UNIVERSIDADE FEDERAL DO CEARÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

ISMAEL SOARES DA SILVA

BURIEDBRAINS: UM AMBIENTE MULTIAGENTE INSPIRADO EM ROGUELIKE PARA AVALIAÇÃO DE MEMÓRIA E TOMADA DE DECISÃO

SOBRAL

2025

ISMAEL SOARES DA SILVA

BURIEDBRAINS: UM AMBIENTE MULTIAGENTE INSPIRADO EM ROGUELIKE PARA AVALIAÇÃO DE MEMÓRIA E TOMADA DE DECISÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia da computação do Centro de Ciências e Tecnologia (CCT) da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia da computação.

Orientador: Prof. Dr. Thiago Iachiley Araújo de Souza

SOBRAL

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S654b Soares da Silva, Ismael.
BuriedBrains: Um Ambiente Multiagente Inspirado em Roguelike para Avaliação de Memória e Tomada de Decisão / Ismael Soares da Silva. – 2025.
102 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Sobral, Curso de Engenharia da Computação, Sobral, 2025.
Orientação: Prof. Dr. Thiago Iachiley Araújo de Souza.

1. Aprendizado por Reforço. 2. Ambientes Parcialmente Observáveis. 3. Sistemas Multiagente. 4. Comportamento Emergente. 5. Inteligência Artificial. I. Título.

CDD 621.39

ISMAEL SOARES DA SILVA

BURIEDBRAINS: UM AMBIENTE MULTIAGENTE INSPIRADO EM ROGUELIKE PARA AVALIAÇÃO DE MEMÓRIA E TOMADA DE DECISÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia da computação do Centro de Ciências e Tecnologia (CCT) da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia da computação.

Aprovada em: 28 de Janeiro de 2026

BANCA EXAMINADORA

Prof. Dr. Thiago Iachiley Araújo de Souza (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Me. Erick Aguiar Donato
Universidade Federal do Ceará (UFC)

Prof. Dr. Artur de Oliveira da Rocha Franco
Universidade Federal do Ceará (UFC)

Deo Optimo Maximo

AGRADECIMENTOS

Aos meus pais. Nos momentos de minha ausência dedicados à graduação e aos projetos, vocês me ensinaram que o futuro é decidido pela forma como alocamos nossos recursos no presente.

Às personas Shabriri e "T-Rex", pelas sugestões 'opcionais' que serviram de solo firme para uma redação mais madura e confiante neste trabalho.

Ao meu orientador, Prof. Dr. Thiago Iachiley, pela confiança e pela autonomia concedida ao projeto.

À Universidade Federal do Ceará (UFC), Campus de Sobral, e ao Curso de Engenharia da Computação, pelo ambiente e pelos recursos que propiciaram minha formação.

Agradeço a todos os professores cuja paixão era pelas pessoas, e não apenas pelo ato de ensinar. Devo a vocês parte essencial de minha formação como mente pensante e ser humano.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, pela adequação do *template* LaTeX utilizado neste trabalho, facilitando o atendimento às normas da biblioteca da UFC.

Aos bibliotecários da Universidade Federal do Ceará: Francisco Edvander Pires Santos, Juliana Soares Lima, Izabel Lima dos Santos, Kalline Yasmin Soares Feitosa e Eliene Maria Vieira de Moura, pela revisão e normalização deste *template*.

Ao aluno Thiago Nascimento, da Universidade Estadual do Ceará, que elaborou a versão original do *template* do qual este trabalho foi adaptado.

Gratias ago Deo, qui mentem illuminavit

“I am grateful.”

(Kentaro Miura (Berserk))

RESUMO

Este trabalho apresenta o desenvolvimento e a validação do *BuriedBrains*, um ambiente de simulação multiagente fundamentado em geração procedural e observabilidade parcial, estruturado por meio de grafos dinâmicos híbridos. O objetivo principal consiste em mensurar o desempenho de agentes de Aprendizado por Reforço (RL) na resolução de problemas de planejamento de longo prazo e generalização sob condições de *permadeath* e restrição de recursos.

A topologia do ambiente, inspirada em mecânicas do gênero *roguelike* — especificamente o título *Buriedbornes* (Nussygame, 2016) —, utiliza Grafos Acíclicos Dirigidos (DAGs) para as Zonas de Progressão e grafos cíclicos para os Santuários. Essas estruturas demandam gerenciamento de inventário e habilidades, uma vez que as transições de estado resultam em alterações persistentes em variáveis como *Karma* e na probabilidade de sobrevivência do agente.

A metodologia consistiu em um estudo comparativo para isolar o impacto da recorrência temporal, contrastando uma arquitetura reativa (PPO) com uma recorrente (LSTM). Ambos os modelos foram otimizados via *Hyperparameter Optimization* (HPO) e integrados a mecanismos de *Self-Attention*. Os dados sugerem que as características do *BuriedBrains* restringem o desempenho de modelos reativos sob as condições testadas. Essa limitação reflete-se na estabilização do PPO em um patamar de 65% de Variância Explicada (EV), resultado que se manteve constante mesmo com o aumento da complexidade da rede ou a expansão do horizonte de amostragem.

O agente recorrente apresentou desempenho 24% superior nos testes de Otimização de Hiperparâmetros e atingiu 80% de variância explicada. Os resultados sugerem que a arquitetura foi capaz de sustentar políticas voltadas à especialização de atributos e apresentar padrões de interação multiagente. Análises via ferramentas de visualização identificaram a convergência para a estabilização em estados de não-agressão recíproca, em que a memória permitiu a otimização da longevidade do episódio em detrimento de interações de soma zero. Os resultados obtidos com o *BuriedBrains* sugerem que a integração de mecânicas de *roguelike* estabelece um ambiente no qual a reatividade simples apresenta limitações, sugerindo a necessidade de arquiteturas que incorporem mecanismos de persistência de estados e processamento de sequências temporais.

Palavras-chave: Aprendizado por Reforço. Ambientes Parcialmente Observáveis. Sistemas Multiagente. Comportamento Emergente. Inteligência Artificial.

ABSTRACT

This work presents the development and validation of *BuriedBrains*, a multi-agent simulation environment based on procedural generation and partial observability, structured through hybrid dynamic graphs. The primary objective is to measure the performance of Reinforcement Learning (RL) agents in solving long-term planning and generalization problems under conditions of *permadeath* and resource constraints.

The environment's topology, inspired by roguelike mechanics — specifically the title *Buriedbornes* (Nussygame, 2016) —, utilizes Directed Acyclic Graphs (DAGs) for Progression Zones and cyclic graphs for Sanctuaries. These structures demand inventory and skill management, as state transitions result in persistent changes to variables such as *Karma* and the agent's survival probability.

The methodology consisted of a comparative study to isolate the impact of temporal recurrence, contrasting a reactive architecture (PPO) with a recurrent one (LSTM). Both models were refined through Hyperparameter Optimization (HPO) and integrated with Self-Attention mechanisms. Data suggests that the characteristics of *BuriedBrains* restrict the performance of reactive models under the tested conditions, as reflected by PPO's stabilization at a 65% Explained Variance (EV) plateau, which remained constant despite increases in network complexity or sampling horizon. The recurrent agent demonstrated 24% superior performance in HPO tests and reached 80% explained variance. Results suggest the architecture was capable of sustaining policies focused on attribute specialization and exhibiting multi-agent interaction patterns. Visualization-based analysis identified convergence toward reciprocal non-aggression states, where memory enabled the optimization of episode longevity over zero-sum interactions. *BuriedBrains* thus serves as a benchmark for evaluating long-term temporal dependency resolution and decision-making in autonomous systems.

Keywords: Reinforcement Learning. Partially Observable Environments. Multi-agent Systems. Emergent Behavior. Artificial Intelligence.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

s_t	Estado do ambiente no instante de tempo t
a_t	Ação selecionada pelo agente no instante de tempo t
r_t	Recompensa recebida no instante de tempo t
$R(s,a)$	Função de recompensa, que retorna o valor esperado da recompensa r_t
G_t	Retorno (soma das recompensas futuras descontadas) a partir de t
$\pi(a\|s)$	Política do agente, que mapeia estados a uma distribuição de probabilidade sobre ações
π^*	Política ótima que maximiza o retorno esperado
γ	Fator de desconto de recompensas futuras
$\mathbb{E}[\cdot]$	Operador de valor esperado
θ	Parâmetros da rede neural que representa a política
$L^{CLIP}(\theta)$	Função objetivo substituta do PPO
$\hat{A}_t$	Estimativa da função de vantagem no instante t
ε	Hiperparâmetro de *clipping* do PPO
α	Taxa de aprendizado (*learning rate*) do otimizador
β	Coeficiente de entropia para incentivo à exploração
λ	Parâmetro do Estimador de Vantagem Generalizada (GAE)
$\|\nabla\|_{max}$	Limiar de truncamento do gradiente (*Gradient Clipping*)
$\mathscr{S}, \mathscr{A}, \Omega$	Conjuntos de estados, ações e observações, respectivamente
$\mathscr{T}, \mathscr{R}, \mathscr{O}$	Funções de transição, recompensa e observação, respectivamente
b_t	Crença (*belief*) do agente sobre o estado no instante t
o_t	Observação parcial recebida no instante t
$G = (V,E)$	Grafo composto por um conjunto de vértices V e arestas E
$\mathscr{V}, \mathscr{E}$	Conjunto de vértices e de arestas do grafo, respectivamente
p_{kz}	Probabilidade de conectividade no modelo de Erdős-Rényi (Santuário)
$\mathscr{N}(v_t)$	Vizinhança do vértice v_t (conjunto de vértices adjacentes)
$d(u,v)$	Distância geodésica (caminho mínimo) entre os vértices u e v

C_D, C_B, C_C	Métricas de centralidade (Grau, Intermediação e Proximidade)
$S(v)$	Índice de relevância topológica (*Score*) para poda de nós
$\omega_1, \omega_2, \omega_3$	Pesos da combinação linear para o cálculo do $S(v)$
z_t	Estado de reputação (Karma) de um agente no disco de Poincaré em t
$\mathbb{D}$	Disco de Poincaré, onde $\|z\| < 1$
$d_H(z_1, z_2)$	Distância hiperbólica entre dois estados de reputação
ν	Elemento de fibra contextual (especificação do contexto da reputação)
$\mu(z_t, a)$	Vetor de *drift* (tendência determinística) da reputação
W_t	Processo de Wiener (Movimento Browniano padrão)
ρ	Fator de escala para o ruído estocástico (difusão)
∇_H	Operador de gradiente no espaço hiperbólico
h_t	Vetor de estado oculto de uma rede recorrente
$\mathbf{Q}, \mathbf{K}, \mathbf{V}$	Matrizes de Consulta, Chaves e Valores (*Self-Attention*)
$\text{Attention}(\cdot)$	Função de atenção por produto escalar escalonado
f	Andar atual da simulação (variável de profundidade)
$\Lambda(f)$	Multiplicador de escalonamento logístico para o andar f
V_{max}	Valor assintótico superior de atributos (teto de escalonamento)
M_{budget}	Multiplicador de intensidade global do orquestrador
δ	Taxa de Redução de Dano (*Damage Reduction*)
ϕ	Coeficiente de *Floor Tax* (penalidade por profundidade)
τ	Fator de escalonamento por *Tier* de inimigo
ζ	Amplitude da flutuação estocástica de combate

SUMÁRIO

1 INTRODUÇÃO

Os avanços no Aprendizado por Reforço (Reinforcement Learning – RL) têm impulsionado o desenvolvimento de agentes computacionais capazes de aprender e se adaptar a diferentes tarefas, consolidando essa abordagem como um dos pilares da inteligência artificial moderna. Entretanto, a generalização em Processos de Decisão de Markov Parcialmente Observáveis (POMDPs) impõe desafios estruturais à tomada de decisão, sendo característica principal de processos de decisão de vasta margem de aplicações em RL. Nestes cenários, o agente opera sob incerteza de estado, recompensas esparsas e a necessidade de atribuição de crédito temporal, definida pela dependência entre ações imediatas e desfechos em estados futuros (SUTTON; BARTO, 2018; LAKE *et al.*, 2017).

Este trabalho apresenta o desenvolvimento do *BuriedBrains*, um ambiente de simulação fundamentado em mecânicas do gênero *roguelike* e inspirado no título *Buriedbornes* (Nussygame, 2016). O sistema foi concebido para incorporar características não-markovianas, implementando condições de terminação definitiva do episódio e disponibilidade limitada de recursos. Estas propriedades atuam como variáveis que requerem a persistência de estados internos para a progressão do agente no simulador.

A utilização de estruturas de jogos como plataformas experimentais segue procedimentos metodológicos estabelecidos em ambientes como *StarCraft II* (VINYALS *et al.*, 2019). O gênero *roguelike* é recorrente na literatura para a análise de generalização, como observado no *NetHack Learning Environment* (KÜTTLER *et al.*, 2020), enquanto o *Neural MMO* (SUAREZ *et al.*, 2019) explora sistemas de sobrevivência e progressão. O *BuriedBrains* associa a geração procedural a dependência temporal, partindo do pressuposto de que padrões comportamentais específicos emergem da busca pela maximização de duração de episódios em ambientes estocásticos, visto que a perda é definitiva, e a geração procedural adiciona não-linearidade ao escopo.

A validade do ambiente como plataforma de RL fundamenta-se em quatro propriedades integradas:

1. **Geração Procedural (PCG):** Exige a formulação de políticas generalizáveis, mitigando a eficácia da memorização de estados;
2. **Atribuição de Crédito Temporal:** Associa a seleção de parâmetros iniciais a resultados tardios no horizonte do episódio;
3. **Observabilidade Parcial (POMDP):** Requer o processamento de informações que não

estão contidas integralmente na observação imediata;

4. **Terminação Definitiva e Escassez de Sinais:** Resulta em um gradiente de aprendizado com ruído elevado, dado o custo do erro e a esparsidade de recompensas positivas.

O objetivo central deste trabalho consiste no desenvolvimento do *BuriedBrains* e na caracterização de suas dinâmicas operacionais. Para inferir as propriedades do ambiente em termos de *baseline*, a metodologia inclui uma análise comparativa entre duas arquiteturas: PPO (*Proximal Policy Optimization*) e *RecurrentPPO*.

A investigação avalia o impacto da introdução de uma camada recorrente baseada em *Long Short-Term Memory* (LSTM) na política do agente. Os resultados sugerem que as limitações de desempenho da arquitetura puramente reativa (PPO) indicam a natureza não-markoviana do sistema.

1.1 Justificativa

A motivação deste estudo reside na lacuna entre o processamento eficiente de estados imediatos e a otimização de políticas em horizontes temporais extensos no Aprendizado por Reforço (RL). Embora mecanismos de *Self-Attention* permitam o processamento de grandes volumes de dados em observações pontuais, a análise isolada do estado atual não garante a convergência para políticas que maximizem o retorno acumulado em cenários de alta complexidade.

O ambiente *BuriedBrains* impõe restrições de recompensa que limitam a convergência de arquiteturas cujas políticas adotam abordagens estritamente reativas. A aplicação de uma curva logística de escalonamento adversarial e modelagem de recompensa deliberadamente esparsa indica que políticas sem persistência de estado apresentam desempenho subótimo diante do incremento da dificuldade inicial. O aumento progressivo do dano infligido por agentes de níveis superiores estabelece uma condição ambiental que requer a otimização uma política que associe diversas camadas de atributos do agente (e.g.: inventário, uso de habilidades, tempos de recarga) para maior acumulativo de recompensas por episódio.

A introdução de um sistema de reputação baseado em geometria hiperbólica (*Karma*) e das interações multiagente adiciona variáveis temporais e relacionais ao processo de decisão. O gerenciamento de inventário (Ações 5–8) é utilizado para operacionalizar a mecânica de sinalização custosa (*Costly Signaling*). Todos esses fatores exigem que o processo de Atribuição de Crédito Temporal avalie ações que refletem não-estacionariedade (a política de agentes muda constantemente), compensando estatisticamente o incremento na taxa de sobrevivência em

estados subsequentes.

Dessa forma, a investigação avalia se a otimização de políticas neste ambiente é favorecida pela retenção de contexto temporal, o que indiciaria modelagem de observações não-Markovianas. Ao manter o extrator de características por atenção constante tanto para arquiteturas reativas quanto recorrentes, o *BuriedBrains* isola a memória como variável experimental, permitindo assim analisar se a persistência de estados é um requisito para a emergência de padrões de cooperação em sistemas multiagente sob escassez de recursos.

1.2 Objetivos

1.2.1 Objetivo Geral

Apresentar o ambiente *BuriedBrains* como plataforma para a análise de dependência temporal em políticas de Aprendizado por Reforço (RL). A investigação utiliza mecanismos de *Self-Attention* para isolar a influência da persistência de estados recorrentes em relação ao processamento de observações pontuais.

1.2.2 Objetivos Específicos

- Formalizar a estrutura do ambiente por meio da Teoria dos Grafos, modelando a progressão procedural em Grafos Acíclicos Dirigidos (DAGs) e definindo as zonas de interação multiagente;
- Estabelecer um *baseline* de desempenho comparativo entre as arquiteturas PPO e RecurrentPPO (LSTM), mantendo invariante o `AttentionFeatureExtractor` de 198 dimensões para isolar o impacto de arquiteturas baseadas em recorrência;
- Avaliar a estabilidade das estimativas de gradiente das políticas sob o regime de escalonamento de dificuldade e escassez de recompensas;
- Investigar a mecânica de descarte de itens como sinalização entre agentes, analisando seu impacto na eficácia da atribuição de crédito temporal sob o sistema de reputação implementado.

1.3 Estrutura do trabalho

Este trabalho organiza-se em cinco capítulos. O Capítulo 1 contextualiza o problema e delimita os objetivos. O Capítulo 2 aborda a fundamentação teórica sobre Processos de Decisão de Markov Parcialmente Observáveis (POMDPs) e arquiteturas de RL. O Capítulo 3 detalha a especificação técnica do ambiente e das arquiteturas testadas. O Capítulo 4 apresenta os dados experimentais e a análise comparativa entre os modelos. O Capítulo 5 sintetiza as observações e indica possíveis extensões para a pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo estabelece as bases teóricas para o desenvolvimento do ambiente *BuriedBrains* e a análise de agentes inteligentes. A fundamentação abrange os princípios de Aprendizado por Reforço, sistemas multiagente, interação estratégica e emergência de comportamentos. Adicionalmente, discutem-se os desafios de observabilidade parcial e a validação de ambientes simulados.

2.1 Aprendizado por reforço (RL)

O Aprendizado por Reforço (RL) caracteriza-se pela interação entre um agente e um ambiente em etapas discretas. A cada passo de tempo t, o agente recebe o estado s_t, executa uma ação a_t segundo sua política $\pi(a|s)$ e recebe um sinal de recompensa r_t, resultando na transição para o estado s_{t+1}. Neste paradigma, o objetivo consiste em determinar uma política que maximize o retorno, definido como o valor esperado da soma das recompensas acumuladas ao longo do tempo (SUTTON; BARTO, 2018).

O retorno (G_t) para o instante t consiste na soma das recompensas futuras, ponderadas pelo fator de desconto $\gamma \in [0,1]$, conforme a Equação 2.1. O parâmetro γ regula a influência de recompensas imediatas em comparação a recompensas futuras.

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1}$$

O processo de otimização visa determinar a política π^* que maximiza o valor esperado do retorno considerando a distribuição de estados, conforme a Equação 2.2.

$$\pi^* = \arg\max_{\pi} \mathbb{E}[G_t|\pi] \tag{2.2}$$

2.1.1 Proximal Policy Optimization (PPO)

O algoritmo *Proximal Policy Optimization* (PPO) integra a classe de métodos de *policy gradient* e é aplicado em problemas de RL devido à sua estabilidade durante o treinamento e ao equilíbrio entre complexidade de implementação e eficiência amostral (SCHULMAN *et al.*, 2017). O método baseia-se na otimização de uma função objetivo substituta, empregando um mecanismo de *clipping* para restringir a magnitude das atualizações da política em cada iteração.

Em diversos cenários de controle, o PPO mantém-se como uma linha de base válida, apresentando estabilidade superior a métodos de otimização de política mais recentes, mesmo em ambientes complexos (PARK *et al.*, 2024). Tal evidência fundamenta a utilização do PPO como algoritmo de controle neste estudo.

A função objetivo do PPO, em sua formulação com *clipping*, é definida pela Equação 2.3.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t \right) \right] \tag{2.3}$$

Nesta expressão, $\hat{\mathbb{E}}_t$ denota a expectativa empírica sobre as amostras de treinamento, θ representa os parâmetros da política e $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ denota a razão de probabilidade entre a política atual e a política de coleta (*old*). O termo $\hat{A}_t$ corresponde à estimativa da função de vantagem no instante t e ε é o hiperparâmetro que delimita o intervalo de *clipping*. Este mecanismo restringe a magnitude das mudanças na política (policy updates), visando mitigar instabilidades durante o processo de treinamento.

2.2 Sistemas Multiagente (MAS)

Sistemas Multiagente (*Multi-Agent System* - MAS) consistem em estruturas computacionais integradas por múltiplos agentes autônomos que operam em um ambiente compartilhado. Cada unidade opera sob objetivos específicos e autonomia decisória, de modo que o comportamento global do sistema emerge das interações locais entre os agentes. A área de MAS analisa mecanismos de cooperação, coordenação e competição estabelecidos nesse contexto (WOOLDRIDGE, 2009).

O ambiente *BuriedBrains* opera como um sistema multiagente, especificamente em cenários de interação competitiva (PvP - *Player versus Player*). Nessas condições, a tarefa do agente torna-se mais complexa que a otimização de trajetórias individuais, incorporando a interação estratégica com outros agentes autônomos. As ações dos demais agentes introduzem dinâmicas de não-estacionariedade que impactam o desempenho e o alcance dos estados de meta estabelecidos

2.3 Teoria dos Jogos

A Teoria dos Jogos consiste na modelagem matemática de interações estratégicas entre agentes racionais (MYERSON, 1991). Este campo de estudo analisa cenários em que o resultado das ações de um agente é condicionado pelas decisões de terceiros. O **Dilema do Prisioneiro** ilustra uma situação em que a busca pelo interesse individual resulta em um equilíbrio subótimo em comparação à cooperação mútua.

Dinâmicas semelhantes são observadas em interações entre agentes no ambiente *BuriedBrains*. Nestas interações, os agentes deparam-se com escolhas análogas a um dilema social. A cooperação, como a negociação de passagem, pode resultar em ganhos mútuos de utilidade. Contudo, a defecção oferece o potencial de maiores ganhos individuais, sob o risco de respostas punitivas e perdas bilaterais. A arquitetura do ambiente é estruturada para que essas tensões estratégicas emerjam das regras de interação estabelecidas.

2.4 Modelagem de reputação

O modelo proposto utiliza a geometria hiperbólica para descrever a dinâmica de confiança em sistemas multiagente. O estado reputacional de cada agente é mapeado como um ponto z no disco de Poincaré, definido por $\mathbb{D} = \{z \in \mathbb{C} : |z| < 1\}$. Essa representação é adotada devido à propriedade do espaço hiperbólico em acomodar estruturas hierárquicas e relações de proximidade em variedades de curvatura constante negativa (NICKEL; KIELA, 2017).

As coordenadas polares de z definem os parâmetros operacionais da reputação:

- **Magnitude ($|z|$):** Indica o grau de estabilidade ou convergência da reputação. Valores de magnitude próximos à unidade sugerem estados reputacionais com baixa variância residual e menor sensibilidade a flutuações estocásticas.
- **Argumento ($\arg(z)$):** Representa a orientação do comportamento observado, diferenciando tendências como cooperação ou defecção no sistema.

A dissimilaridade entre dois estados reputacionais, z_1 e z_2, é quantificada pela métrica hiperbólica isométrica (CANNON *et al.*, 1997), conforme apresentado na Equação 2.4:

$$d_H(z_1, z_2) = \operatorname{arcosh}\left(1 + \frac{2|z_1 - z_2|^2}{(1-|z_1|^2)(1-|z_2|^2)}\right) \tag{2.4}$$

Uma característica central desta métrica é o crescimento exponencial da distância conforme o vetor se aproxima da fronteira do disco ($|z| \to 1$). O modelo faz uso desta geometra

para formalizar a decréscimo da taxa de variação posicional em estados extremos. Enquanto na origem (neutralidade) variações de baixa magnitude resultam em mudanças posicionais mensuráveis, em regiões periféricas o sistema requer um volume maior de interações para induzir deslocamentos significativos nas coordenadas hiperbólicas.

A evolução temporal do estado reputacional é descrita como um processo de difusão na variedade Riemanniana. A dinâmica segue uma Equação Diferencial Estocástica (SDE)baseada em processos de fluxo de gradiente em variedades Riemannianas (BONNABEL, 2013), definida na Equação 2.5:

$$dz = (1 - |z|^2) \cdot (-\nabla\Phi(z)dt + \sigma dW_t) \tag{2.5}$$

2.5 Emergência controlada em sistemas autônomos

A emergência define-se como a manifestação de propriedades sistêmicas não descritas nas regras de transição ou nas funções de recompensa. Tais padrões decorrem da interação entre agentes individuais. Em sistemas multiagente e modelos de larga escala, a análise foca na identificação de comportamentos auto-organizados ausentes na especificação algorítmica inicial (TANG; KEJRIWAL, 2025).

No ambiente *BuriedBrains*, a arquitetura não estabelece trajetórias prévias. Restrições a interações específicas, como a defecção e a predação, são aplicadas ao sistema para que se observe a convergência das políticas de ação. Embora as ações básicas (equipar, soltar, atacar) sejam tratadas de forma atômica no espaço de ações, sua execução sequencial e contextual é processada pelo sistema de recompensas. Deste processo derivam-se padrões de trajetória passíveis de análise quantitativa em relação aos objetivos do ambiente.

