**UNIVERSIDADE FEDERAL DE MINAS GERAIS**
**Instituto de Ciências Exatas**
**Programa de Pós-Graduação em Ciência da Computação**

Tiago Negrisoli de Oliveira

# Aprendizado por reforço para ajuste dinâmico de dificuldade em jogos de luta

Belo Horizonte
2025

Tiago Negrisoli de Oliveira

# Aprendizado por reforço para ajuste dinâmico de dificuldade em jogos de luta

**Versão Final**

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Luiz Chamowicz

Belo Horizonte
2025

Oliveira, Tiago Negrisoli de.

O48a Aprendizado por reforço para ajuste dinâmico de dificuldade em jogos de luta [recurso eletrônico] / Tiago Negrisoli de Oliveira – 2025.

1 recurso online (95 f. il.) : pdf.

Orientador: Luiz Chaimowicz.

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.

Referências: f. 86-95.

1. Computação - Teses. 2. Jogos eletrônicos - Teses. I. Chaimowicz , Luiz. II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. III. Título.

CDU 519.6*08 (043)

Ficha catalográfica elaborada por Célio Resende Diniz, bibliotecário CRB 6/2403 - Universidade Federal de Minas Gerais – ICEx.


UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# FOLHA DE APROVAÇÃO

# APRENDIZADO POR REFORÇO PARA AJUSTE DINÂMICO DE DIFICULDADE EM JOGOS DE LUTA

## TIAGO NEGRISOLI DE OLIVEIRA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. Luiz Chaimowicz - Orientador
Departamento de Ciência da Computação - UFMG

Prof. Anderson Rocha Tavares
Instituto de Informática - UFRGS

Prof. Lucas Nascimento Ferreira
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 04 de abril de 2025.

---

Documento assinado eletronicamente por **Luiz Chaimowicz**, **Professor do Magistério**

**Superior**, em 07/04/2025, às 12:45, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Anderson Rocha Tavares**, **Usuário Externo**, em 09/04/2025, às 17:09, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Lucas Nascimento Ferreira**, **Professor(a)**, em 14/04/2025, às 11:21, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4086659** e o código CRC **2F04FEC9**.

**Referência:** Processo nº 23072.219904/2025-25 SEI nº 4086659

*Dedico esta dissertação a todos familiares, amigos e professores que me auxiliaram nesta jornada.*

# Agradecimentos

A jornada até a conclusão desta dissertação foi marcada por desafios, aprendizados e, acima de tudo, pelo apoio de pessoas especiais, às quais expresso minha profunda gratidão.

Primeiramente, agradeço à minha família, pelo amor incondicional, paciência e incentivo em todos os momentos. Vocês foram minha fundação, inspiração e motivação para continuar, mesmo nos momentos mais difíceis.

Aos meus amigos, que me deram constante apoio, compreensão e momentos de descontração durante a jornada. Em especial, esse trabalho é dedicado à memória de Bárbara, uma amiga querida que partiu, mas que a presença e inspiração se mantêm vivas em meu coração. Seu carinho, palavras de encorajamento e amizade foram e sempre serão fundamentais para mim.

Ao meu orientador, sou profundamente grato pela paciência, dedicação e por compartilhar o conhecimento. Seu apoio e orientação foram determinantes para o desenvolvimento deste trabalho e para meu crescimento acadêmico e pessoal.

Por fim, agradeço a todos que, direta ou indiretamente, contribuíram para que este trabalho se tornasse realidade. A cada um de vocês, meu mais sincero obrigado.

"*Perfeclty balanced, as all things should be.*"
(Thanos)

# Resumo

Ajuste Dinâmico de Dificuldade (do inglês Dynamic Difficulty Adjustment - DDA) é uma técnica de automaticamente ajustar parâmetros de um jogo, como itens, mapas ou comportamento dos oponentes, com o intuito de prover ao jogador experiências desafiadoras e engajantes. O objetivo é manter o balanceamento e garantir um nível ótimo de entretenimento. Neste trabalho, investigamos o uso de aprendizado por reforço no ajuste dinâmico de dificuldade. Para isso, criamos um agente para um jogo de luta com objetivo de manter sua habilidade relativa equiparável com a do jogador. Durante a partida o agente altera sua proficiência para ser equivalente ao usuário. Para tal são controlados dois parâmetros distintos: Balanceamento e Competitividade. Balanceamento se refere à dificuldade percebida pelo jogador ao longo da interação com o sistema. Competitividade está relacionado ao resultado da tarefa. A fim de lidar com ambas as métricas, foi proposta uma função de recompensa que guia o agente a ter a habilidade relativa similar ao jogador e assim manter a partida balanceada. Além disso, uma penalidade atribuída durante o treino foi introduzida com objetivo de restringir a taxa de vitória do agente para o nível desejado. Dessa forma, gerando um oponente que não é tão fraco nem tão forte. Somado a isso, foram investigados formas de gerar um comportamento robusto capaz de generalização contra jogadores não vistos durante o treinamento. Técnicas de regularização são exploradas para aprimorar a desempenho do agente e sua adaptabilidade. Resultados apontam que a regularização possui efeitos positivos na generalização, entretanto não é suficiente para gerar um comportamento capaz de balancear o jogo para um conjunto com diversos jogadores. Assim, um esquema de treinamento utilizando *Selfplay* foi proposto para aprimorar ainda mais a capacidade de generalização do método, sem a necessidade de dados humanos ou de comportamentos previamente programados.

**Palavras-chave:** Aprendizado por Reforço. Ajuste de Dificuldade Dinâmico. Treinamento Baseado em População. Generalização.

# Abstract

Dynamic Difficulty Adjustment (DDA) is a technique that automatically adjusts game parameters, such as items, maps, or opponent behavior, to provide players with challenging and engaging experiences. The goal is to maintain balance and ensure an optimal level of entertainment. This work proposes a reinforcement learning agent for a fighting game to create an opponent whose skill level matches the player's relative ability. To achieve this, two distinct parameters are controlled: Balance and Competitiveness. Balance refers to the difficulty perceived by the player throughout the interaction with the system. Competitiveness is related to the outcome of the task. To address both metrics, a reward function was designed to guide the agent toward having a skill level comparable to the player, thus keeping the match balanced. Additionally, a penalty applied during training was introduced to limit the agent's win rate to the desired level, resulting in an opponent that is neither too weak nor too strong. Furthermore, methods to create robust behavior capable of generalizing against unseen players during training were investigated. Regularization techniques were explored to improve the agent's performance and adaptability. Results indicate that regularization has positive effects on generalization; however, it is insufficient to produce behavior capable of balancing against a diverse set of players. Consequently, a training scheme using self-play was proposed to further enhance the method's generalization capacity without requiring human data or pre-programmed behaviors.

