UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Luís Henrique Chesani

AVALIAÇÃO DE MÉTODOS DE MACHINE LEARNING PARA CONTROLE DE DIFICULDADE EM UM JOGO ESTILO SOULSLIKE

Santa Maria, RS
2025

Luís Henrique Chesani

AVALIAÇÃO DE MÉTODOS DE MACHINE LEARNING PARA CONTROLE DE DIFICULDADE EM UM JOGO ESTILO SOULSLIKE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da computação, Área de Concentração em Área de concentração do CNPq, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da computação**.

Orientador: Prof. Joaquim Vinicius Carvalho Assunção

Santa Maria, RS
2025

Luís Henrique Chesani

AVALIAÇÃO DE MÉTODOS DE MACHINE LEARNING PARA CONTROLE DE DIFICULDADE EM UM JOGO ESTILO SOULSLIKE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da computação, Área de Concentração em Área de concentração do CNPq, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da computação**.

Aprovado em 1º de abril de 2025:

_____

Joaquim Vinicius Carvalho Assunção, Dr. (UFSM)
(Presidente/Orientador)

_____

Cesar Tadeu Pozzer, Dr. (UFSM)

_____

Sergio Luis Sardi Mergen, Dr. (UFSM)

Santa Maria, RS
2025

RESUMO

AVALIAÇÃO DE MÉTODOS DE MACHINE LEARNING PARA CONTROLE DE DIFICULDADE EM UM JOGO ESTILO SOULSLIKE

AUTOR: Luís Henrique Chesani
Orientador: Joaquim Vinicius Carvalho Assunção

Este projeto tem como objetivo explorar e avaliar diferentes métodos de machine learning (GéRON, 2022) para controlar dinamicamente os níveis de dificuldade em um jogo de *soulslike* desenvolvido na Unity. Utilizando o Machine Learning - Agents (ML-Agents), uma ferramenta de machine learning da Unity, o estudo se concentra na implementação de algoritmos de aprendizado de máquina para ajustar a dificuldade do jogo de acordo com o desempenho do jogador em tempo real. O foco é investigar como tais técnicas de aprendizado por reforço podem ser aplicadas para adaptar a experiência do usuário, proporcionando um equilíbrio entre desafio e diversão. A pesquisa também visa analisar o impacto dessas abordagens na experiência do jogador, na fluidez do jogo e na longevidade da experiência de jogo. A comparação entre diferentes modelos de machine learning permitiu simular diferentes níveis de dificuldade devido à diferença nos métodos de aprendizagem, garantindo que os jogadores encontrem um nível de desafio adequado, evitando frustração ou falta de interesse devido às chamadas "dificuldades artificiais", que se baseiam em simplesmente aumentar e diminuir os *status dos inimigos.*

Palavras-chave: *Machine Learning. Jogos. Simulação de dificuldade.*

ABSTRACT

EVALUATION OF MACHINE LEARNING METHODS FOR DIFFICULTY CONTROL IN A SOULSLIKE-STYLE GAME

AUTHOR: Luís Henrique Chesani
ADVISOR: Joaquim Vinicius Carvalho Assunção

This project aims to explore and evaluate different machine learning methods (GéRON, 2022) for dynamically controlling difficulty levels in a soulslike game developed in Unity. Using Machine Learning - Agents (ML-Agents), a machine learning toolkit by Unity, the study focuses on implementing machine learning algorithms to adjust the game's difficulty in real-time based on player performance. The objective is to investigate how such techniques of reinforcement learning can be applied to adapt the user experience, ensuring a balance between challenge and enjoyment. The research also seeks to analyze the impact of these approaches on player experience, game flow, and the longevity of gameplay. By comparing different machine learning models, the study will simulate various difficulty levels resulting from distinct learning methods, ensuring that players encounter an appropriate level of challenge while avoiding frustration or disengagement caused by so-called "artificial difficulties," which are typically based on simply increasing or decreasing enemy stats.

Keywords: *Machine Learning. Games. Difficulty adjust.*

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A experiência do jogador em jogos eletrônicos é profundamente impactada pelo nível de dificuldade que o jogo apresenta. Tradicionalmente, os jogos ajustam a dificuldade de forma estática, muitas vezes por meio de alterações nos atributos dos inimigos, como vida, dano e resistência. Embora essa abordagem seja funcional, ela pode levar a experiências de jogo desbalanceadas, frustrantes ou pouco envolventes, principalmente quando não considera a habilidade e o estilo individual de cada jogador.

Com os avanços em Machine Learning (ML), surge a possibilidade de tornar a dificuldade dos jogos mais adaptativa, respondendo ao desempenho do jogador e oferecendo uma curva de aprendizado mais equilibrada e personalizada.

Este trabalho propõe a avaliação de diferentes métodos de Machine Learning aplicados ao controle de dificuldade em jogos, buscando uma correlação entre diferentes algoritmos para diferentes dificuldades, utilizando como base um jogo do gênero soulslike desenvolvido na Unity Engine. Foram exploradas abordagens como Behavioral Cloning (BC), uma técnica supervisionada que visa imitar comportamentos de jogadores humanos, e algoritmos de aprendizado por reforço, como Proximal Policy Optimization (PPO) e Soft Actor-Critic (SAC), com o uso da ferramenta ML-Agents da Unity(Unity Technologies, 2023).

O uso do ML-Agents se destaca por ser uma solução prática e acessível que permite a integração direta de algoritmos de aprendizado de máquina ao ambiente de desenvolvimento Unity. Essa ferramenta abstrai diversas complexidades da implementação de agentes inteligentes, oferecendo uma interface intuitiva para treinar e avaliar comportamentos por meio de simulações em tempo real. Assim, ela representa uma ponte entre os conceitos teóricos de Machine Learning e a aplicação concreta no desenvolvimento de jogos, possibilitando que desenvolvedores concentrem seus esforços no design e na análise de resultados, sem a necessidade de construir toda a infraestrutura de aprendizado do zero.

1.1 DEFINIÇÃO DO PROBLEMA

As abordagens para solução de dificuldades artificiais, embora simples e amplamente utilizadas, tendem a ignorar as diferenças individuais de habilidade, experiência e estilo de jogo dos jogadores. Como consequência, podem gerar experiências desbalanceadas: ou muito fáceis e entediantes, ou excessivamente difíceis e frustrantes. Este trabalho tem como objetivo verificar se os algoritmos da ferramenta ML-Agents, da Unity, podem ser solução para este tipo de problema em jogos do estilo soulslike. Especificamente se

cada um destes algoritmos (PPO, SAC, e BC) podem ser utilizados como representações distintas de níveis de dificuldade no jogo.

1.2 OBJETIVOS

Objetivo Geral

Avaliar o desempenho de diferentes técnicas de Machine Learning — especificamente, Behavioral Cloning (BC), Proximal Policy Optimization (PPO) e Soft Actor-Critic (SAC) — na tarefa de ajustar a dificuldade de um jogo soulslike. O objetivo é investigar como cada técnica responde à variação de habilidade entre jogadores, propondo alternativas mais flexíveis e personalizadas em relação aos métodos tradicionais de balanceamento de dificuldade. Para isso, foi desenvolvido um ambiente de jogo do tipo soulslike na Unity, integrado à ferramenta ML-Agents, permitindo a simulação e o treinamento de agentes inteligentes. Essas escolhas foram feitas pois jogos do gênero não costumam ter ajuste de dificuldade, e também a praticidade do ML-Agents da Unity. Ao final, pretende-se identificar se as abordagens podem ser correlacionadas à diferentes níveis de dificuldade.