Variáveis como atribuição de dano, tempos de recarga (*cooldowns*) e o sistema de reputação (*karma*) definem as condições de contorno para a operação dos agentes. A ocorrência de estratégias de interação não previstas sugere a convergência resultante da otimização das políticas (π) diante da estrutura de incentivos e limites ambientais.

2.6 Ambientes Parcialmente Observáveis (POMDP)

Em aplicações práticas, o estado real do ambiente pode não ser diretamente acessível ao agente. Essa condição é modelada por meio de Processos de Decisão de Markov Parcialmente

Observáveis (*Partially Observable Markov Decision Processes* - POMDP) (KAELBLING *et al.*, 1998). Um POMDP é definido pela 8-tupla apresentada na Equação 2.6:

$$(\mathscr{S}, \mathscr{A}, \mathscr{T}, \mathscr{R}, \Omega, \mathscr{O}, b_0, \gamma) \tag{2.6}$$

Nesta definição, $\mathscr{S}$ representa o conjunto de estados, $\mathscr{A}$ o de ações, $\mathscr{T}$ a função de transição e $\mathscr{R}$ a função de recompensa. Os componentes específicos que caracterizam a observabilidade parcial incluem o conjunto de observações Ω, a função de probabilidade de observação $\mathscr{O}$, a distribuição de crença inicial b_0 e o fator de desconto γ. Em contraste com o MDP, o agente recebe uma observação $o_t \in \Omega$ em vez do estado $s_t \in \mathscr{S}$. A estimativa do estado atual, portanto, é computada a partir do histórico acumulado de observações e ações.

2.7 Agentes com memória: Redes Neurais Recorrentes

Ambientes parcialmente observáveis demandam arquiteturas de rede neural capazes de processar e integrar observações ao longo de séries temporais. As Redes Neurais Recorrentes (*Recurrent Neural Networks* - RNNs) são aplicadas para que o agente retenha informações de observações passadas (HOCHREITER; SCHMIDHUBER, 1997). A operação básica de uma RNN é descrita pela Equação 2.7. Nessa formulação, o estado oculto h_t no instante t é definido como uma função do estado anterior h_{t-1} e da entrada corrente o_t.

$$h_t = f(W_{hh}h_{t-1} + W_{xh}o_t + b_h) \tag{2.7}$$

As matrizes W_{hh} e W_{xh} representam os pesos da rede, b_h denota o vetor de viés, e f denota uma função de ativação não linear. A arquitetura *Long Short-Term Memory* (LSTM) expande esse princípio ao utilizar mecanismos de portões (*gates*) para controlar o fluxo de dados. Os mecanismos de *gating* auxiliam na preservação de dependências temporais de longo prazo.

Em cenários de longo horizonte temporal, a recorrência facilita o processo de Atribuição de Crédito Temporal (*Temporal Credit Assignment*). Essa estrutura permite a representação da relação entre ações pretéritas, como a coleta de itens, e recompensas obtidas em etapas posteriores do episódio. Dessa forma, mitiga-se a perda de informações relevantes em tarefas com horizonte temporal extenso.

2.8 Grafos aleatórios: o modelo de Erdős–Rényi

A geração procedural de ambientes com variabilidade estocástica sob parâmetros definidos utiliza o modelo de grafo aleatório Erdős–Rényi (E-R), denotado por $G(n,p)$. Neste modelo (ERDőS; RéNYI, 1959), o grafo é construído com n vértices e cada aresta potencial é incluída de forma independente com probabilidade p. O parâmetro p determina:

- A densidade esperada do grafo: $E[|E|] = \binom{n}{2} p$
- O limiar de conectividade global: $p_c = \frac{\ln n}{n}$ (transição para a conectividade global)

A transição de fase do modelo E-R é aplicada neste estudo. Quando p excede p_c, verifica-se a formação de um componente gigante que conecta $\Theta(n)$ vértices (BOLLOBáS, 2001). Esta propriedade fundamenta:

1. A manutenção de conectividade entre as salas do ambiente
2. O controle da variabilidade estrutural via intervalo $p \in [0.25, 0.35]$
3. A geração de configurações espaciais não determinísticas

2.8.1 Análise Estrutural: Métricas de Centralidade

O refinamento das estruturas geradas utiliza métricas de centralidade baseadas em Análise de Redes Sociais (FREEMAN, 1978) para a execução de poda heurística.

Tabela 1 – Métricas de centralidade aplicadas

Métrica	**Descrição Operacional**
$C_D(v) = \frac{\deg(v)}{n-1}$	Centralidade de Grau do vértice v
$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$	Frequência de intermediação de caminhos
$C_C(v) = \frac{n-1}{\sum_{u \neq v} d(u,v)}$	Proximidade média aos demais nós

O processo de poda é orientado pelo cálculo:

$$\text{Score}(v) = \alpha C_D(v) + \beta C_B(v) + \gamma C_C(v) \tag{2.8}$$

Os pesos α, β, γ regulam a ênfase em propriedades de:

- Conectividade local (C_D)
- Intermediação de trajetórias (C_B)
- Dispersão e proximidade global (C_C)

Esta configuração visa manter as propriedades estruturais preponderantes do grafo ao mitigar a redundância de arestas. O método demonstra a viabilidade de gerar ambientes com

propriedades topológicas controladas, mantendo a complexidade necessária para a navegação do agente.

2.9 Validação de ambientes

A validação de um novo ambiente de simulação e a análise de sua complexidade são etapas essenciais na pesquisa em RL. Este trabalho adota duas práticas para este fim.

2.9.1 Mecanismos de Atenção (Self-Attention)

A incorporação de mecanismos de atenção em arquiteturas de *Deep Reinforcement Learning* (Deep RL) baseia-se em princípios do modelo *Transformer* (VASWANI *et al.*, 2017). O mecanismo de *Self-Attention* atribui pesos variáveis a diferentes componentes do vetor de observação. Dessa forma, o sistema atribui maior relevância estatística a elementos específicos da entrada, como entidades ou objetos de interesse, reduzindo a influência de componentes ruidosos independentemente da posição espacial no dado de entrada.

O mapeamento da atenção utiliza projeções lineares representadas pelas matrizes de consulta (Q), chaves (K) e valores (V). A operação consiste no cálculo do produto escalar (pontuação de similaridade) entre a consulta e as chaves, processada por uma função *softmax* e aplicada aos valores:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.9}$$

Onde d_k representa a dimensão das chaves e atua como fator de escala para evitar a saturação da função *softmax* e garantir a estabilidade dos gradientes durante o treinamento.

No ambiente *BuriedBrains*, esse mecanismo auxilia na extração de características com maior correlação com o sinal de recompensa, precedendo as camadas de decisão da política de ação. Diferente de redes densas (MLPs) com pesos estáticos ou redes convolucionais (CNNs) restritas a dependências locais, a atenção estabelece dependências globais entre elementos da observação.

2.9.2 Agentes de Referência (Baseline)

A caracterização de novos ambientes de aprendizado por reforço envolve a avaliação de agentes de referência. A implementação de arquiteturas estabelecidas, como o algoritmo

PPO utilizando camadas recorrentes do tipo Long Short-Term Memory (LSTM), processadas via retropropagação através do tempo (*Backpropagation Through Time*)é utilizada para estabelecer patamares de desempenho (baselines) iniciais. Esse procedimento fornece um referencial de comparabilidade, permitindo situar a complexidade do ambiente em relação a marcos da literatura de *benchmarking* (BELLEMARE *et al.*, 2013; VINYALS *et al.*, 2019; KÜTTLER *et al.*, 2020).

3 METODOLOGIA

Este capítulo apresenta os procedimentos para o desenvolvimento e validação do ambiente de simulação *BuriedBrains*. A metodologia compreende três etapas: a modelagem formal via Teoria dos Grafos e Processos de Decisão de Markov Parcialmente Observáveis (POMDP); a implementação do simulador para Aprendizado por Reforço; e a definição da arquitetura dos agentes com o respectivo protocolo experimental.

O sistema mantém as propriedades mecânicas e topológicas de jogos roguelike. Pretende-se estabelecer a formulação do ambiente que permita avaliar como os agentes lidam com dependências temporais e com a escolha de ações voltadas a horizontes de longo prazo.

3.1 Modelagem formal do ambiente

O *BuriedBrains* baseia-se em um grafo dinâmico $G = (V, E)$. Nesta estrutura, V representa o conjunto de vértices (salas) e $E \subseteq V \times V$ o conjunto de arestas (conexões). A interação do agente com o ambiente é formalizada como um POMDP, definido pela tupla $(\mathscr{S}, \mathscr{A}, \mathscr{T}, \mathscr{R}, \Omega, \mathscr{O}, \gamma)$.

Em contraste com ambientes de observabilidade total, a política do agente computa ações condicionada a uma observação local $o_t \in \Omega$, gerada pela função de observação $\mathscr{O}(o_t|s_t)$. Esta observação contém os atributos do vértice atual v_t e dados parciais da vizinhança imediata $\mathscr{N}(v_t)$.

3.1.1 Modelagem da topologia procedural

A modelagem utiliza alternância topológica para diversificar os padrões de navegação exigidos.

O módulo `map_generation.py` utiliza a biblioteca *NetworkX* para a modelagem topológica do ambiente. A navegação é estruturada sob dois paradigmas distintos de grafos: um sistema de progressão linear para o modo Agente-versus-Ambiente (*Player-versus-Environment* (PvE)) e uma configuração de centralidade para interações multiagente de soma não-zero em topologia específica Agente-versus-Agente (*Player-versus-Player* (PvP)).

O sistema gera duas estruturas distintas conforme o nível de profundidade d, que determina a frequência de aparição de Santuários:

- **Zonas de progressão (DAGs):** Nos andares de exploração, o grafo é um Grafo Acíclico

Dirigido (DAG). Esta estrutura impõe transições irreversíveis; a ausência de arestas de retorno estabelece, portanto, um custo de oportunidade inerente à trajetória do agente.

- **Santuários e arenas (Grafos Cíclicos):** Nos andares de interação, a topologia é um grafo não-dirigido conexo. A geração baseia-se no modelo $G(n,p)$ de Erdős–Rényi aplicando-se um critério de aceitação para assegurar um componente gigante conexo. Essa configuração permite a circulação entre vértices, requisito para mecânicas de distanciamento e aproximação entre agentes.

3.1.1.1 Topologia das zonas de progressão

Nas zonas de progressão, o ambiente opera como um DAG, denotado por $G_P = (V_P, E_P)$. Nesta topologia, cada vértice possui, no máximo, dois sucessores diretos. Cada aresta dirigida $(v, v') \in E_P$ define o suporte da função de transição $\mathscr{T}(s'|s,a)$, limitando os estados alcançáveis à vizinhança imediata no grafo.

Ao percorrer uma aresta, o ramo alternativo torna-se inacessível. Tal característica visa operacionalizar a irreversibilidade de seleção de caminhos. A Figura 1 exemplifica a estrutura sob a perspectiva do agente. Estando no vértice $v_{2,2}$, as opções de transição são:

$$(v_{2,2}, v_{2,3,A}) \quad \text{e} \quad (v_{2,2}, v_{2,3,B})$$

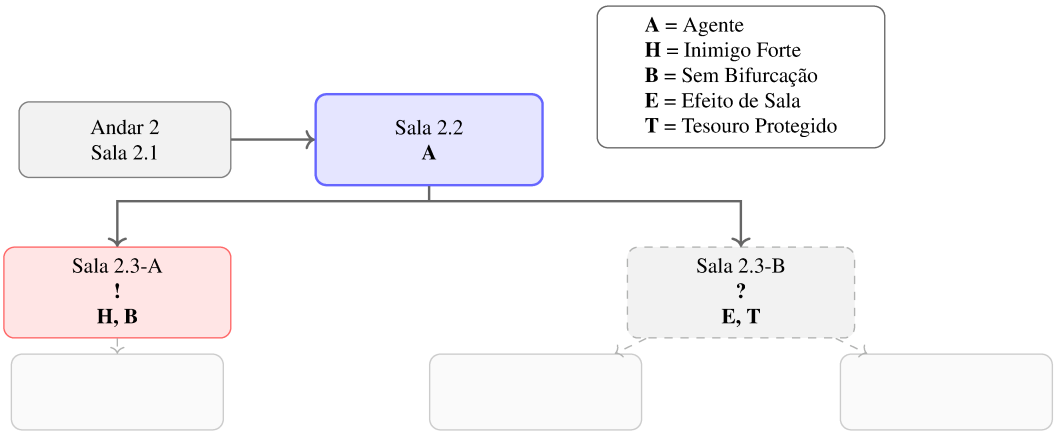
A rede neural do agente recebe como entrada atributos parciais dos vértices sucessores para orientar a seleção da próxima ação:

- $\psi(v_{2,3,A}) = \{\text{agente adversarial de alta dificuldade, grau de saída} = 1\}$;
- $\psi(v_{2,3,B}) = \{\text{modificador de sala, recurso protegido}\}$.

A função de mapeamento $\psi : V_P \to \mathscr{P}(\mathbb{A})$ associa o vértice a um subconjunto do espaço de atributos $\mathbb{A}$.

O conteúdo detalhado dos vértices sucessores permanece oculto até a transição efetiva.

Figura 1 – Representação da topologia de um andar de progressão como um Grafo Acíclico Dirigido (DAG), ilustrando a visão parcial do agente no vértice $v_{2,2}$ e os atributos dos vértices sucessores.



Fonte: Elaborado pelo autor (2025).

3.1.1.2 Topologia do Santuário

A cada $\mathscr{K}$ andares de progressão (parâmetro padrão $\mathscr{K} = 20$), o ambiente utiliza uma topologia denominada Santuário. Esta estrutura é destinada à interação multiagente, com mecânicas de pareamento baseadas em sistemas de invasão e encontros pontuais presentes em *Dark Souls* (FromSoftware, 2011) e *Buriedbornes* (Nussygame, 2016).

A coordenação dos encontros ocorre por uma fila de sincronização (*matchmaking queue*). Agentes que atingem níveis de profundidade f que sejam múltiplos de $\mathscr{K}$ são retidos. O sistema instancia o Santuário e inicia o episódio de interação multiagente, removendo ambos os agentes da fila, e alocando-os nas extremidades do santuário gerado.

O grafo do Santuário é gerado pelo modelo aleatório de Erdős–Rényi $G(n,p)$, aplicando-se um procedimento de Poda Baseada em Centralidade (*Centrality-Based Pruning*). Para cada vértice v, define-se um índice de relevância topológica $S(v)$ por meio da combinação linear de três métricas de rede normalizadas:

$$S(v) = \omega_1 \cdot C_D(v) + \omega_2 \cdot C_B(v) + \omega_3 \cdot C_C(v) \tag{3.1}$$

As variáveis $C_D(v)$, $C_B(v)$ (Intermediação) e $C_C(v)$ (Proximidade) quantificam a conectividade local, o fluxo de caminhos mínimos e a acessibilidade do nó, ponderadas pelos pesos ω_i.

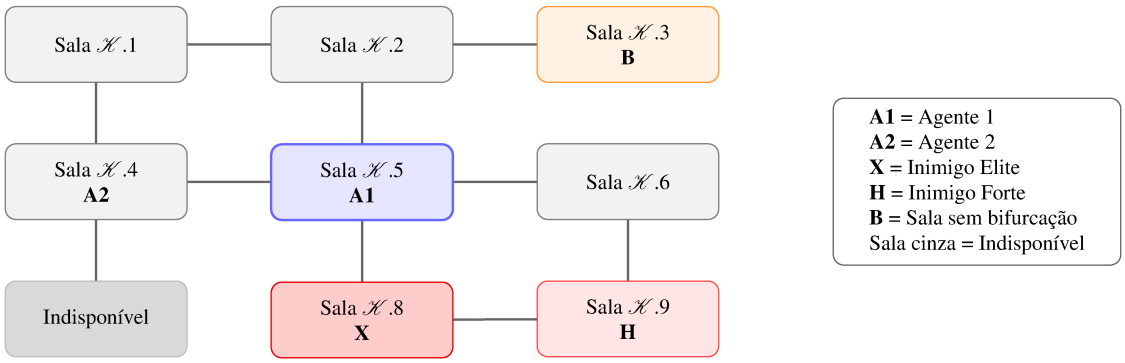
A organização desta arena baseia-se em dois critérios de classificação de nós:

- **Nó central** (`is_center`)**:** definido pela centralidade de grau ($C_D(v)$), assegurando que o ponto central possua a maior conectividade local imediata da arena.

- **Nós de saída (`is_exit`):** Estabelecidos nos vértices que apresentam a maior distância geodésica em relação ao centro. Esta disposição espacial requer que as trajetórias de saída interceptem a zona central, o que eleva o potencial de encontros multiagente na região de maior conectividade.

A Figura 2 apresenta a estrutura resultante. A restrição de observabilidade parcial (POMDP) permanece ativa: o agente acessa informações apenas do vértice atual e da vizinhança imediata ($d = 1$). A ausência de dados sobre a posição global do agente adversarial caracteriza o cenário de observabilidade parcial necessário para a investigação de padrões de interação social, como cooperação e confronto, sob incerteza.

Figura 2 – Exemplo de topologia de um andar **K** (Santuário), representada por um grafo não-direcionado de ordem 9.



Fonte: Elaborado pelo autor (2025).

3.1.1.3 Gerenciamento de memória

Para otimizar o consumo de recursos computacionais em simulações com múltiplos agentes, o sistema incorpora a remoção de estruturas de dados excedentes por meio do método `_handle_exploration_turn` no script `env.py`. Ao processar a transição para um nó sucessor em uma estrutura de progressão, o componente gerenciador elimina as ramificações adjacentes e seus respectivos descendentes do grafo associado ao agente (`nx.descendants`). O procedimento prioriza a eficiência no uso de memória através do descarte de caminhos inacessíveis.

3.1.2 Modelagem do agente

O agente é definido por um vetor de estado interno $\mathscr{S}_{agent} \in \mathbb{R}^n$, que compreende atributos de estado variáveis. Diferente de arquiteturas para *Gridworlds* convencionais (SUTTON; BARTO, 2018), o *BuriedBrains* demanda a seleção de ações sob restrições de recursos de longo prazo e a otimização da alocação de atributos para a progressão no episódio.

Atributos e Estado Inicial: No início de cada episódio ($t = 0$), o agente é instanciado no Nível 1 com o seguinte perfil de atributos:

- **Pontos de vida (*Health Points*(HP)):** 300;
- **Parâmetros ofensivos:** Dano base 10, probabilidade de acerto de 90% e 5% de probabilidade de acerto crítico (multiplicador de $2,0\times$);
- **Parâmetros defensivos:** Probabilidade de evasão base de 5% e defesa nula;
- **Reputação (κ):** Variável $\kappa \in \mathbb{C}$ inicializada em $0+0i$, representando o alinhamento de interação do agente.

3.1.2.1 Curva de progressão

A evolução do agente é regida por uma função seccionada que define o requisito de experiência (XP_{req}) para a transição de nível $L \in \mathbb{Z}^+$. Esta estrutura visa modular a escala de dificuldade entre os estágios iniciais, intermediários e avançados do experimento:

$$XP_{req}(L) = \begin{cases} 100 + (L \cdot 25), & \text{se } L \leq 10 \\ 400 + (L-10) \cdot 40, & \text{se } 10 < L \leq 40 \\ 1600 + (L-40) \cdot 100, & \text{se } L > 40 \end{cases} \tag{3.2}$$

A progressão de nível resulta em incrementos fixos nos atributos, conforme especificado na Tabela 2. Tais incrementos são aplicados de forma determinística para isolar o efeito das decisões do agente sobre o desempenho final.

Tabela 2 – Parâmetros de progressão de atributos do agente

Atributo	**Incremento (Δ_L)**	**Descrição Técnica**
HP Máximo	$+10$	Aumento do limite máximo da variável de saúde.
Dano Base	$+2$	Incremento aditivo na magnitude de saída ofensiva.
Defesa	$0,1$	Aumento na taxa percentual de mitigação de dano.

Fonte: Elaborado pelo autor (2025).

Progressão de atributos

A Tabela 3 apresenta a parametrização do sistema de progressão no *BuriedBrains*. A calibração da curva de experiência (XP) define que o centésimo andar corresponda ao nível 20 do agente.

Esta configuração resulta em um ritmo de evolução decrescente. O escalonamento da função de custo de nível determina que os primeiros 10 níveis ocorram em intervalos médios

de 5 andares. Níveis subsequentes demandam intervalos de sobrevivência extensos. A estrutura de progressão estabelece que a defesa passiva atinja $1,9\%$ de mitigação no centésimo andar.

Este valor de mitigação é inferior ao incremento de dano projetado para agentes adversariais e chefes, cujas funções de dano seguem crescimentos logístico e exponencial.

Nestas condições, a eficácia do agente associa-se a variáveis externas à progressão linear de atributos:

1. **Otimização de inventário:** Obtenção de itens com modificadores de estado para compensar a reduzida mitigação base;
2. **Processamento temporal de recompensas:** Priorização da sobrevivência de longo prazo em relação a retornos imediatos;
3. **Interação multiagente:** Uso do alinhamento reputacional κ na modulação de confrontos e cooperação.

A configuração do ambiente desacopla a aquisição de políticas de sobrevivência da evolução estatística de atributos. O modelo indica a necessidade de decisões fundamentadas em contexto e memória de longo prazo.

Tabela 3 – Parâmetros de progressão do agente (andares 1 a 100)

Andar	Nível	HP Máximo	Dano Base	Defesa (%)
1	1	300	10	0,00%
10	3	320	14	0,20%
20	5	340	18	0,40%
30	7	360	22	0,60%
40	9	380	26	0,80%
50	11	400	30	1,00%
60	13	420	34	1,20%
70	15	440	38	1,40%
80	16	450	40	1,50%
90	18	470	44	1,70%
100	20	490	48	1,90%

Fonte: Elaborado pelo autor (2025).

Escalonamento procedural de agentes adversariais Buscando estabelecer um incremento não-linear de dificuldade, o *BuriedBrains* utiliza uma função logística (sigmoide) para o escalonamento de integridade (HP) e dano das unidades agentes adversariais até o andar 500. O valor de um atributo V no andar f é definido por:

$$V(d) = \left(\frac{V_{max}}{1 + e^{-k_{\text{slope}}(d - f_0)}} \right) - \text{offset} \tag{3.3}$$

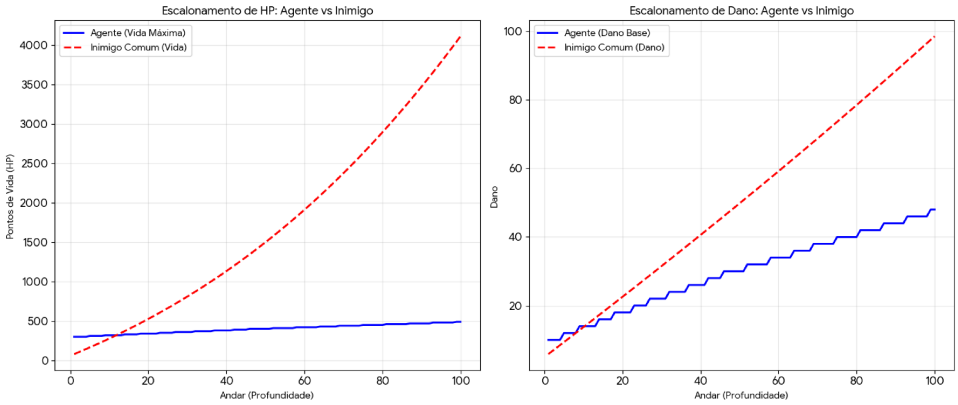
Nesta formulação, V_{max} representa o valor teto do estágio, k_{slope} a inclinação da curva $(0,015)$ e f_0 o ponto de inflexão (200).

Adicionalmente, aplica-se uma taxa de dano fixo por andar ($D_{flat} = d \cdot 0.8$) para limitar a viabilidade de políticas baseadas exclusivamente em mitigação de dano.

Divergência de Parâmetros A disparidade entre a progressão do agente e o escalonamento procedural dos agentes adversariais é uma característica projetada para induzir o desenvolvimento de padrões de decisão multivariados. Enquanto os atributos base do agente apresentam crescimento linear (+10 HP e +2 Dano por nível), os agentes adversariais seguem a curva logística de crescimento.

Como indicado na Figura 3, a partir do andar 40, os parâmetros de HP e dano dos agentes adversariais divergem da capacidade nominal do agente. Esta disparidade caracteriza um limiar de divergência operacional: a progressão além deste patamar requer a otimização do uso itens modificadores de estado e a utilização da memória recorrente para a antecipação de riscos. Neste cenário, a política do agente deve considerar a gestão do alinhamento κ, visto que a probabilidade de sucesso em confrontos diretos apresenta tendência decrescente conforme a evolução dos andares.

Figura 3 – Comparativo entre o escalonamento dos atributos do agente e dos agentes adversariais ao longo dos andares



Fonte: Elaborado pelo autor (2025).