**Keywords:** Reinforcement Learning. Dynamic Difficulty Adjustment. Population-based Trainning. Generalization.

# Lista de Figuras

# Lista de Tabelas

# Sumário

# Capítulo 1

# Introdução

## 1.1 Motivação

Nas últimas décadas, os jogos de computador se tornaram um dos principais meios de entretenimento, tornando-se um hábito comum nas vidas das pessoas [40]. Dados disponibilizados em [1] mostram que o número de jogadores aumentou consistentemente desde 2015 a uma taxa média de 5.6%, totalizando 2.69 bilhões de gamers no mundo ao final de 2020. Estima-se que em 2025, a indústria de jogos mundial terá uma receita de 268.81 bilhões de dólares. A utilização de jogos se expande para além de entretenimento, com aplicações para fins educacionais [27, 51, 16, 16], visando uma forma lúdica de aprender uma tarefa, e saúde [86, 3, 6, 5] para tratamento de doenças ou reabilitação de movimentos.

Jogos possuem diversas definições e muitas das vezes contraditórias [50]. Tekinbas e Zimmerman [97] descrevem como "sistemas em que pessoas engajam em um conflito artificial, definido por regras, e geram um resultado quantificável". Já Crawford [17] define um jogo como um sistema formal fechado que subjetivamente representa um subconjunto da realidade. O sistema formal é dito ter regras explícitas. Fechado por ser autocontido e não requerer referências externas aos agentes. Embora uma definição exata não exista, conforme a indústria de jogos continuamente cresce, jogos têm sido alvo de muitas pesquisas visando entender aspectos únicos desse meio de entretenimento. Uma grande questão que tem sido alvo de estudos é "Por que jogos são divertidos?". Koster [50] discute em seu livro, na perspectiva de design, o assunto. Para o autor, humanos são inerentemente reconhecedores de padrões. A todo momento estamos tentando identificá-los e entendê-los. Como jogos são sistemas matemáticos fechados com um conjunto de regras bem definidas, são ambientes que nos geram tais padrões. Dessa forma, buscamos aprender aquele sistema. Jogos com uma estratégia conhecida e clara para ganhar são desinteressantes.

Malone [61] estudou quais os fatores que geram motivação em jogos. Ele realizou experimentos com versões modificadas do Breakout [9], onde cada versão teve uma ou mais características do original removidas. Com a avaliação de voluntários, ele conseguiu

verificar quais desses fatores são mais influentes na diversão. Ele definiu três fatores como principais: desafio, fantasia e curiosidade. Desafio está relacionado ao objetivo. Ambos são fatores de motivação por se relacionarem à autoestima do jogador. Ter sucesso em uma tarefa difícil faz com que a pessoa sinta-se melhor consigo mesma. Já o fracasso tem o efeito oposto, desencorajando a realização da tarefa. Dessa forma, o processo deve ter níveis variáveis de dificuldade que se ajuste com a habilidade do jogador. Fantasia é o aspecto que gera "imagens mentais de coisas que não estão presentes aos sentidos ou na experiência real da pessoa envolvida". Ela é o fator que transporta o jogador para longe dos problemas reais e o insere no mundo proposto pelo jogo. Ela pode ser provida por objetos físicos ou situações vivenciadas no universo do videogame. Por fim, a curiosidade é relacionada ao entendimento e expectativas sobre o cenário e estado atual do jogador. Tais ambientes devem ser surpreendentes e novos na visão do jogador, mas não devem ser completamente incompreensíveis. Dessa forma, eles devem almejar um nível ótimo de complexidade de informação. Definida, geralmente, por um mundo em que é possível prever cenários futuros, mas que em alguns momentos ocorre a quebra de expectativa.

Mihaly Csiksentmihalyi, psicólogo de origem croata, desenvolveu uma teoria de experiência ótima na realização de atividades [18]. Essa é baseada no conceito de Flow. O autor o define como "estado no qual a pessoa está tão envolvida em uma atividade que nada além da tarefa importa. A experiência em si é tão prazerosa que, mesmo a um alto custo, irá realizá-la pelo simples fato de fazê-la". Entre vários fatores que devem estar presentes para a pessoa alcançar esse estado, o principal é o desafio. A tarefa sendo executada deve ser proporcional à habilidade da pessoa para que não ocorra tédio, em cenários muito fáceis, ou ansiedade, níveis de dificuldade muito altos.

Um aspecto que leva ao entretenimento, identificado em todos os modelos apresentados, é o *desafio*. Nos modelos, é necessário ter um nível ótimo de dificuldade em relação à habilidade do agente executando a tarefa. Desde sempre, a indústria de jogos tentou resolver esse problema com um mecanismo de escolha de dificuldade. Um conjunto finito e estático de possíveis graus de desafio, geralmente entre fácil, médio ou difícil, é apresentado ao jogador. Ele deve então escolher qual o melhor que lhe satisfaz. Esse método possui alguns problemas. Jogos são usualmente utilizados por um grupo de jogadores diversos com estilos e proficiência diferentes. Com isso, é improvável que um conjunto limitado e estático de dificuldades consiga apresentar experiências apropriadas para todos os níveis de habilidades. Outro problema é que esse método entrega a responsabilidade de balanceamento para o jogador, assumindo que a avaliação de sua própria habilidade sobre o jogo seja correta. Por fim, para criar um nível de desafio, os designers devem descrever fatores que afetam a dificuldade e, em seguida, encontrar as melhores configurações para cada nível. Esse processo deve ser realizado para cada oponente e fase do jogo, requerendo testes e uma definição do que se espera em cada nível, aumentando consideravelmente o tempo e a complexidade do desenvolvimento. Dessa forma, métodos que consigam realizar

esse balanceamento de forma automática são desejáveis.

## 1.2 Ajuste Dinâmico de Dificuldades

Recentemente, muitas pesquisas tiveram como foco criar métodos para encontrar um nível ótimo de desafio de forma automática. A área de Ajustes de Dificuldade Dinâmica (do inglês *Dynamic Difficulty Adjustment* - DDA) emergiu para lidar com esses problemas e prover ao jogador uma experiência de jogatina personalizada e engajante. Sistemas que utilizam esses métodos modificam automaticamente aspectos do jogo para ajustar o desafio para ser compatível com a habilidade do jogador. Ajustar a dificuldade em tempo real garante que o jogo não se apresente nem muito fácil, nem muito difícil. DDA visa aprimorar a satisfação do jogador, imersão e entretenimento, mantendo a relação de desafio e proficiência do jogador num nível ótimo.

Uma possível abordagem para a implementação de um sistema DDA é o uso de aprendizado por reforço (do inglês *Reinforcement Learning* - RL). RL modela um processo de decisão em que um agente interage continuamente com um ambiente. Esse agente recebe informação do ambiente por meio de observações e pode realizar uma ação. Essa interação causa uma transição do ambiente de um estado para outro. Durante o processo, o agente recebe um valor chamado de recompensa que reflete a transição de estado. O objetivo do aprendizado por reforço é aprender a maximizar a soma esperada total de recompensas ao longo do tempo através de suas ações. Embora esse campo tenha alcançado grandes resultados em outros domínios, como derrotar campeões mundiais em jogos como Dota 2 [73] e gerar agentes melhores que a maioria dos jogadores humanos em StarCraft [102], relativamente poucos trabalhos foram explorados na aplicação em DDA.