Objetivos Específicos

- *Desenvolver um ambiente de jogo do tipo soulslike na Unity que permita integração com agentes inteligentes via ML-Agents;*
- *Implementar agentes de treino utilizando as técnicas BC, PPO e SAC, simulando comportamentos de inimigos adaptativos;*
- *Projetar métricas para avaliar o desempenho dos agentes em termos de adaptação à habilidade do jogador, fluidez de jogabilidade e engajamento;*
- *Realizar testes controlados com diferentes perfis de jogadores e analisar os resultados obtidos;*
- *Comparar as abordagens de aprendizado com base em suas vantagens, limitações e impacto na experiência do usuário;*
- *Discutir as implicações dos resultados obtidos para o desenvolvimento de sistemas de dificuldade adaptativa em jogos futuros;*
- *Avaliar a importância do uso do ML-Agents como uma ferramenta prática e acessível para desenvolvedores, por oferecer integração nativa com a Unity, suporte a múltiplos algoritmos de aprendizado e uma estrutura modular que facilita a criação, treinamento e avaliação de agentes inteligentes em ambientes interativos.*

1.3 ORGANIZAÇÃO DESTA MONOGRAFIA

No próximo capítulo, é apresentado o conceito de dificuldade dinâmica em jogos, contextualizando seu papel no design de jogos modernos e os problemas associados às abordagens tradicionais. No Capítulo 3, é descrita a implementação do jogo do tipo souls-like na Unity, que servirá como ambiente de testes para os agentes de Machine Learning. No Capítulo 4, é abordada a integração da ferramenta ML-Agents com o jogo e o funcionamento da comunicação entre o ambiente e os agentes externos. O Capítulo 5 é dedicado à descrição detalhada da implementação dos agentes de Machine Learning, incluindo os métodos utilizados: Behavioral Cloning (BC), Proximal Policy Optimization (PPO) e Soft Actor-Critic (SAC). No Capítulo 6 é explicada a metodologia de testes utilizada para avaliar o desempenho e a efetividade dos diferentes modelos. Os Capítulos 7 e 8 apresentará os testes realizados e seus respectivos resultados, analisando o impacto das abordagens propostas na experiência do jogador. Todas as seções são revisitadas no Capítulo 9, onde também é apresentada a conclusão final com base no estudo desenvolvido. Por fim, no último capítulo são discutidas possíveis direções para trabalhos futuros que possam aprimorar e expandir esta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos fundamentais que embasam o desenvolvimento deste trabalho. Inicialmente, é feita uma contextualização sobre o gênero soulslike, com ênfase em suas principais características e desafios de design. Em seguida, são discutidos trabalhos relacionados ao ajuste de dificuldade em jogos digitais, com foco nas abordagens que utilizam inteligência artificial e aprendizado de máquina. Essa fundamentação fornece o suporte necessário para a definição da metodologia e para a análise dos resultados obtidos.

2.1 GÊNERO SOULSLIKE

O termo soulslike refere-se a um subgênero de jogos eletrônicos inspirado fortemente na série Souls, desenvolvida pela FromSoftware, com destaque para títulos como Demon's Souls (FromSoftware, 2009), Dark Souls (FromSoftware, 2011) e seus sucessores. Esses jogos se destacaram por oferecerem uma experiência desafiadora e punitiva, porém sem ajuste de dificuldade.

Com base nas principais mecânicas encontradas nesse tipo de jogo, foram selecionadas as seguintes ações para compor a base comportamental dos agentes neste trabalho:

Jogador (humano):

1. ***Andar**: movimento comum para todas as direções, com velocidade constante;*
2. ***Desviar**: esquiva com invulnerabilidade temporária;*
3. ***Travar a mira**: a câmera fica centralizada no inimigo, facilitando acertar a direção do ataque;*
4. ***Defender**: bloqueio dos ataques do inimigo. se feito no momento exato, gera um parry, senão a defesa normal é acionada e o jogador toma dano reduzido sem perder o estado de defesa;*
5. *Atacar: ataque corpo a corpo.*

Inimigo (agente):

1. ***Andar**: vai em direção ao jogador humano;*
2. ***Atacar**: diferentes variações de ataque são detalhadas na seção 3.2 (é nessas variações que os algoritmos do ML-Agents são testados).*

2.2 TRABALHOS RELACIONADOS

A adaptação dinâmica da dificuldade em jogos é um tema recorrente na literatura de desenvolvimento de jogos e aprendizado de máquina, sendo abordado por diversas metodologias que buscam equilibrar desafio e engajamento. Este trabalho se insere nesse contexto, com foco na comparação de diferentes técnicas de Machine Learning aplicadas à modulação da dificuldade de forma inteligente.

(MACHADO, 2013) propôs uma metodologia baseada em aprendizado de máquina para modelagem de jogadores, destacando a importância de compreender padrões de comportamento para personalizar a experiência de jogo. De forma semelhante, (SAMPAYO-VARGAS et al., 2013) investigaram os efeitos da dificuldade adaptativa em jogos educacionais, concluindo que ajustes baseados em desempenho podem melhorar significativamente a motivação e a aprendizagem dos usuários.

Mais recentemente, (FUCHS; GIESEKE; DOCKHORN, 2024) propuseram uma abordagem híbrida que combina Imitation Learning e Reinforcement Learning para ajustar dinamicamente a dificuldade com base no estilo individual de cada jogador. O trabalho destaca a personalização como fator-chave na longevidade da experiência de jogo.

(RAHIMI et al., 2023) aplicaram Reinforcement Learning contínuo em um jogo de memória visual, com foco na adaptação da dificuldade em tempo real, demonstrando benefícios em termos de fluidez e retenção do jogador. Essa abordagem também mostra o potencial do RL em ambientes sensíveis ao tempo e à performance do usuário.

Outra contribuição relevante é de (OR; KOLOMENKIN; SHABAT, 2021), que utilizaram redes neurais profundas para implementar um sistema de Dynamic Difficulty Adjustment (DDA) com restrições de experiência do usuário e consistência de jogabilidade. Esse estudo reforça a ideia de que o ajuste da dificuldade não deve comprometer a coerência do jogo.

No cenário nacional, (OLIVEIRA; CHAIMOWICZ, 2023) exploraram o uso de aprendizado por reforço para DDA em jogos digitais, apresentando uma metodologia de avaliação baseada em métricas objetivas e subjetivas. Seu trabalho fornece um ponto de partida importante para experimentos comparativos entre diferentes algoritmos.

(TAGLIARO, 2022) apresentou uma implementação de sistemas de dificuldade adaptativa voltada para jogos desafiadores, com foco em uma experiência personalizada e equilibrada para o jogador. Utilizando conceitos de adaptação dinâmica em tempo real, o autor explorou estratégias que ajustam o comportamento de inimigos com base no desempenho do jogador, em um cenário inspirado na lógica de jogos do tipo soulslike. O trabalho reforça a importância de considerar aspectos subjetivos da experiência do usuário, propondo soluções que vão além da simples alteração de atributos numéricos, como vida e dano, e que priorizam a manutenção do engajamento e da fluidez do jogo. Essa proposta se alinha

com os objetivos deste projeto, ao evidenciar o potencial de abordagens adaptativas na construção de desafios mais responsivos e imersivos.

A presente pesquisa se diferencia ao realizar uma análise comparativa entre três abordagens distintas — Behavioral Cloning, Proximal Policy Optimization e Soft Actor-Critic — dentro de um ambiente controlado construído na Unity, especificamente voltado para o gênero soulslike. Ao utilizar o ML-Agents como ferramenta de treinamento, busca-se entender qual técnica oferece melhor adaptação ao perfil do jogador, contribuindo para a construção de experiências de jogo mais imersivas e equilibradas.

2.3 MÉTODOS DE APRENDIZADO DE MÁQUINA DO ML-AGENTS

O ML-Agents é uma ferramenta da Unity que permite aplicar técnicas de aprendizado de máquina diretamente em jogos criados na engine. Ela oferece diferentes algoritmos que ajudam a treinar agentes para aprender comportamentos, seja por tentativa e erro (aprendizado por reforço) ou por imitação (aprendizado supervisionado).

Neste trabalho, foram usados três métodos principais: Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC) e Behavioral Cloning (BC). Os métodos de aprendizado por reforço, como PPO e SAC, são baseados em um sistema de recompensas[1]*, no qual o agente recebe prêmios por ações que o aproximam de seu objetivo, incentivando comportamentos desejados. Já o Behavioral Cloning (BC) utiliza aprendizado supervisionado para imitar diretamente comportamentos observados, sem o uso explícito de recompensas.*

A seguir, descrevemos suas ideias básicas e como funcionam na prática.

2.3.1 Proximal Policy Optimization (PPO)

PPO é um método de aprendizado por reforço que ensina o agente a tomar decisões com base em recompensas recebidas ao interagir com o ambiente. Ele funciona de maneira iterativa:

- *Primeiro, o agente executa algumas ações e coleta dados sobre essas ações e as recompensas obtidas.*
- *Depois, o algoritmo ajusta a "política" do agente (que é como ele decide o que fazer) para aumentar as chances de repetir ações que deram bons resultados e evitar ações ruins.*

[1] Recompensas maiores representam um maior desempenho de acordo com as métricas utilizadas.

- *Para evitar mudanças muito bruscas, o PPO limita o quanto a política pode ser alterada a cada atualização, garantindo estabilidade no aprendizado.*

Exemplo simples: *Imagine um agente que está aprendendo a andar em um labirinto. Ele recebe uma recompensa positiva toda vez que avança na direção correta e uma penalidade se bater em uma parede. O PPO usa essas recompensas para ajustar as decisões do agente, incentivando movimentos que o levem mais perto da saída sem mudar radicalmente seu comportamento a cada tentativa.*

2.3.2 Soft Actor-Critic (SAC)

SAC é outro método de aprendizado por reforço, mas com uma abordagem que incentiva o agente a explorar o ambiente de forma mais ampla e segura. Ele faz isso adicionando um "bônus" para a aleatoriedade nas ações, o que ajuda o agente a evitar ficar preso a estratégias ruins.