3.1.2.2 Economia de itens e otimização de inventário

O processo de otimização em *BuriedBrains* ocorre por meio da definição de perfis de atributos, na qual são balanceados atributos ofensivos e utilitários. O sistema baseia-se em quatro níveis de raridade normalizados: *Common* (0,25), *Rare* (0,50), *Epic* (0,75) e *Legendary* (1,0). Esses valores operam como multiplicadores para os atributos passivos e para o desempenho das

interações mecânicas. O inventário é segmentado em quatro categorias funcionais:

1. **Equipamentos de combate (*Weapon* e *Armor*):** Estabelecem o perfil operacional do agente, variando entre especializações focadas em dano por segundo (*Damage Per Second - DPS*) ou em mitigação/absorção e defesa passiva. Itens de raridade superior, como a *Sword of the Forgotten King*, introduzem modificadores percentuais que permitem mitigar o escalonamento de atributos dos agentes adversariais em fases avançadas do episódio.
2. **Artefatos e itens de interação Social:** Além de bônus utilitários, como a ressurreição via *Phoenix Feather*, os itens são utilizados como meio de troca no Santuário. O descarte desses itens para sinalizar intenções não agressivas configura uma dinâmica de **Sinalização Custosa**. Nesse cenário, a política do agente opera o descarte de um bônus passivo para reduzir a probabilidade de dinâmicas de soma zero.
3. **Módulos de habilidade (*Skill Tomes*):** Possibilitam a otimização do conjunto de ações ativas (espaços 0–3). A seleção impõe uma análise de custo de oportunidade entre o uso de habilidades de baixo consumo de recursos e tempo de recarga reduzido, ou habilidades de alto impacto, como o *Meteor Strike*, que demandam controle estrito de *cooldown*. São aprendidas em ordem fixa (ex.: na primeira vez substitui a habilidade 0; na segunda, a habilidade 2; exceto texttt'Wait').
4. **Consumíveis:** Oferecem modificadores de atributos imediatos e temporários em uma situação de combate. Uso único.

O sistema decrementa a recarga de uma habilidade fora de combate através de ações de movimento (e.g., equipar um item ou se mover para outra sala).

A variedade de mecânicas passivas e gatilhos condicionais favorece a otimização de políticas que maximizem sinergias (e.g., priorizar mitigação percentual se o HP base for baixo). A Tabela 4 resume os impactos dessas categorias no modelo de sobrevivência.

Tabela 4 – Tipologia de efeitos e mecânicas de equipamentos e consumíveis

Grupo	Impacto no Estado	Exemplo (Raridade)
Ofensiva	Modificadores passivos de atributos e multiplicadores de dano final.	*Ragnar's Wrath* (Lendário)
Defensiva	Redução percentual de dano e aumento de vida máxima (HP).	*Armor of Eternity* (Lendário)
Controle	Gatilhos que aplicam atordoamento (*Stun*) ou dano contínuo (*DoT* (*Poison/Burn*)).	*Shadowfang* (Lendário)
Suporte	Cura imediata e modificadores temporários de atributos (*buffs*).	*Elixir of Rejuv.* (Épico)
Utilitário	Reflexão de dano.	*Mirrorplate* (Épico)

Fonte: Elaborado pelo autor (2025).

3.1.3 Espaço de Ação

O agente utiliza um espaço de ação discreto, definido pelo conjunto $\mathscr{A} = \{0, \ldots, 13\}$. No ambiente *BuriedBrains*, as ações mapeiam o conjunto de decisões discretas para transições de estado e interações no ambiente. A Tabela 5 detalha a função de cada dimensão desse espaço.

Tabela 5 – Definição funcional do espaço de ações (14 dimensões)

Índice	Mnemônico	Descrição Funcional
0 – 3	`SKILL_1-4`	Execução de habilidades. No Santuário, o uso de ações ofensivas contra outros agentes ativa o modo competitivo.
4	`INTERACT`	Equipamento de itens, aprendizado de habilidades ou realização de transações nos Santuários.
5 – 8	`DROP_ITEM`	Descarte de Arma, Armadura, Artefato ou Consumível no Santuário.
9 – 12	`MOVE_0-3`	Transição entre vértices adjacentes no grafo atual.
13	`CONSUMABLE`	Utilização de itens consumíveis que aplicam modificadores temporários ao vetor de estado do agente.

Fonte: Elaborado pelo autor (2025).

3.1.4 Modelagem da observação parcial (POMDP)

No ambiente *BuriedBrains*, a interação do agente com o ambiente é modelada como um *Processo de Decisão de Markov Parcialmente Observável* (POMDP). A cada passo temporal, o sistema fornece uma observação local $o_t \in \mathbb{R}^{198}$. O processamento da entrada, caracterizada pela heterogeneidade e pela pela estrutura multidimensional do vetor de entrada pela estrutura multidimensional do vetor de entrada, é realizado pelo `AttentionFeatureExtractor`, um extrator de características fundamentado em mecanismos de *Self-Attention*.

A arquitetura mapeia segmentos da observação para 11 tokens de representação latente (embeddings) de dimensão 96. Essa configuração permite que o modelo pondere a relevância mútua entre diferentes segmentos do vetor de entrada. A composição da observação está estruturada nos seguintes blocos:

1. **Estado das habilidades (índices 0-39):** Codifica 4 *slots* de habilidades, com cada *slot* representado em 10 dimensões. O `SkillEncoder` extrai variáveis de dano, tempo de recarga (*cooldown*) normalizado, magnitude de efeitos e categorias *one-hot* (Dano, Cura, *Buff*, *Debuff*, *Stun* e *DoT*).
2. **Estado interno (40-42):** Contém a razão atual de HP (*Health Points*), o nível do agente e o progresso de experiência (XP) em relação ao próximo nível.
3. **Contexto PvE e coleta (43-49):** Descreve o estado de combate, a presença de elementos

interativos no ambiente (como fontes ou baús), o HP relativo do agente adversarial e o valor de raridade do item disponível para coleta.

4. **Variáveis de interação multiagente (50-56):** Bloco relativo à Zona $\mathscr{K}$ (Santuário). Informa a localização na arena, a presença de outros agentes, disparidade de níveis e a representação numérica da reputação (*Karma*) do agente adversarial.
5. **Navegação e topologia (57-121):** Codifica o nó atual e os quatro nós adjacentes (indexados cardinalmente de 1 a 4). Cada nó possui 13 atributos, incluindo indicadores de acessibilidade, heurística de periculosidade relativa e efeitos ambientaisos quatro nós adjacentes (indexados cardinalmente de 1 a 4). Cada nó possui 13 atributos, incluindo indicadores de acessibilidade, heurística de periculosidade relativa e efeitos ambientais (como *Heat*, *Fog* ou *Evasion Zone*) processados pelo `EffectEncoder`.
6. **Metadados de equipamento e sinais (122-129):** Contém a raridade dos itens equipados e identifica sinalizações de agentes adversariais no Santuário, como o descarte de itens ou indicadores binários de neutralidade comportamental.
7. **Codificação de inventário (130-197):** O `ItemEncoder` processa as características dos quatro equipamentos ativos (Arma, Armadura, Artefato e Consumível). O vetor de 17 dimensões por item inclui bônus passivos normalizados e o mapeamento de efeitos *on-hit*, a exemplo de *Poison* ou *Bleed*.

Esta estruturação objetiva compor uma representação de estado suficiente para a otimização de políticas de longo prazo, contrastando com domínios focados em respostas reativas de curto prazo.

3.1.5 Modelagem de recompensas e incentivos

A política de um agente é otimizada maximizando o retorno acumulado R_t. Para balizar a convergência do modelo em relação às dinâmicas de exploração e interação, o ambiente utiliza uma função de recompensa composta por múltiplos componentes, conforme descrito na Tabela 6.

A estrutura de recompensas foi projetada para mitigar a morosidade na convergência comumente associada a sinais esparsos em ambientes de alta dimensionalidade. Além dos reforços terminais de vitória ou derrota, o sistema incorpora sinais intermediários baseados na exploração de novas áreas e penalidades fixas por intervalo de tempo (*time-step*). O objetivo dessa implementação é reduzir a convergência para ótimos locais de estagnação em estados de

recompensa nula durante um episódio.

Caso a vida do agente (HP) atinja o limiar $HP \leq 0$, o método `_respawn_agent` é executado, instanciando o agente em um novo grafo de progressão. Nesse processo, os atributos de estado são redefinidos para os valores iniciais, com exceção da variável de reputação, conforme define o método `create_initial_agent` em `progression.py`. Este parâmetro permanece persistente até a execução de um reinício global da simulação pelo *Gym*, que redefine todos os agentes simultaneamente. A duração de um episódio global é definida pela quantidade de passos individual por agente `max_episode_steps` em `train.py`.

Quanto a reputação como atributo permanente, tal escolha de projeto tem por finalidade atenuar dificuldades na atribuição de crédito. Dado que as interações entre agentes ocorrem com baixa frequência, o sinal resultante apresenta alta esparsidade. A persistência do dado permite, portanto, a acumulação de informações ao longo de múltiplos ciclos de vida do agente no ambiente.

A estruturação da função de recompensa $\mathscr{R}(s,a)$ é apresentada na Tabela 6. O design utiliza a técnica de *reward shaping* para mitigar o problema de recompensas esparsas, atribuindo valores intermediários a transições de estado que precedem o objetivo terminal.

Tabela 6 – Função de Recompensa ($\mathscr{R}(s,a)$) e estrutura de incentivos

Evento / Condição	ΔR	**Lógica de Operação (*Reward Shaping*)**
Existência e navegação		
Custo de Existência (PvE)	$-0{,}5$	Penalidade por *time step* para atenuar políticas de estagnação em combate.
Custo de Existência (Santuário)	$-0{,}1$	Penalidade inicial para mitigar políticas de estagnação no Santuário.
Transição de Estado (Sucesso)	$+5{,}0$	Estímulo à transição entre vértices adjacentes e à exploração da topologia.
Descoberta de Vértice Inédito	$+0{,}5$	Recompensa por exploração de novas salas no grafo.
Movimento Redundante	$-1{,}5$	Penalidade por ciclos imediatos entre dois nós no Santuário.
Ação Inválida / Saque Ineficaz	$-5{,}0$ a $-10{,}0$	Desincentivo a ações inválidas ou tentativas de saque sem alvo local.
Estados terminais e progressão		
Interrupção da Trajetória (Morte)	-300	Penalidade terminal para desincentivar a reinicialização de atributos.

Continua na próxima página

Evento / Condição	ΔR	**Lógica de Operação (*Reward Shaping*)**
Alcance de Marco (*Milestone*)	$+100$	Bônus por profundidade no grafo a cada 10 andares alcançados.
Início de Confronto PvE	$+10{,}0$	Incentivo ao engajamento em combate ao detectar um agente adversarial ambiental.
Combate		
Dano Líquido Causado	$+0{,}6$/hp	Recompensa densa vinculada à investida ofensiva.
Dano Líquido Sofrido	$-0{,}5$/hp	Penalidade densa para incentivar mitigação de dano.
Eliminação de Inimigo	$+100$	Incentivo por remoção de ameaça ambiental.
Incremento de Nível	$+50{,}0$	Reforço para evolução do agente.
Gestão de inventário		
Uso de Poção de Cura	$+10{,}0 + 0{,}1\Delta$	Recompensa proporcional ao HP restaurado.
Consumo de Elixir	$+20{,}0$	Bônus fixo para restauração completa.
Ativação de Buffs	$+5{,}0$ a $+50{,}0$	Incentivo ao uso de modificadores de atributos.
Uso de Item Redundante	$-2{,}0$	Penalidade por uso sem alteração de estado.
Falha no Uso de Item	$-5{,}0$	Penalidade por tentativa inválida.
Aquisição de Habilidade	$+100$	Incentivo à expansão do espaço de ações.
Substituição de Equipamento	$+75 + 100\Delta_v$	Incentivo à melhoria de raridade.
Interação entre agentes (Santuário)		
Troca de Itens	$+200$	Incentivo à cooperação.
Encontro de Oponente	$+100$	Recompensa por co-localização.
Saída Cooperativa	$+100$	Bônus na ausência de agressão.
Ruptura de Cooperação	-100	Penalidade por defecção.
Sinalização de Estado	$-2{,}0$	Custo para evitar saturação.
Vitória / Derrota PvP	$+200/-300$	Recompensa em confrontos diretos.
Vitória em Desvantagem	± 50 a 100	Incentivo a risco estratégico.
Vitória Predatória	-50 por dif. nível	Desincentivo à predação.
Penalidade por estagnação		

Continua na próxima página

Evento / Condição	ΔR	**Lógica de Operação (*Reward Shaping*)**
Pressão do Santuário	$-0{,}1$ a -10	Penalidade temporal para evitar estagnação.

Fonte: Elaborado pelo autor (2025).

A recompensa de troca de itens é mediada por uma variável de estado que mapeia itens já transacionados na zona atual, impossibilitando ciclos de trocas redundantes.

3.2 Engenharia e arquitetura do simulador

O *BuriedBrains* emprega o princípio de Separação de Preocupações (*Separation of Concerns*). Esta organização desacopla as regras do ambiente — como combate, economia e geração procedural — da interface de aprendizado por reforço.

Os scripts detalhados neste capítulo estão disponíveis no repositório do projeto (SILVA, 2025) na versão 2.3.2.

3.2.1 Visão geral da arquitetura (Hub-and-Spoke)

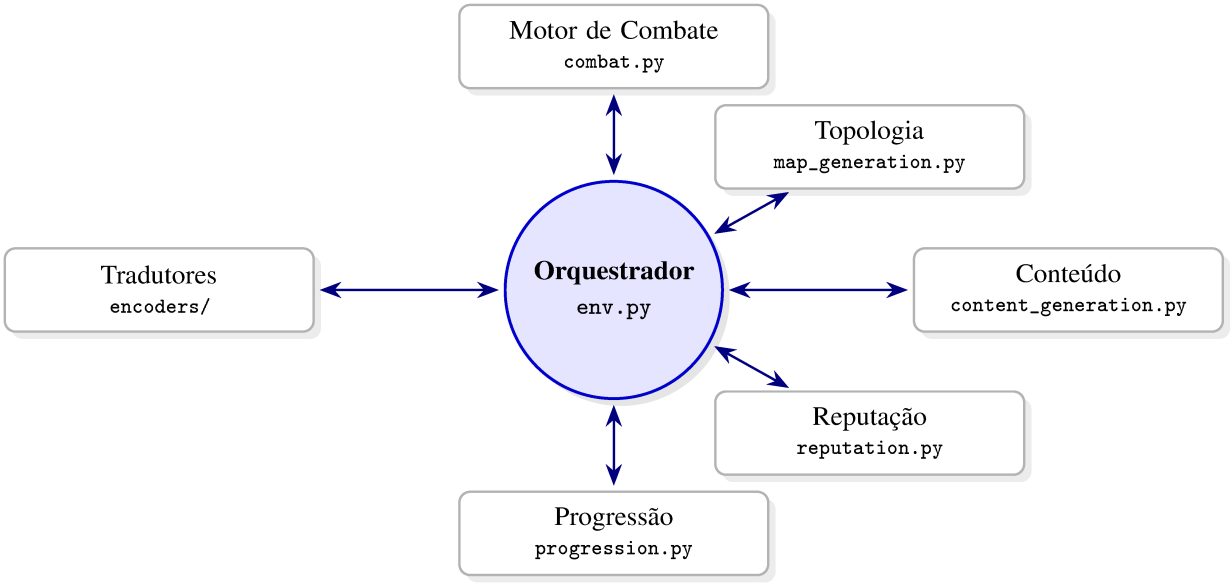
O simulador opera sob um modelo *Hub-and-Spoke*. O módulo `env.py` atua como o componente central (*Hub*), responsável pela gestão do estado global e pela comunicação com o algoritmo de aprendizagem. Os demais módulos atuam como componentes periféricos (*Spokes*) especializados em regras de domínio específicas.

Nesta topologia, o núcleo processa as transições de estado do episódio. O processamento matemático e as rotinas algorítmicas de suporte são executados pelos módulos periféricos, o que resulta na descentralização das responsabilidades.

- **Gestão de fluxo:** O *Hub* recebe transições de estado, processa a ação do agente e invoca o módulo competente.
- **Desacoplamento técnico:** Alterações na lógica de dano no `combat.py` ou na curva de progressão de níveis no `progression.py` não exigem modificações na estrutura da interface *Gymnasium*.
- **Mapeamento de dados:** Os *Encoders* atuam como camadas de tradução, convertendo dados brutos dos catálogos em vetores numéricos que o *Hub* agrega na observação final.

A Figura 4 ilustra essa interdependência modular.

Figura 4 – Representação da arquitetura modular *Hub-and-Spoke* do simulador



Fonte: Elaborado pelo autor (2025).

3.2.2 Arquitetura de módulos funcionais

A lógica de domínio foi segmentada em módulos independentes de forma a mitigar a concentração de responsabilidades no núcleo do sistema (*God Object*). A invocação desses componentes ocorre de acordo com o estado atual da aplicação:

- **Módulo de progressão** (`progression.py`)**:** Responsável pelo escalonamento dos atributos de agentes adversariais via curvas logísticas, implementadas na função `logistic_scale`. Este componente processa o acúmulo de experiência e a atualização dos atributos do agente durante o avanço de nível.
- **Módulo de combate** (`combat.py`)**:** Opera como uma máquina de estados discreta voltada ao processamento da lógica de turnos. A unidade isola o cálculo de acertos, a mitigação de dano e a resolução de turnos de combate.
- **Geração procedural de conteúdo** (`content_generation.py`)**:** Baseia-se no modelo de orçamento por sala (*Room Budget*). A partir do andar atual, o módulo gera o conteúdo de uma sala, visando a distribuição estocástica dos elementos.
- **Topologia e navegação** (`map_generation.py`)**:** Emprega a biblioteca *NetworkX* para a estruturação do ambiente. O sistema gera grafos acíclicos dirigidos (DAGs) para os percursos PvE e aplica modelos de Erdős-Rényi na definição das arenas destinadas ao cenário de soma não zero.

3.2.3 Codificação de atributos e inventário (Encoders)

Os codificadores convertem dados semânticos e estruturados em tensores densos. Atuam como interface entre o estado do simulador e a rede neural. Esta camada processa a informação de forma normalizada, mitigando a sensibilidade a alterações no catálogo de dados

O módulo `item_encoder.py` mapeia atributos de equipamentos em vetores numéricos de dimensão fixa ($\mathbb{R}^{17}$). A implementação utiliza um mecanismo de calibragem prévia ao treinamento. Essa estrutura possibilita o processamento de expansões no catálogo sem a necessidade de ajustes manuais em parâmetros de normalização.

3.2.3.1 Codificação automática de itens

O método `_calibrate` percorre o catálogo global de equipamentos em formato *YAML* antes do início do treinamento. O procedimento extrai os valores máximos (*ceilings*) referentes a quatro atributos passivos:

- `flat_damage_bonus` e `damage_modifier`;
- `flat_hp_bonus` e `damage_reduction`.

Estes valores são armazenados no dicionário `self.max_stats`. No método `encode`, tais variáveis operam como denominadores para o mapeamento dos atributos estatísticos ao intervalo $[0, 1]$. Essa normalização visa mitigar desequilíbrio na magnitude dos gradientes com ordens de grandeza elevadas, como pontos de vida (HP), sobre o gradiente da rede neural em relação a atributos de menor escala.

3.2.3.2 Mapeamento da proprioecepção do agente

O vetor de saída ($\mathbb{R}^{17}$) é estruturado em três blocos funcionais que representam as propriedades técnicas e os efeitos do item:

1. **Atributos escalares (5 dimensões):** Contém os quatro atributos estatísticos normalizados e a probabilidade de ativação (*chance*) de efeitos secundários (*on-hit* ou *on-hurt*).
2. **Categoria (3 dimensões):** Utiliza codificação *one-hot* para distinguir as categorias *weapon, armor* e *artifact*.
3. **Indicadores de efeito (9 dimensões):** Mapeia a presença de efeitos específicos via codificação multi-hot: *stun, burn, poison, bleed, reflect, revive, vulnerable, blind* e *sunder*.

Essa estrutura fornece ao modelo dados sobre a atributos que caracterizam a semân-

tica operacional de cada item. Esta decomposição tem por objetivo permitir que a política de decisão associe sinalizações de efeitos específicos (como a presença de *revive*) de forma distinta dos valores brutos de atributos defensivos ou ofensivos.

3.2.3.3 Mapeamento de habilidades

O módulo `skill_encoder.py` converte as propriedades das habilidades em um vetor denso de 9 dimensões. Essa representação visa codificar o o tipo de dano, efeitos e a duração de cada habilidade configurada no catálogo.

Normalização via catálogo global

O `SkillEncoder` executa uma etapa de calibragem que processa o arquivo `skill_and_effects` no início da execução. O procedimento extrai os valores máximos para:

- **Dano base (`damage`):** Valor de referência para a normalização de habilidades ofensivas.
- **Tempo de recarga (`cd`):** Limite superior para o intervalo de ativação da técnica.

Durante a codificação, os atributos são normalizados por esses valores máximos. Esse processo normaliza habilidades de alta intensidade e baixa frequência para um intervalo de magnitude equivalente a ataques de baixa intensidade e alta frequência.

Extração de potência e efeito

O codificador extrai o parâmetro `potency` a partir do *ruleset* de efeitos. Nos casos em que o catálogo de habilidades define apenas a *tag* do efeito (e.g., *burn*, *life_drain*), o módulo recupera do dicionário de regras para obter os valores numéricos associados:

1. **Identificação de tags:** O módulo mapeia as categorias de efeito vinculadas à habilidade.
2. **Atribuição de valores:** O codificador acessa o `effect_ruleset` para extrair a magnitude (`potency`), como o percentual de recuperação de vida ou multiplicadores de dano contínuo, além da duração do efeito.
3. **Projeção:** Os valores extraídos são normalizados e inseridos nas dimensões do vetor. Essa estrutura fornece dados para que o modelo processe o impacto funcional de habilidades de controle (e.g., *stun*) mesmo na ausência de dano direto.

Representação das dimensões do encoder

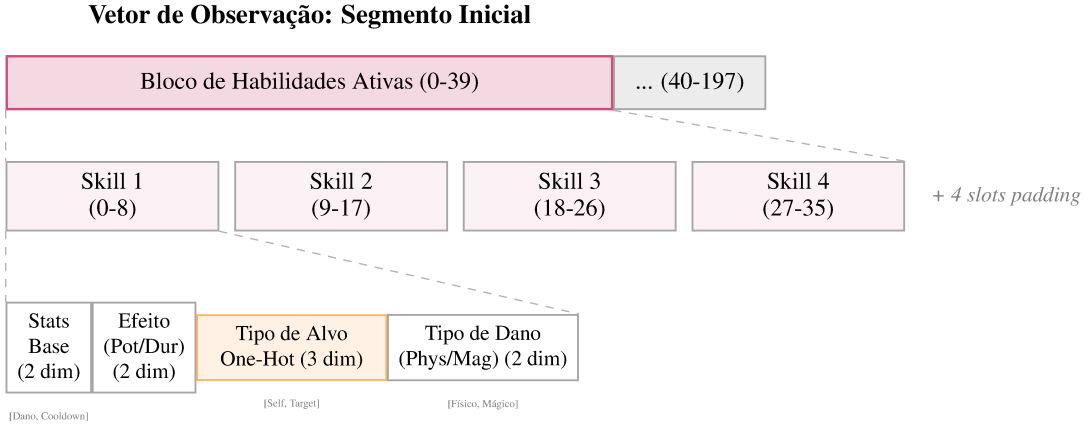
O vetor resultante ($\mathbb{R}^9$) encapsula as características funcionais da habilidade, sendo composto por: (1) dano normalizado; (2) *cooldown* base normalizado; (3) *potency* do efeito principal; (4) duração do efeito; (5-7) mapeamento *one-hot* do tipo de alvo (*self, target, AOE*); e (8-9) representação binária (multi-hot) de tipo de dano (*physical* e *magical*).

Essa padronização auxilia a convergência para políticas de seleção de ações cujas estimativas de valor associam o custo temporal à magnitude dos efeitos de cada habilidade.

Mapeamento de habilidades

O segmento inicial do vetor de observação (índices 0 a 39) armazena a representação do conjunto de habilidades do agente. Conforme indicado na Figura 5, o `SkillEncoder` processa cada técnica de forma individual. O módulo extrai metadados do arquivo `skill_and_effects.yaml` e os converte em um vetor de 9 dimensões que parametriza os custos, efeitos e categorias da ação.

Figura 5 – Estrutura do bloco de habilidades detalhando a segmentação das características de cada habilidade



Fonte: Elaborado pelo autor (2025).

3.2.3.4 Mapeamento do estado espacial e navegação

O módulo `effect_encoder.py` constitui um subcomponente da camada de percepção espacial. Este mapeia as condições de terreno em vetores *one-hot* de 9 dimensões. No módulo de interface do ambiente `env.py`, esses dados são agrupados em um bloco de 65 dimensões (índices 57 a 121 do vetor de observação). Esse conjunto parametriza o estado do nó atual e de até quatro nós adjacentes.

Estrutura do sub-bloco de sala

Cada nó visível é representado por um sub-bloco de 13 características. A organização dos dados segue a estrutura abaixo:

1. **Viabilidade (Índice +0):** Indica se o nó pertence ao grafo local ou se consiste em preenchimento (*padding*) para a manutenção de dimensões fixas no tensor.
2. **Indicador de ameaça (Índice +1):** Registra a presença de agentes adversariais não jogáveis (*Non-Playable-Characters* - NPCs) ou de agentes adversários.