## 1.3 Objetivos e Contribuições

Neste trabalho, a aplicação de aprendizado por reforço no contexto de ajuste dinâmico de dificuldade é investigada. Assim, o principal objetivo é gerar um agente capaz de balancear o jogo para oponentes com habilidades relativas e estilos variados. Dessa forma, esse trabalho busca investigar duas perguntas:

- Como modelar o problema de ajuste dinâmico de dificuldade para aprendizado por

reforço?

- Como generalizar o modelo para uma variedade de tipos de jogadores?

Várias contribuições para a área são mostradas neste trabalho.

- **Uma nova função de recompensa foi proposta**: guia o agente a ter uma habilidade relativa próxima ao oponente.
- **Introdução de um mecanismo de penalidade**: permite limitar a taxa de vitória do agente para um nível desejado. Essa provê mais controle sobre o comportamento resultante do treinamento, permitindo o ajuste do nível de dificuldade e competitividade que se deseja obter para melhor compatibilidade com a habilidade do jogador.
- **Generalização em aprendizado por reforço foi explorada**: técnicas de regularização foram investigadas.
- **Esquema de treinamento proposto**: um modelo de treinamento utilizando *self-play* foi proposto. O treinamento procura gerar oponentes com diferentes estilos e habilidades relativas para aprimorar a capacidade do agente de lidar com ampla variedade de jogadores.

Os resultados apontam que a modelagem do agente através das funções recompensa gera agentes capazes de balancear o jogo para as métricas. Essa primeira parte gerou uma publicação no Simpósio Brasileiro de Games e Entretenimento Digital (SBGames) de 2023 [20].

Além disso, o treinamento proposto demonstrou conseguir aumentar o desempenho e a adaptabilidade do agente, aprimorando, consequentemente, a habilidade de balancear o jogo contra diferentes oponentes. Os experimentos foram conduzidos em um jogo competitivo de luta para avaliar o nosso método.

## 1.4 Organização da Dissertação

Essa dissertação está organizada em seis capítulos. O capítulo **??** apresenta tanto o referencial teórico de aprendizado por reforço quanto a revisão bibliográfica com foco em ajuste dinâmico de dificuldade. O capítulo 4 apresenta o simulador utilizado, apresentando conceitos gerais do gênero de jogos de luta e específicos do simulador, assim como apresenta a modelagem do ambiente para o treinamento em RL. O capítulo 3 apresenta a

metodologia utilizada neste trabalho para gerar o agente DDA. O capítulo 6 apresenta os experimentos realizados para demonstrar o funcionamento do método proposto e avaliar a capacidade do agente final. Por fim, o capítulo 7 apresenta as conclusões e possibilidades de trabalhos futuros.

# Capítulo 2

# Aprendizado por Reforço

Aprendizado por reforço é o processo de aprender o mapeamento de situações em ações para maximizar um valor numérico chamado recompensa [95]. A entidade que toma as decisões e aprende é chamada agente. Tudo que o agente interage e externo a ele é denominado ambiente. Dessa forma, existe a contínua interação no qual o agente recebe alguma representação do ambiente como uma observação e executa uma ação. O ambiente então é modificado para um novo estado, e o agente recebe uma recompensa que reflete essa mudança. A imagem 2.1 ilustra essa interação. Assim, o agente é apresentado com decisões sequenciais, e deve aprender quais ações tomar para que a recompensa obtida ao longo desse processo seja máxima.

O objetivo de um agente de aprendizado por reforço é maximizar a recompensa recebida ao longo do tempo. Conforme ele interage com o ambiente, o agente observa sequências de estado, recompensa e próximo estado: $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$. Essa sequência é chamada de trajetória. Formalmente, podemos definir o conjunto das recompensas $R_i$ que se quer maximizar pela função de retorno $G_t$. Em geral, o objetivo é maximizar o valor esperado do retorno $G_t$ definido na equação (2.1).

$$G_t = \sum_{i=0}^{T-t-1} \gamma^i R_{t+i+1} \tag{2.1}$$

Em que T é o último período da interação entre o agente e o ambiente, incluindo $T = \infty$. $\gamma \in [0, 1]$ é chamado de taxa de desconto. Esse permite que ambientes não episódicos (nos quais não existe um estado terminal) sejam modelados como uma maxi-

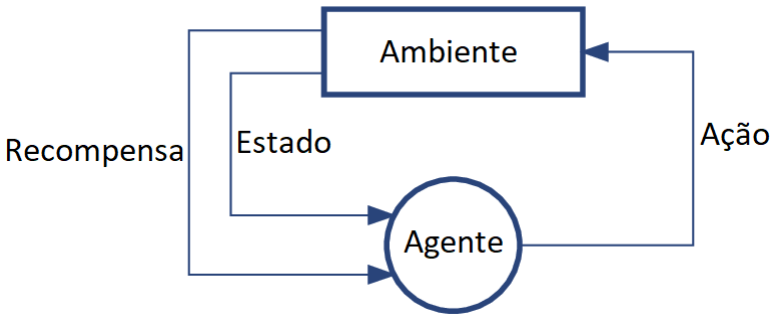



Figura 2.1: Processo de interação do agente com ambiente.

mização do valor esperado do retorno. O efeito pode ser interpretado de duas formas. A primeira é que ele representa a probabilidade do episódio acabar no tempo $i$. Outra visão é a importância que o agente tem sobre recompensas imediatas em relação às futuras. Com $\gamma = 0$, apenas a recompensa imediata é considerada. Com $\gamma = 1$, não existe diferença entre recompensas imediatas e futuras, todas têm a mesma importância. Dessa forma, a taxa de desconto controla o horizonte que o agente irá considerar.

Podemos formalizar matematicamente o problema de aprendizado por reforço através do Processo de Decisão de Markov (do inglês Markov Decision Process-MDP). MDP é uma formulação de sequências de escolhas que afetam não apenas recompensas imediatas, como também próximas situações a serem experienciadas e que, consequentemente, afeta recompensas futuras. Esse é definido como uma tupla $(S, A, T, R, \gamma)$. O espaço de estados $S$ é definido como o conjunto de todos os possíveis estados do problema. Para que um conjunto seja considerado válido, é necessário que todos os possíveis estados $s \in S$ satisfaçam a propriedade de Markov. Essa propriedade estabelece que um estado $s_t$ contém todas as informações necessárias para modelar estados futuros. Assim, a probabilidade de alcançar $s_{t+1}$ a partir de $s_t$ depende exclusivamente do estado atual, sem influência de estados anteriores. Essa propriedade é formalmente descrita na equação (2.2).

$$P(s_{t+1} = s' | s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0) = P(s_{t+1} = s' | s_t, a_t) \tag{2.2}$$

em que $P$ representa a probabilidade de transição entre estados, e $s_i$ é o estado do agente no tempo $i$. A equação expressa que a probabilidade de alcançar um novo estado $s'$ no tempo $t + 1$, dado todo o histórico de estados e ações $(s_t, a_t, s_{t1}, a_{t1}, ..., s_0, a_0)$, é idêntica à probabilidade de alcançar $s'$ considerando apenas o estado atual $s_t$ e a ação $a_t$. O conjunto $A$ é chamado de espaço de ações. Ele é definido como o conjunto finito de ações que podem ser realizadas a cada momento. A função de transição $T(s_{t+1}|s_t, a_t)$ mapeia a probabilidade de alcançar $s_{t+1}$ dado $s_t$ e a ação $a_t$. $R(s_t, a_t)$ é a função de recompensa imediata ao executar uma ação em um determinado estado.

