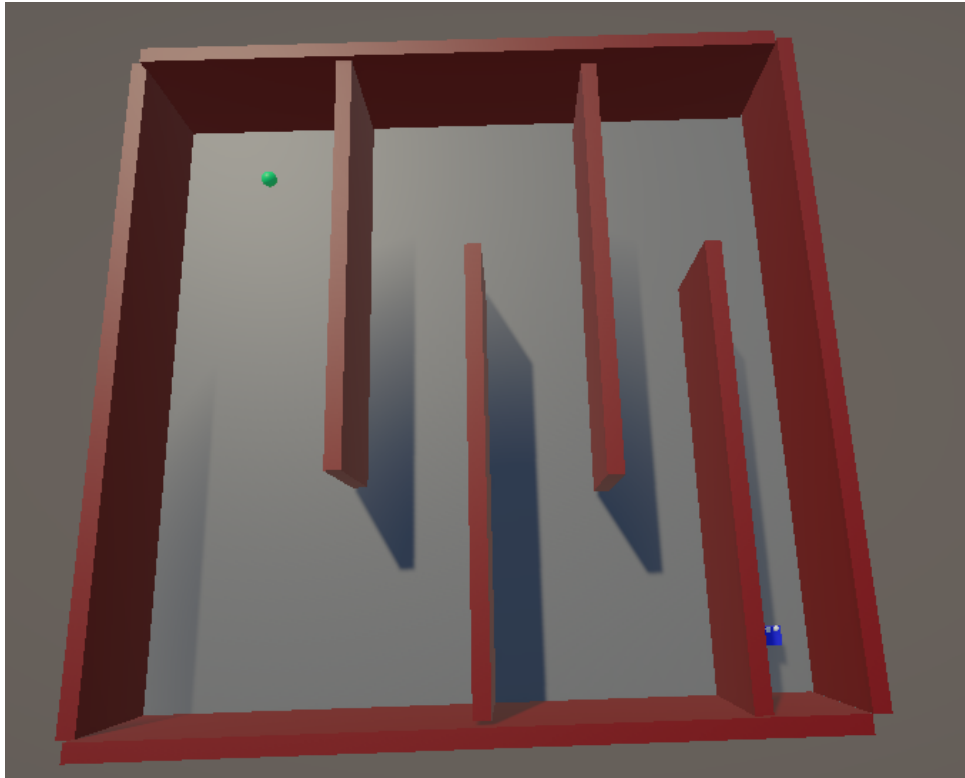
O segundo modelo de treinamento utilizou um algoritmo genético para tomar as decisões de movimentação do agente no ambiente. O treinamento ocorreu com os seguintes passos: primeiramente o ambiente foi configurado no Unity e para isso foi utilizado um mapa construído de forma semelhante ao modelo de treinamento anterior (Figura 5), composto de curvas e obstáculos os quais o agente deveria desviar, sendo penalizado em caso de colisão e contendo um objetivo que, ao ser tocado, concedia ao agente uma recompensa. As regras do treinamento foram construídas da seguinte forma:

- Uma recompensa grande quando o agente atingisse o objetivo, sendo ela de 1,0;
- Recompensas proporcionais à aproximação do objetivo entre o passo anterior e o atual;
- Penalidades proporcionais ao distanciamento do objetivo entre o passo anterior e o atual;
- Penalização forte ao colidir com os objetos rotulados com Wall, inicialmente em 1,0; e
- Penalização pequena por passo de treinamento, sendo 0,0005 para cada passo efetuado.

4.2.1 Processo de treinamento: dificuldades e ajustes

Na primeira etapa de treinamento, os agentes conseguiam se mover, mas a evolução era lenta. Após cerca de 20 gerações, a população ainda apresentava comportamentos aleatórios. A principal causa foi a escala de *fitness* inadequada, já que agentes que batiam rapidamente eram penalizados de forma igual aos que exploravam parcialmente o mapa. A solução foi normalizar a recompensa pela distância ao objetivo, permitindo diferenciar indivíduos promissores de ineficazes.