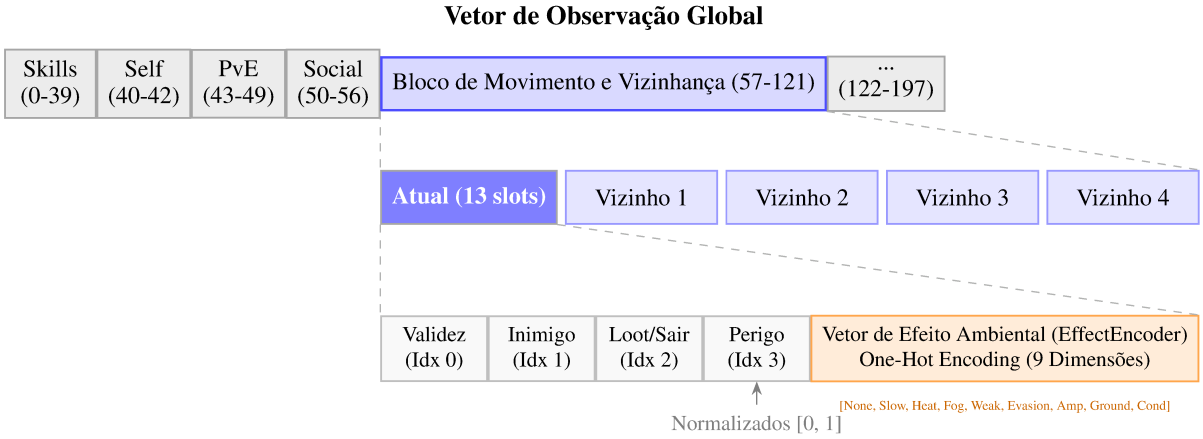
3. **Indicador de recompensa (Índice +2):** Sinaliza a existência de recursos, como itens, eventos de tesouro, fontes de recuperação ou a saída desbloqueada.
4. **Nível de perigo (*d_level*) (Índice +3):** Escala valores atribuídos via discretização normalizada: 1,0 para chefes (*bosses*); 0,66 para agentes adversariais de elite ou outros agentes; e 0,33 para agentes adversariais comuns.
5. **Vetor de efeito (Índice +4 a +12):** Composto pelas nove dimensões geradas pelo `EffectEncoder`, que especificam o modificador ambiental do nó.

Processamento de navegação e transições de Estado

A inclusão dos dados referentes aos nós adjacentes possibilita o processamento da relação entre custo e recompensa das transições de estado. A integração do indicador de perigo (*d_level*) com o vetor de efeito (e.g., *conductive field*) permite que a política de decisão incorpore a dependência funcional entre bônus de dano de categorias específicas de agentes adversariais a modificadores ambientais.

Essa organização dos dados favorece a estabilidade do treinamento do modelo ao fornecer variáveis para o mapeamento de padrões espaciais pela rede neural. Ao processar simultaneamente as métricas de perigo e os efeitos ambientais dos nós adjacentes e do nó atual, o sistema pode otimizar a política de navegação.

Figura 6 – Estrutura hierárquica do vetor de observação



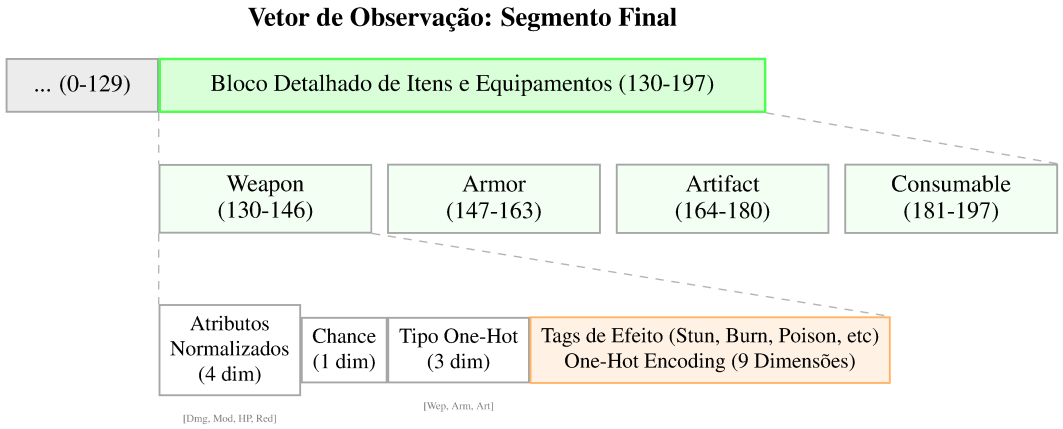
Fonte: Elaborado pelo autor (2025).

3.2.3.5 Mapeamento de itens e habilidades

Enquanto o bloco de movimento codifica informações de topologia, os blocos de equipamentos (índices 130 a 197) e habilidades (índices 0 a 39) armazenam as variáveis de estado do agente.

A Figura 7 detalha a estrutura do bloco de equipamentos, evidenciando como o `ItemEncoder` decompõe um item em uma assinatura digital única de 17 dimensões.

Figura 7 – Decomposição do bloco de equipamentos



Fonte: Elaborado pelo autor (2025).

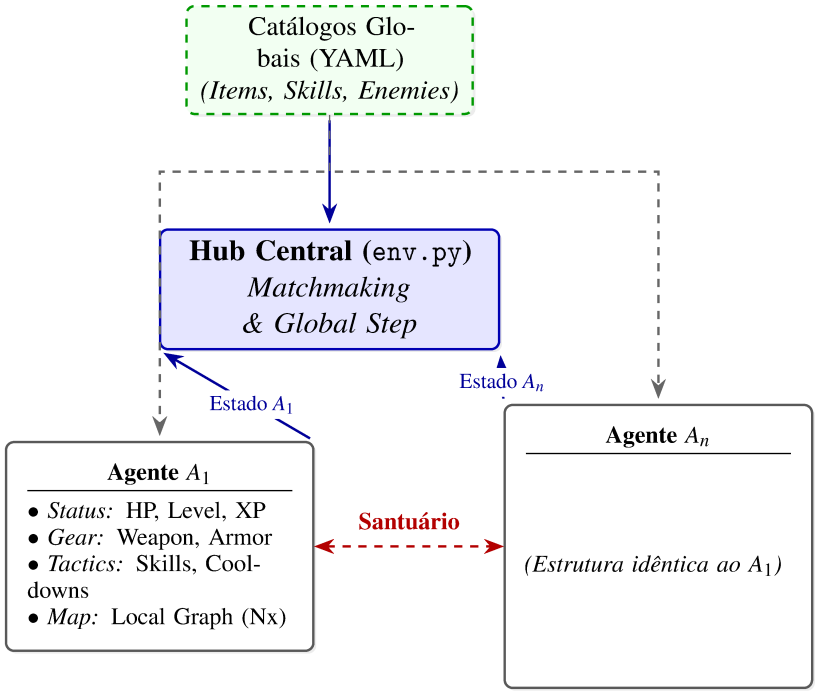
3.2.4 Gerenciamento de estado e persistência multiagente

O sistema adota uma estrutura de particionamento de estado, na qual o dicionário `agent_states` atua como o registro central de dados. A indexação das informações é realizada por meio do identificador único `agent_id`. Este modelo de organização possibilita a execução simultânea de múltiplos agentes em instâncias paralela e mitigar a ocorrência de efeitos colaterais decorrentes de vazamento de estados ou disputa por recursos durante a progressão.

O isolamento lógico impede que operações executadas por um Agente *A* interfiram na topologia ou nos recursos disponíveis para um Agente *B*. Ações como a coleta de itens ou a poda de nós do grafo (*node pruning*) permanecem restritas ao escopo do agente individual, dado que cada grafo de progressão é exclusivo de um agente por vez. Dessa forma, a integridade da exploração individual é mantida até a etapa de pareamento contextual no Santuário.

A Figura 8 ilustra a taxonomia e a organização desta persistência distribuída.

Figura 8 – Diagrama de interação Agente-Ambiente com estados isolados e orquestração pelo Hub Central



Fonte: Elaborado pelo autor (2025).

O estado de cada agente é estruturado em quatro camadas de persistência:

- **Atributos do agente:** Armazena as variáveis do vetor de estado do agente. Os dados são processados pelo módulo `progression.py` para o cálculo de incrementos de atributos por meio de curvas de crescimento logístico associadas ao avanço de nível.
- **Estrutura do inventário:** Mapeia os equipamentos alocados nas categorias de *Weapon, Armor, Artifact* e *Consumable*. A persistência é integrada ao componente `ItemEncoder`, que executa a normalização de bônus passivos e efeitos *on-hit* no início de cada passo.
- **Navegação local:** Cada agente mantém uma instância própria de `nx.DiGraph` no dicionário `graphs`. Durante o deslocamento em uma zona de progressão, o simulador aplica a poda de nós (*Node Pruning*), removendo sucessores não visitados para reduzir a alocação de memória.
- **Tempos de recarga de habilidades (*Cooldowns*):** O vetor de estados do agente mantém um registro de tempos de recarga para as habilidades disponíveis. O valor é decrementado tanto em ações de combate quanto em turnos de movimentação.

Esta estrutura de persistência é o que possibilita a extração de métricas de telemetria individuais para integração com os *callbacks* de monitoramento via *TensorBoard*.

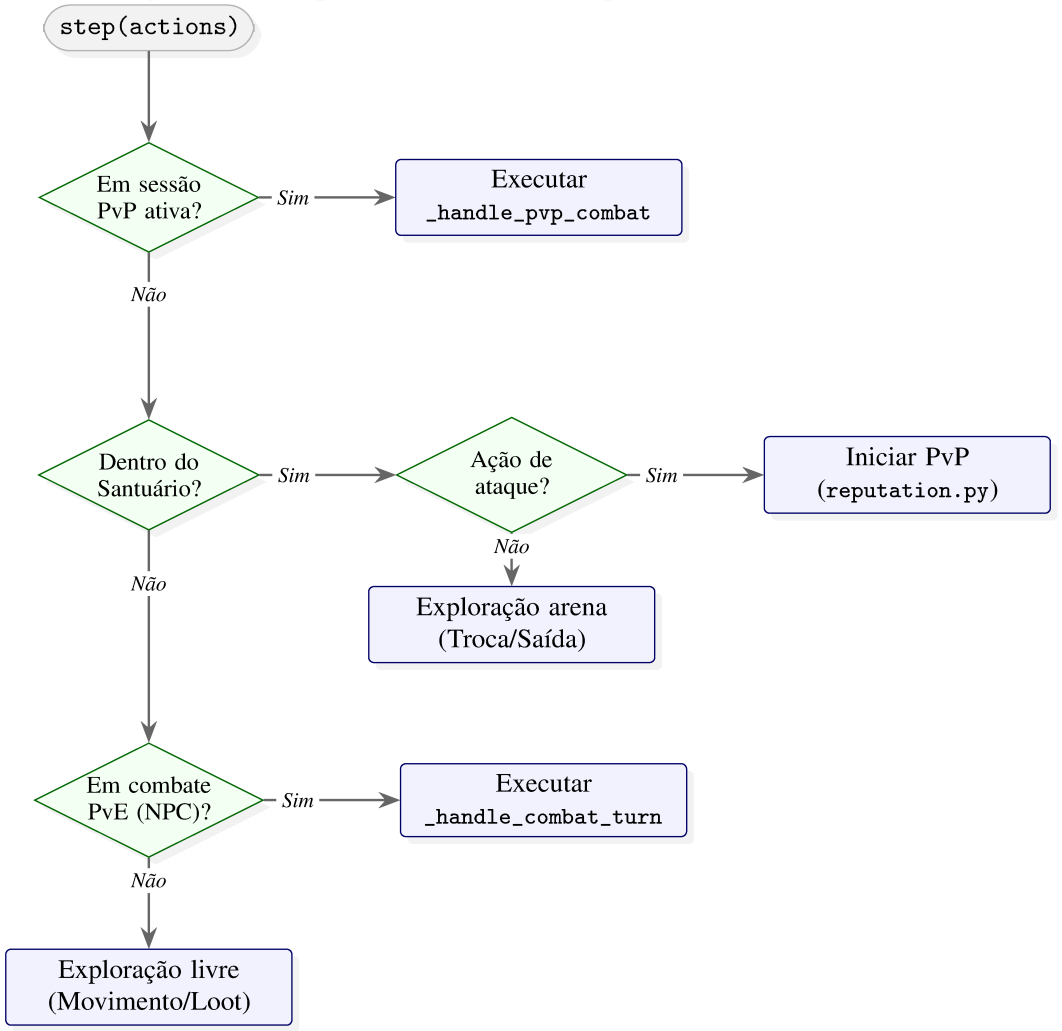
3.2.5 Ciclo de operação e máquina de estados (`step`)

A função `step()` constitui a lógica operacional do simulador. Diferente de ambientes de aprendizado por reforço com regras estáticas, o *BuriedBrains* alterna entre três estados operativos, associadas por agente:

- **Exploração:** Estado padrão de progressão. O orquestrador monitora a navegação no grafo e invoca o módulo `content_generation.py` para geração de conteúdo em salas visitadas.
- **Combate:** Iniciado após o encontro com um agente adversarial ambiental. Neste estado, as ações de movimentação são suspensas e o fluxo é transferido ao módulo `combat.py`. O processamento de turnos ocorre de forma atômica até a resolução do confronto.
- **Santuário:** Ocorre em marcos de progressão pré-definidos pela variável `sanctum_floor`. O agente é realocado para uma instância de Santuário compartilhada. Neste contexto, são avaliadas regras de interação multiagente, como transação de itens ou ações hostis.

O sistema prioriza sessões de conflito ativo (`pvp_sessions`) sobre o estado de exploração. Esta hierarquia tem por objetivo a manutenção da sincronia entre instâncias em execuções paralelas utilizando múltiplos núcleos (`num_cores`). A estrutura de decisão percorrida em cada passo de tempo é detalhada na Figura 9:

Figura 9 – Fluxograma da máquina de estados do step



Fonte: Elaborado pelo autor (2025).

3.2.6 Módulos da interface do ambiente

Esta subseção descreve a implementação das regras que definem a simulação física e as interações táticas do ambiente.

3.2.6.1 Balanceamento de dificuldade

O módulo `progression.py` executa as funções matemáticas definidas na 3.1.2. A modelagem estabelece o equilíbrio entre o agente e o ambiente.

Implementação do escalonamento

O cálculo dos incrementos de atributos por nível e do multiplicador logístico (Λ_{floor}) baseia-se nas definições da subseção 3.1.2. O simulador aplica o método `logistic_scale` para atualizar o dicionário de estados dos agentes adversariais a cada nova instância de andar.

Chaos Mode e Floor Tax

Ao ultrapassar o limite crítico de exploração, o simulador ativa mecânicas de pressão para encerrar trajetórias estagnadas:

1. **Chaos Mode (andar > 500):** Após o andar 500 (`CHAOS_FLOOR_THRESHOLD`), o escalonamento logístico é substituído por um crescimento **exponencial puro**. Os atributos dos agentes adversariais passam a ser multiplicados por um fator de $1,035^{(d-500)}$, tornando a sobrevivência matematicamente insustentável em curtos intervalos de tempo.
2. **Floor Tax (dano fixo):** Para evitar que agentes com alta defesa ignorem o dano de agentes adversariais fracos, o sistema aplica um **Dano Fixo de Piso** em cada ataque recebido:

$$D_{flat} = d \times 0,8 \tag{3.4}$$

Este valor é somado ao dano final e ignora parcialmente as reduções de armadura, garantindo que o ambiente sempre exerça uma força de desgaste (*attrition*) proporcional à profundidade alcançada.

3.2.6.2 Sistema de combate e resolução de turnos

O módulo `combat.py` processa a resolução de confrontos entre agentes e agentes adversariais controlados pelo ambiente ou entre múltiplos agentes. O componente opera como uma máquina de estados que isola a lógica de conflito da navegação pelo grafo. Para preservar a integridade do estado mestre, o motor utiliza estruturas de dados temporárias geradas pela função `initialize_combatant`. Esta função instancia um perfil de atributos transientes, consolidando os valores base do agente, bônus de equipamentos e modificadores ambientais da sala atual.

Combate e resolução de turnos

A resolução de combate no *BuriedBrains* foi concebida sobre o modelo de combate turno a turno. A dinâmica fundamenta-se na ordenação de habilidades (*cooldowns*) e no controle de mitigação de danos em cenários de escalonamento de atributos. O ambiente apresenta variáveis estocásticas e modificadores de cenário que condicionam a otimização da política de seleção de ações do agente a cada encontro. A sequência de processamento de um turno segue a ordem estabelecida:

1. **Modificadores de ambiente e Efeitos Passivos:** O sistema aplica modificadores de terreno, como *Slow Terrain* ou *Heat*, antes da ação do agente. Nesta etapa, processam-se danos contínuos (*Damage over Time* - DoT), regeneração (*Heal over Time* - HoT) e a

verificação do estado do agente (HP). Mecanismos de ressurreição (*Revive*) são acionados se o HP atingir o valor zero durante esta fase.

2. **Turnos negados e recarga (*Cooldowns*):** O sistema identifica estados de restrição de controle, como o *Stun*. Independentemente da execução de uma ação ativa no turno, os tempos de recarga de todas as habilidades são decrementados. Essa mecânica impõe uma penalidade temporal que influencia a eficiência da função objetivo do agente.
3. **Uso de consumíveis:** O agente pode utilizar itens de efeito imediatos. O uso desses recursos permite a restauração de HP ou a aplicação de incrementos temporários de atributos (*buffs*), auxiliando na compensação do aumento de poder dos agentes adversariais.
4. **Cálculo de ação e dano:** A probabilidade de acerto, $P(hit)$, é definida pela relação entre Precisão e Evasão, operando no intervalo $[5\%, 95\%]$. O dano final é determinado por uma função multiplicativa que integra o dano base da habilidade, modificadores de equipamento e a redução de dano (*Armor*) do alvo, sendo esta última limitada a um teto de 70%.

Para mitigar a dominância de configurações focadas exclusivamente em defesa, o ambiente aplica o *Floor Tax*. Esta mecânica consiste em um bônus passivo de dano de agentes adversariais oriundo da profundidade (f), com incremento linear. O dispositivo atua como uma restrição ambiental que induz o agente a otimizar a velocidade de resolução dos confrontos em vez de priorizar apenas a resistência passiva.

Fluxo de resolução de turnos

A resolução de cada turno é centralizada na função `execute_action`. O método mapeia as ações selecionadas pelos agentes em alterações de estado numéricas, seguindo um fluxo de validações e cálculos estocásticos:

1. **validação de negação de turno e recarga:** Antes do processamento de dano, o sistema invoca a função `is_stunned` para verificar a existência de efeitos de controle ativos. Simultaneamente, valida-se se o tempo de recarga (*cooldown*) da habilidade é nulo. Caso uma das condições de impedimento seja detectada, a execução é interrompida e registrada em log.
2. **Resolução de acerto e evasão:** A probabilidade de sucesso da ação decorre da relação entre a precisão (*accuracy*) do atacante e a capacidade de evasão (*evasion*) do alvo. Visando a manutenção da estocasticidade e a continuidade do aprendizado no modelo MARL, aplicam-se limites operacionais (*hard caps*):
    - **Precisão mínima (`CAP_MIN_ACCURACY`):** Fixada em 10%, visando impedir a impos-

sibilidade estatística de acerto.

- **Esquiva máxima (`CAP_MAX_EVASION`):** Limitada a 75%, para mitigar estados de invulnerabilidade durante o treinamento.

3. **Cálculo de dano:** O valor bruto (D_{total}) é processado por uma cadeia de multiplicadores que integram a base da habilidade, atributos e modificadores. A determinação do valor segue a Equação 3.5:

$$D_{\text{total}} = (V_{\text{base}} + \text{Atrib}_{\text{bonus}}) \times \text{Mod}_{\text{global}} \times \text{Mod}_{\text{tipo}} \times \text{Crit} \tag{3.5}$$

Onde:

V_{base}**:** Valor nominal da habilidade extraído do catálogo `enemies_and_events.yaml`.

$\text{Atrib}_{\text{bonus}}$**:** Soma do dano estatístico do agente com bônus fixos de itens.

$\text{Mod}_{\text{global}}$**:** Modificador escalar aplicado ao atacante por efeitos passivos ou consumíveis.

Mod_{tipo}**:** Fator de especialização (físico ou mágico) sensível às propriedades da sala.

Crit: Coeficiente de acerto crítico, definido por `CRIT_MULTIPLIER_DEFAULT`.

4. **Mitigação e dano efetivo:** O valor D_{total} é ajustado pela redução defensiva do alvo para produzir o dano final (D_{final}). O sistema impõe um teto de mitigação de 70% (`CAP_MAX_DMG_REDUCTION`). A aplicação da regra `MIN_DEFENSE_MULTIPLIER` estabelece que o alvo receba ao menos 10% do dano original, visando a convergência do episódio e evitando a estagnação do estado de combate.

Efeitos persistentes

Após a resolução do dano direto, o motor executa a fase de manutenção através da função `resolve_turn_effects_and_cooldowns`. Essa fase é crítica para a profundidade estratégica do ambiente MARL, lidando com:

- **Gatilhos de reação (*On-Hit / On-Being-Hit*):** O sistema processa efeitos automáticos vinculados a itens, como o *Reflect* (devolve parte do dano ao atacante) ou a aplicação de *debuffs* condicionais.
- **Estados temporais (DoTs/HoTs):** São aplicados danos ou curas contínuas (*Burn, Poison, Regeneration*) baseados na *potency* do efeito, seguidos pelo decremento da duração dos *status* ativos.
- **Ressurreição:** O motor verifica a tag *Revive* em equipamentos. Se ativa, impede a morte imediata do combatente, restaurando uma porcentagem de HP (padrão de 50%) e consumindo o uso do efeito para o restante do combate.

Consumíveis e Reward Shaping

O uso de consumíveis é processado pelo método `use_consumable`, que atribui recompensa de acordo com o uso para reforço positivo explícito (*Reward Shaping*). O agente recebe *feedback* positivo proporcional à eficiência da cura (evitando recompensas por cura desperdiçada em HP cheio) e bônus por ativação para consumíveis que modificam temporariamente atributos, como o *God Mode*, que eleva temporariamente todos os atributos defensivos e ofensivos a 2x.

Limites estritos de atributos Para garantir a estabilidade numérica e o desafio em níveis avançados, o sistema impõe limites rígidos (*Hard Caps*) aos atributos defensivos. A Redução de Dano (δ) é limitada a 70%, impedindo a invulnerabilidade passiva do agente.

A magnitude do dano desferido por um agente adversarial no andar f, denotada por D_{final}, é definida pela composição do dano base escalado (D_{base}) e a penalidade ambiental (D_{floor}), sujeita a uma variância estocástica η:

$$D_{final} = \lfloor (D_{base} + (f \cdot \phi \cdot \tau)) \cdot \eta \rfloor, \quad \eta \sim \mathscr{U}(1-\zeta, 1+\zeta) \tag{3.6}$$

Onde $\phi = 0,8$ representa o coeficiente do *Floor Tax*, $\tau \in \{1,0; 1,2; 1,5\}$ é o fator de escalonamento por *Tier* (Comum, Elite ou Boss) e $\eta \sim \mathscr{U}(1-\varepsilon, 1+\varepsilon)$ introduz uma variância definida pelo *tier* do agente adversarial. Esta definição se aplica somente aos agentes adversariais gerados nas zonas de progressão.

Arquétipos emergentes A configuração das restrições matemáticas em conjunto com o catálogo de itens favorece a emergência de três políticas predominantes:

- **Dano explosivo (*Burst/DPS*):** Baseia-se no acúmulo de bônus fixos (*Flat Damage*) e multiplicadores (*Damage Modifier*). O objetivo técnico é a minimização do tempo de combate. Associa-se ao uso de itens como: *Ragnar's Wrath* e *Sword of the Forgotten King*.
- **Controle e desgaste (*Control/DoT*):** Fundamenta-se na negação de turnos ao adversário (*Stun*) e na aplicação de efeitos de dano contínuo (*Poison* e *Burn*). Habilidades como *Poison Cloud* ou o item *Shadowfang* permitem a exploração dessa especialização emergente.
- **Sustentação (*Tank/Sustain*):** Nesta especialização, a política do agente converge para a seleção de incrementos incrementos fixos em defesa passiva, ou sustentação através de habilidades de cura como *Life Drain* ou *Healing Wave*. Esta estratégia pode ser menos comum pois apresenta dependências temporais complexas que dificultam a atribuição de crédito pela política em meio ao ruído de transições de estado nos modos de combate e navegação.

3.2.6.3 Geração de conteúdo procedural

Alocação e filtragem de candidatos

A geração de uma sala inicia-se com o cálculo do Orçamento Total de Sala (B_{total}), que define o orçamento do algoritmo para instanciar entidades. A fórmula utilizada combina um valor base estático com um escalonamento linear baseado na profundidade da exploração:

$$B_{total} = (B_{base} + (d \times Lambda(f))) \times M_{budget} \tag{3.7}$$

Onde:

B_{base}**:** Orçamento base fixado em 100.0 unidades (`BASE_BUDGET`).

f**:** O andar atual onde a sala está sendo gerada.

$Lambda(f)$**:** Fator de escalonamento (`FLOOR_SCALING`), definido em 10.0 unidades por andar.

M_{budget}**:** Multiplicador de intensidade, que permite ao orquestrador ajustar a dificuldade global do episódio.

Antes da seleção por peso (*Weighted Random Selection*). Cada entidade no catálogo possui o atributo `min_floor`. O algoritmo descarta automaticamente qualquer candidato cujo requisito de andar seja superior ao andar atual do agente ($d < min\_floor$), impedindo que agentes adversariais de *Tier* elevado (como Bosses e Elites) surjam em estágios iniciais e gerem encontros matematicamente impossíveis.

Distribuição de itens para saque

O sistema de distribuição de itens utiliza gatilhos para associar a recompensa ao nível de dificuldade imposta ao agente. Quando o motor de jogo instancia agentes adversariais de dificuldade elevada (*Elite* ou *Boss*), as probabilidades de recompensa (`WEIGHT_MULT`) sofrem ajuste contextual:

- **Morbid Treasure:** Ativado em salas de maior dificuldade, com o peso de probabilidade ampliado em 20 vezes. Este ajuste é o que possibilita a obtenção de itens das categorias *Epic* e *Legendary*.
- **Módulos de habilidade:** A obtenção de novos módulos de habilidade (*SkillTomes*) é condicionada ao tipo de baú instanciado:

$$P(Skill) = \begin{cases} 0,40, & \text{se } \textit{Morbid Treasure} \\ 0,25, & \text{se } \textit{Treasure} \end{cases} \tag{3.8}$$

Mecanismos de compensação: Double Drop e Floor Loot

Para mitigar a variância negativa, o simulador implementa dois recursos de controle:

1. **Saque Duplo (*Double Drop*):** Indica uma probabilidade fixa de 25% (`CHANCE_EXTRA_ITEM`) para que salas com evento de saque forneçam um item adicional. A medida visa reduzir o tempo de otimização de inventário do agente.
2. **Itens de solo (*Floor Loot*):** Independentemente do orçamento da sala ou da presença de baús, há 25% de probabilidade (`CHANCE_FLOOR_LOOT`) de geração de consumíveis diretamente no cenário. Este recurso favorece a continuidade da exploração em andares avançados, provendo um fluxo regular de itens de recuperação.