- *O agente aprende não só a maximizar a recompensa, mas também a manter um comportamento diversificado (exploratório).*
- *Isso ajuda a encontrar soluções melhores e mais robustas, principalmente em ambientes com ações que podem variar em uma escala contínua (por exemplo, controlar a velocidade de um personagem).*

Exemplo simples: *Voltando ao labirinto, o agente SAC não apenas tenta andar para frente, mas também experimenta diferentes direções e velocidades, buscando novas rotas e evitando ficar repetindo o mesmo caminho sem sucesso.*

2.3.3 Behavioral Cloning (BC)

BC é uma técnica diferente: em vez de aprender sozinho, o agente aprende a imitar um especialista. Ele recebe exemplos de como o especialista age em diferentes situações e tenta replicar essas ações.

- *Os dados de treinamento são pares de "estado do ambiente" e "ação do especialista".*
- *O agente aprende uma função que, dado um estado, prevê a ação que o especialista tomaria.*

Exemplo simples: *Se você mostrar a um agente vídeos de alguém jogando o labirinto e as decisões que essa pessoa tomou, o agente vai tentar copiar esse comportamento, sem testar sozinho as ações.*

2.3.4 Resumo dos Métodos

Método	Tipo	Como Aprende	Vantagens	Desvantagens
PPO	Aprendizado por reforço	Ajusta a política com base em recompensas obtidas em suas próprias interações, limitando mudanças bruscas	Estável e eficaz em muitos tipos de problema	Exige bastante interação com o ambiente
SAC	Aprendizado por reforço	Maximiza a recompensa e a entropia para balancear *exploration*[1] e *exploitation*[2]	Aprendizagem eficiente, bom para ações contínuas	Mais complexo de implementar e ajustar
BC	Aprendizado supervisionado	Imita o comportamento de um especialista com base em exemplos	Simples e rápido, sem necessidade de interagir com o ambiente	Pode falhar em situações não cobertas nos exemplos, não aprende com erros

Tabela 1 – Resumo dos métodos utilizados no ML-Agents

[1] *Exploration* consiste em testar novas ações ou caminhos, mesmo que inicialmente menos promissores, com o objetivo de descobrir estratégias potencialmente melhores a longo prazo.

[2] *Exploitation* refere-se à ação de escolher a opção que o agente já conhece como a mais vantajosa com base em experiências anteriores, maximizando recompensas imediatas.

3 AMBIENTE DO JOGO

Este capítulo apresenta o ambiente de jogo desenvolvido para os testes e avaliações das técnicas de Machine Learning abordadas neste trabalho. O jogo, construído na Unity Engine, é do tipo soulslike, com foco em confrontos entre o jogador e inimigos, denominados bosses. A proposta do ambiente é simular um cenário de alta dificuldade, exigindo decisões rápidas e domínio das mecânicas por parte do jogador. A seguir, temos uma captura de tela (Figura 1) demonstrando a estética visual, barras de vida e stamina do player na parte superior (vermelho e verde, respectivamente), barras de vida (em vermelho), assim como o jogador e o inimigo (as barras azul, do jogador, e amarela, do agente, não foram implementadas).

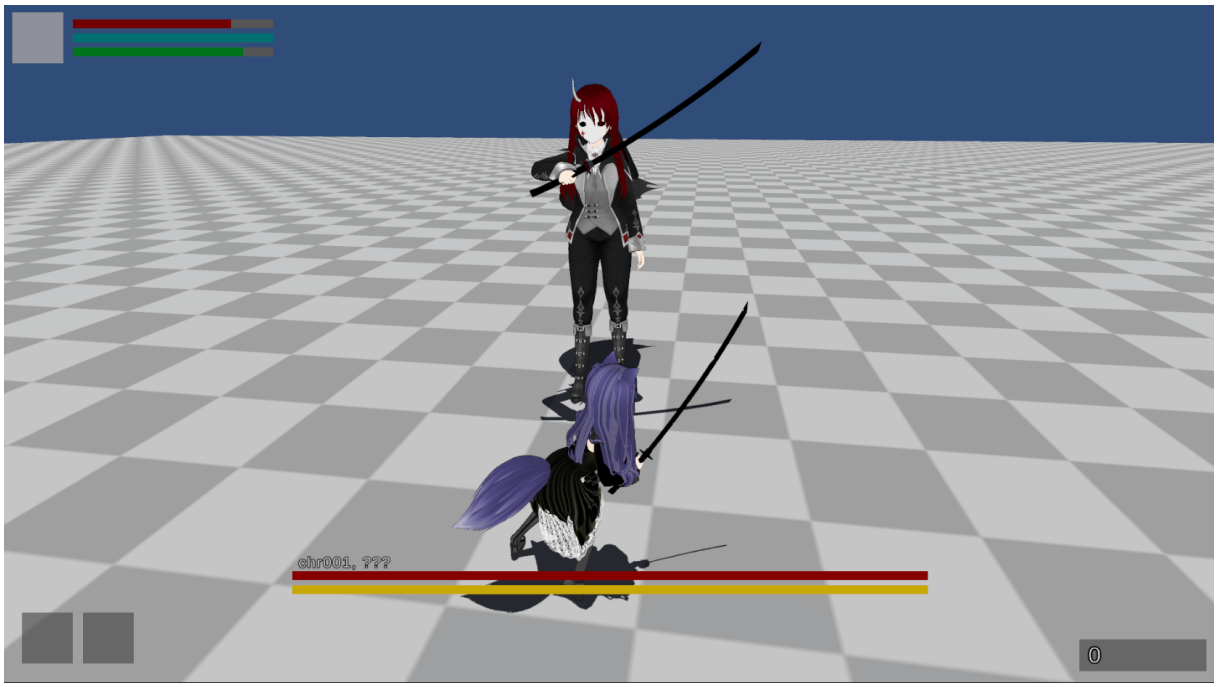

Figura 1 – Aparência do Jogo

3.1 MECÂNICAS DO JOGADOR

O jogador dispõe de um conjunto de ações ofensivas e defensivas, além de recursos limitados que influenciam diretamente sua performance em combate. As ações disponíveis são:

- ***Esquiva**: permite ao jogador escapar de ataques inimigos por meio de um movimento evasivo, consumindo parte da barra de stamina.*

- ***Defesa**: reduz ou anula o dano de ataques frontais, com consumo variável de stamina dependendo da força do golpe.*
- ***Dash**: movimento rápido de reposicionamento, utilizado tanto ofensivamente quanto para fugir de situações perigosas.*
- ***Parry (Defesa Perfeita)**: ação defensiva de alta precisão que, quando realizada no tempo exato, quebra momentaneamente a defesa do inimigo, abrindo espaço para um contra-ataque.*

O jogador possui dois atributos principais:

- ***Vida (HP)**: determina sua sobrevivência. Ao atingir zero, o jogador é derrotado.*
- ***Stamina**: consumida ao realizar esquivas, defesas e ataques; sua gestão é essencial para a sobrevivência e eficiência nos combates.*

3.2 COMPORTAMENTO DO INIMIGO

Para representarmos de maneira mais clara e precisa o movimento do agente, podemos usar uma máquina de estados (Figura 2) que mostra as ações disponíveis e as transições de estado que são representados pelas letras $\psi, \alpha, \beta, \lambda, \gamma, \omega$. Com exceção dos estados S10, S20 e BDash, todos podem ser encerrados para voltar ao Walk, caso o algoritmo selecione ψ.