O agente deve aprender a escolher ações durante a interação com o ambiente. Formalmente, definimos uma função chamada política $\pi$ que mapeia cada estado $s \in S$ e a ação $a \in A$ a uma probabilidade $\pi(a|s) = P[A_t = a|S_t = s]$ de executar a ação $a$ no estado $s$. O objetivo é encontrar a política ótima $\pi^*$ que maximize o retorno esperado. Grande parte dos métodos de aprendizado por reforço funciona tentando estimar o quão bom é o atual estado ou ações. Um estado/ação ser bom é definido por recompensas esperadas futuras e por uma política $\pi$. Em um MDP, podemos definir, formalmente, a função $V(s)$ na equação (2.3).

$$V_\pi(s) = E_\pi \left[ G_t \mid S_t = s \right] = E_\pi \left[ \sum_{i=0}^{\infty} \gamma^i R_{t+k+1} \mid S_t = s \right] \tag{2.3}$$

Ela é a função valor do estado. Ela representa o retorno descontado esperado por alcançar $s$ e seguir a política $\pi$ em seguida. Similarmente, podemos definir uma função de valor para ações $Q(s, a)$ na equação (2.4).

$$Q_\pi(s, a) = E_\pi \left[ G_t \mid S_t = s, A_t = a \right] = E_\pi \left[ \sum_{i=0}^{\infty} \gamma^i R_{t+k+1} \mid S_t = s \right] \tag{2.4}$$

A função $Q$ é chamada de função valor da ação. Ela denota o retorno descontado esperado escolhendo $a$ em $s$ e seguindo a política $\pi$ nos próximos estados. Valores ótimos das funções de valor do estado $V^*_\pi(s) = max_\pi V_\pi(s), \forall s \in S$ e do par estado-ação $Q^*_\pi(s, a) = max_\pi Q_\pi(s, a), \forall s \in S, \forall a \in A$ são as funções que retornam o maior valor esperado da recompensa para todos os estados ou par estado-ação. A política ótima dado $V^*$ e $Q^*$ é escolher a ação que maximize o valor do próximo estado. Dessa forma, estimar essas funções leva a encontrar $\pi^*$.

Um problema fundamental em aprendizado por reforço é o dilema de *exploraition-exploitation*. Conforme o agente aprende com o ambiente, a política é aprimorada. Nos momentos de decisões, a fim de maximizar a recompensa, ele deve seguir a melhor estratégia descoberta até então. Entretanto, apenas escolher a melhor ação definida por $\pi$ pode levar a ótimos locais. Isso porque o espaço de soluções pode não ter sido explorado e a melhor estratégia ainda não foi encontrada pelo agente. Assim, deve existir um meio-termo entre escolher a melhor estratégia (*exploit*) e explorar novas ações (*explore*).

## 2.1 Proximal Policy Optimization

O objetivo do agente de aprendizado por reforço é encontrar uma política que maximize a recompensa esperada ao longo das interações. Existem diversas formas de resolver esse problema, entretanto pode-se separar em dois grandes grupos. O primeiro são métodos baseados nas funções de valor $V(s)$ e $Q(s, a)$. Esses têm como alvo estimar essas funções e, a partir delas, obter a política. O segundo grupo são métodos baseados na política, que buscam encontrar diretamente a função $\pi$.

Métodos mais tradicionais que estimam as funções de valor utilizam a versão tabular. As funções de cada estado ou par estado-ação são armazenadas em uma tabela. Embora tenham garantias teóricas de convergência, em cenários mais complexos, se torna inviável a construção e armazenamento da tabela. Além do problema de memória, existe a questão de generalização. Em cenários com grande espaço de estados, é improvável que um estado específico seja amostrado diversas vezes. Assim, um modelo com capacidade de avaliar estados similares sem a necessidade de amostragem de todo o espaço de estados

é desejável. Para tal, a solução mais utilizada atualmente é a aproximação de função. Mais especificamente, a função de valor ou política são parametrizadas por um conjunto de parâmetros $\theta$. O objetivo é encontrar o vetor de parâmetros $\theta^*$ que maximize a recompensa esperada. A fim de estimar esse vetor, o método atualiza os parâmetros do modelo em direção ao gradiente de uma função de desempenho $J(\theta)$ assim como descrito na equação (2.5).

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta J(\theta_t) \tag{2.5}$$

A função $J(\theta)$ também chamada de objetivo, define o desempenho do modelo atual. No caso de aprendizado por reforço, é baseada na recompensa obtida pelo agente. Dado um MDP, uma política parametrizada $\pi_\theta$ diferenciável e uma trajetória $\tau$, a função para maximizar a recompensa é dada por (2.6).

$$J(\theta) = E_{\tau \sim p(\tau)}[G(\tau)] \tag{2.6}$$

Em que $E$ é o valor esperado da função de retorno $G(\tau)$ com a distribuição de probabilidade $p(\tau)$ da trajetória do agente. Embora defina o objetivo teórico, essa equação não é diretamente aplicável.

O *Policy Gradient Theorem* provê uma forma analítica do gradiente $\nabla_\theta J(\theta)$ de forma que é possível estimar os gradientes e otimizar diretamente a política. Entretanto, na prática, essas estimativas possuem muito ruído, assim, diversas técnicas são aplicadas para reduzir a variância dos gradientes ao mesmo tempo, mantendo o *bias* baixo [52]. Uma dessas é a utilização de um *baseline* no cálculo da estimativa dos gradientes. Com isso, a função mais utilizada atualmente é apresentada na equação (2.7).

$$\nabla_\theta J(\theta) = E_t[\nabla_\theta log \pi_\theta(a_t|s_t) A(s,a)_t] \tag{2.7}$$

Em que $A(s,a) = Q(s,a) - V(s)$ é chamada de *advantage*. Essa função determina o valor de uma ação $a$ no estado $s$ comparado a escolher outra aleatoriamente ditada pela política $\pi(a|s)$. Diversos métodos foram propostos utilizando essa formulação. Entre eles, o estado da arte atualmente *Proximal Policy Optimization* (PPO) [85], sendo o utilizado neste trabalho. O método é derivado de outro TRPO [84] que introduz uma restrição à função objetivo. Quando aplicado diretamente à equação (2.7), resulta em grandes atualizações na política e muitas vezes prejudicial ao treinamento. A restrição aplicada visa limitar o tamanho da atualização provinda da função objetivo. Embora tenha apresentado bons resultados, a implementação pode ser muitas vezes complexa. O método PPO introduz o *Clipped Surrogate Objective* que funciona similar à restrição, mas é aplicado diretamente à função objetivo. Essa é dada pela equação (2.8).

$$J(\theta) = E_t[min(r_t(\theta)A(s,a), clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A(s,a)] \tag{2.8}$$

Em que $\epsilon$ é um hiperparâmetro e $r_t(\theta)$ é a razão entre as probabilidades da nova política sobre a antiga. Essa razão verifica a taxa de divergência entre as duas versões. A função *clip* limita o valor $r_t(\theta)$ entre $[1-\epsilon,\ 1+\epsilon]$, evitando grandes atualizações da política. Além disso, o mínimo entre a versão limitada e padrão é utilizado. Assim, a atualização é um limite inferior da função padrão.