As definições técnicas detalhadas nos arquivos `enemies_and_events.yaml`, `equipment_catal` e `skill_and_effects.yaml` podem ser consultadas no repositório do projeto (SILVA, 2025) (v2.3.2).

Currículo implícito de agentes adversariais

A distribuição procedural de agentes adversariais estabelece um currículo de treinamento implícito ao agente. O atributo `min_floor` delimita a complexidade ambiental, permitindo a estabilização de políticas base antes da exposição a transições de estado de maior complexidade e regimes de recompensa esparsos.

A Tabela 7 apresenta os estágios desse escalonamento, integrando os parâmetros do ambiente aos conceitos de aprendizagem por reforço:

Tabela 7 – Escalonamento da complexidade adversarial por profundidade

Fase	**Piso ($d \geq$)**	**Oponente**	**Abstração de RL Correspondente**
1	0	*Zombie*	Mapeamento Ação-Recompensa Imediata
2	25	*Krul*	Dependência de Estado (Variáveis Exógenas)
3	85	*Lava Golem*	Robustez à Variância Estocástica
4	160	*Ashen Monarch*	Atribuição de Crédito em Horizontes Longos
5	300	*The Forgotten One*	Generalização sob Múltiplas Restrições

Fonte: Elaborado pelo autor (2025).

O escalonamento é uma estratégia para promover a otimização da política (π_θ), atenuando a convergência prematura a mínimos locais em diferentes níveis de dificuldade:

- **Fase 1 - atribuição de recompensa:** A interação inicial com unidades *Zombie* ($d = 0$) favorece o aprendizado da relação entre a ação de ataque e o retorno positivo imediato. Nesta etapa, a política tende a convergir para estratégias de maximização de dano local.
- **Fase 2 - transição de estado:** O agente adversarial *Krul* ($d = 25$) funciona como uma

barreira de atributos. A ausência da ação `EQUIP` para itens de maior raridade torna a probabilidade de vitória estatisticamente baixa. A otimização da política exige o mapeamento de estados contendo equipamentos superiores, visto que estes são agregados a maiores probabilidades de transição para estados de recompensa.

- **Fase 3 - otimização de estados de sobrevida:** O *Lava Golem* ($d = 85$) introduz variância por meio de dano explosivo. Este comportamento penaliza políticas que operam com baixas margens de segurança, favorecendo a otimização de sobrevida *buffer* de HP para compensar ataques de elevada magnitude.
- **Fase 4 - atribuição de crédito temporal:** O *Ashen Monarch* ($d = 160$) aplica efeitos de dano contínuo (*Heat*). A otimização exige que a função de valor compute o impacto de estados de calor atuais em eventos terminais futuros. Isso requer que o cálculo do retorno acumulado (G_t) considere horizontes temporais extensos.
- **Fase 5 - generalização:** A unidade *The Forgotten One* ($d = 300$) combina múltiplas mecânicas restritivas (*Soul Drain*, *Stun* e *Blind*). A configuração do agente adversarial desencoraja o *overfitting* em ações singulares, favorecendo uma política capaz de alternar entre comportamentos defensivos e ofensivos sob condições variáveis.

3.2.6.4 Sistema de reputação e Interação Multiagente

O componente `reputation.py` processa a dinâmica de reputação (Karma) com base em geometria hiperbólica. O sistema mapeia o estado dos agentes no espaço métrico do **Disco de Poincaré** ($|z| < 1$). Esta abordagem possibilita a representação de estados de polarização comportamental e variância de trajetórias em um plano complexo contínuo.

Integração da equação diferencial estocástica (SDE)

A variação temporal da reputação de um agente é modelada por uma **equação diferencial estocástica (SDE)** de Langevin, integrada pelo método de Euler-Maruyama. A posição do Karma z_t no tempo t é definida por:

$$\mathrm{d}z_t = \mu(z_t, \text{ação})\mathrm{d}t + \rho\mathrm{d}W_t \tag{3.9}$$

$\mu(z_t, \textbf{ação})$**: vetor de drift**, que estabelece o direcionamento do Karma a partir das interações registradas.

dt**:** Passo de tempo da simulação, fixado em 0.1 (`dt`).

$\sigma_{noise}\, dW_t$**:** Componente estocástico (*Ruído de Wiener*), com escala $\sigma_{noise} = 0.01$. O termo introduz flutuações que emulam a incerteza dos processos de avaliação social e evitam a convergência prematura para estados estacionários ou equilíbrios estáticos

Mapeamento de interações em vetores de drift

O módulo converte eventos discretos, como trocas e combates, em vetores de influência que deslocam o Karma em direção a atratores específicos. A função `saint_villain_drift` calcula a resultante baseada nos seguintes pontos de convergência:

- **Ações de cooperação (`"pró-sociais"`):** Geram um deslocamento em direção ao atrator $z_{saint} = 0,95 + 0j$. Eventos de *Trade Success* aplicam este vetor, aproximando o estado z do limite positivo do eixo real ($Re(z) \to 1$).
- **Ações de conflito (`"antissociais"`):** Direcionam o estado para o atrator $z_{villain} = -0,95 + 0j$. Instâncias de traição (`REW_BETRAYAL`) ou eliminações assimétricas geram deslocamentos de magnitude elevada em direção à região negativa do plano.
- **Ações neutras (`"neutral"`):** Exercem atração em direção à origem ($0 + 0j$). Este mecanismo estabelece uma tendência de normalização da reputação para o estado inicial na ausência de interações frequentes.

O parâmetro `attraction` (0,5) regula a intensidade dos deslocamentos. O valor de `attraction` impõe que a evolução da reputação ocorra de forma incremental, embora o sistema atribua pesos superiores a eventos de traição para simular a assimetria na estabilidade da confiança em sistemas competitivos.

Interação entre agentes no santuário

A topografia dos Santuários condiciona as interações entre agentes às restrições espaciais do ambiente. No sistema *BuriedBrains*, a coordenação não utiliza comandos diretos de negociação, mas ocorre via manipulação de inventário. Itens das categorias **arma, armadura, artefato ou consumível** funcionam como vetores de sinalização. O descarte de um item funcional (Ações 5–8) resulta na redução de recursos ofensivos ou defensivos via política do agente, o que indica uma sinalização de não hostilidade por meio da diminuição da capacidade competitiva do agente.

A resolução não hostil no sistema opera sob duas modalidades:

- **Barganha por troca (*Bargain Trade*):** Define-se pelo descarte simultâneo de itens por ambos os agentes e posterior coleta do recurso depositado pelo agente adversarial. O sistema invalida a oferta caso um agente recupere o item que ele próprio descartou, o que

mitiga vieses de exploração de mecânicas. A conclusão da sequência atribui $+200$ pontos de recompensa e incremento duplo na métrica de *Karma* aos participantes.

- **Barganha por tributo (*Toll*):** Interação unilateral em que um agente descarta um item e o agente adversarial o recolhe sem oferecer contrapartida material. O doador recebe incremento na métrica de *Karma*, enquanto o receptor amplia seu inventário. Esse processo estabelece um custo de passagem que atua como desincentivo ao confronto direto.

Mecânicas de conflito e penalidades de reputação

A execução de agressões (Ações 0–3) ativa um estado de combate *PvP* irreversível. O sistema aplica sanções a estados específicos de interação: a **traição** (ataque após sinalização do agente adversarial) e a **perfídia** (ataque após o registro de sinalização do agente). Ambas as ações resultam em penalidade de -100 na recompensa e redução dos índices de *Karma*.

As variações na reputação são proporcionais à disparidade de nível entre os agentes:

1. **Vitória sob disparidade positiva:** O agente que derrota um agente adversarial de nível superior (diferença > 10) recebe bônus elevado de *Karma*.
2. **Vitória sob disparidade negativa:** A vitória sobre agentes de nível inferior (diferença < 10) gera decréscimos de *Karma* proporcionais à diferença de poder.

Saída e Purgo

A liberação do acesso à saída ocorre mediante a ocupação do mesmo vértice pelos agentes ou a conclusão de uma sequência de barganha. Para mitigar a ausência de resolução temporal no encontro, o mecanismo de **Purgo** é acionado após 30 passos na arena. Este protocolo introduz uma redução progressiva de 5% do HP a cada 5 turnos, o que induz o encerramento da interação pelo risco de eliminação por inatividade dos agentes. Se um dos agentes for eliminado pelo ambiente sem ter havido o encontro, a porta de saída é liberada automaticamente.

Mapeamento da reputação no vetor de observação

Para possibilitar o processamento pelo modelo, o número complexo z integra o vetor de observação do agente ($\mathbb{R}^{198}$) por meio de sua decomposição cartesiana. O módulo de percepção expõe dois indicadores do agente adversarial:

1. **Componente real ($\mathbf{Re}(z)$):** Representa a tendência comportamental entre cooperação e agressão.
2. **Componente imaginária ($\mathbf{Im}(z)$):** Desvios no ângulo indicam oscilações na reputação do agente, sinalizando a transição entre estados de reciprocidade e potencial hostilidade.

3.2.7 Paralelismo e orquestração multiagente

O *throughput* de geração de experiências no *BuriedBrains* é sustentado por uma infraestrutura de paralelismo estruturada em dois níveis. O primeiro nível opera a concorrência de agentes em uma única instância do ambiente. O segundo nível utiliza multiprocessamento para escalar a coleta de trajetórias em múltiplos núcleos de CPU, permitindo a geração de transições em escala.

3.2.7.1 Interface de instâncias e o `MultiAgentWrapper`

O *BuriedBrains* utiliza o `MultiAgentWrapper`, um componente desenvolvido para servir como interface de compatibilidade entre a lógica multiagente do simulador e a Application Programming Interface (API) de vetorização (`VecEnv`) da biblioteca *Stable Baselines3*. O design do sistema garante a independência entre processos paralelos: agentes vinculados a um processo não interagem com agentes de instâncias distintas. Essa separação previne o vazamento de informações entre trajetórias, garantindo que as interações entre agentes fiquem restritas ao escopo de sua própria instância.

O controle de sincronismo é implementado via uma barreira de sincronização no método `step`. Este mecanismo pausa a progressão do simulador até que as ações de todos os `num_agents` sejam recebidas, processando então a transição para o estado subsequente. Tal procedimento tem por objetivo evitar condições de corrida durante a manipulação de variáveis de estado compartilhadas, como o controle de ocupação de salas ou a modificações topológicas no ambiente.

3.2.7.2 Escalabilidade e demandas de hardware

O segundo nível de paralelismo é implementado por meio da classe `SubprocVecEnv`, que distribui réplicas do ambiente em processos independentes na CPU. A densidade de agentes simultâneos é condicionada pela arquitetura selecionada e pela capacidade do hardware utilizado.

Desta forma, o sistema permite simulações com múltiplos agentes, sendo o limite de escalabilidade restrito pela relação entre a quantidade de núcleos de processamento (`num_cores`) e o volume de memória disponível para os estados da LSTM. Para prevenir a vazamentos de memória e manter a estabilidade da execução, o design do simulador inclui um procedimento que executa a desalocação de instâncias de agentes ao término de cada episódio global ou local,

liberando os recursos alocados.

3.2.7.3 Monitoramento, Callbacks e registro de dados

O monitoramento do treinamento no *BuriedBrains* utiliza uma arquitetura de *callbacks* integrada ao *TensorBoard*. Em ambientes multiagente com recompensas esparsas, a análise baseada exclusivamente na recompensa média pode não capturar nuances da dinâmica de aprendizado ou a instabilidade nas políticas de ação. A classe `CustomLoggingCallback` coleta métricas que associam o desempenho quantitativo aos padrões de interação e trajetórias dos agentes no simulador.

Métricas de desempenho e sobrevivência

Este grupo de indicadores monitora a adequação do agente às restrições do simulador:

- `custom/avg_floor_reached`: Registra o andar médio atingido, representando a métrica de progressão no ambiente.
- `custom/avg_level` e `custom/win_rate`: Indicam a a assimilação entre risco e recompensa.
- `custom/rate_invalid_actions`: A redução nesta taxa sugere a conformidade do modelo com as limitações estruturais e espaciais do ambiente.

Métricas de combate

Estes registros descrevem a interação entre o agente e os agentes adversariais conforme o escalonamento da dificuldade:

- `custom/avg_damage_dealt` e `combat/avg_highest_crit`: Monitoram a incidência de bônus de atributos e modificadores de equipamentos para compensar o escalonamento de pontos de vida dos agentes adversariais.
- `combat/avg_pve_duration` e `avg_pvp_duration`: Avaliam a duração média das instâncias de combate. Durações reduzidas estão associadas a políticas de alta densidade de dano (*burst*), ao passo que confrontos extensos caracterizam comportamentos de mitigação e persistência.
- `combat/avg_healing_received`: Mensura o acionamento de habilidades de regeneração e itens de suporte para mitigar o dano residual acumulado entre salas.

Métricas de otimização de inventário

Estas métricas descrevem o gerenciamento de recursos e a interação do agente com elementos gerados proceduralmente:

- `custom/avg_chests_opened`: Quantifica a frequência de interação com eventos de saque, pfornecendo subsídios para a caracterização de perfis de trajetória, distinguindo entre a progressão direta e comportamentos de exploração para aquisição de recursos.
- `custom/avg_equipment_swaps` e `custom/avg_skill_upgrades`: Registram a frequência de modificação nos atributos internos e no conjunto de habilidades disponíveis ao agente durante o episódio.
- `custom/avg_consumables_used` e `custom/avg_wait_actions`: Monitoram a utilização de itens de consumo e a frequência de ações de inação (*no-op*), que podem indicar a otimização de políticas frente a tempos de recarga (*cooldowns*) ou a estados de espera estratégica.

Métricas de interações entre agentes

O ambiente *BuriedBrains* fornece métricas para a análise de interações entre os agentes. As métricas sociais quantificam a frequência e a tipologia dos encontros:

- `social/total_arena_encounters` e `total_pvp_combats`: Registram a presença de agentes em zonas comuns e a taxa de transição desses encontros para estados de combate direto.
- `social/total_bargains`, `_trade` e `_toll`: Diferenciam categorias de troca de recursos, distinguindo entre trocas bilaterais e descartes unilaterais.
- `total_betrayals` e `total_cowardice_kills`: Registram, respectivamente, a transição de estados cooperativos para competitivos em uma mesma trajetória e a incidência de ataques em cenários de alta assimetria de integridade (*HP*) entre os agentes.
- `social/avg_final_karma`: Indica o valor médio acumulado da reputação ao final dos episódios.

Análise qualitativa por registro de dados

O simulador inclui uma funcionalidade destinada ao armazenamento de trajetórias individuais de alto desempenho, denominada *Hall da Fama*. O sistema retém os dados de uma amostra da trajetória de agentes (configurada por padrão em 500 últimos registros para cada agente).

Este registro organiza os agentes em ordem decrescente com base nos seguintes critérios de classificação:

1. **Andar máximo alcançado**: Mensura a progressão máxima de andares atingida no ambiente.

2. **Nível individual máximo**: Indica a evolução do estado do agente e aumento de atributos.
3. **Oponentes derrotados**: Registra a eficácia em instâncias de combate contra adversários do ambiente.

Este conjunto de registros permite que a avaliação dos modelos considere variáveis de interação e padrões de sobrevivência, complementando a análise baseada estritamente no retorno acumulado da função de recompensa.

3.3 Interface de monitoramento e validação (Visualizer)

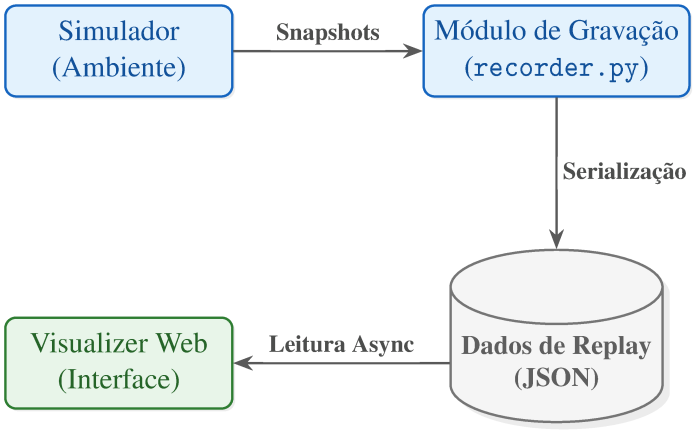
Ambientes multiagente apresentam desafios concernente à depuração e à interpretação de resultados devido à estocasticidade e à alta dimensionalidade dos dados. Métricas quantitativas isoladas podem omitir nuances comportamentais resultantes das políticas. No *BuriedBrains*, a análise qualitativa utiliza um *Visualizer* implementado com tecnologias *web* (JavaScript, HTML5 e CSS3). A ferramenta permite a avaliação das políticas aprendidas ao permitir a identificação depermitir a identificação de padrões de movimentação, alocação de recursos e dinâmicas de interação entre os agentes.

3.3.1 Pipeline de dados e desacoplamento de arquitetura

O projeto do *Visualizer* adota uma arquitetura independente da execução do treinamento. Essa estrutura visa mitigar a concorrência por recursos de hardware (CPU/GPU) entre a renderização visual e o treinamento do modelo. O fluxo de informações está segmentado em três componentes (Figura 10):

1. **Gravação (`recorder.py`):** Registra o estado dos agentes (`agent_states`) em cada passo temporal da simulação.
2. **Serialização:** Transforma os dados brutos em arquivos JSON contendo trajetórias dos agentes, grafos e logs de execução.
3. **Renderização (`main.js`):** Processa os arquivos serializados para reconstruir os episódios por meio de módulos especializados.

Figura 10 – Fluxo de dados entre o simulador e o visualizer



Fonte: Elaborado pelo autor (2025).

3.3.2 Pipeline de captura e serialização

A coleta de dados é realizada por meio do script `run_recording.py`. Este componente instancia o ambiente `BuriedBrainsEnv` e utiliza uma política pré-treinada (`RecurrentPPO`, via biblioteca *Stable Baselines3*) para a amostragem de ações no simulador. Durante a execução, a classe `BuriedBrainsRecorder` monitora e armazena as transições de estado.

3.3.2.1 Coleta de dados de estado

O `BuriedBrainsRecorder` possui acesso ao estado global do ambiente (`agent_states`). Através do método `record_step`, o componente registra os atributos dos agentes ativos ao final de cada turno, incluindo:

- **Estado:** níveis de HP, habilidades, inventário, experiência (XP) e nível;
- **Reputação:** posição do agente no plano complexo (partes real e imaginária);
- **Contexto:** classificação do modo de exibição (`EXPLORATION`, `COMBAT_PVE`, `COMBAT_PVP` ou `ARENA`) definida pela localização e estado do agente.

3.3.2.2 Conversão semântica

O pipeline converte as saídas numéricas da rede neural em rótulos identificáveis para análise. O gravador associa o índice da ação selecionada ao nome correspondente nos catálogos do sistema (ex: conversão do índice 1 para a ação "*Heavy Blow*").

O *recorder* é responsável pela extração do vetor. de observação bruta (`raw_obs`) de 198 dimensões. Esse registro permite que o visualizador apresente os dados de entrada

que compõem o vetor de observação no instante da decisão, incluindo a posição de vizinhos e modificadores ambientais. Esses dados permitem a análise de trajetórias detalhada na Seção **??**.

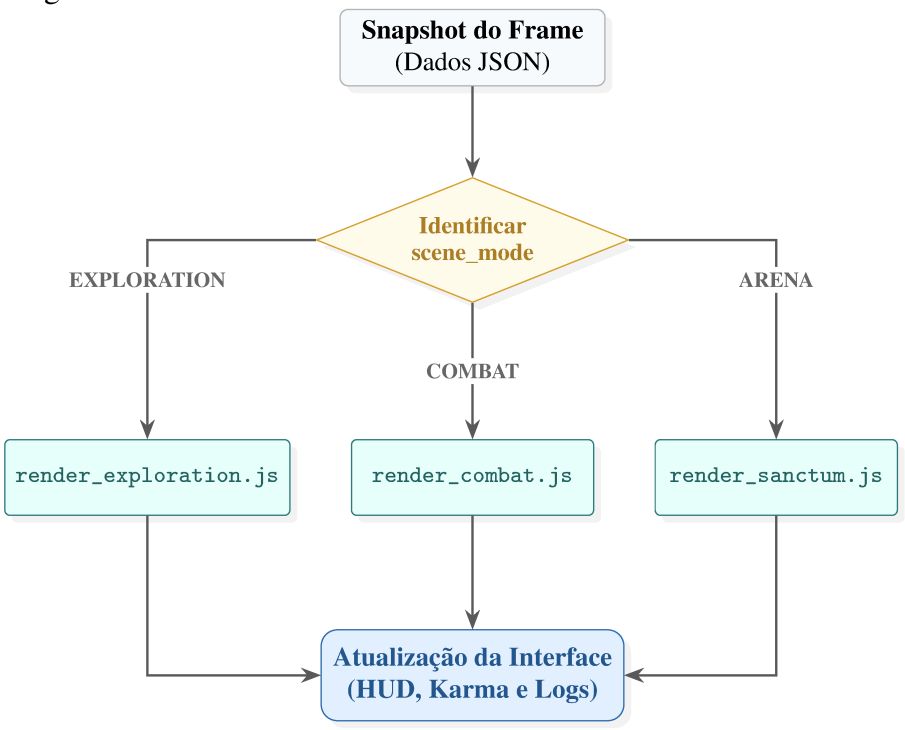
3.3.2.3 Estrutura do arquivo JSON

Ao término da simulação, o método `save_to_json` exporta os dados em um arquivo estruturado cronologicamente. Cada entrada na lista principal do JSON corresponde a um passo temporal, mapeando identificadores únicos (`agent_id`) aos seus respectivos estados. Essa estrutura permite o carregamento do episódio para navegação não-linear e alternância de perspectivas entre agentes durante o *replay*.

3.3.2.4 Arquitetura do front-end e roteamento de Cenas

A interface do *Visualizer* emprega uma estrutura modular para o processamento de telemetria, visando manter a responsividade do navegador durante a exibição de volumes elevados de dados. A função `renderFrame` centraliza a lógica de exibição. A cada atualização na linha do tempo, o sistema verifica o `scene_mode` do agente e aciona o motor de renderização correspondente, conforme ilustrado na Figura 11.

Figura 11 – Lógica de roteamento dinâmico de cenas do Visualizer



Fonte: Elaborado pelo autor (2025).

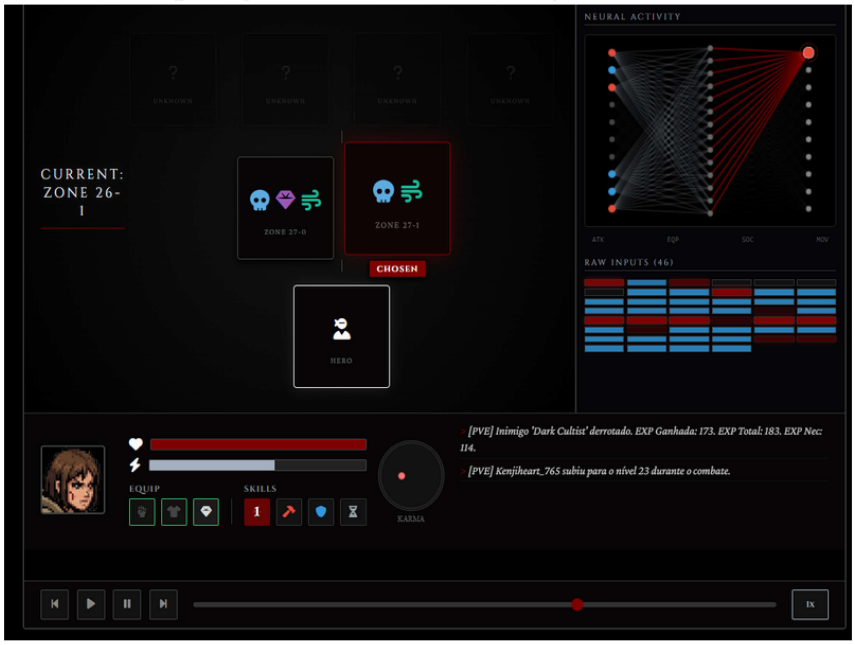
O sistema alterna entre três visualizações principais:

Exploração: Organiza o progresso do agente exibindo salas adjacentes com ícones de perigo (caveiras), recursos (gemas) e modificadores (vento). A interface destaca a decisão da política com o rótulo *"CHOSEN"*.

Combate (PvE e PvP): Reconstrói os confrontos em arena, utilizando barras de vida e *feedback* visual de dano (tremores e flashes), essencial para validar a execução de habilidades.

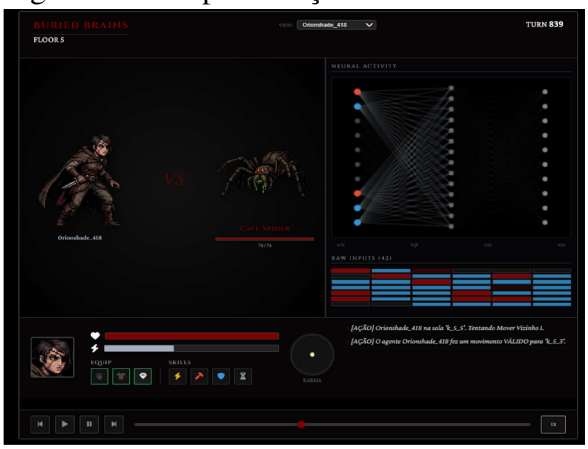
Santuário: Exibe o mapa global em uma grade 3×3, desenhando conexões entre salas e permitindo visualizar a densidade de agentes e itens em tempo real.