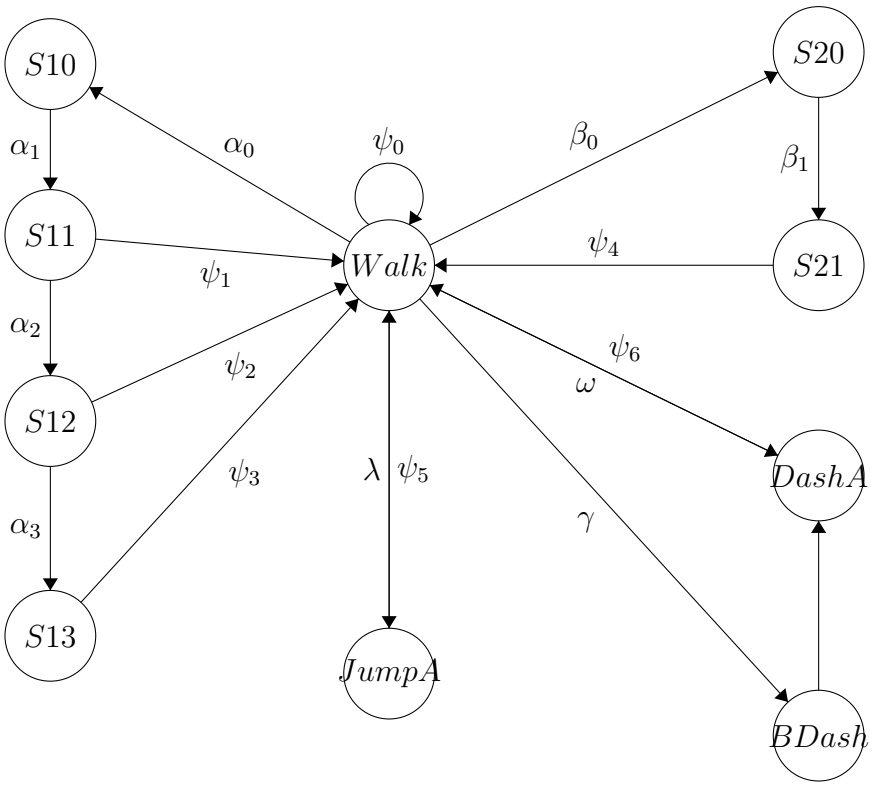

Figura 2 – Máquina de estados do inimigo controlado por IA.

O inimigo controlado por agentes inteligentes também conta com um conjunto de ações complexas, sendo um subconjunto de ações[1] *presente no jogo Dark Souls 1 e projetadas para desafiar o jogador e testar a adaptabilidade dos modelos de Machine Learning.*

- ***Esquiva****: Usada defensivamente para evitar ataques do jogador, representada pelo estado BDash quando recebe γ do estado Walk, irá transicionar incondicionalmente para o DashA.*
- ***Combos de ataque (2 tipos)****: Sequências ofensivas com múltiplos golpes, podendo ser interrompidas a qualquer momento conforme a decisão do agente, a sequência de estados S10 e S20 simbolizam os golpes iniciais, o algoritmo decide se segue a sequência (α ou β) ou se encerra o combo prematuramente, retornando ao Walk ($\psi_1 ... \psi_4$).*
- ***Ataque perfurante com dash****: O estado DashA é responsável por um ataque linear e rápido, tornando difícil a esquiva. Existe a transição regrada por ψ_6, porém retorna incondicionalmente ao estado de caminhada.*
- ***Ataque com pulo e impacto****: O inimigo salta e atinge o chão, gerando lanças que surgem do solo, forçando o jogador a se reposicionar, representado pelo estado JumpA.*

[1] O ataque com pulo implementado no agente foi uma liberdade criativa adotada com base em movimentos semelhantes presentes em jogos do gênero.

- ***Movimento (andar)**: O estado Walk é usado para aproximação ou reposicionamento é tratado como o estado default. As transições representadas por ψ indicam o retorno ao estado Walk.*

  O inimigo possui os seguintes atributos:

- ***Vida (HP)**: seu valor decrescente representa o progresso do jogador no combate.*
- ***Cooldown**: Variável que impede de atacar desenfreadamente.*

3.3 IMPLEMENTAÇÃO DO JOGO

Esta seção descreve a implementação prática do jogo utilizado como ambiente de testes para os agentes inteligentes.

3.3.0.1 Decisão de Ação

Ao final de cada ação executada pelo inimigo (por exemplo, após completar um combo ou concluir um ataque com pulo), o agente recebe o estado atual do ambiente. Esse estado contém informações relevantes, como a posição do jogador, a distância entre inimigo e jogador, a stamina disponível, o tempo desde a última ação, entre outras variáveis que definem o contexto.

*Esses dados compõem o vetor de entrada do agente, que os utiliza em conjunto com os **pesos previamente treinados** por um dos algoritmos para inferir a ação mais adequada à situação.*

O fluxo básico da decisão pode ser representado pelos seguintes passos [2]:

```
função decidir_ação(estado_atual):
    // Recebe o vetor de características do estado atual do ambiente
    entrada_agente = extrair_características(conjunto_de_observáveis)

    // Usa o modelo treinado para prever a melhor ação
    ação_escolhida = modelo.predict(entrada_agente)

    retornar ação_escolhida
```

[2] Detalhes são apresentados na Seção 4.1.2.

3.3.0.2 Execução da Ação Escolhida

A ação escolhida pelo agente é então enviada para a máquina de estados do inimigo, que gerencia tanto a execução da animação correspondente quanto as regras de comportamento atreladas àquela ação (ex: iniciar um combo, defender, esquivar, usar um ataque especial).

Ao término da execução da ação, um novo ciclo de decisão é iniciado, garantindo que o agente reavalie o estado do ambiente para decidir o próximo movimento.

Essa execução pode ser ilustrada pelos seguintes passos:

```
função executar_ação(ação):
    // Atualiza a máquina de estados para refletir a nova ação
    inimigo.maquina_estados.atualizar_estado(ação)

    // Inicia a animação associada
    inimigo.animator.rodar_animação(ação)

    // Aguarda o término da ação antes de prosseguir
    esperar até inimigo.ação_atual.concluída

    // Sinaliza que o agente pode decidir nova ação
    sinalizar_fim_de_ação()
```

3.3.0.3 Ciclo Completo do Agente

A interação contínua entre decisão e execução pode ser descrita pelo ciclo principal do agente:

```
enquanto inimigo.vivo:
    estado_atual = obter_estado_ambiente()

    ação = decidir_ação(estado_atual)

    executar_ação(ação)
```

Este modelo baseado em ciclos permite que o agente se adapte dinamicamente ao comportamento do jogador, reagindo a mudanças no posicionamento, no esgotamento de stamina, tentativas de ataque ou defesa, entre outras condições do jogo.

4 METODOLOGIA

A metodologia adotada neste trabalho é composta por etapas práticas e analíticas, voltadas à construção de um ambiente de testes e à avaliação de modelos de Machine Learning para controle de dificuldade em jogos digitais. O foco está no desenvolvimento de um jogo soulslike, no qual os inimigos principais (bosses) são controlados por agentes inteligentes treinados com diferentes técnicas de aprendizado de máquina.

4.1 DESENVOLVIMENTO DO AMBIENTE DE JOGO

Inicialmente, foi desenvolvido um jogo do tipo soulslike utilizando a Unity Engine, uma plataforma amplamente utilizada na indústria de jogos. O projeto foi inteiramente concebido e implementado pelo autor deste trabalho, que atuará como desenvolvedor do jogo e responsável por toda a lógica de interação, modelagem do ambiente e integração com os agentes inteligentes.

O jogo foi projetado de forma a simular uma experiência desafiadora, com foco em combates contra chefes (bosses), que representam os principais desafios enfrentados pelo jogador.

Os inimigos foram configurados para atuarem como agentes controlados externamente por modelos de Machine Learning. Essa integração é feita por meio da ferramenta ML-Agents, disponibilizada pela própria Unity, que permite a conexão entre o ambiente de jogo e algoritmos de treinamento.

4.1.1 Modelos de Machine Learning Utilizados

Nesta subseção, são apresentadas as propriedades dos algoritmos que motivaram sua seleção, considerando sua adequação aos objetivos deste estudo.

- ***Proximal Policy Optimization (PPO)**: algoritmo com foco em otimizar o desempenho do agente enquanto mantém estabilidade durante o treinamento, utilizando uma função de perda que limita atualizações muito abruptas nas políticas para garantir aprendizado mais estável e confiável.*
- ***Soft Actor-Critic (SAC)**: método de aprendizado por reforço profundo baseado em políticas estocásticas e aprendizado off-policy, que utiliza o princípio de entropia máxima para incentivar a exploração durante o treinamento. Essa abordagem permite que o agente tome decisões mais robustas em ambientes complexos e estocásticos,*

reduzindo o risco de convergir para políticas subótimas e aumentando a estabilidade e eficiência do aprendizado.

- ***Behavioral Cloning (BC)**: método de aprendizado supervisionado que permite aos agentes imitarem comportamentos pré-definidos, com base em dados coletados de jogadores humanos ou scripts especializados. No contexto do trabalho, ele foi treinado com demonstrações do modelo PPO já treinado. Esse método foi treinado por 1/5[1] dos passos dos demais, com o intuito de simular uma dificuldade mais fácil, pois é um algoritmo menos adequado para a solução do problema.*

Inicialmente, espera-se que a dificuldade de cada algoritmo reflita o quão adequeado ele seria para a solução do problema em questão, sendo PPO > SAC > BC. O PPO se faz o mais adequado por trabalhar com ações discretas e evitar explosão de valores. Para o funcionamento do SAC houve uma tradução dos valores discretos em contínuos, enquanto o BC busca imitar o comportamento do PPO.