O PPO faz parte de um terceiro grupo de métodos chamado *Actor-Critic*. Esse possui dois principais componentes: *Actor* que aprende a política e o *Critic* que aprende a função de valor $V$. Intuitivamente, o ator seleciona as ações, enquanto o crítico avalia a qualidade dessas decisões e utiliza essa avaliação para aprimorar a política. O PPO é amplamente utilizado devido à sua estabilidade e eficiência no treinamento de agentes de aprendizado por reforço. Por essas propriedades, foi o método utilizado neste trabalho.

## 2.2 Treinamento Baseado em População

Um método que se tornou popular para gerar políticas robustas sem conhecimento prévio da tarefa e com capacidade de generalização é o treinamento baseado em população (do inglês *Population-based* - PB)[58]. O método mantém uma população de agentes que exploram o ambiente em paralelo e aprendem entre si, de modo a aprimorar o desempenho coletivo. Este tipo de treinamento oferece diversos benefícios, como escalabilidade e diversidade de estratégias. Self-play (SP), um tipo específico de método PB, inicialmente introduzido por Arthur Samuel [83], considera apenas um agente com política $\pi$ sendo otimizado, e para os outros no ambiente a política mais recente deste é utilizada. Assim, todos os oponentes possuem o mesmo comportamento. Embora consiga gerar políticas fortes, o método pode eventualmente esquecer comportamentos úteis previamente adquiridos no treinamento. Esse efeito é chamado de *catastrophic forgetting*.

Uma variação do método que ameniza esse problema foi apresentada por Bansal et. al. [10]. Os autores propuseram utilizar versões passadas do agente durante o treinamento. O trabalho considerou a probabilidade $\delta$ de selecionar um oponente utilizando a política mais recente. Em testes, verificaram que, para $\delta = 1$, o treinamento resultou em uma política mais fraca, enquanto para valores $\delta = 0.5$, no ambiente mais complexo, resultou na melhor política. Assim, considerar o histórico aprimora a política final. Entretanto, o treinamento utilizando apenas versões anteriores do agente leva a políticas que comumente alcançam mínimos locais e que têm dificuldade de generalização contra diferentes oponentes [47].

Outra variação introduzida em [46] mantém diversos agentes treinando em paralelo, onde cada um otimiza sua própria política. O conjunto de versões históricas é

compartilhado, e os oponentes são selecionados a partir desses. Naturalmente, seja pela inicialização aleatória, mecanismo de exploração ou melhor resposta, cada processo irá explorar um local distinto no espaço de estratégias, e potencialmente encontrará distintas soluções. Assim, o procedimento aumenta a capacidade de generalização dos agentes. Hernandez et. al [41] propuseram um framework para definir o treinamento *self-play* em ambientes MARL. O trabalho define três principais componentes. A primeira é a noção de zoológico $\pi^o$, que define o conjunto de possíveis comportamentos que podem ser utilizados a cada episódio pelos outros agentes no ambiente. O segundo é a distribuição de amostragem de políticas $\Omega$, que define a probabilidade de cada política em $\pi^o$ ser escolhida. Por fim, uma função $G$ chamada de *gating function*, que determina se uma nova cópia da política $\pi$ sendo atualizada será incluída no zoológico. Os autores descrevem o *loop* de treinamento, em que, ao início do episódio, um novo comportamento será atribuído ao oponente, amostrado seguindo a distribuição $\Omega$. Após a otimização da política $\pi$, a função $G$ determina se essa será adicionada a $\pi^o$. Esse processo é repetido até o fim do treinamento.

Trabalhos como [89, 101, 90, 8, 15] utilizaram o treinamento *self-play* e obtiveram agentes capazes de vencer jogadores profissionais em seus respectivos jogos. O AlphaStar [101] aplicado para StarCraft II [29] utilizou um esquema de treinamento com três agentes: *Main Agent*, *League Exploiters* e *Main Exploiters*. O *Main Agent* utiliza o treinamento *self-play* contra todos os oponentes na liga. Os League Exploiters são utilizados para encontrar políticas que os oponentes no conjunto não conseguem vencer. Por fim, os Main Exploiters treinam contra o Main Agent de modo a identificar fraquezas e ser a resposta ótima contra ele. Dessa forma, o sistema identifica de forma automática fraquezas no agente principal, aprimorando o desempenho. Berner et. al. [73] desenvolveram um agente com o treinamento *self-play* para o jogo Dota 2 [99]. A política resultante conseguiu derrotar campeões mundiais do jogo. Para tal, utilizaram diversas técnicas. Uma dessas foi o treinamento por *self-play*. Os autores treinaram o agente contra versões de si, em que 80% das vezes um conjunto de políticas recentes era amostrado, e 20% políticas históricas eram utilizadas. Silver et. al. [89] desenvolveram um agente (AlphaGO) para o jogo Go, considerado um dos mais complexos dentre os clássicos, como xadrez. Para alcançar esses resultados, foram executadas três etapas de treinamento. A primeira foi otimizar a rede com dados de jogadores humanos experientes. Após essa etapa, o treinamento self-play foi utilizado, a fim de aprimorar a seleção de ações. O agente enfrentava versões derivadas de si. A última etapa foi treinar um preditor que avaliava o valor de $V(s)$ em cada estado. A política resultante teve grandes resultados tanto contra soluções prévias do problema quanto contra o campeão europeu do jogo. Posteriormente, em [90], uma versão sem dados humanos (AlphaZero) foi desenvolvida, com apenas o treinamento self-play. Essa conseguiu ganhar 100% das vezes contra essa versão antiga.

### 2.2.1 Diversidade

Em cenários multi-agente, a diversidade de estratégias faz um papel crucial na performance [42]. Em especial nos ambientes com forte não-transitividade, promover a diversidade de políticas é essencial para encontrar soluções ótimas [55]. Em cenários como esses, o agente deve manter um conjunto de comportamentos diferentes para evitar que o oponente explore fraquezas de uma estratégia única. Dessa forma, incentivar a pluralidade estratégica auxilia no processo de aprendizado por cobrir uma variedade de diferentes soluções pelos agentes e torná-los robustos ao encontrar novos oponentes. Em [62], foi avaliado o efeito de variações no ambiente e diversidade de comportamentos na generalização. O trabalho mostrou que a diversidade está relacionada com o aumento de performance e generalização em alguns cenários testados. Mesmo em ambientes que não tiveram aumento significativo da performance, não houve efeitos negativos.