Figura 5 – Estrutura do mapa de treinamento do algoritmo genético



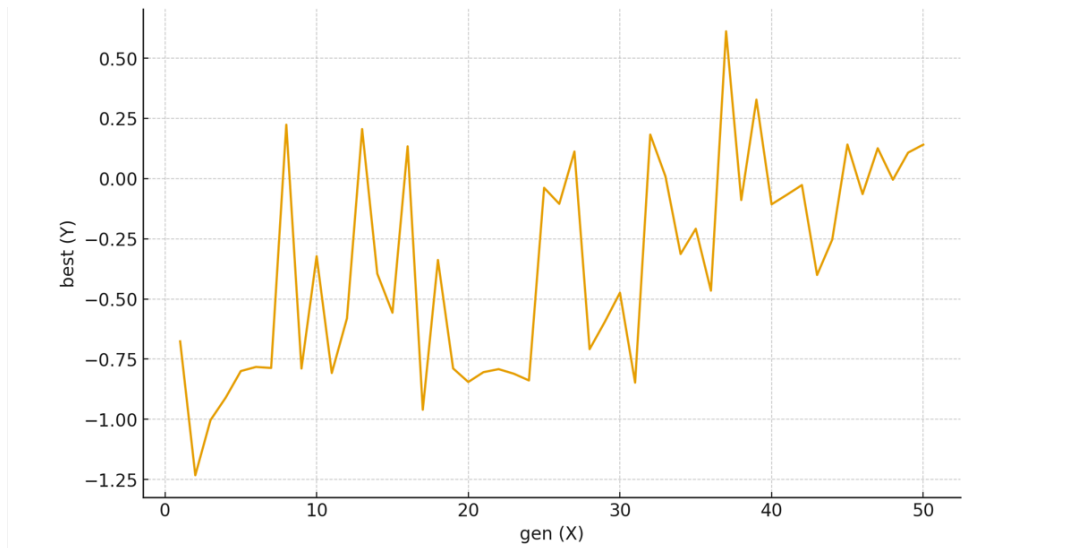
Fonte: Elaborada pelo autor.

Após a resolução dos problemas da etapa anterior utilizando recompensas proporcionais, observou-se que entre as gerações 40 e 50, apresentava-se uma convergência prematura: todos os agentes adotaram o mesmo padrão de rotação, colidindo logo após o início do episódio, evidenciado na Figura 6. A causa foi a falta de diversidade genética provocada por uma taxa de mutação baixa (0,02) e elitismo alto. O problema foi resolvido ajustando a taxa de mutação para 0,05 e introduzindo mutação gaussiana apenas em parte da população, preservando o comportamento dos melhores agentes, mas injetando variabilidade no restante. Com a implementação da mutação gaussiana e o aumento da taxa de mutação, o agente teve um resultado ligeiramente melhor que em casos anteriores, porém com um progresso total abaixo do esperado.

Observou-se então que a função de *fitness* baseada apenas na chegada ao objetivo final gerava um cenário de *feedback* esparsos: a maioria dos indivíduos não recebia reforço suficiente para orientar a evolução, o que resultava em trajetórias curtas e estagnação antes da metade do percurso. Além disso, para o agente atingir o objetivo, ele teria que se distanciar do objetivo em alguns momentos devido ao modo que o labirinto foi organizado, obtendo assim uma recompensa negativa mesmo com o comportamento correto.