Cada modelo foi treinado individualmente em cenários equivalentes e com parâmetros semelhantes, a fim de permitir comparações justas entre os métodos.

4.1.2 Entradas para o Treinamento dos Agentes

Para que os modelos de aprendizado de máquina possam tomar decisões eficazes durante o treinamento, é necessário fornecer um conjunto de observações representativas do estado do ambiente. Essas observações constituem os pesos que são utilizados pelos agentes para aprender políticas de comportamento adequadas.

No contexto deste projeto, foram coletadas e fornecidas ao agente as seguintes informações:

- ***Distância entre jogador e inimigo**: valor escalar que representa a proximidade entre as duas entidades no campo de batalha;*
- ***Posição do jogador e do inimigo**: coordenadas espaciais normalizadas de ambos os personagens no ambiente tridimensional;*
- ***Estado de ataque**: valor booleano indicando se o jogador (ou inimigo) está atualmente executando um ataque;*
- ***Estado de defesa**: valor booleano indicando se o jogador está utilizando uma ação defensiva no momento;*

[1] Em ambientes de teste utilizados nos estudos, foi observado que 1/5 dos passsos de treinamento do PPO já representavam um nível de desafio considerável para um jogador humano — no caso, o próprio autor. No entanto, esse desempenho ainda se distanciava significativamente daquele alcançado pelo modelo PPO.

- ***Stamina do jogador**: valor numérico que representa a quantidade atual de energia disponível para execução de ações como ataques, desvios e defesa;*
- ***Rotação do jogador**: orientação do personagem no espaço, representada de forma adequada ao formato de entrada aceito pelo modelo (por exemplo, ângulo ou vetor de direção).*

Esses dados foram coletados continuamente durante os episódios de treinamento e atualizados a cada iteração[2], totalizando 500.000 passos[3] de treinamento para os algoritmos PPO e SAC, enquanto o algoritmo BC foi treinado por apenas 100.000 passos. O BC foi treinado por um número reduzido de passos porque sua proposta era servir como uma solução mais simples e direta, exigindo menos iterações para aprendizado inicial. Já os 500.000 passos aplicados ao PPO e SAC foram definidos com base em observações empíricas durante os testes preliminares: nesse ponto, os modelos já apresentavam uma recompensa acumulada estável e considerada satisfatória para os objetivos da tarefa, não sendo necessário estender o treinamento além disso[4]. A seleção dessas variáveis visa oferecer ao agente uma percepção suficiente do ambiente para que ele possa aprender estratégias que considerem tanto posicionamento quanto estados internos do jogador, como recursos disponíveis e ações em execução.

4.1.3 Coleta e Avaliação dos Dados

Essa diferença exigiu que fossem implementadas adaptações no sistema de controle dos agentes para que a interação humana pudesse ocorrer de forma consistente em todos os casos. No caso do SAC, as ações contínuas geradas pelo modelo foram mapeadas para os comandos de controle correspondentes no ambiente de jogo, respeitando a faixa e a granularidade das variáveis contínuas, como ângulos de direção ou intensidades de movimento.

Já para o PPO, que emula um espaço de ações discretas, o modelo seleciona diretamente comandos discretos pré-definidos (como mover ou atacar). Para uniformizar a avaliação e facilitar a coleta dos dados de interação humana, foi necessário adaptar a interface de entrada do jogo para aceitar esses diferentes tipos de controle, garantindo que os jogadores pudessem interagir adequadamente com os agentes treinados, independentemente da natureza contínua ou discreta das ações.

Durante as partidas, foi gerado um relatório automático contendo métricas relevantes para análise de desempenho, incluindo:

[2] Iterações são tomadas de ações do agente com base nos algoritmos.
[3] Passos (*Steps*) as iterações realizadas pelo agente na tomada de decisões
[4] Treinamento conduzido por uma CPU Ryzen 5 - 7600.

- ***Proporção de vitórias e derrotas***.
- ***Número de tentativas até a primeira vitória***.

Essas informações são registradas automaticamente ao final de cada combate, e armazenadas para posterior análise. Os participantes interagiram com o ambiente de jogo em sessões controladas. Cada participante enfrentou agentes treinados com os diferentes algoritmos de aprendizado (PPO, SAC e BC), sendo que a distribuição das demos com agentes foi definida de forma randomizada para cada jogador.

4.1.4 Análise Comparativa

Após a coleta dos relatórios de combate para cada algoritmo, os dados foram agrupados e organizados com base nas respectivas métricas. Para garantir uma comparação justa e estatisticamente relevante, foi utilizada a média dos resultados obtidos em diferentes sessões de jogo.

A análise comparativa permitiu identificar qual modelo apresentou o comportamento mais eficiente, considerando critérios como curva de aprendizado e consistência de vitórias. Essa abordagem busca revelar qual técnica de aprendizado oferece maior potencial para gerar agentes adaptativos e desafiadores dentro da proposta de um jogo do gênero soulslike.

5 RESULTADOS

Os testes foram realizados com a participação de 12 pessoas interessadas, que interagiram com o ambiente de jogo em sessões controladas. Ao todo, foram conduzidos 306 desafios entre o jogador e a máquina, distribuídos de forma aproximadamente equitativa entre os métodos de aprendizado de máquina avaliados: Behavioral Cloning, Proximal Policy Optimization e Soft Actor-Critic. Cada participante enfrentou agentes treinados com os diferentes algoritmos, permitindo a coleta dos dados referentes a quantidade de vitórias/derrotas, consistência após abater o agente pela primeira vez e curva de aprendizado medida pela quantidade de tentativas até derrotar o agente.

5.0.1 Progresso do treinamento

Esta subseção apresenta o progresso de treinamento de cada algoritmo, os hiperparâmetros utilizados, o tempo necessário para concluir o treinamento e as métricas empregadas para acompanhar sua evolução. Os gráficos apresentados nesta seção foram gerados com o TensorBoard, uma ferramenta amplamente utilizada para visualizar métricas de aprendizado e monitorar o desempenho de modelos durante o treinamento. O ML-Agents, utilizado neste trabalho, já gera nativamente os relatórios compatíveis com o TensorBoard, facilitando o acompanhamento do progresso dos agentes. A linha de maior opacidade nos gráficos representa a curva suavizada, com o objetivo de simplificar a visualização dos dados.

5.0.1.1 PPO

O treinamento foi realizado off-line, antes da fase de testes, concluindo os 500.000 passos em aproximadamente 51,19 minutos. Esse tempo é considerado rápido em comparação com os outros algoritmos avaliados, considerando que o ambiente apresenta complexidade moderada e que o ML-Agents é otimizado para gerar relatórios e métricas em tempo real com o TensorBoard.

A evolução da recompensa acumulada pode ser visualizada na Figura 3, onde o eixo X representa os passos de treinamento e o eixo Y a recompensa obtida pelo agente. Observa-se uma tendência geral de crescimento na recompensa, com flutuações típicas do processo de aprendizado por reforço. A partir de aproximadamente 200.000 passos, nota-se maior estabilidade e aumento consistente da recompensa, indicando que o agente passou a adotar uma política mais eficiente. Próximo ao final do treinamento, por volta dos

450.000 passos, o valor da recompensa se estabiliza, sugerindo que o agente alcançou um comportamento satisfatório para os objetivos propostos.

A Figura 4 apresenta a duração média dos episódios ao longo do treinamento. Embora no início os episódios fossem mais longos — com duração superior a 1000 passos —, houve uma redução gradual nesse valor. Essa diminuição é um indicativo de que o agente aprendeu a concluir os episódios de forma mais eficiente, realizando ações mais eficazes em menor tempo. Nos estágios finais do treinamento, a duração dos episódios estabilizou-se em torno de 650 a 700 passos (value), o valor Smoothed representa o mesmo valor para a curva suavizada.

A análise conjunta das duas figuras mostra que, ao mesmo tempo em que a recompensa aumentava, a duração dos episódios diminuía, o que caracteriza um ganho de desempenho com maior eficiência. Assim, o limite de 500.000 passos mostrou-se adequado, visto que os indicadores já se encontravam estáveis ao final do processo de aprendizado.