Promover a diversidade tem o papel crítico em ambientes *Multi-agent reinforcement learning* (MARL) para gerar alta performance no treino, prevenindo que os agentes enfrentem uma mesma estratégia repetidamente, e em testes, mantendo a população robusta contra oponentes não vistos durante o treinamento [57]. Com isso, existem diversos trabalhos que investigam esse tópico e propõem definições de quantificar diversidade. Os autores de [55] dividem em dois tipos de métricas. O primeiro *Behavior Diversity (BD)* constrói métricas baseadas na trajetória ou pares estado-ação. A forma mais intuitiva de contabilizar a diversidade em aprendizado por reforço é avaliando diretamente a política dos agentes. Assim, para uma determinada trajetória, utilizam-se métricas para comparar as distribuições de probabilidade. Dessa forma, a diversidade estratégica é determinada pela divergência no mapeamento de ações.

Na literatura, diversas métricas tradicionais de comparação de distribuição de probabilidades foram empregadas para avaliar a disparidade de políticas. Alguns exemplos são: divergência de Kullback–Leibler [107], divergência de Jensen-Shannon [60], distância de Hellinger e métrica de Wasserstein [13]. Outros trabalhos, como [77] e [56], propuseram novas métricas. O primeiro avaliava a diversidade conjunta da população baseada nas ações tomadas de cada indivíduo. A métrica denominada *Diversity via Determinants (DvD)* compara os indivíduos em pares com o uso de um kernel que forma uma matriz. A diversidade é o determinante dessa matriz que, geometricamente, representa o volume do paralelepípedo no espaço de características definido pela escolha do kernel. O objetivo é maximizar esse volume para cobrir uma maior diversidade de estratégias. O segundo apresentou uma função de objetivo dupla. A ideia é conseguir separar as métricas de habilidade relativa e estilo a fim de otimizar ambas, além de ter um meio de comparação entre agentes. A definição de estilo é baseada nas diferenças dos estados ao longo de uma trajetória quando enfrentados oponentes distintos. Um problema de métricas do

tipo *Behavior Diversity* é que não se considera a recompensa de cada estratégia. Assim, pode ser ineficiente em cenários onde uma pequena mudança na estratégia resulta em uma grande mudança na recompensa final, como labirintos.

O segundo tipo é denominado *Response Diversity (RD)*. Este se baseia no resultado empírico. A ideia é que a diferença de resultados obtidos na população contra diferentes oponentes revela a diversidade de respostas estratégicas entre os indivíduos. Um exemplo desse tipo de métrica é o trabalho de Perez et. al. [68], que descreve uma nova métrica para avaliar diversidade. Essa se baseia em uma interpretação geométrica de jogos modelados por meio de *Determinantal Point Process* (DPP). DPP é um *framework* probabilístico originário do campo de física quântica que modela a probabilidade de um subconjunto de itens serem amostrados de um conjunto base em que é preferível amostras diversificadas. No trabalho, a métrica de similaridade era baseada nos diferentes desempenhos dos agentes devido a características únicas no comportamento. Garnelo et. al. [32] utilizou uma matriz de performance da população via avaliações por pares. Os autores descrevem que a métrica mais importante é a diversidade efetiva, que mede as diferentes estratégias que apresentam bons resultados. Essa métrica é determinada a partir do equilíbrio de Nash e da matriz de performance. A diversidade efetiva captura a diversidade das melhores estratégias.

# Capítulo 3

# Ajuste Dinâmico de Dificuldade

O Ajuste Dinâmico de Dificuldade (Dynamic Difficulty Adjustment – DDA) é uma área de pesquisa focada no desenvolvimento de métodos que modifiquem aspectos do jogo em tempo real. Seu principal objetivo é proporcionar um nível de desafio ideal para cada jogador, tornando a experiência mais envolvente e satisfatória. Um sistema DDA é composto por dois elementos principais: métrica de dificuldade e mecanismo de ajuste. A métrica de dificuldade busca quantificar a percepção de desafio do jogador em um valor numérico, servindo como referência para o sistema determinar se deve aumentar ou reduzir a dificuldade. O mecanismo de ajuste define como o sistema altera os parâmetros do jogo para equilibrar a dificuldade, garantindo que ela permaneça adequada à proficiência do jogador. Segundo Gustavo Andrade et al. [7], um sistema DDA eficaz deve atender a três requisitos essenciais. (i) O sistema deve identificar e ajustar-se à proficiência do jogador o mais rápido possível. (ii) É necessário monitorar continuamente o desempenho ao longo do tempo. (iii) As adaptações realizadas devem se manter críveis e não perceptíveis. Com essas características, o sistema mantém o nível ideal de desafio ao longo de toda a interação, ajustando-se à progressão do jogador sem que as mudanças sejam óbvias, quebrando a imersão. Pesquisas e aplicações em DDA têm ganhado maior foco nos últimos anos. Segundo [108], o número de artigos sobre o assunto aumentou em três vezes no período de 2012 para 2017. Diversas técnicas já foram estudadas e aplicadas a diversos cenários distintos. As próximas seções aprofundam a discussão sobre os componentes de um sistema DDA e exploram suas aplicações tanto em pesquisas acadêmicas quanto em jogos comerciais.

## 3.1 Percepção de Dificuldade

Flow [18], teoria desenvolvida por Csikszentmihalyi, é um estado no qual a pessoa se encontra totalmente imersa em uma atividade, a ponto de nada além da tarefa importar. Esse estado ocorre quando há equilíbrio entre o nível de habilidade do indivíduo e o desafio

proposto, evitando tanto o tédio quanto a frustração. Em seus estudos, o pesquisador notou que, frequentemente, as atividades são descritas como positivas quando uma ou mais das seguintes características estão presentes:

- Existe a possibilidade de ser cumprida;
- É possível ter concentração durante sua realização;
- Tem objetivos claros;
- Possui feedback imediatos;
- Não se percebe o ambiente externo;
- Existe um senso de controle sobre as ações;
- A percepção de si desaparece;
- Sensação de passagem de tempo é alterada.

Quando esses fatores ocorrem no decorrer da tarefa, a pessoa entra no estado de Flow. Csiksentmihalyi afirma que o proveito de alguma atividade aparece quando o nível de habilidade da pessoa é equiparável à dificuldade da tarefa. Quando é muito fácil, gera-se tédio. Em contrapartida, ao ser muito complexa, ocorre a frustração. Dessa forma, o prazer emerge na fronteira entre os dois, ambientes cujo desafio é balanceado com a capacidade da pessoa. A Figura 3.1 mostra de forma gráfica a teoria, em que, conforme o nível de habilidade aumenta, para se manter no estado, o desafio deve aumentar também. Sweetser e Wyeth [96] propuseram o modelo GameFlow visando identificar elementos-chave que contribuem para o entretenimento do jogador. No total, oito fatores foram derivados da pesquisa. Esses elementos são fortemente relacionados aos descritos na teoria de Flow. Dessa forma, os autores transportaram o conceito para o contexto de jogos.