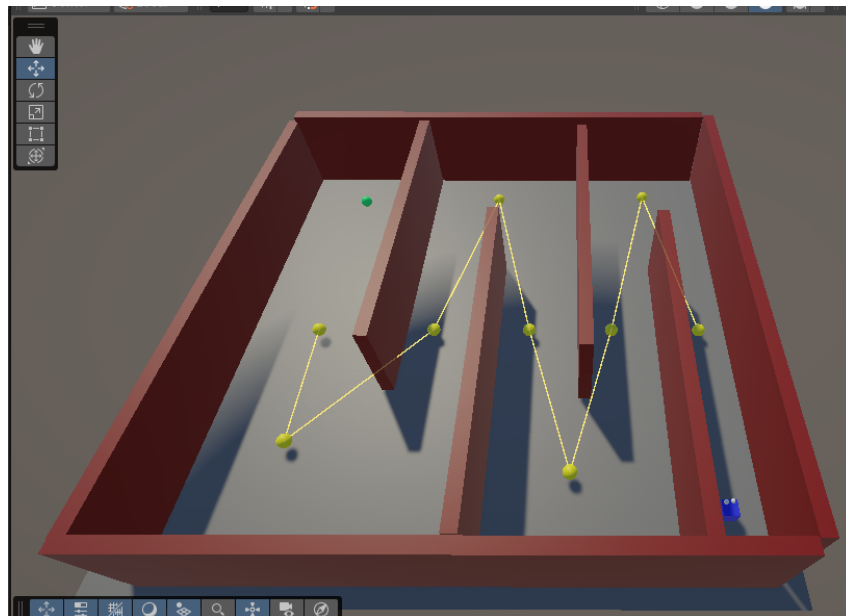
Para contornar esse problema, foram inseridos nove *waypoints* da maneira exemplificada na Figura 7, cada *waypoint* atingido pelo agente renderia uma recompensa positiva no *fitness* do mesmo ao final da geração. Essa densificação do sinal de retorno funcionou

Figura 6 – Gráfico referente à convergência entre as gerações 40 e 50



Fonte: Elaborada pelo autor.

como uma forma de reforço positivo para o agente (SAVID et al., 2023). Com isso mesmo quando o agente não concluía a rota, ele acumulava ganhos proporcionais ao avanço, permitindo diferenciar melhor os indivíduos durante a seleção. Após ajustes, algumas execuções apresentaram indivíduos capazes de alcançar dois ou três *waypoints* consecutivos, mas o comportamento geral manteve-se abaixo do esperado.

Figura 7 – Imagem do circuito com a adição dos *waypoints*

Fonte: Elaborada pelo autor.

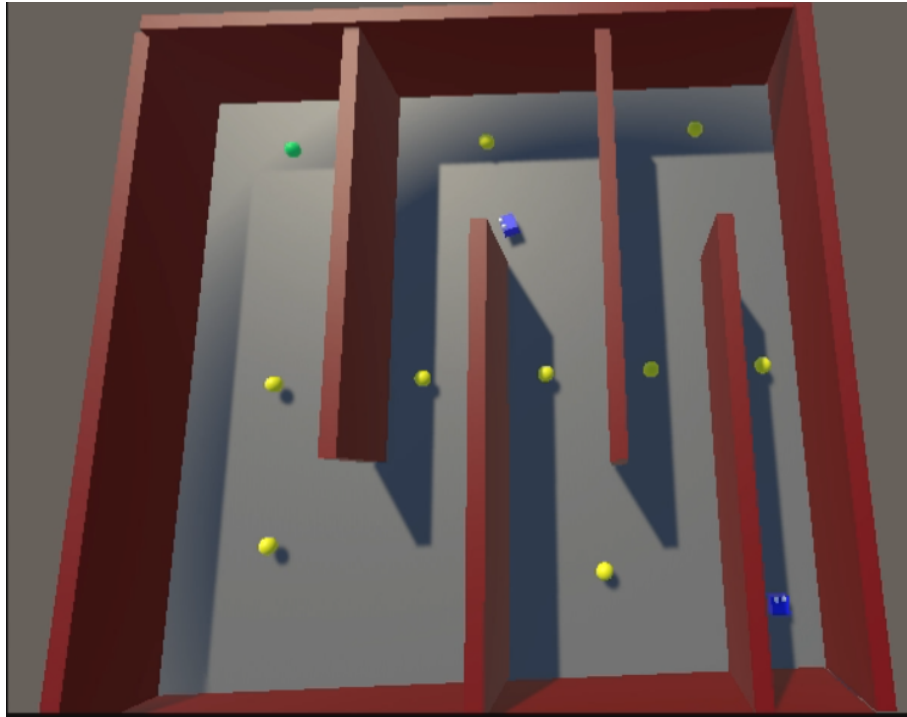
Diante desse resultado, foram realizadas alterações na rede neural, substituindo os 12 neurônios da camada oculta, que anteriormente utilizavam a função de ativação