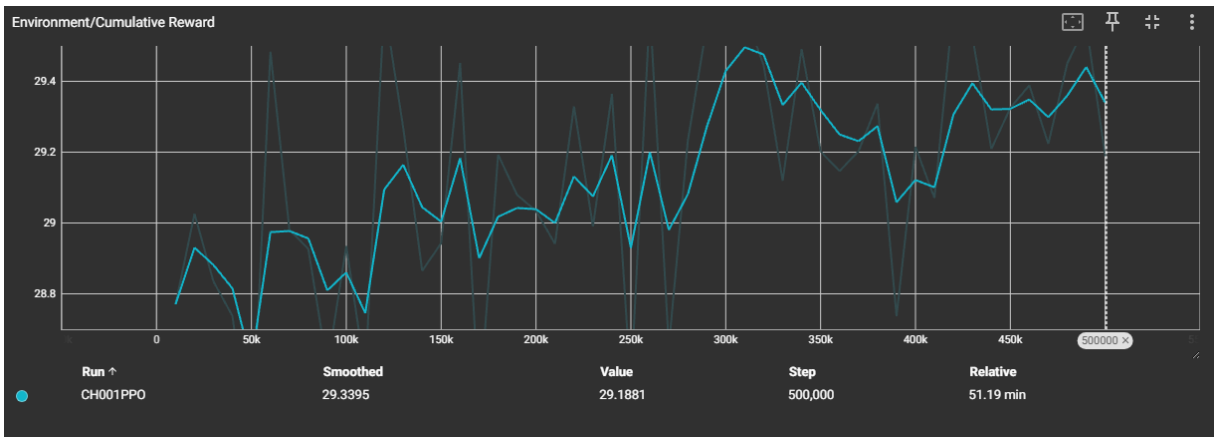

Figura 3 – Recompensa ao longo do treinamento - PPO

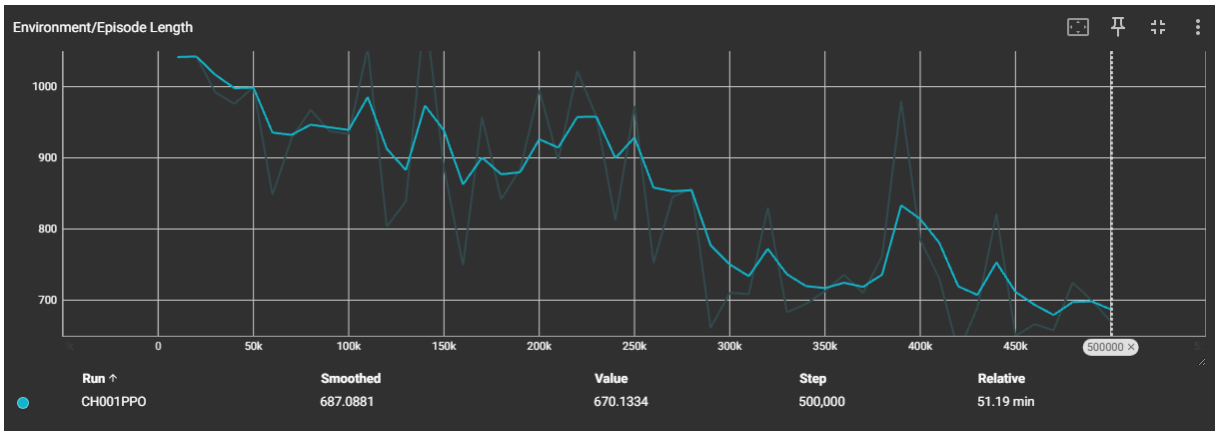

Figura 4 – Duração de episódio ao longo do treinamento - PPO

5.0.1.2 SAC

Em comparação ao algoritmo anterior, o treinamento com o SAC foi significativamente mais custoso em termos de tempo computacional, totalizando aproximadamente 2 horas e 8 minutos para alcançar os 500.000 passos. Essa diferença está relacionada à dinâmica do agente durante o treinamento: por permanecer longos períodos com recompensas baixas e executar ações menos eficazes nos estágios iniciais, os episódios tiveram, em média, maior duração.

A Figura 5 apresenta a evolução da recompensa acumulada ao longo do treinamento. Embora haja uma oscilação perceptível, é possível observar uma tendência geral de crescimento, especialmente após os 350.000 passos. O valor suavizado ao final do processo se aproxima de 29.4, indicando que o SAC eventualmente convergiu para uma política comparável à do PPO em termos de desempenho. No entanto, essa convergência ocorreu de forma mais tardia e instável, o que justifica o maior tempo necessário de treinamento.

Na Figura 6, observa-se a duração média dos episódios ao longo do treinamento. Inicialmente, os episódios ultrapassavam 1100 passos, reduzindo de forma progressiva ao longo do tempo. Assim como no caso do PPO, essa redução sugere um aprendizado gradual de ações mais eficazes e rápidas. Entretanto, a linha de tendência indica uma maior variabilidade e uma estabilização menos clara, refletindo o comportamento mais errático do agente durante parte do treinamento.

Em conjunto, as figuras mostram que o SAC foi capaz de aprender uma política eficaz, mas exigiu mais tempo de treinamento e apresentou maior instabilidade durante a aprendizagem, especialmente nos estágios intermediários. Ainda assim, o desempenho final é satisfatório e demonstra a capacidade do algoritmo de adaptação mesmo em ambientes com maior variabilidade de estados e ações.

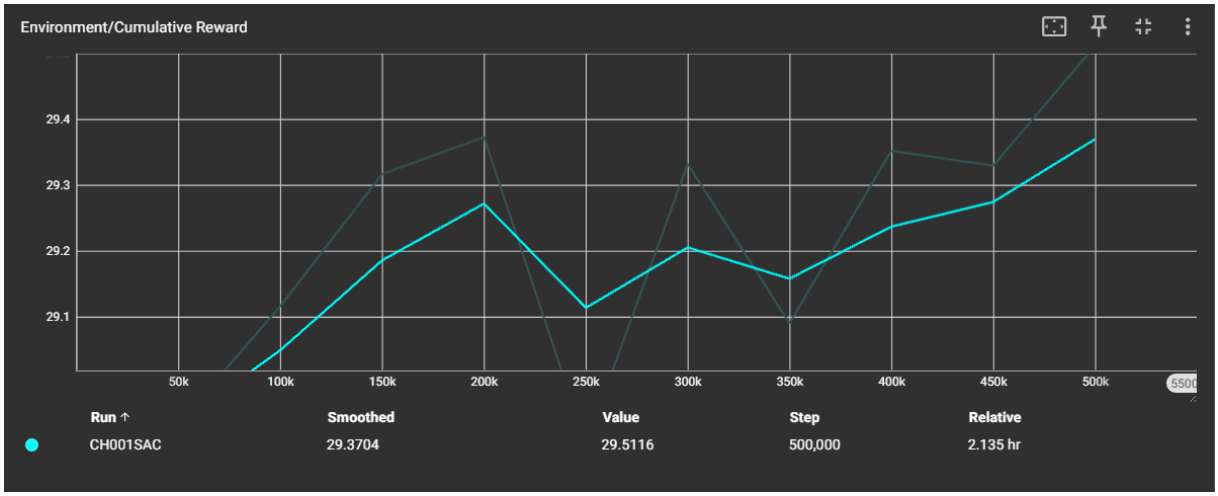

Figura 5 – Recompensa ao longo do treinamento - SAC

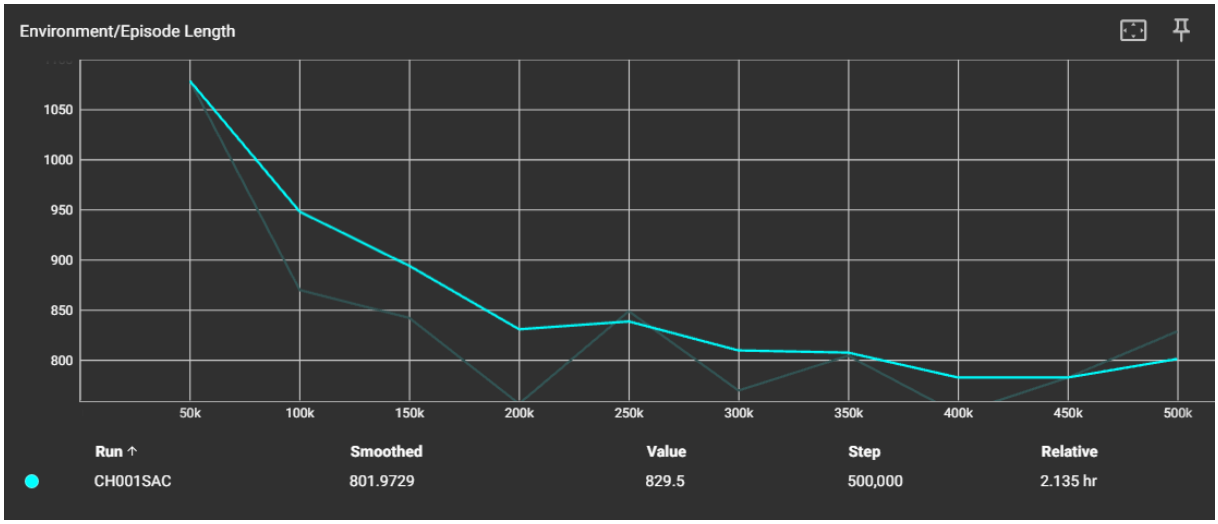

Figura 6 – Duração de episódio ao longo do treinamento - SAC

5.0.1.3 BC

Como citado na subseção 4.1.1, o algoritmo BC foi treinado por apenas 100.000 passos, com o objetivo de servir como uma abordagem mais simples e direta, baseada em aprendizado por imitação. O tempo total de treinamento foi de aproximadamente 11,76 minutos, consideravelmente menor que os demais algoritmos, visto que o processo se baseia na observação das ações tomadas pelo agente treinado com PPO.