Figura 12 – Interface de exploração: detalhe das salas adjacentes e caminho selecionado

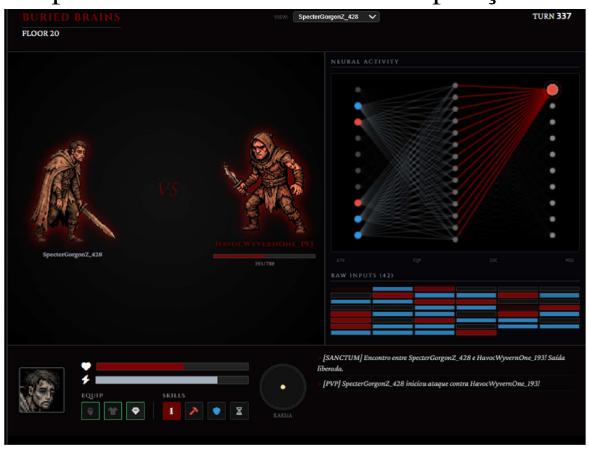


Fonte: Elaborado pelo autor (2025).

Figura 13 – Representações do módulo de combate para diferentes contextos de oposição



(a) Combate Agente-Máquina (PvE)



(b) Combate Agente-Agente (PvP)

Fonte: Elaborado pelo autor (2025).

Figura 14 – Interface do Santuário: visualização de topologia e ocupação da arena

Fonte: Elaborado pelo autor (2025).

3.3.2.5 Representação visual e telemetria neural

A representação dos agentes utiliza uma biblioteca de ativos gráficos gerados por inteligência artificial (GOOGLE, 2025). O módulo `asset_loader.js` utiliza um algoritmo de *hash* para manter a consistência dos arquétipos (ex: *Mage, Brute, Explorer*) vinculados a cada agente durante o episódio, auxiliando o rastreamento de trajetórias individuais.

O controle da exibição é gerenciado pelo `PlaybackManager`, que permite a navegação bidirecional e o ajuste da taxa de quadros do *replay*. Este componente opera em conjunto com o painel de visualização de ativações neurais (`render_brain.js`), que apresenta o estado interno do modelo.

A interface integra uma funcionalidade que permite selecionar qualquer agente ativo para monitoramento. Ao alterar o foco, os painéis de telemetria e o modo de cena são atualizados para refletir o vetor de observação e as ativações internas do agente selecionado. Essa característica auxilia a análise de interações e conflitos sob diferentes configurações de entrada.

A telemetria é composta por quatro módulos principais:

- **Painel de atividade neural (`render_brain.js`):** representa graficamente a estrutura da rede, indicando as ativações da camada de saída e a magnitude dos sinais de entrada. A visualização indica padrões de ativação vinculados a comportamentos específicos;
- **Malha de observações do agente (*Raw Inputs*):** exibe as 198 dimensões do vetor de

observação em uma grade visual. Este módulo permite a verificação dos componentes de vizinhança, modificadores e estados próprios que compõem o vetor de entrada no instante da seleção da ação;

- **Painel de controle:** monitora os parâmetros de estado (HP, XP), o nível de Karma e o tempo de recarga de habilidades, fornecendo dados sobre a progressão do agente;
- **Logs de eventos recentes:** registra as transições de estado e interações em formato de texto. O módulo complementa os dados numéricos com uma sequência cronológica de eventos que auxilia a depuração do fluxo lógico da simulação.

A integração entre a representação gráfica e os dados brutos da rede neural auxilia na identificação de padrões entre as variáveis do ambiente e as ações executadas pelo modelo para auxílio da análise qualitativa.

3.4 Infraestrutura de treinamento e análise

O módulo `train.py` é responsável pelo processamento de dados, a extração de características e a execução de políticas de aprendizado multiagente.

3.4.1 Arquitetura de extração de características por Atenção

O módulo `AttentionFeatureExtractor` processa o vetor de observações do agente no ambiente por meio de mecanismos de atenção. Informações de habilidades, movimentação e propriocepção são mapeadas por *encoders* de sub-redes. A motivação para a escolha de um mecanismo de atenção se baseia no fato de que, em arquiteturas de camadas densas, sinais discretos de baixa dimensionalidade podem sofrer atenuação quando processados em conjunto com *embeddings* de alta dimensionalidade provenientes de itens e habilidades (VASWANI *et al.*, 2017).

Este extrator é aplicado tanto ao *baseline* PPO quanto à arquitetura RecurrentPPO. Essa padronização sugere que que eventuais variações de desempenho decorrem da persistência temporal da política, e não de diferenças na capacidade de representação das observações.

3.4.2 Tokenização e projeção linear

O vetor de observação de 198 dimensões é segmentado em 11 tokens de 18 dimensões cada. Este particionamento agrupa blocos de dados específicos, como o inventário (índices 130–

197) e os sensores do Santuário (índices 50–56), em subconjuntos distintos para o processamento.

Cada token $t_i \in \mathbb{R}^{18}$ é projetado para um espaço latente de dimensão $d_{model} = 96$ por meio de uma projeção linear. A projeção é definida pela aplicação de uma função de ativação não linear sobre a transformação linear:

$$e_i = \text{ReLU}(W_p t_i + b_p) \tag{3.10}$$

onde $W_p \in \mathbb{R}^{96\times 18}$ representa a matriz de pesos de projeção e b_p o vetor de viés. A sequência resultante de embeddings $E = (e_1, e_2, \ldots, e_{11})$ é utilizada como entrada para o mecanismo de atenção.

3.4.3 Rede de Self-Attention

A classe `AttentionFeatureExtractor` implementa a atribuição de pesos para os componentes da observação em relação à sua representação global. Por meio do mecanismo de produto escalar escalonado (*scaled dot-product attention*), o computa as projeções de Consulta (Q), Chave (K) e Valor (V), processando a saída conforme a Equação 3.11:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.11}$$

Onde d_k denota a dimensão das chaves, utilizada como fator de escala para estabilização dos gradientes. Esta estrutura favorece a preservação de sinais que poderiam sofrer atenuação em camadas densas globais.

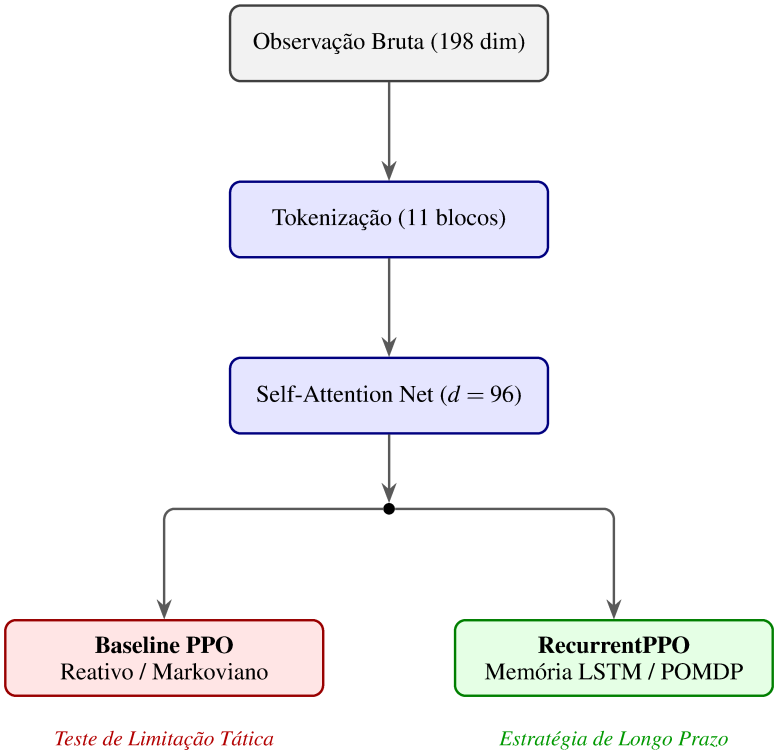
3.4.4 Definição de baselines

A avaliação do *BuriedBrains* baseia-se na comparação entre duas configurações distintas, estabelecidas para isolar o impacto da memória no desempenho do agente:

- **PPO (baseline Markoviano):** Esta configuração atua como uma linha de base reativa. O algoritmo assume a Propriedade de Markov, operando sob a premissa de que a observação atual contém as informações necessárias para a tomada de decisão. No contexto deste experimento, o uso dessa arquitetura indica as limitações de políticas sem memória em tarefas que demandam o rastreamento de recursos ao longo do tempo.
- **RecurrentPPO (abordagem para POMDP):** Constitui a arquitetura central da proposta. Após a extração de características pelo mecanismo de atenção, os dados são processados

por uma camada *LSTM* (Long Short-Term Memory). Esta camada mantém um estado oculto (h_t) que condensa a trajetória de observações passadas para subsidiar a política de decisão. Desta forma, a arquitetura atua como um modelo de referência inversa, sugerindo que o ambiente possui características não-Markovianas.

Figura 15 – Fluxo de processamento compartilhado e diferenciação entre a arquitetura proposta e o baseline markoviano



Fonte: Elaborado pelo autor (2025).

3.4.5 Otimização de hiperparâmetros (HPO)

O framework Optuna foi utilizado para a Otimização de Hiperparâmetros (Hyperparameter Optimization - HPO), dada a sensibilidade de algoritmos de Aprendizado por Reforço a essas configurações. Este procedimento foi adotado para reduzir o impacto de parâmetros subótimos e permitir que o desempenho da arquitetura recorrente seja atribuído à sua capacidade de processamento de dependências temporais.

3.4.5.1 Metodologia de busca e algoritmo TPE

O algoritmo *Tree-structured Parzen Estimator* (TPE) foi aplicado na amostragem de parâmetros. Este método modela a distribuição das configurações de melhor desempenho para orientar as iterações (*trials*) subsequentes de forma probabilística.

3.4.5.2 Espaço de busca e parâmetros calibrados

O script `optimize.py` define o intervalo de busca para os componentes dos algoritmos *PPO* e *RecurrentPPO*:

- **Taxa de aprendizado (*learning_rate*):** Amostrada em escala logarítmica entre 10^{-6} e 10^{-3}, visando o equilíbrio entre a velocidade de convergência e a estabilidade numérica do gradiente.
- **Tamanho do lote (*batch_size*) e passos (*n_steps*):** Definidos em potências de 2, variando de 64 a 512 para o lote e 256 a 2048 para os passos. A escolha busca conciliar a utilização de memória de vídeo (VRAM) com a estabilidade das atualizações da política.
- **Coeficiente de entropia (*ent_coef*):** Amostrado entre 0,00001 e 0,05. Este parâmetro regula o balanço entre a exploração (*exploration*) de novos estados e a explotação (*exploitation*) de ações previamente mapeadas.
- **Fatores de desconto (γ e λ):** O fator de desconto γ (0,9 a 0,999) e o parâmetro Generalized Advantage Estimation (GAE) λ (0,8 a 1,0) foram incluídos para ajustar a atribuição de crédito em relação ao horizonte temporal dos episódios.

O processo de otimização totalizou 50 *trials*. Cada iteração compreendeu um treinamento de 200.000 passos, seguido pela avaliação de 10 episódios completos. A execução foi realizada de forma sequencial (`N_JOBS=1`) para manter a estabilidade da GPU e evitar falhas por insuficiência de memória.

Para reduzir o custo computacional do processo de busca, foi aplicado o algoritmo `MedianPruner`. Este método de poda monitora o desempenho parcial de cada iteração e interrompe precocemente os experimentos que apresentem resultados inferiores à mediana das execuções anteriores.

A aplicação desta técnica permite a alocação de recursos em regiões do espaço de busca que indicam maior probabilidade de convergência para lidar com as dependências temporais do ambiente *BuriedBrains*. As configurações derivadas deste processo de otimização fundamentam algumas das análises experimentais subsequentes.

3.4.6 Ambiente de execução e protocolos de experimentação

Esta subseção descreve as configurações de hardware, o ambiente de software e o protocolo de execução dos treinamentos finais. Os dados apresentados referem-se à versão

`v2.3.2` do *BuriedBrains*.

3.4.6.1 Infraestrutura de hardware

Os experimentos foram realizados em um notebook para tarefas de alto desempenho Dell G15 5530. Configuração do hardware utilizado:

- **Processador:** Intel Core i5-13450HX (10 núcleos, 16 *threads*), com frequência máxima de 4,60 GHz.
- **Memória RAM:** 16 GB DDR5 de 4800 MHz.
- **GPU:** NVIDIA GeForce RTX 3050 Mobile (6 GB GDDR6).
- **Drivers e CUDA:** Driver NVIDIA versão 581.32 com suporte à arquitetura CUDA 13.0.

3.4.6.2 Sistema operacional e dependências

O sistema operacional utilizado foi o Windows 11 (versão 25H2), com o interpretador Python 3.14.0. As bibliotecas utilizadas e suas versões estão listadas no arquivo `requirements.txt` do repositório (SILVA, 2025). As principais dependências incluem:

- **Aprendizado por reforço:** *Stable Baselines3* (SB3) e *Gymnasium*. Estas bibliotecas gerenciam a interface dos ambientes e a execução dos algoritmos PPO e RecurrentPPO.
- **Cálculo tensorial e redes neurais:** *PyTorch*, utilizado na implementação das camadas neurais e mecanismos de *Self-Attention*.
- **Processamento de dados:** *NumPy*, *Pandas* e *NetworkX*. Estes pacotes realizam a gestão de estados e o processamento de grafos procedurais.
- **Monitoramento:** *Matplotlib*, *Seaborn* e *TQDM* para a geração de logs e visualização do progresso.

3.4.6.3 Definição dos experimentos

Foram conduzidas cinco baterias de experimentos utilizando a versão `2.3.2` do simulador. O protocolo objetiva a criação de cenários com propriedades que permitam comparar o *baseline* não recorrente (*PPO*) com a arquitetura recorrente (*RecurrentPPO*).

Cada experimento foi repetido com as *seeds* 111, 222, 333, 444, 555 e 666. Parâmetros omitidos nas seções subsequentes adotam as definições padrão do arquivo `train.py`. A normalização de observações e recompensas foi aplicada via `VecNormalize`, com as flags

`norm_obs=True` e `norm_reward=True`. O truncamento de recompensas seguiu a configuração `clip_reward=NORM_REWARD_CLIP`.

A análise da métrica `ep_mean_rew_` é realizada em conjunto com os demais indicadores na seção 3.2.7.3. Os arquivos de automação (`.bat`) usados nos treinamentos estão disponíveis no Apêndice A.

Experimento Magnolia

Este experimento utiliza hiperparâmetros obtidos via otimização com *Optuna* para um cenário de 32 agentes (4 ambientes com 8 agentes cada) e 20×10^6 passos temporais. O objetivo é caracterizar o desempenho de referência das arquiteturas sob densidade populacional média e mesmas regras de ablação do cenário Ne'helit, com as configurações de parâmetros otimizadas.

- **RecurrentPPO (LSTM):** Taxa de aprendizado (α) de $2,06 \times 10^{-4}$, `n_steps` = 128, `batch_size` = 256, `n_epochs` = 5, $\gamma = 0,99$, $\lambda = 0,9$, `ent_coef` (β) = $5,57 \times 10^{-4}$, `max_grad_norm` = 0,77, `clip_range` = 0,2, `lstm_hidden_size` = 256 e `net_arch` = [256, 256].
- **PPO (baseline):** $\alpha = 6,16 \times 10^{-5}$, `n_steps` = 512, `batch_size` = 128, `n_epochs` = 7, $\gamma = 0,99$, $\lambda = 0,95$, `ent_coef` (β) = $1,95 \times 10^{-3}$, `max_grad_norm` = 0,5, `clip_range` = 0,3 e `net_arch` = [256, 256].

Experimento Bee Swarm

Esta bateria avalia a estabilidade da convergência sob ruído amostral elevado. O teste emprega 1.000 agentes simultâneos (10 ambientes com 100 agentes cada) e 10×10^6 passos temporais. A frequência de atualização foi elevada em 16 vezes em relação ao experimento de referência, visando identificar instabilidades ou colapso da política sob frequência de atualização elevada:

- **Parâmetros de rollout:** Horizonte de 32 passos temporais, `batch_size` = 8192 e `n_epochs` = 20.
- **Otimização:** $\alpha = 3 \times 10^{-4}$, coeficiente de entropia (β) = 0,01 e fator GAE (λ) = 0,95.

Experimento Be'helit

O foco deste experimento é a capacidade de capturar dependências temporais de longo prazo. Com uma população de 160 agentes (10 ambientes com 16 agentes) e 5×10^6 passos temporais, o horizonte de atualização foi expandido para 256 passos. A configuração objetiva testar os limites da retropropagação truncada através do tempo (*Truncated BPTT*) e a

convergência da arquitetura reativa (PPO):

- **Configuração de treino:** `batch_size` = 5120, `max_grad_norm` = 0,5 e `n_epochs` = 6.
- **Ajuste de hiperparâmetros:** $\alpha = 1 \times 10^{-4}$ para alta densidade de agentes e $\beta = 0,01$ para incentivo à exploração.

Experimento Ne'helit

Por padrão, o ambiente premia o encontro entre agentes e o dano infligido e dano recebido. Este estudo de ablação verifica a manifestação de comportamentos competitivos na ausência de recompensa imediata. O cenário utiliza 160 agentes (10 ambientes com 16 agentes) e 20×10^6 passos temporais. Apenas o modelo recorrente foi treinado neste cenário, para ser comparado com o cenário Be'helit:

- **Modificadores de recompensa:** `REW_DMG_DEALT_SCALAR` = 0,0, `REW_DMG_TAKEN_SCALAR` = 0,0 e `REW_MEET_BONUS` = 0,0.
- **Parâmetros de treino:** `batch_size` = 5120, `n_steps` = 256 e $\alpha = 1 \times 10^{-4}$.

Experimento Heiritsu

O cenário Heiritsu analisa o impacto da rotatividade de Santuários na convergência dos modelos. Assume parâmetros baseados no resultado do Optuna, escalados para mais agentes. O treinamento utiliza 96 agentes (6 ambientes com 16 agentes) e 20×10^6 passos temporais:

- **Arquitetura:** `lstm_hidden_size` = 256 e `net_arch` = [256, 256].
- **Parâmetros de treino:** $\alpha = 2 \times 10^{-4}$, `batch_size` = 1024, $\beta = 0,001$ e `max_grad_norm` = 0,77.

Tabela 8 – Resumo dos parâmetros experimentais por bateria de teste

Experimento	**Agentes**	**Ambientes**	**Horizonte**	**Batch**	**LR (α)**	**Entropia (β)**	**Epochs**	**Timesteps**
Magnolia (LSTM)	32	4x8	128	256	$2,06 \times 10^{-4}$	$5,57 \times 10^{-4}$	5	20M
Magnolia (PPO)	32	4x8	512	128	$6,16 \times 10^{-5}$	$1,95 \times 10^{-3}$	7	20M
Bee Swarm	1000	10x100	32	8192	3×10^{-4}	0,01	20	10M
Be'helit	160	10x16	256	5120	1×10^{-4}	0,01	6	5M
Ne'helit	160	10x16	256	5120	1×10^{-4}	0,01	6	20M
Heiritsu	96	6x16	128	1024	2×10^{-4}	0,001	6	20M

Fonte: Elaborado pelo autor (2025).

4 RESULTADOS E DISCUSSÕES

Este capítulo apresenta a análise dos dados coletados nas cinco baterias de experimentos realizadas no ambiente *BuriedBrains*. O objetivo consiste em verificar a relação entre a capacidade de sobrevivência dos agentes e suas respectivas arquiteturas neurais, a fim de inferir sobre o caráter não-Markoviano do ambiente.

As métricas estão divididas em quatro eixos: desempenho macroscópico, estabilidade do modelo, eficácia dos equipamentos e padrões de comportamento emergente controlado.

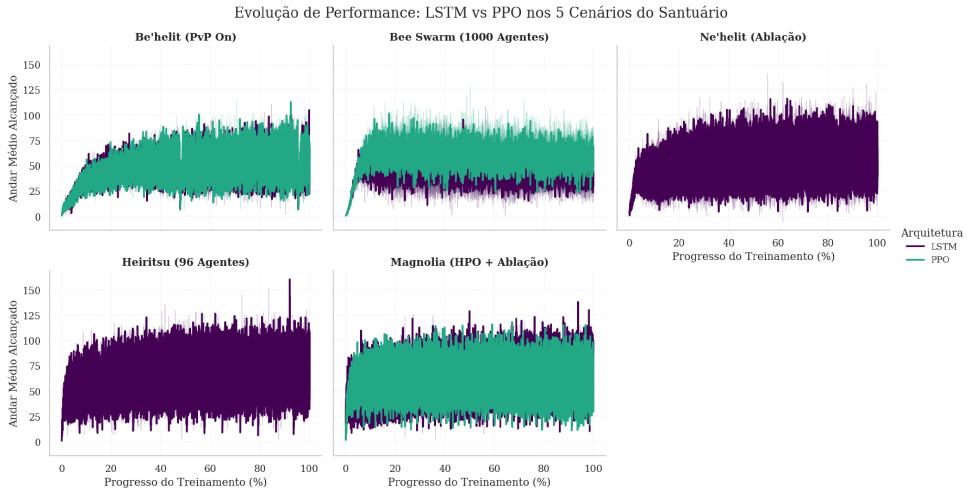
4.1 Macro-Performance e validação estatística

Esta seção descreve a progressão dos agentes entre os andares do ambiente. Compara-se o limite de desempenho das arquiteturas PPO e LSTM em cenários de hiperparametrização controlada. A análise verifica se a inclusão de memória recorrente resulta em aumento da profundidade alcançada, especialmente em contextos que exigem a relação de eventos em longos intervalos de tempo.

4.1.1 Análise comparativa de média de andares alcançados e o limite Markoviano

A Figura 16 exibe o desempenho médio dos agentes nos cinco cenários testados. Os dados indicam que a arquitetura LSTM apresenta uma trajetória de aprendizado com menor oscilação e alcança níveis superiores na maioria dos testes. A exceção ocorre no experimento *Bee Swarm*. Neste caso, a restrição do horizonte temporal ($N_{steps} = 32$) pode ter influenciado negativamente no aprendizado do modelo recorrente, via saturação da camada oculta.

Figura 16 – Evolução do andar médio alcançado por cenário e arquitetura



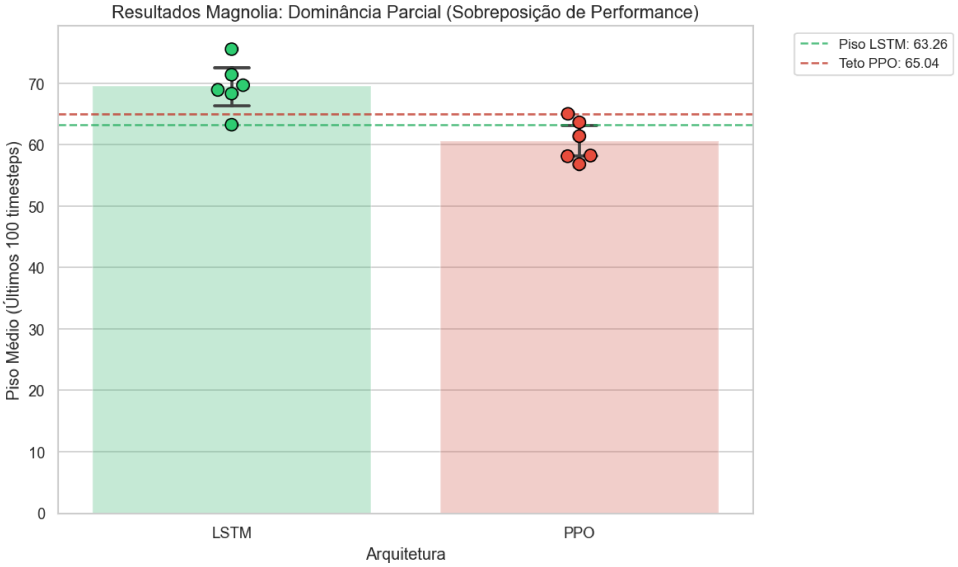
Fonte: Elaborado pelo autor (2025).

No modelo reativo, identifica-se uma estabilização da curva de progressão próximo ao andar 60. A arquitetura LSTM apresenta uma redução gradual na taxa de sobrevivência, o que sugere maior capacidade de ajuste às variáveis procedurais do ambiente. Em contrapartida, o PPO exibe um declínio acentuado de desempenho. Este padrão indica uma possível limitação da política puramente Markoviana. Sem o estado oculto (*hidden state*) para o processamento de contextos prévios, o agente PPO demonstra dificuldade em associar ações executadas em estágios iniciais com a sobrevivência em níveis de maior dificuldade. No cenário Magnolia, especificamente, observou-se um ganho relativo de 10,1%, enquanto no Bee Swarm, houve uma divergência de -32.6% em relação ao PPO.

4.1.2 Análise de variância entre sementes

Realizou-se uma análise cruzada entre seis sementes aleatórias para mensurar a influência da variância estocástica nos resultados. Os dados consolidados na Figura 17 apontam uma superioridade estatística da arquitetura recorrente em comparação ao modelo reativo no cenário *Magnolia*.

Figura 17 – Meta-análise de desempenho por semente aleatória no experimento Magnolia



Fonte: Elaborado pelo autor (2025).

A análise registra uma Probabilidade de Superioridade (Common Language Effect Size) de 94,4% para o modelo LSTM. Observou-se que, em todas as iterações, o desempenho mínimo registrado pelo LSTM foi superior à média obtida pelo PPO.