sigmoid, pela ativação *ReLU* como mostrada na equação 4.1, com o objetivo de aprimorar o processo de aprendizado (GLOROT; BORDES; BENGIO, 2011). Além disso, a camada de saída, inicialmente composta também por neurônios com ativação *sigmoid*, cujo retorno variava entre 0 e 1 e era posteriormente extrapolado para o intervalo de -1 a 1 utilizado pelo agente, foi modificada para utilizar a função *tanh*, que nativamente produz saídas no intervalo de -1 a 1 evidenciadas na equação 4.2, proporcionando uma resposta mais adequada ao controle de movimentação do agente. Mesmo com os ajustes da rede neural o agente continuou apresentando baixo desempenho, nos melhores cenários chegando apenas até o quinto *waypoint* como mostra a Figura 8.

$$\text{ReLU}(z) = \begin{cases} 0, & \text{se } z < 0, \\ z, & \text{se } z \geq 0, \end{cases} \quad \text{onde } z = \mathbf{w}^\top \mathbf{x} + b. \quad (4.1)$$

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad \text{para qualquer } z \in \mathbf{R}, \quad \text{onde } z = \mathbf{w}^\top \mathbf{x} + b. \quad (4.2)$$

Figura 8 – Imagem do agente que completou 5 *waypoints*



Fonte: Elaborada pelo autor.

4.2.2 Conclusão Parcial

Em vista dos fatos previamente analisados, percebe-se que o método utilizando algoritmo genético se mostrou pouco eficaz para tal tarefa do modo em que foi configurado

para esse experimento, necessitando de testes e ajustes para retornar um desempenho mediano no cumprimento do objetivo proposto. Esse comportamento implica ciclos prolongados de experimentação, com sucessivas tentativas de calibragem dos parâmetros evolutivos e reavaliação contínua do progresso obtido, a fim de evitar estagnação e melhorar gradualmente a solução encontrada. Outro ponto negativo é a não integração nativa com o Unity, sendo necessária a construção de todo o algoritmo em Python para que o método possa ser utilizado. Essa ausência de suporte direto eleva o esforço de engenharia, uma vez que demanda a implementação completa do fluxo de avaliação e evolução, além do gerenciamento da comunicação entre o ambiente e o processo externo, o que aumenta a complexidade operacional e o tempo de desenvolvimento.

5 Conclusão

Levando em conta os resultados alcançados, o aprendizado por reforço apresentou como principal ponto positivo a facilidade de configuração técnica, uma vez que o ML-Agents Toolkit integra Unity e Python de forma nativa, padronizando a coleta de observações, a aplicação de ações e o registro de métricas. Por outro lado, o desempenho do aprendizado por reforço mostrou-se altamente dependente do desenho das recompensas e penalizações, exigindo testes extensivos e ajustes iterativos, como por exemplo, balancear aproximação ao objetivo, custo por passo e penalização por colisão, para alcançar estabilidade e resultados consistentes. Esse comportamento evidencia que, mesmo com o suporte do ML-Agents, ainda é necessário conhecimento técnico sólido em análise de treinamentos e design de jogos para se obter bons resultados.

No caso do algoritmo genético, observou-se baixa eficácia para a tarefa nas condições configuradas neste experimento, demandando testes e ajustes para atingir um desempenho apenas mediano. Entre os fatores limitantes estiveram a convergência prematura, a sensibilidade da função de *fitness* e a necessidade de densificar o sinal de retorno para evitar estagnação. Além disso, a ausência de integração nativa com o Unity impôs a implementação completa do ciclo evolutivo em Python (avaliação, seleção, *crossover* e mutação), aumentando a complexidade da orquestração e da sincronização com o editor.