A Figura 7 apresenta a evolução da recompensa acumulada ao longo do treinamento. Apesar da curta duração do processo, os resultados finais foram bastante semelhantes aos obtidos pelos algoritmos baseados em aprendizado por reforço. A recompensa suavizada atinge valores próximos de 29.3, com certa oscilação natural ao longo do tempo, mas sem indicar perdas significativas de desempenho.

A Figura 8 mostra a duração média dos episódios durante o treinamento. Assim como nos outros casos, houve uma redução progressiva na duração, com uma média final em torno de 715 passos por episódio. Esse comportamento sugere que o agente foi capaz de imitar com sucesso as estratégias observadas, atingindo um grau de eficiência próximo ao dos demais métodos, mesmo sem o processo iterativo de exploração e ajuste de política característico do aprendizado por reforço.

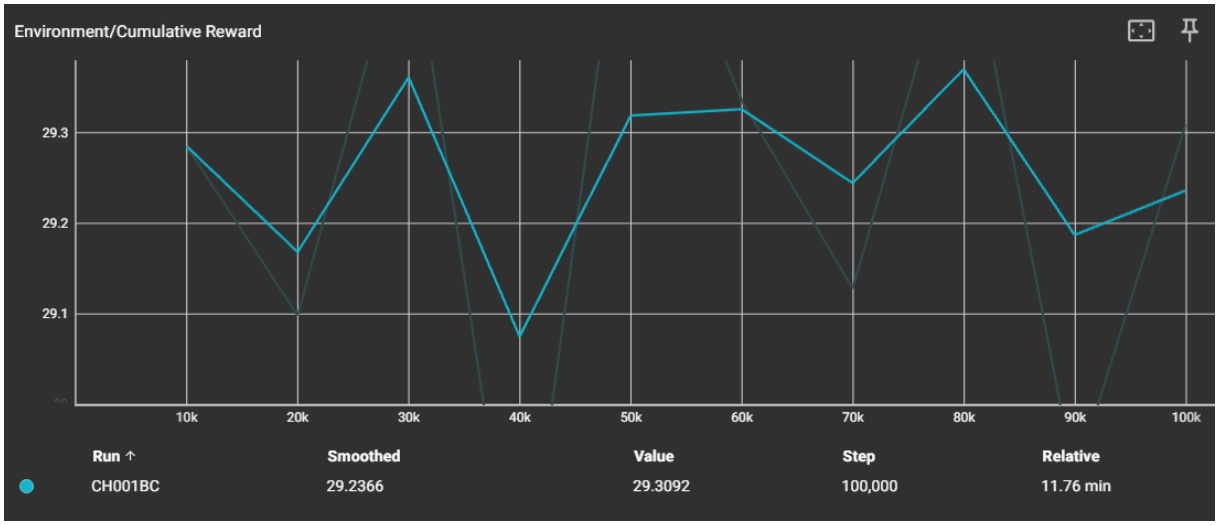

Figura 7 – Recompensa ao longo do treinamento - BC

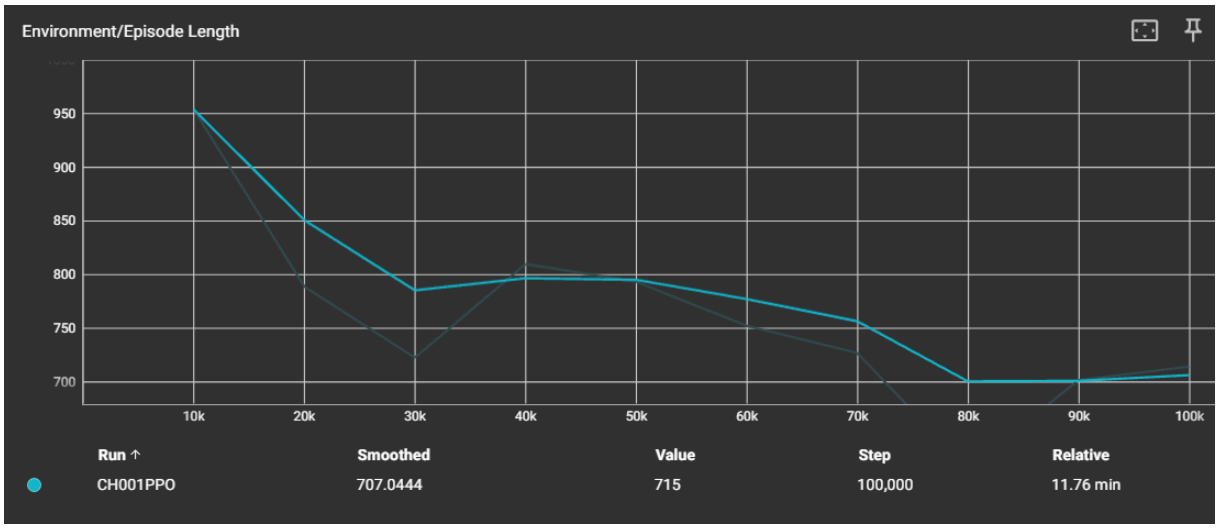

Figura 8 – Duração de episódio ao longo do treinamento - BC

5.0.2 Desempenho

A Figura 9 apresenta a taxa de vitórias e derrotas obtidas pelos diferentes algoritmos de aprendizado testados no experimento.

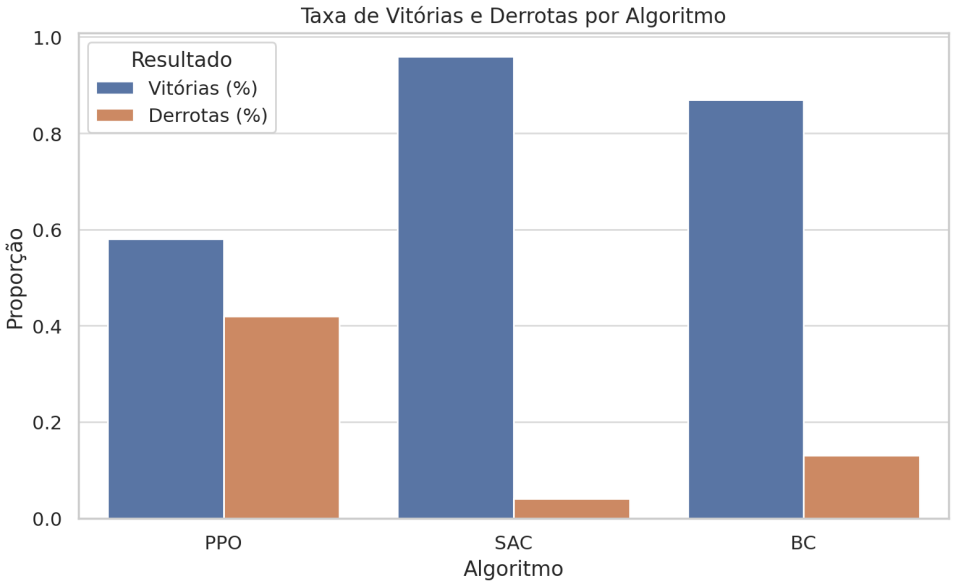

Figura 9 – Taxa de vitórias por algoritmo de aprendizado.

Com a amostra de aproximadamente de 102 execuções por algoritmo, observa-se que o nível de dificuldade é dado por SAC > BC > PPO, em contradição com o esperado na subseção 4.1.1.