Sob diferentes condições de inicialização, como verificado na semente `S666`, a arquitetura recorrente manteve a estabilidade dos resultados. O PPO, no entanto, apresentou maior sensibilidade aos ruídos do ambiente. Os dados sugerem que a memória atua como um fator de estabilização, permitindo que o agente mantenha uma política de ação menos errática sob condições de incerteza procedural.

Apesar do baixo número de seeds, os resultados demonstram estabilidade de convergência, evidenciando que a arquitetura recorrente não depende tanto de sorte inicial para alcançar bom desempenho.

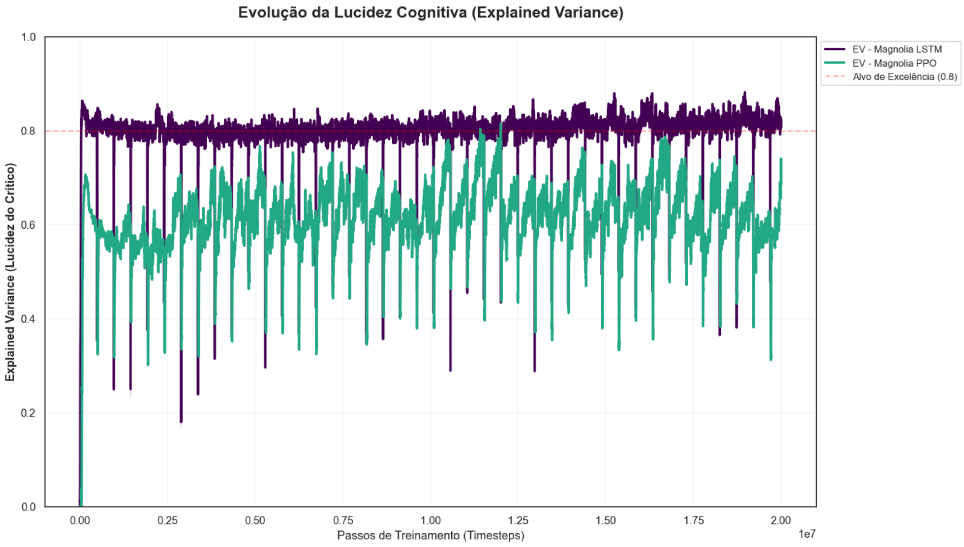
4.2 Análise de estimativa de valor e estabilidade da política

Nesta seção, são analisadas as métricas que descrevem a convergência dos modelos e a previsibilidade do ambiente. A análise verifica a hipótese de que a memória recorrente mitiga o impacto da opacidade temporal no ambiente *BuriedBrains*.

4.2.1 Análise de variância explicada (EV)

A métrica de Variância Explicada (EV) mede a precisão com que a rede de valor (*Critic*) estima o retorno das recompensas futuras. Em Processos de Decisão de Markov Parcialmente Observáveis (POMDP), esta métrica indica a qualidade do ajuste do modelo à dinâmica do cenário. A Figura 18 apresenta o comparativo de EV no experimento *Magnolia*. O modelo LSTM atingiu o patamar de 0,80, enquanto o PPO estabilizou-se em 0,65, sendo o teto 1,00.

Figura 18 – Comparativo de Variância Explicada (EV) entre as arquiteturas no cenário Magnolia



Fonte: Elaborado pelo autor (2025).

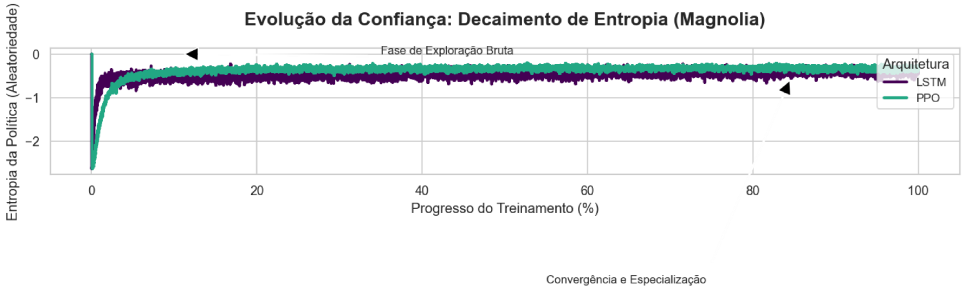
A diferença observada entre as arquiteturas sugere um limite de previsibilidade para modelos reativos. O teto de 0,65 alcançado pelo PPO indica que a rede não captura aproximadamente 35% da lógica de valor do ambiente, mesmo com a janela de *rollout* configurada para 512 passos. Este erro residual de estimativa pode ser associado à natureza não-Markoviana do cenário. No Santuário, estados com representações visuais idênticas podem possuir valores distintos em função de variáveis de estado não observadas no instante atual, como o descarte de itens de inventário. A estrutura de memória do LSTM permite a integração dessas dependências temporais, reduzindo o erro de Bellman na estimativa de valor.

4.2.2 Análise de entropia da política e convergência

A entropia da política, detalhada na Figura 19, quantifica a diversidade de ações executadas pelos modelos durante o treinamento. O PPO convergiu para uma entropia final de -0,33, valor inferior aos -0,39 registrados pelo LSTM. Nota-se que o PPO operou com um

coeficiente de exploração (`ent_coef`) de $1,95 \times 10^{-3}$, valor superior ao utilizado na arquitetura recorrente.

Figura 19 – Decaimento de entropia da política durante o treinamento (Magnolia)



Fonte: Elaborado pelo autor (2025).

Este resultado sugere uma convergência para políticas locais no PPO. A dificuldade em processar dependências temporais pode levar o agente reativo a priorizar políticas decisão que não escalam com a dificuldade do ambiente, reduzindo a exploração de ações alternativas. Já o LSTM manteve maior variabilidade de ações ao final do treinamento. Este comportamento sugere que a memória auxilia na discriminação de contextos específicos, permitindo a manutenção de uma política de ações mais diversificada sem perda de desempenho. Ou seja, a memória pode ter sido fator importante para que o agente recorrente continuasse explorando novas políticas, o que é essencial para o progresso no ambiente.

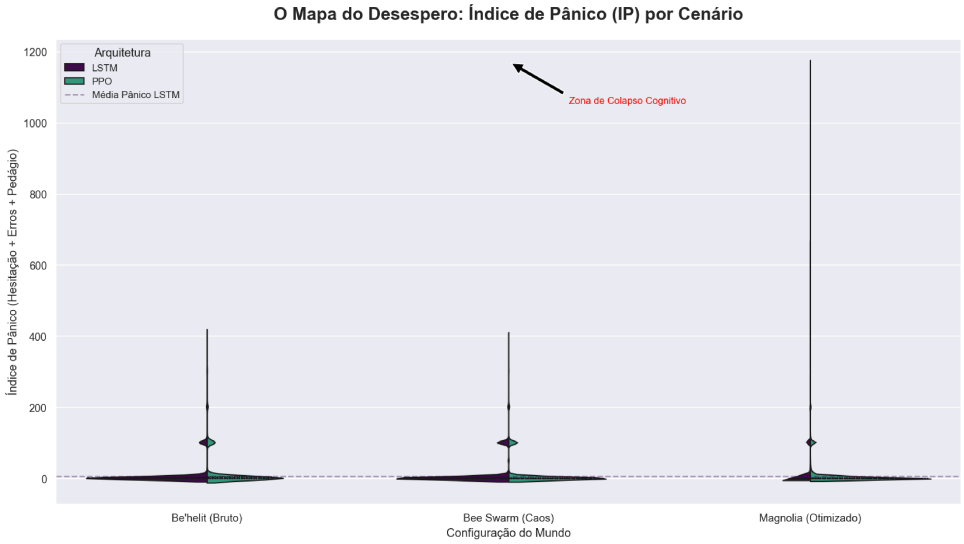
4.2.3 Análise de colapso de política

O Índice de Pânico (IP), é uma métrica personalizada definida neste trabalho como a taxa de ações inválidas por passo, medindo a estabilidade da política do agente. De acordo com a Figura 20, o modelo LSTM apresentou IP médio de 4,4, enquanto o PPO registrou 9,2 no cenário *Magnolia*, e a composição analítica deste índice é formalizada pela Equação 4.1:

$$IP = A_{wait} + (10 \cdot R_{invalid}) + (100 \cdot T_{bargains}) \tag{4.1}$$

Em que A_{wait} representa a média de ações de espera, $R_{invalid}$ a taxa de ações inválidas e $T_{bargains}$ o custo total de pedágios em trocas sociais (bargains). A ponderação exponencial dos termos reflete a gravidade de cada evento para a estabilidade sistêmica da política.

Figura 20 – Índice de Pânico: comparação de instabilidade de política

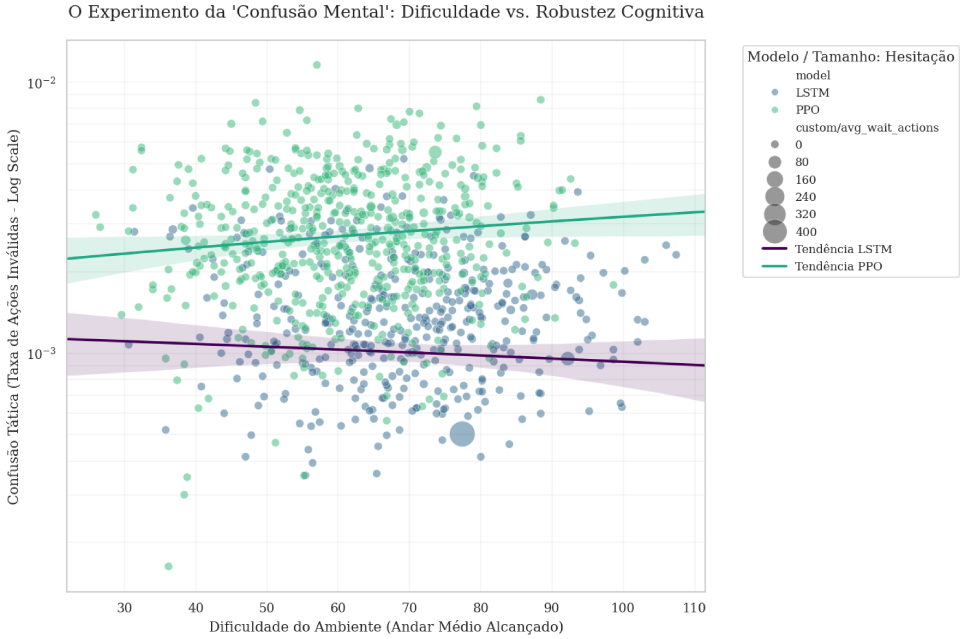


Fonte: Elaborado pelo autor (2025).

A magnitude do índice para o PPO é consistente com a hipótese de ambivalência de estado. A ausência de memória pode gerar instabilidade na seleção de ações ao encontrar observações insuficientes para a tomada de decisão. O agente recorrente apresenta menor taxa de erro, sugerindo que o contexto temporal contribui para a seleção de ações válidas de acordo com as restrições do ambiente.

De forma análoga, a análise da Figura 21 correlaciona a taxa de ações inválidas com a progressão entre os andares. A arquitetura PPO apresenta aumento na taxa de erro conforme a dificuldade do ambiente cresce ($r = 0,0911$).

Figura 21 – Correlação entre profundidade (andares) e taxa de ações inválidas



Fonte: Elaborado pelo autor (2025).

O modelo LSTM apresentou estabilidade na métrica ($r = -0,0365$). Este resultado evidencia que a memória recorrente contribui para a manutenção do desempenho operacional apesar do aumento da complexidade procedural. A manutenção de baixas taxas de erro em andares avançados sugere o aprendizado de padrões de controle que não se degradam com o incremento dos atributos dos oponentes ou da densidade de obstáculos.

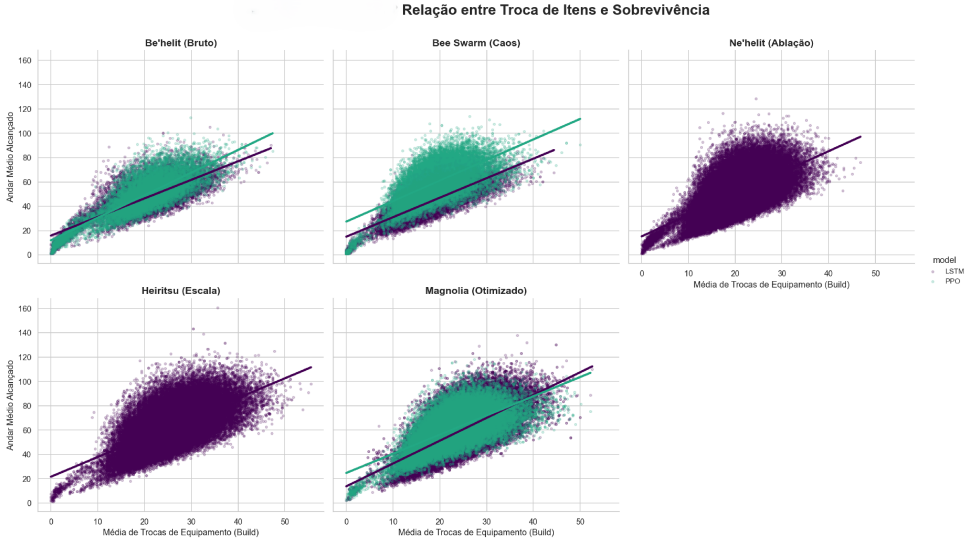
4.3 Análise de otimização de inventário

Para esta seção, é investigado se o tempo de sobrevivência dos agentes no *Buried-Brains* associa-se a otimização de inventário ou se decorre de variações estatísticas do ambiente. No Santuário, a substituição de equipamentos (*swaps*) apresenta um custo de oportunidade elevado. A otimização de inventário do agente exige o equilíbrio entre a depreciação dos itens atuais e o ganho potencial de longo prazo.

4.3.1 Análise de correlação entre otimização de inventário e progressão

Utilizou-se uma análise de regressão linear entre a frequência de trocas de itens e o andar máximo alcançado. A Figura 22 apresenta essa relação. O coeficiente angular da reta (*slope*) funciona como um indicador da eficiência marginal de cada troca executada pelo agente.

Figura 22 – Regressão linear entre frequência de trocas e profundidade alcançada



Fonte: Elaborado pelo autor (2025).

No experimento *Magnolia*, a arquitetura LSTM apresentou um *slope* de 1,87, enquanto o *baseline* PPO registrou 1,57. Os dados sugerem que cada troca realizada pelo modelo LSTM resulta em um incremento médio de 1,87 níveis de sobrevivência. No modelo PPO, o retorno por ação de troca é aproximadamente 19% inferior.

Essa diferença indica que o modelo recorrente executa trocas com maior impacto para a duração de episódio, se adaptando ao currículo implícito de agentes adversariais. A estrutura de memória interna possibilita a manutenção do estado latente do inventário de itens. Isso favorece a seleção de equipamentos que complementam lacunas específicas de atributos. O PPO, por assumir percepção Markoviana do estado atual, opera com predomínio de ganho imediato (*greedy selection*). Este padrão resulta em trocas com menor utilidade ao longo prazo, limitando a profundidade alcançada nos testes.

4.4 Baseline autômato determinístico em ambiente simplicado

Os resultados anteriores sugerem que a otimização de inventário é um componente essencial para alcançar andares maiores. Esta seção busca medir a disparidade entre o limite teórico-matemático e o desempenho empírico, com foco na mecânica de rotação de equipamentos como variável para a sobrevivência em andares elevados.

4.4.1 Formulação do teste de baseline em ambiente simplificado

Para estabelecer um referencial teórico, foi projetada simulação de combate estática e isolada. O objetivo foi quantificar o impacto do escalonamento de itens no andar médio alcançado pelo agente. O experimento consistiu em 2.000 simulações por estrato, visando reduzir a interferência da variância do sistema nos resultados.

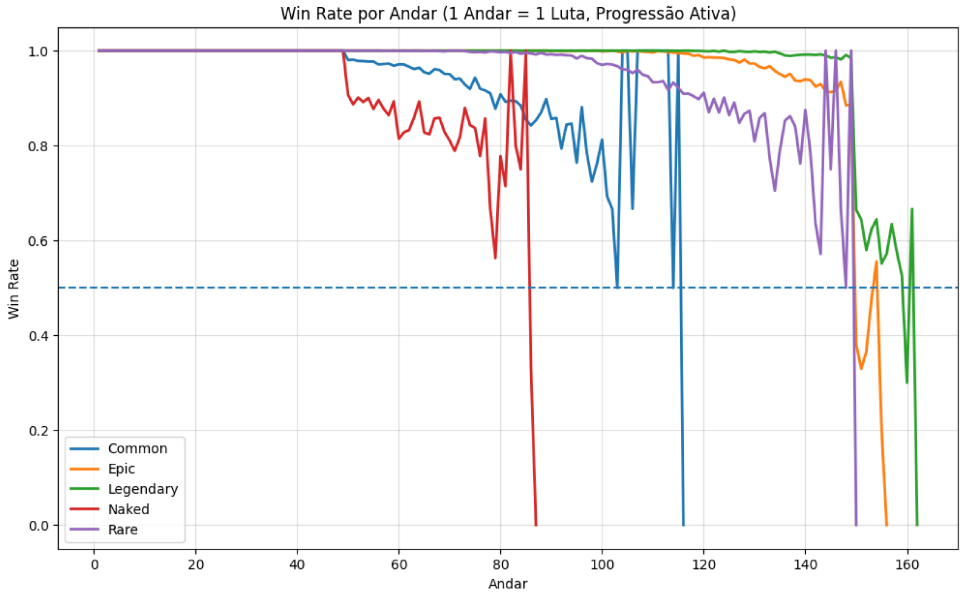
A modelagem, detalhada no notebook `Análise de Progressão de Atributos do Agente versus Ambiente.ipynb` (disponível em (SILVA, 2025)), utilizou cinco estratos: *Naked* (sem itens), *Common*, *Rare*, *Epic* e *Legendary*. Diferente do ambiente dinâmico, esta simulação operou sob condições de cenário ideal: vitalidade regenerada ao início de cada confronto, tempos de recarga (*cooldowns*) reiniciados e ausência de efeitos negativos persistentes entre encontros, nem efeitos de sala, assim como uma progressão linear (o agente não precisou escolher entre salas).

O autômato determinístico foi projetado com a heurística:

- defesa (*Stone Shield*) caso $HP < 60\%$;
- dano pesado (*Heavy Blow*);
- ataque rápido (*Quick Strike*);
- repouso (*Wait*).

A Figura 23 apresenta valores que indicam o limiar no qual o escalonamento adversárial supera a taxa de progressão de atributos do agente.

Figura 23 – Taxa de vitória por andar em ambiente estático



Fonte: Elaborado pelo autor (2025).

Os resultados sugerem que, mesmo sob condições ideais, o Tier *Legendary* apresenta perda de consistência próximo ao 160º andar. Contudo, no ambiente real, a arquitetura LSTM mantém uma progressão média de 70 andares, com picos de desempenho atingindo o teto de 160 andares.

Este desempenho pode estar associado a Variância Explicada (EV) de 80%, indicando uma estimativa de valor mais segura pela rede crítica que gera atualizações mais suaves na política. A eficiência observada pode estar associada à integração de características espaciais: a política recorrente identifica estados de menor probabilidade de transição para derrota e localizações de itens de restauração. sugerindo que a sobrevivência depende da mitigação do atrito durante a progressão.

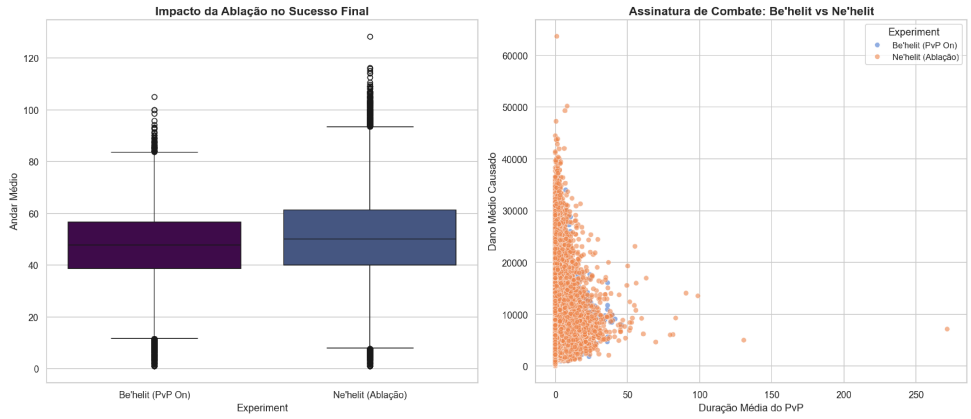
4.5 Análise de padrões de interações de soma não-zero

Esta seção analisa como a arquitetura dos agentes influencia as estratégias de manutenção de estado e a interação com outros agentes na simulação.

4.5.1 Análise da ablação de recompensas por iniciativa competitiva

O estudo de ablação realizado entre os experimentos *Be'helit* (incentivo direto ao combate) e *Ne'helit* (remoção de bônus por dano e encontro entre agentes) sugere uma mudança na eficiência tática dos agentes. Conforme apresentado na Figura 24, a ausência de reforço para o conflito não resultou em redução da frequência de engajamento em combate, mas em maior celeridade no encerramento dos confrontos.

Figura 24 – Comparativo de letalidade e duração de combates após ablação de bônus PvP



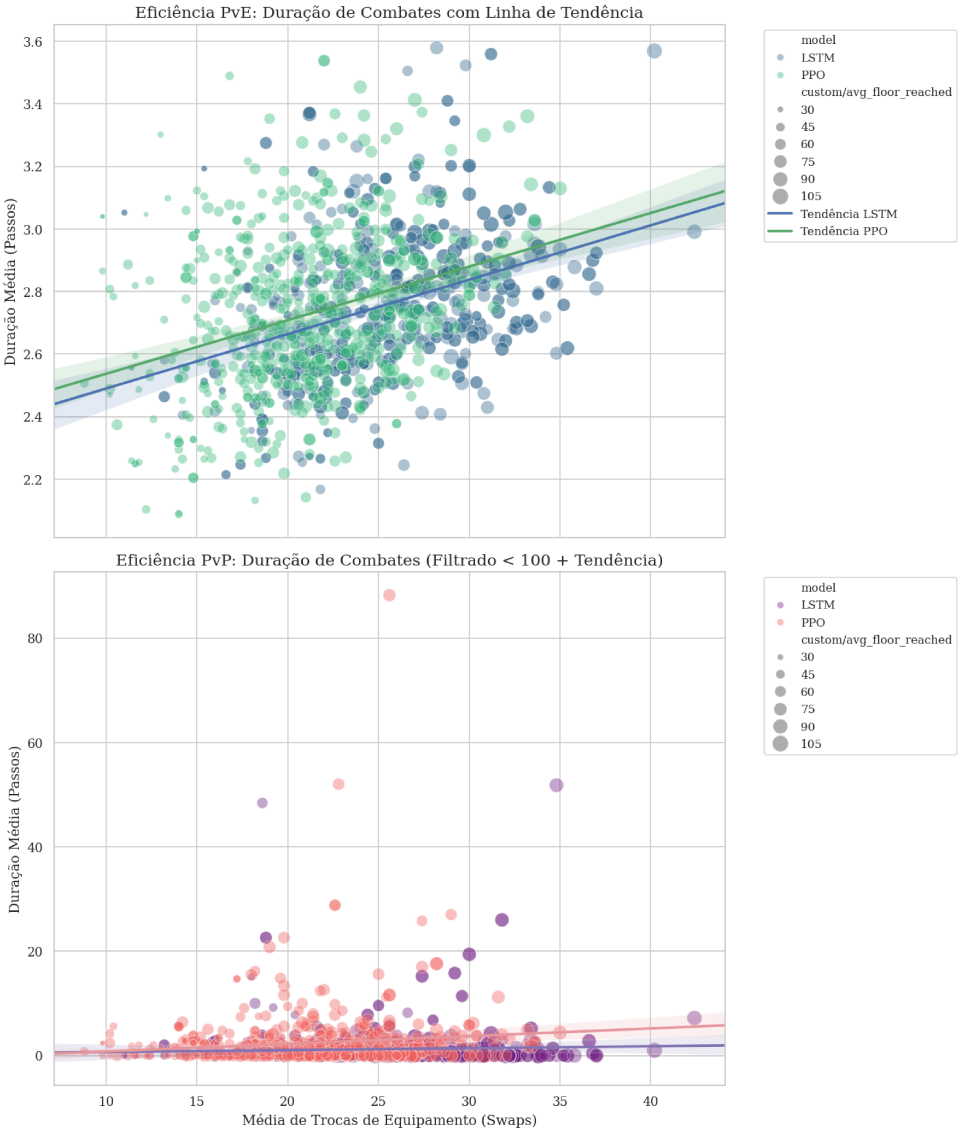
Fonte: Elaborado pelo autor (2025).

No cenário *Be'helit*, o modelo LSTM apresentou tendência a prolongar as trocas de dano, comportamento associado à maximização de recompensas como `REW_DMG_DEALT`. Com a transição para o *Ne'helit*, registrou-se redução de 65% na duração média das lutas e aumento de 17% no dano total infligido por ciclo. Os dados sugerem que, sem o incentivo para o acúmulo dano afligido, a política converge para uma configuração que minimiza o tempo de exposição ao risco ambiental. Sem uma recompensa de maximização de dano, combates entre agentes são apenas estados de desgaste, e assim, a política tende a convergir para a valorização de ações que minimizem a janela de tempo de vulnerabilidade.

4.5.2 Análise da correlação entre duração de combates e otimização de inventário

A Figura 25 detalha a correlação entre a frequência de substituição de equipamentos (*Swaps*) e o tempo de resolução de conflitos PvE e PvP.

Figura 25 – Correlação entre trocas de equipamento e duração de encontros



Fonte: Elaborado pelo autor (2025).