Comparativamente, o aprendizado por reforço mostrou uma melhor relação entre esforço de implementação e resultado quando acompanhado de curadoria cuidadosa de recompensas e, quando necessário, de técnicas como curriculum learning. O algoritmo genético, embora útil para exploração ampla do espaço de soluções, exigiu maior engenharia e maior sinalização de progresso para evitar colapsos populacionais e evoluções sem direção. Em síntese, para o problema de locomoção com desvio de obstáculos em ambiente 3D, o aprendizado por reforço se mostrou mais adequado no conjunto de ferramentas e restrições adotadas, enquanto o algoritmo genético mostrou-se dependente da infraestrutura adicional e calibrações mais profundas para atingir níveis semelhantes de desempenho.

Para trabalhos futuros, sugere-se investigar variantes de aprendizado por reforço, como SAC e TD3 (FUJIMOTO; HOOFF; MEGER, 2018), uma vez que esses algoritmos tendem a apresentar ganhos de estabilidade em cenários com dinâmica mais sensível, além de permitir políticas com maior suavidade nas ações. Também se recomenda empregar esquemas mais robustos de normalização e escalonamento de recompensas, com padronização consistente ao longo do treino para reduzir oscilações. Para o algoritmo genético, sugere-se utilizar outras configurações de redes neurais, explorando variações de

profundidade, larguras de camada e funções de ativação, bem como outros métodos de definição e propagação dos resultados, com ajustes no critério de avaliação e na forma de agregação do desempenho ao longo do episódio para tornar o *feedback* mais informativo e menos esparsos. No plano sensorial, pode-se testar entradas complementares, como raycasts direcionais, variando alcance, ângulos e densidade de amostragem, de modo a ampliar a percepção do agente em torno de obstáculos e melhorar a tomada de decisão em trechos críticos do percurso.

Referências

- AZOUBEL, P. B.; PINA, L. V.; DEMAISON, A. L.; CAMPOS, L. F. d. A. Game design e hci: A importância de estudos e pesquisa no processo de desenvolvimento de jogos digitais. In: *Anais do 12º Congresso Brasileiro de Pesquisa e Desenvolvimento em Design*. São Paulo: Blucher, 2016. (Blucher Design Proceedings, 2), p. 4410–4418. ISSN 2318-6968. Acesso em: 27 mar. de 2025. Disponível em: <https://pdfs.semanticscholar.org/2fbd/1f3b13c3315f361e2eafb8157e6d9f388e2c.pdf>.
- BENGIO, Y.; LOURADOUR, J.; COLLOBERT, R.; WESTON, J. Curriculum learning. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2009. (ICML '09), p. 41–48. ISBN 9781605585161. Acesso em 20 ago. de 2025. Disponível em: <https://doi.org/10.1145/1553374.1553380>.
- CONNOLLY, T. M.; BOYLE, E. A.; MACARTHUR, E.; HAINEY, T.; BOYLE, J. M. A systematic literature review of empirical evidence on computer games and serious games. *Computers Education*, v. 59, n. 2, p. 661–686, 2012. ISSN 0360-1315. Acesso em 22 out. de 2025. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0360131512000619>.
- FUJIMOTO, S.; HOOF, H. van; MEGER, D. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. Acesso em 22 out. de 2025. Disponível em: <https://arxiv.org/abs/1802.09477>.
- GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. In: GORDON, G.; DUNSON, D.; DUDÍK, M. (Ed.). *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Fort Lauderdale, FL, USA: PMLR, 2011. (Proceedings of Machine Learning Research, v. 15), p. 315–323. Acesso em 15 set. de 2025. Disponível em: <https://proceedings.mlr.press/v15/glorot11a.html>.
- HALEEM, M. S. *Advances in Artificial Intelligence, Machine Learning and Deep Learning Applications*. Coventry: Eletronics, 2023. Acesso em: 27 mar. 2025. Disponível em: <https://www.mdpi.com/2079-9292/12/18/3780>.
- HENDERSON, P.; ISLAM, R.; BACHMAN, P.; PINEAU, J.; PRECUP, D.; MEGER, D. *Deep Reinforcement Learning that Matters*. 2019. Acesso em 22 out. de 2025. Disponível em: <https://arxiv.org/abs/1709.06560>.
- JULIANI, A.; BERGES, V.-P.; VCKAY, E.; GAO, Y.; HENRY, H.; MATTAR, M.; LANGE, D. *Unity: A General Platform for Intelligent Agents*. 2018. Acesso em 03 abr. de 2025. Disponível em: <https://ui.adsabs.harvard.edu/abs/2018arXiv180902627J/abstract>.
- MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G.; PETERSEN, S.; BEATTIE, C.; SADIK, A.; ANTONOGLOU, I.; KING, H.; KUMARAN, D.; WIERSTRA, D.; LEGG, S.; HASSABIS, D. Human-level control through deep reinforcement learning. *Nature*, v. 518, n. 7540, p. 529–533, feb 2015. Acesso em 22 out. de 2025. Disponível em: <https://www.nature.com/articles/nature14236>.