O ajuste dinâmico de dificuldade se baseia nessa ideia, garantindo que um jogo mantenha um nível de desafio ideal para o jogador, proporcionando uma experiência envolvente. Essa é uma tarefa desafiadora, pois deve atender jogadores com diferentes níveis de habilidade, experiências e motivações [25]. Bartle [11], em um dos primeiros modelos para categorizar jogadores, identificou tipos e suas respectivas motivações investigando jogos MUDs (do inglês *Multi-user Dungeons and Dragons*). Em seu trabalho, ele identifica quatro tipos de jogadores, baseados nas interações: *Achievers*, *Explorers*, *Socialisers* e *Killers*. *Achievers* querem aperfeiçoar suas habilidades e conquistar todos os desafios do jogo. *Explorers* são jogadores que buscam explorar o mundo e coletar detalhes sobre o ambiente virtual. *Socialisers* tem interesse em interagir com outros jogadores e buscam a parte social dos jogos. *Killers* buscam a competitividade contra outros jogadores.

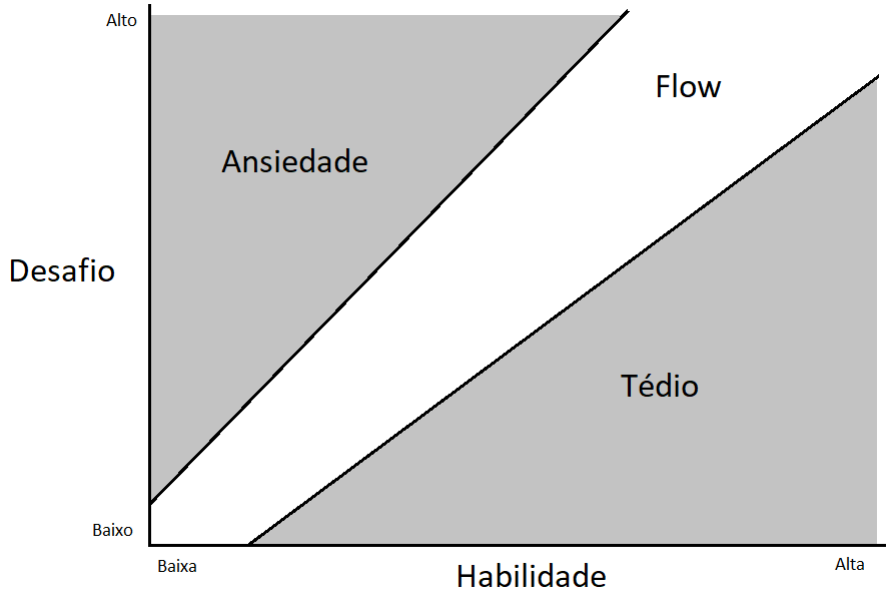



Figura 3.1: Modelo da teoria de Flow.

Outro modelo foi introduzido por Lennart E. Nacke et al. [67]. Esse, denominado BrainHex, pretende prover uma tipologia de preferências de jogatina baseada em aspectos neurobiológicos. Os autores definem sete tipos diferentes de jogadores: *Seeker*, *Survivor*, *Daredevil*, *Mastermind*, *Conqueror*, *Socialiser* e *Achiever*. Os *Seekers* têm curiosidade sobre o mundo do jogo. *Survivors* têm motivação provindo de experiências intensas muito relacionadas a jogos de terror. *Daredevil* são jogadores que buscam viver no limite e tiram proveito de situações de risco. *Mastermind* preferem o lado estratégico e na solução de puzzles, realizando ações o mais eficiente possível. *Conqueror* preferem situações adversas, inimigos quase impossíveis. Eles buscam desafios que sejam difíceis de serem vencidos. Ambos *Socialiser* e *Achiever* são similares à teoria de Bartle.

Por fim, Quantic Foudry uma empresa de pesquisa de marketing com foco em motivação de *gamers*, combinando ciência dos dados e ciência social, desenvolveu um modelo [2] com tipos de jogadores e suas respectivas motivações. Para tal, analisaram um total de 1.25 milhões de jogadores e reivindicam que resolvem problemas estatísticos presentes em propostas anteriores. Eles encontraram seis grupos de motivação em que cada um deles possui dois fatores:

- *Action*
  - *Destruction* está relacionado ao entretenimento provindo de caos e destruição.
  - *Excitement* são jogadores que curtem jogos rápidos e intensos que provem adrenalina constantemente.
- *Social*

  - *Competition* se refere ao proveito de jogar contra outros jogadores em cenários competitivos.
  - *Community* define preferencia sobre colaboração e socialização com outros jogadores.

- *Mastery*
  - *Challenge* define a preferência sobre jogos que requerem muito da habilidade do usuário.
  - *Strategy* são jogadores que focam mais no lado de decisão e planejamento.

- *Achievement*
  - *Completion* procuram terminar tudo que um jogo tem a oferecer.
  - *Power* buscam o poder no contexto do mundo do jogo.

- *Immersion*
  - *Fantasy* são jogadores que buscam experiências que o permitam sentir-se o personagem no mundo.
  - *Story* almejam estórias elaboradas com personagens multidimensionais e complexos.

- *Creativity*
  - *Design* são jogadores que buscam expressar sua individualidade através do jogo, tirando proveito da customização.
  - *Discovery* são jogadores que gostam de experimentar no mundo do jogo em que estão inseridos, tentando coisas novas e ver suas consequências.

Com uma ampla variedade de motivações e tipos de jogadores, definir singularmente percepção de dificuldade se torna um grande desafio. Entretanto, para um sistema DDA é necessária uma forma de verificar o desbalanceamento de uma partida em tempo real. Dvir Ben Or et al. [74] identificou em trabalhos prévios na área três principais formas de aproximar a dificuldade percebida pelo usuário em uma métrica.