Tabela 2 – Desempenho dos algoritmos para uma amostra de 306 tentativas (102 por algoritmo)

Algoritmo	**MDT**	**V**	**D**
PPO	4.20	0.58	0.42
SAC	9.23	0.96	0.04
BC	10.80	0.87	0.13

A coluna MDT representa o número médio de tentativas até a primeira derrota do agente, a coluna V representa a proporção de vezes que o agente venceu o jogador, D representa o complemento.

5.0.3 Análise

Ao contrário do que se pensava em 4.1.1, a ordem de dificuldade sendo SAC > BC > PPO, essa subseção busca explicar os motivos.

- ***PPO**: Sendo completamento o contrário do esperado, por mais que tenha obtido os melhores dados em treinamento contra um agente simulando o jogador, o PPO se mostrou um algoritmo considerado fácil em relação às outras opções, além de ser derrotado a primeira vez mais rapidamente, ele venceu o jogador consideravelmente*

menos. Imagina-se que se deve ao fato de ter aprendido com maestria os movimentos do agente de treinamento, porém, ao enfrentar um jogador real, ao usar um mesmo ataque em uma situação parecida, o jogador começa a se acostumar e seu comportamento fica previsível.

- ***SAC**: Como mostrado em 5.0.1, o SAC se mostrou pouco otimizado em relação a solução do problema, em conjunto com a ampla exploração*[1]*, é obtido um algoritmo que raramente irá tomar a mesma decisão em uma situação semelhante, tornando difícil a predição de movimentos por parte do jogador. Essa teoria é reforçada analisando que SAC e BC possuem um MDT semelhante, porém a taxa de vitórias do algoritmo SAC é extremamente alta.*

- ***BC**: Por mais que tenha sido treinado observando o agente de PPO, os 100000 passos de treinamento, que tinham como objetivo deixar o agente mais fácil (pois esperava-se que o PPO iria ter o melhor desempenho), se provaram insuficientes para copiar as ações do agente observado, acabando com um agente muito diferente e imprevisível, obtendo como resultado um agente mais difícil que o agente que teria seu comportamento clonado.*

É importante notar que, mesmo que os algoritmos tenham sido divididos randomicamente entre os interessados no experimento, pessoas diferentes tendem a ter desempenhos diferentes em determinados jogos, também podendo influenciar nos resultados.

Essa possibilidade de escolher entre agentes controlados por algoritmos distintos promove uma experiência personalizada e menos previsível, o que é especialmente relevante para jogos do gênero soulslike, onde o desafio é um elemento central da experiência. Dessa forma, o trabalho contribui para o desenvolvimento de sistemas de dificuldade mais orgânicos, capazes de preservar o interesse e a motivação do jogador ao longo do tempo, evitando a sensação de repetição e monotonia.

[1] Exploração alta se refere a uma maior quantidade de possibilidades tentadas de maneiras randômicas.

6 CONCLUSÃO

Este trabalho teve como objetivo investigar e comparar diferentes algoritmos de aprendizado de máquina — Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC) e Behavior Cloning (BC) — para o controle de dificuldade em um ambiente de jogo do tipo soulslike desenvolvido com Unity e integrado ao ML-Agents.

Os resultados obtidos mostraram que, contrariamente às expectativas iniciais, o SAC apresentou o maior nível de desafio para o jogador, seguido pelo BC e, por último, pelo PPO. Essa inversão evidencia a importância de considerar não apenas a eficiência e estabilidade de aprendizado dos algoritmos, mas também seu impacto direto no nível de imprevisibilidade e adaptabilidade do agente, elementos cruciais para tornar a experiência do jogador mais divertida e menos previsível.

A técnica de Behavior Cloning (BC), mesmo treinada com uma amostra reduzida e derivada do PPO, acabou apresentando um comportamento final significativamente distinto do esperado, adquirindo uma dinâmica de luta única e não tão previsível. Já o SAC, ao priorizar uma exploração mais ampla de estados e ações, criou uma experiência de combate particularmente desafiadora e menos previsível.

Assim, conclui-se que as abordagens de aprendizado por reforço e por imitação utilizadas neste trabalho se mostraram capazes de representar diferentes níveis de dificuldade de forma inicial, com potencial para contribuir para o engajamento do jogador em cenários específicos. No entanto, seus impactos na satisfação e na experiência geral do jogador exigem investigações mais aprofundadas.

7 TRABALHOS FUTUROS

Embora o trabalho ofereça uma análise exploratória sobre o tema, algumas limitações e oportunidades para estudos futuros foram identificadas:

- ***Análise com mais jogadores e amostras ampliadas**: Realizar testes com um número maior e mais variado de jogadores para aumentar a significância estatística e verificar diferenças de estilo de jogo.*
- ***Combinação de abordagens**: Explorar o uso de abordagens híbridas (imitation learning + reinforcement learning) para aproveitar o melhor dos dois mundos e verificar se uma técnica pode compensar as limitações da outra.*
- ***Adaptação dinâmica em tempo real**: Investigar a implementação de ajuste de dificuldade durante a própria partida, adaptando o agente às mudanças no nível de habilidade do jogador em tempo real.*
- ***Outros algoritmos e arquiteturas**: Testar algoritmos mais avançados de aprendizado por reforço e arquiteturas de rede neural para verificar seu impacto no nível de dificuldade e adaptabilidade do agente.*
- ***Experiência do jogador**: Ampliar o escopo para considerar métricas subjetivas, como satisfação e frustração, para melhorar a compreensão do impacto das abordagens testadas sobre a experiência humana.*

Em síntese, este trabalho representa um primeiro passo na exploração do uso de diferentes métodos de aprendizado — por reforço e por imitação — como forma de simbolizar e implementar diferentes níveis de dificuldade no contexto de jogos, especialmente no gênero soulslike, contribuindo para tornar as interações entre jogadores e agentes inteligentes cada vez mais naturais, desafiadoras e adaptadas ao perfil e habilidade de cada jogador.

Além dos pontos já mencionados, os resultados obtidos neste trabalho indicam que a utilização de diferentes algoritmos de aprendizado de máquina para controlar a dificuldade pode contribuir significativamente para manter o jogador engajado. Isso ocorre porque a variação nos comportamentos dos agentes, gerada pela diversidade dos métodos (PPO, SAC e BC), permite oferecer níveis de dificuldade que fogem do modelo tradicional e artificialmente parametrizado, apresentando desafios mais naturais e dinâmicos.

REFERÊNCIAS

FromSoftware. ***Demon's Souls****. [S.l.]: Sony Computer Entertainment, 2009. Game [PlayStation 3].*

____. ***Dark Souls****. [S.l.]: Bandai Namco Entertainment, 2011. https://en.bandainamcoent.eu/dark-souls/dark-souls. Game [PlayStation 3, Xbox 360, Microsoft Windows].*

FUCHS, R.; GIESEKE, R.; DOCKHORN, A. Personalized dynamic difficulty adjustment – imitation learning meets reinforcement learning. ***arXiv preprint arXiv:2408.06818****, 2024. Disponível em: <https://arxiv.org/abs/2408.06818>https://arxiv.org/abs/2408.06818.*

GéRON, A. ***Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow****. 3. ed. Sebastopol, CA: O'Reilly Media, 2022.*

MACHADO, M. C. Uma metodologia para modelagem de jogadores baseada em aprendizado de máquina. Universidade Federal de Minas Gerais, 2013.

OLIVEIRA, T. N. d.; CHAIMOWICZ, L. Investigating reinforcement learning for dynamic difficulty adjustment. In: ***Anais do Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames)****. [S.l.: s.n.], 2023. p. 66–75.*

OR, D. B.; KOLOMENKIN, M.; SHABAT, G. Dl-dda – deep learning based dynamic difficulty adjustment with ux and gameplay constraints. ***arXiv preprint arXiv:2106.03075****, 2021. Disponível em: <https://arxiv.org/abs/2106.03075>https://arxiv.org/abs/2106.03075.*

RAHIMI, M. et al. Continuous reinforcement learning-based dynamic difficulty adjustment in a visual working memory game. ***arXiv preprint arXiv:2308.12726****, 2023. Disponível em: <https://arxiv.org/abs/2308.12726>https://arxiv.org/abs/2308.12726.*

SAMPAYO-VARGAS, S. et al. The effectiveness of adaptive difficulty adjustments on students' motivation and learning in an educational computer game. ***Computers & Education****, Elsevier, v. 69, p. 452–462, 2013.*

TAGLIARO, L. R. G. An implementation of adaptive difficulty systems for challenging video games. 2022.

Unity Technologies. ***Unity ML-Agents Toolkit****. 2023. https://github.com/Unity-Technologies/ml-agents. Acessado em: 10 jul. 2025.*