- MÜLLER, A. A.; BITTENCOURT, J. R. Ud3 - unity decision tree integrator: Uma extensão de Árvores de decisão para unity. In: *2021 IEEE Latin America Conference on Computing, Communications and Electronics (LACCE)*. IEEE, 2021. p. 65–70. Acesso em: 27 mar. de 2025. Disponível em: <https://www.sbgames.org/sbgames2014/papers/computing/short/20_allcomputingshortpages.pdf>.
- ONTANON, S.; SYNNAEVE, G.; URIARTE, A.; RICHOUX, F.; CHURCHILL, D.; PREUSS, M. Rts ai problems and techniques. In: _____. [s.n.], 2015. ISBN 978-3-319-08234-9. Acesso em: 27 mar. de 2025. Disponível em: <https://link.springer.com/rwe/10.1007/978-3-319-08234-9_17-1>.
- SAVID, Y.; MAHMOUDI, R.; MASKELIŪNAS, R.; DAMAŠEVIČIUS, R. Simulated autonomous driving using reinforcement learning: A comparative study on unity's ml-agents framework. *Information*, v. 14, n. 5, 2023. ISSN 2078-2489. Acesso em 30 ago. de 2025. Disponível em: <<https://www.mdpi.com/2078-2489/14/5/290>>.
- SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. *Proximal Policy Optimization Algorithms*. 2017. Acesso em 20 ago. de 2025. Disponível em: <<https://arxiv.org/abs/1707.06347>>.
- STANLEY, K. O.; MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evol. Comput.*, MIT Press, Cambridge, MA, USA, v. 10, n. 2, p. 99–127, jun. 2002. ISSN 1063-6560. Acesso em 07 set. de 2025. Disponível em: <<https://doi.org/10.1162/106365602320169811>>.
- TERVO, A. *Effects of curriculum learning on maze exploring DRL agent using Unity ML-Agents*. 69 p. Dissertação (Master's thesis) — University of Turku, Turku, Finland, jun 2022. Acesso em 20 out. de 2025. Disponível em: <<https://www.utupub.fi/handle/10024/154305>>.
- TUPE, K.; SINGH, P.; PARMAR, S.; PANDEY, T. Ai & npc in games. *International Journal of Computer Engineering in Research Trends*, India, v. 3, n. 3, p. 129–133, 2016. Acesso em: 27 mar. 2025. Disponível em: <https://ijcert.org/ems/ijcert_papers/V3I307.pdf>.
- UPDYKE, D.; PODNAR, T.; DOBSON, G.; YARGER, J. *Using Machine Learning to Increase NPC Fidelity*. [S.l.], 2021. Acesso em 27 mar. de 2025. Disponível em: <<https://doi.org/10.1184/R1/14107373>>.
- ZHANG, A.; BALLAS, N.; PINEAU, J. *A Dissection of Overfitting and Generalization in Continuous Reinforcement Learning*. 2018. Acesso em 23 ago. de 2025. Disponível em: <<https://arxiv.org/abs/1806.07937>>.