Os resultados indicam uma distinção na dinâmica de interação entre agentes. Enquanto a duração dos combates PvP no modelo PPO apresenta tendência ascendente e correlação positiva fraca ($r = 0,1260$), o que pode indicar uma tendência de menor eficiência frente a oponentes com maior volume de recursos, a curva do LSTM permanece estável ($r = 0,0433$).

Este padrão é consistente com a hipótese de que a política do agente LSTM possui maior capacidade de assimilação de recompensas esparsas, fruto de otimização eficiente de equipamentos, independentemente da progressão da dificuldade. Enquanto o PPO apresenta combates extensos em níveis avançados, um sinal de atrito por ineficiência de itens, o agente LSTM encerra confrontos entre agentes de forma célere. Este comportamento minimiza a exposição ao risco ambiental e otimiza o tempo disponível para a progressão entre andares.

4.5.3 Análise de convergência sub-ótima para equilíbrio de Nash preguiçoso

A análise de trajetórias identificou a convergência para estados de *Mexican Standoff* em estágios avançados do cenário *Magnolia*. Ao operar em modo determinístico (*model.predict*), agentes LSTM frequentemente selecionam a ação *Wait* de forma persistente a cada passo de tempo ao encontrarem outros agentes.

A ocorrência deste comportamento sugere uma resposta à estrutura de custos do ambiente. No cenário *Magnolia*, a inexistência de recompensas por dano e o risco de término do episódio (*permadeath*) tornam o valor esperado do combate negativo. Refletido em uma Variância Explicada de 80%, a rede de valor do LSTM sinaliza uma convergência para um ponto de equilíbrio local onde a inação mútua apresenta-se como política de maior utilidade esperada.

O contraste com a arquitetura PPO reforça esta interpretação. O modelo reativo, com menor capacidade preditiva (EV de 65%), tende a iniciar combates de baixa eficiência. Já o LSTM converge para uma política altamente avessa ao risco em modo determinístico. O padrão observado indica que o modelo converge para uma política que maximiza a probabilidade de sobrevivência em dilemas de coordenação de soma não-zero. Esse achado é consistente com a proposta do *BuriedBrains* de simular dilemas de preservação de estado, resultando em estabilidade social por aversão ao risco.

4.5.4 Análise da influência do sistema de reputação sobre a profundidade alcançada

No experimento *Magnolia*, o modelo LSTM apresentou Karma negativo médio de -0,6 em níveis de alta dificuldade. Isso sugere que a memória temporal permite ao modelo mapear que a eliminação de potenciais oponentes é estatisticamente mais favorável à integridade do que a coexistência.

Paralelamente, as correlações de negociações (*Bargains*) tornam-se nulas ($r \approx 0,00$) com o aumento da complexidade procedural. O fenômeno indica uma pressão ambiental que desfavorece comportamentos cooperativos/negociais em favor da eficiência operacional individual. O agente recorrente, sensível à escassez de recursos no longo prazo, reduz a frequência de interações sociais não mandatórias para priorizar a manutenção de seus indicadores de estado.

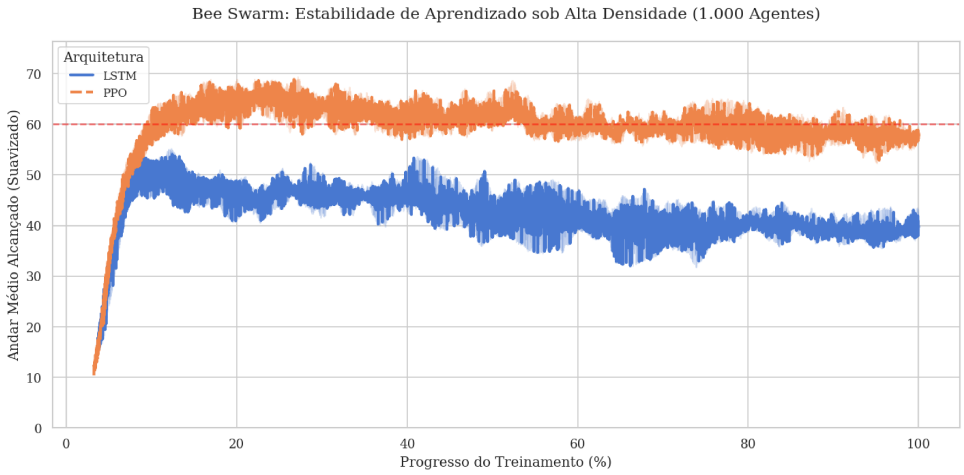
4.6 Limitações de memória e variância em alta amostragem

Embora a arquitetura LSTM apresente desempenho superior em cenários de densidade populacional controlada, o experimento *Bee Swarm* permitiu identificar limites operacionais da memória recorrente em condições de elevada entropia de dados.

4.6.1 Análise do experimento Bee Swarm e saturação de memória oculta

No cenário *Bee Swarm*, que integra 1.000 agentes simultâneos e uma janela de atualização de 32 passos, a arquitetura PPO obteve resultados superiores ao modelo LSTM. A Figura 26 apresenta os dados comparativos deste experimento.

Figura 26 – Desempenho comparativo no cenário de alta densidade Bee Swarm



Fonte: Elaborado pelo autor (2025).

Com um horizonte de retropropagação através do tempo (BPTT) limitado a 32 passos, o modelo recorrente não convergiu para a representação de padrões relevantes em meio às interações de múltiplos agentes. O processamento de dependências temporais em um fluxo de dados predominantemente ruidoso resultou em instabilidade no aprendizado, sendo caracterizada por um aumento na taxa de ações inválidas.

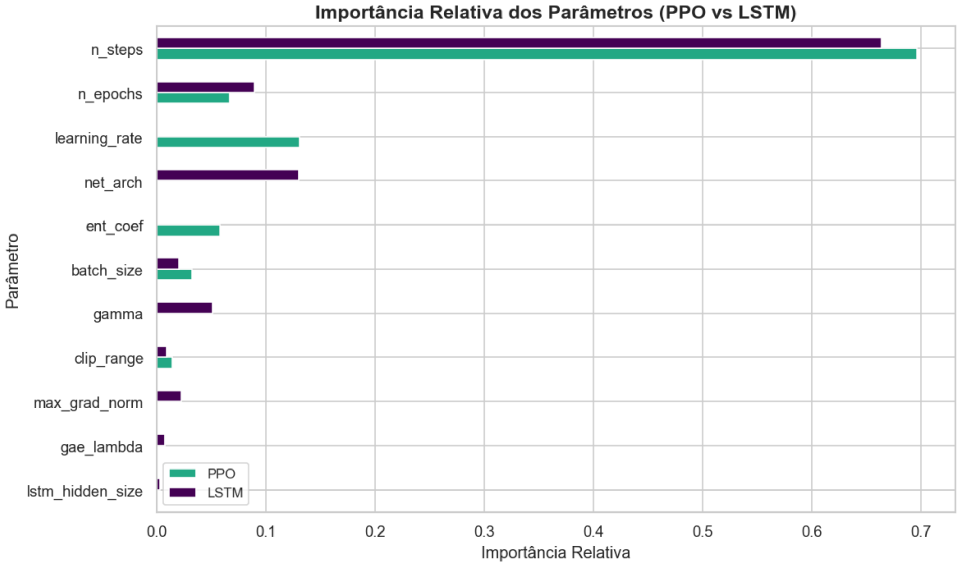
Em contrapartida, o agente PPO demonstrou maior adaptação ao cenário. Por se tratar de uma arquitetura baseada na Propriedade de Markov, o PPO restringe-se ao processamento do estado imediato em detrimento de estados passados. Esse comportamento sugere que, embora a memória facilite a modelagem de estratégias em ambientes de baixa entropia, a reatividade imediata é mais eficaz para a manutenção de métricas de desempenho em cenários de elevada entropia, embora a política emergente possa ser incapaz de alcançar maiores andares.

Com a probabilidade média de sobrevivência de 58 andares no cenário Bee Swarm e 60 nos demais, observa-se que a política aprendida pelo agente PPO converge para um patamar fixo de desempenho. Essa Estagnação sugere a adoção de uma política sub-ótima, simples e sem capacidade de escalabilidade, conforme corroborado pelos demais resultados.

4.6.2 Análise do estudo de otimização de hiperparâmetros do Optuna

A otimização de hiperparâmetros via *Optuna* indicou padrões de convergência que sugerem limitações estruturais na arquitetura PPO. A análise da importância dos parâmetros, detalhada na Figura 27, indica que o desempenho de ambos os modelos é sensível ao horizonte de coleta (`n_steps`), embora por mecanismos distintos.

Figura 27 – Importância relativa dos hiperparâmetros para PPO e LSTM



Fonte: Elaborado pelo autor (2025).

Para o modelo PPO, a configuração ótima convergiu para um horizonte de 512 passos temporais como o parâmetro de maior relevância (0,69). Este valor está correlacionado à demanda por maior volume de amostragem, possivelmente mitigando a ausência de memória interna, sendo uma janela quatro vezes superior à utilizada pelo LSTM (128 passos). Apesar do aumento na amostragem, a Variância Explicada não excedeu 65%, o que sugere um platô de aprendizado para arquiteturas puramente reativas neste domínio.

A sensibilidade à arquitetura neural (`net_arch`) também apresentou disparidades. No modelo LSTM, a configuração da rede é o segundo fator mais influente no desempenho (0,14), enquanto no PPO a arquitetura não consta entre os cinco parâmetros principais.

Este resultado indica que, para agentes recorrentes, a organização das camadas neurais é determinante para a mapeamento de dependências temporais em estados passados. No caso do PPO, o limitador de desempenho parece estar vinculado à natureza Markoviana da entrada de dados, e não à capacidade de processamento das camadas densas. Portanto, o aumento da complexidade da rede não soluciona a ausência de contexto temporal no ambiente *BuriedBrains*.

4.7 Síntese dos Resultados

Esta seção integra os resultados experimentais obtidos, analisando o efeito da memória recorrente no ambiente *BuriedBrains*. A observação conjunta das métricas de desempenho, processamento de estados e padrões de ação sugere que o ambiente possui características não-Markovianas que agentes puramente reativos não são capazes de superar.

A Tabela 9 resume as métricas extraídas dos cenários *Magnolia* e *Bee Swarm*.

Tabela 9 – Síntese comparativa de desempenho e parâmetros operacionais

Dimensão	**PPO (Reativo)**	**LSTM (Recorrente)**
Desempenho	Estagnação no andar 60.	Alcance do nível 70 com tendência de ascensão.
Variância Explicada (EV)	65%: Representação do estado atual.	80%: Captura de dependências temporais na função de valor.
Entropia da Política	0,33: Redução da exploração e tendência ao determinismo.	0,39: Manutenção de diversidade de ações.
Índice de Pânico (IP)	9,2: Alta frequência de troca de comandos.	4,4: Redução da oscilação na política de ações.
Eficiência (Slope)	1,57: Uso imediato de itens.	1,87: Otimização do uso de itens por ganho de retorno.
Sensibilidade HPO	Dependência de *n_steps* (512).	Dependência da arquitetura (*net_arch*).
Padrão de Ação	Dependência estrita do estado atual (s_t).	Convergência para políticas de horizonte temporal estendido.
Escalabilidade	Desempenho estável com 1000 agentes.	Escalabilidade temporal em horizontes extensos.

Fonte: Elaborado pelo autor (2025).

Os dados indicam que a arquitetura PPO, embora apresente estabilidade em cenários de alta densidade de agentes (como o *Bee Swarm*), demonstra limitações na formulação de políticas complexas. O diagnóstico via HPO sugere que, mesmo com a ampliação dos horizontes estatísticos (*n_steps* de 512), o PPO mantém-se limitado a uma Variância Explicada de 65% (que nos outros cenários, baixava para a média de 50%).

Em contraste, o modelo LSTM exibiu uma estimativa de valor mais fidedigna em

relação às transições de estado do ambiente. A manutenção dos níveis de entropia e a redução no Índice de Pânico (IP) sugerem que a memória recorrente favorece a convergência para políticas com maior capacidade de generalização das features do ambiente. A sensibilidade do LSTM à configuração da rede (*net_arch*) evidencia a dependência da camada oculta frente à capacidade de extração de características temporais

Este padrão reflete-se na utilidade das combinações de equipamentos, em que o LSTM apresenta eficiência 19% superior ao *baseline* na otimização de inventário para sobrevivência.

Portanto, é fortalecida a hipótese de que o ambiente *BuriedBrains* de fato propõe complexidade de horizonte temporal para agentes com capacidade de assimilação de crédito temporal convergirem em políticas mais eficientes nos termos abordados.

Estas evidências fundamentam as conclusões e as proposições de experimentação conduzidos.

5 CONCLUSÃO

Este trabalho teve como foco principal o desenvolvimento do *BuriedBrains*, um ambiente multiagente procedural projetado para avaliar o processamento de dependências temporais em cenários de observabilidade parcial. A investigação determinou como as mecânicas de jogos *roguelike* influenciam a aprendizagem de agentes autônomos, utilizando a comparação entre arquiteturas reativas e recorrentes para validar os requisitos de processamento de informação do ambiente.

Os resultados demonstram que o *BuriedBrains* impõe restrições de desempenho a modelos puramente reativos devido à sua estrutura de estados. O modelo PPO manteve um patamar de recompensa constante no andar 60, independentemente de ajustes em hiperparâmetros ou da extensão dos horizontes de coleta (*rollouts*) sugere a convergência para uma política subótima incapaz de escalar para andares maiores. A convergência da Variância Explicada em valores inferiores a 65% no PPO corrobora a hipótese de que o ambiente possui dependências não-Markovianas. Portanto, a tarefa exige a manutenção de estados internos para mitigar a incerteza e a irreversibilidade das trajetórias.

A aplicação da arquitetura LSTM confirmou que a estrutura do *BuriedBrains* demanda memória para a otimização das políticas. O modelo recorrente superou o desempenho do PPO em 24% no teste de avaliação de otimização de hiperparâmetros e em 10,1% no cenário *Magnolia*, indicando que o ambiente favorece algoritmos capazes de realizar a atribuição de crédito em eventos distantes, como a otimização de inventário e a eficiência em turnos de combate. A convergência para políticas de alta agressividade demonstra que a função de recompensa induz pressões táticas que priorizam a sobrevivência via eliminação de inimigos.

O experimento *Bee Swarm* estabeleceu os limites operacionais das políticas aprendidas. Observou-se que o aumento na densidade de agentes pode comprometer a capacidade de representação de dependências temporais no estado oculto da rede. Esse resultado indica que o desempenho é sensível à preservação do sinal de recompensa em cenários com alta densidade de variáveis estocásticas, fator crítico para o estudo de coordenação em sistemas multiagente.

O trabalho cumpriu os objetivos propostos ao fornecer uma plataforma capaz de mensurar a tomada de decisão sob incerteza. O *BuriedBrains* consolida-se como um ambiente para experimentação em Aprendizado por Reforço, permitindo a análise de estabilidade de convergência, capacidade de generalização e representação de estados em sistemas dinâmicos.

Para fins de contribuição acadêmica, o código-fonte e o simulador foram disponibili-

zados sob licença aberta MIT, servindo de fomento pesquisas futuras em ambientes multiagente procedurais inspirados em *roguelike* para agentes de memória.

5.1 Trabalhos Futuros

Com base nos resultados, propõem-se as seguintes frentes de investigação:

- **Arquiteturas de Atenção Temporal:** Implementação de modelos baseados em *Transformers* para avaliar se mecanismos de atenção global mitigam os gargalos de representação observados no LSTM em cenários de alta densidade.
- **Ablação perceptual:** Investigação de como variações no campo de visão dos agentes alteram o desempenho assintótico, quantificando a correlação entre densidade de informação visual e a estabilidade da política.
- **Mecânicas de custo temporal:** Implementação de sistemas de degradação de atributos ou custos para a ação de espera (*Wait*), visando observar como o risco de inação altera a transição entre comportamentos defensivos e ofensivos.
- **Agentes Aadversariais exógenos:** Introdução de inimigos capazes de navegação global para atuar como antagonistas, nos Santuários, potencialmente intensificando dilemas de cooperação e competição.
- **Análise de populações mistas:** Treinamento sob diferentes perfis de recompensa para analisar como sistemas de reputação influenciam estratégias de sobrevivência coletiva.
- **Balanceamento da curva de dificuldade:** Ajuste na progressão dos desafios procedurais para identificar o equilíbrio entre a taxa de aprendizado e a exploração de níveis de profundidade superiores.

REFERÊNCIAS

Nussygame. **Buriedbornes: Dungeon RPG**. 2016. Desenvolvido por ohNussy. Software (Android/iOS/Web). Disponível em: <https://nussygame.com/buriedbornes/>. Acesso em: 28 dez. 2025.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. 2. ed. Cambridge: The MIT Press, 2018.

LAKE, B. M.; ULLMAN, T. D.; TENENBAUM, J. B.; GERSHMAN, S. J. Building machines that learn and think like people. **Behavioral and Brain Sciences**, Cambridge University Press, v. 40, 2017.

VINYALS, O.; BABUSCHKIN, I.; CZARNECKI, W. M. *et al.* Grandmaster level in StarCraft II using multi-agent reinforcement learning. **Nature**, v. 575, p. 350–354, 2019.

KÜTTLER, H.; NARDELLI, N.; MILLER, A. *et al.* The NetHack learning environment. In: **Conference on Neural Information Processing Systems (NeurIPS)**. Vancouver: [s.n.], 2020. v. 34.

SUAREZ, J.; DU, Y.; ISOLA, P.; MORDATCH, I. Neural MMO: A massively multiagent game environment for training and evaluating intelligent agents. **arXiv preprint arXiv:1903.00784**, 2019.

SCHULMAN, J.; WOLSKI FILIP E DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. **arXiv preprint arXiv:1707.06347**, 2017.

PARK, R. *et al.* Unpacking DPO and PPO: Disentangling best practices for learning from preference feedback. **arXiv preprint arXiv:2406.09279**, 2024.

WOOLDRIDGE, M. **An Introduction to MultiAgent Systems**. 2. ed. [S.l.]: John Wiley & Sons, 2009.

MYERSON, R. B. **Game Theory: Analysis of Conflict**. [S.l.]: Harvard University Press, 1991.

NICKEL, M.; KIELA, D. Poincaré embeddings for learning hierarchical representations. In: **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2017. v. 30.

CANNON, J. W.; FLOYD, W. J.; KENYON, R.; PARRY, W. R. Hyperbolic geometry. In: **Flavors of geometry**. [S.l.]: MSRI Publications, 1997. v. 31, p. 59–115.

BONNABEL, S. Stochastic gradient descent on Riemannian manifolds. **IEEE Transactions on Automatic Control**, IEEE, v. 58, n. 9, p. 2217–2229, 2013.

TANG, Y.; KEJRIWAL, M. Overview of emergent abilities in AI. **World Scholars Review**, 2025.

KAELBLING, L. P.; LITTMAN, M. L.; CASSANDRA, A. R. Planning and acting in partially observable stochastic domains. **Artificial Intelligence**, v. 101, n. 1-2, p. 99–134, 1998.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 1997.

ERDőS, P.; RéNYI, A. On random graphs I. **Publicationes Mathematicae Debrecen**, v. 6, p. 290–297, 1959.

BOLLOBáS, B. **Random Graphs**. 2. ed. [S.l.]: Cambridge University Press, 2001.

FREEMAN, L. C. Centrality in social networks conceptual clarification. **Social Networks**, Elsevier, v. 1, n. 3, p. 215–239, 1978.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł.; POLOSUKHIN, I. Attention is all you need. In: **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2017. v. 30.

BELLEMARE, M. G.; NADDAF, Y.; VENESS, J.; BOWLING, M. The Arcade learning environment: An evaluation platform for general agents. **Journal of Artificial Intelligence Research**, v. 47, p. 253–279, 2013.

FromSoftware. **Dark Souls**. Tóquio, Japão: Namco Bandai Games, 2011. Software (PlayStation 3 / Xbox 360).

SILVA, I. **BuriedBrains Roguelike RL Environment**. GitHub, 2025. Disponível em: <https://github.com/maelsilvatt/buriedbrains-roguelike-rl-env>. Acesso em: 29 jan. 2026.

GOOGLE. **Nano Banana Pro 3: Gemini 3 Pro Image Model**. 2025. <https://blog.google/intl/pt-br/produtos/explore-e-encontre-respostas/nano-banana-pro/>. Acesso em: 3 jan. 2026.

SILVA, I. **Repositório do Trabalho de Conclusão de Curso (TCC)**. GitHub, 2025. Disponível em: <https://github.com/maelsilvatt/trabalho-de-conclusao-de-curso-ufc/tree/tcc>. Acesso em: 28 dez. 2025.

APÊNDICE A – SCRIPTS DE EXPERIMENTOS E REPRODUTIBILIDADE

Devido à extensão dos arquivos de lote (*.bat*) e para evitar redundância, este apêndice apresenta a estrutura lógica de execução através do script *Magnolia*. As variações paramétricas de cada bateria adicional, que sobrescrevem as definições do arquivo `train.py`, bem como as instruções de reprodução, estão disponíveis no repositório oficial:

<https://github.com/maelsilvatt/trabalho-de-conclusao-de-curso-ufc/tree/tcc/experimentos>

A.1 Exemplo de Script

Código-fonte 1 – Script do experimento Magnolia

```
@echo OFF
call corevenv\Scripts\activate.bat

:: --- CONFIGURACAO: Magnolia (32 Agentes - Otimizado Optuna) ---
:: Baseado nos resultados exatos do HPO para cenario Sparse.
:: CHAMPION_LSTM_PARAMS = {
::     "learning_rate": 2.06e-4, "n_steps": 128, "batch_size": 256,
::     "n_epochs": 5, "gamma": 0.99, "gae_lambda": 0.9,
::     "ent_coef": 5.57e-4, "max_grad_norm": 0.77, "clip_range": 0.2,
::     "lstm_hidden_size": 256, "net_arch": [256, 256]
:: }
:: CHAMPION_PPO_PARAMS = {
::     "learning_rate": 6.16e-5, "n_steps": 512, "batch_size": 128,
::     "n_epochs": 7, "gamma": 0.99, "gae_lambda": 0.95,
::     "ent_coef": 1.95e-3, "max_grad_norm": 0.5, "clip_range": 0.3,
::     "net_arch": [256, 256]
:: }
:: Rewards ajustadas para ablacao de recompensas PvP
:: REW_DMG_DEALT_SCALAR = 0.0 (APENAS NO PVP. PVE PERMANECE
   INALTERADO)
:: REW_DMG_TAKEN_SCALAR = 0.0  (APENAS NO PVP. PVE PERMANECE
   INALTERADO)
:: REW_MEET_BONUS = 0.0          (destrancar porta)
:: Bonus de saida continua igual (para compensar o tempo gasto)
:: Reward por matar permanece igual (incentivo ao combate)
```

```
:: Versao 2.3.2
set STEPS=20000000
set NUM_AGENTS=32
set VERBOSE=0

echo [FASE 1] Iniciando PPO Champion...
for %%S in (111 222 333 444 555 666) do (
    python _train_optuna_params.py --algo ppo --total_timesteps %
        STEPS% --suffix CHAMPION_PPO_SPARSE_S%%S --num_agents %
        NUM_AGENTS% --seed %%S --verbose %VERBOSE% --no_hall_of_fame
)

echo [FASE 2] Iniciando LSTM Champion...
for %%S in (111 222 333 444 555 666) do (
    python _train_optuna_params.py --algo lstm --total_timesteps %
        STEPS% --suffix CHAMPION_LSTM_SPARSE_S%%S --num_agents %
        NUM_AGENTS% --seed %%S --verbose %VERBOSE% --no_hall_of_fame
)

pause
```

APÊNDICE B – TRANSPARÊNCIA ÉTICA SOBRE O USO DE LLMS COMO FERRAMENTAS DE AUXÍLIO EDITORIAL

Em conformidade com a preocupação de produção textual que inclui o uso de LLMs para auxílio e a insegurança acerca disso, declara-se a utilização da ferramenta **Gemini Pro** como assistente de curadoria editorial, sob supervisão e responsabilidade integral do autor.

B.1 Metodologia de Refinamento e Personas

A redação deste trabalho não foi atribuída a um modelo gerador de texto, mas ferramentas de inteligência artificial foram utilizadas para refinamento técnico e identificação de erros comuns em escrita acadêmica.

- **T-Rex (Sintaxe e ABNTEX2):** Utilizado para orientar a escrita em termos de sobriedade técnica, sinalizando adjetivação desnecessária, padronizando a linguagem probabilística (e.g., substituindo "prova que"por "sugere que") e estruturação LaTeX.
- **Shabriri (Revisão Técnica):** Seu papel foi interpretar um revisor técnico, identificando rastros de antropomorfização, causalidade indevida e imprecisões em definições técnicas locais.

B.2 Autoria e Processo de Trabalho

Para total transparência sobre o processo de escrita que, em algum momento, utilizou-se de ferramentas geradoras, segue em anexo o link para o repositório Github, com histórico de versionamento e revisões.

- **Histórico de Versão (Git):** O repositório registra *commits* incrementais realizados entre junho de 2025 e janeiro de 2026, documentando o progresso de escrita deste documento, assim como notebooks documentando as descobertas que compuseram o capítulo 4.
- **Repositório:** <https://github.com/maelsilvatt/trabalho-de-conclusao-de-curso-ufc>

[1] [2]

[1] *Nota: Este apêndice de transparência — redigido para atestar a autoria humana assistida — quando submetido ao classificador GPTZero, foi avaliado com 100% de probabilidade de origem sintética. Esta observação serve para reforçar as limitações de classificadores estatísticos diante de textos técnicos. Portanto, é reforçada a necessidade de artefatos de histórico do autor, como o repositório do Github.*

[2] *Nota: O rodapé também não foi poupado de ser avaliado com 100% de probabilidade de origem sintética.*