### 3.1.1 Medições de sinais físicos

O primeiro grupo identificado utiliza medições de reações físicas do usuário para estimar a dificuldade, assim realizado em trabalhos como [54, 92, 19, 76, 12, 66, 78]. Por

exemplo, Anders Drachen et al. [26] estudaram a correlação de medições de batimentos cardíacos (do inglês *heart rate* - HR) e atividade eletrodérmica (do inglês *electrodermal activity* - EDA) com as experiências subjetivas reportadas por usuários. O objetivo era verificar a validade dessas métricas para inferir a excitação do usuário perante o jogo e concluíram que existe uma correlação significante entre os fatores. Adi Stein et al. [92], por exemplo, utilizaram um eletroencefalograma para ler a atividade mental do jogador e identificar o estado em que ele se encontra. Quando era detectada uma redução na excitação, o sistema DDA atuava ajustando a dificuldade do jogo. Usualmente, trabalhos desse tipo coletam diferentes sinais físicos e utilizam um classificador para avaliar o estado emocional do usuário. Os classificadores visam prever o nível de ansiedade ou excitação para que o método DDA possa modificar o jogo de acordo. Embora resultados promissores foram apresentados por esse tipo de métrica, é necessário um equipamento especializado e um ambiente controlado para serem utilizados. Assim, a aplicabilidade em cenários reais e para o público geral é reduzida. Uma alternativa foi proposta por trabalhos como [64, 4], em que utilizam expressões faciais do usuário para avaliar o seu estado mental. Embora não tão preciso quanto sinais físicos, o equipamento necessário é bem mais amplo e acessível. Por fim, outra proposta é a utilização da avaliação do próprio usuário sobre sua emoção atual sobre o jogo. Em [30], os autores utilizaram autoavaliações de jogadores durante o período de jogo para modificar a dificuldade. Uma vantagem desse método é o controle do usuário sobre a dificuldade que será oferecida, mas para que isso ocorra, interrupções são necessárias para a coleta de opinião. Os autores discutiram a possibilidade de inserir em diálogos de NPCs esse questionário para reduzir esse efeito.

### 3.1.2 Estados do jogo

O segundo tipo de métrica identificado baseia-se na avaliação dos estados do jogo [21, 24, 43, 44, 86, 53, 65]. Métodos dessa categoria definem um conjunto de estados considerados ideais, e o sistema de DDA atua ajustando os parâmetros do jogo para manter o jogador dentro desses estados. Para isso, heurísticas são empregadas com o objetivo de identificar quais estados são desejáveis e quais devem ser evitados. Ambos trabalhos [100, 22] utilizaram a métrica de diferença de vida dos personagens para tratar os estados desejáveis. Vang [100], em específico, desenvolveu três variações do método de busca de Monte Carlo (do inglês Monte-Carlo Tree Search - MCTS). Em uma das variações, o valor do estado é avaliado baseado na diferença de vida dos personagens. Os estados cuja diferença é pequena são avaliados positivamente, e o método escolhe ações que levam a esses estados. Yannakakis e Hallam [105] analisaram o número de etapas até o

jogador perecer no jogo de Pac-Man. Os autores argumentaram que, observando a média de número de passos próximo do mínimo, resultam em jogos muito difíceis, enquanto esse valor aproximadamente igual ao máximo indica um jogo muito fácil. Assim, baseado na variação desse fator, o método modificava a dificuldade. Por fim, em trabalhos como [72, 103], a diferença de pontuação foi a métrica avaliada. O primeiro foi aplicado em um jogo RTS. A pontuação foi calculada baseada em fatores do jogo, como número de tropas, construções, entre outros. O segundo trabalho modifica a dificuldade individual para cada jogador em uma partida baseada na pontuação de cada um.

### 3.1.3 Desempenho do jogador

Por fim, o último tipo de métrica foca no desempenho do jogador. Os métodos se baseiam na ideia principal de que usuários com maior habilidade devem enfrentar jogos mais difíceis. Com isso, os métodos tentam prever, de alguma forma, a proficiência do jogador e toma ações baseadas de acordo com essa previsão [109, 74, 63, 81, 103, 37, 23, 106, 82, 38]. Goetschalckx [33] modelou o problema de adaptação de jogos em um *Partially Observable Markov Decision Process* (POMDP) em que alguns aspectos não são diretamente observáveis. Esses fatores considerados pelo trabalho são: habilidade, entretenimento e tipo de jogador. O sistema mantém um estado de crenças, uma distribuição de probabilidade sobre todos os estados. Em trabalhos como [98, 104, 35, 93], geraram diferentes fases do jogo de modo a apresentar desafios apropriados para o grau de proficiência do jogador. Nestes, variáveis do jogo são usadas para modelar a habilidade, como progresso total, probabilidade de vitória e tempo total na fase. Assim, baseado na proficiência calculada, a fase é adaptada para melhor ajustar às habilidades do jogador.

## 3.2 Mecanismo de Ajuste

Sistemas DDA podem intervir em diferentes aspectos do jogo para atingir seu objetivo. A abordagem escolhida depende tanto do tipo de jogo quanto das decisões dos desenvolvedores. Em muitos casos, pode-se modelar de diferentes formas o ajuste em um mesmo jogo.

Trabalhos como [44, 74, 109, 86] focaram em modificar variáveis do jogo. Especificamente, Hunicke e Chapman [44], em um dos primeiros trabalhos na área, criaram o

sistema Hamlet. Ele manipulava entidades tanto em cena (modificando dano, vida entre outros) quanto inimigos que ainda surgiriam na fase para evitar estados indesejados de desbalanceamento. Sekhavat [86] implementou um mecanismo de ajuste dinâmico de dificuldade em jogos terapêuticos. O sistema adapta variáveis como o tamanho e a velocidade dos objetos, além de outros fatores, com o objetivo de otimizar a efetividade do tratamento.

Outros trabalhos focaram em modificar o cenário em si para torná-lo mais fácil ou difícil. Miguel et al. [35] utilizou uma abordagem probabilística Bayesiana para modificar a dificuldade do mapa de ambos os jogos Sudoku e um *Rouguelike*. O método ajusta a dificuldade variando o posicionamento de paredes e oponentes no *Roguelike*, enquanto, no caso do Sudoku, altera as posições das células previamente preenchidas. Xue et. al. [104] criou um sistema para o jogo Candy Crush. Com um gráfico probabilístico, a chance de um jogador concluir, falhar ou desistir de uma certa fase são calculadas. Baseado nesse modelo, sementes aleatórias utilizadas para gerar um nível são amostradas para ser compatível com a habilidade do atual jogador. Assim, configurações de fases mais adequadas ao usuário são escolhidas.

Por fim, outro aspecto que métodos DDA podem atuar é no comportamento de personagens não jogáveis (do inglês *Non-Playable Character* - NPC), alterando sua proficiência como em [103, 79, 37, 88, 70]. Silva [88] desenvolveu um método DDA aplicado para um jogo *Multiplayer online battle arena* (MOBA). O sistema proposto alterna entre três possíveis níveis de dificuldade (fácil, médio e difícil) definido por *scripts*, comportamentos fixos pré-programados. Para decidir qual utilizar a cada momento, o desempenho do jogador e do oponente é calculado e a diferença é utilizada para avaliar se o jogo é fácil, balanceado ou difícil. Noblega et al. [70] propôs uma abordagem baseada em aprendizado por reforço para balancear o jogo. O autor focou na habilidade relativa entre o agente e o jogador, propondo uma função de recompensa que orienta o comportamento do agente a manter a diferença de vida entre os dois dentro de um limite máximo.

## 3.3 Métodos

Nesta seção, são apresentados alguns dos métodos utilizados, tanto na indústria quanto na pesquisa, para o desenvolvimento de sistemas de ajuste dinâmico de dificuldade.

### 3.3.1 Aplicações comerciais

Ajuste dinâmico de dificuldade já foi utilizado na indústria de games em grandes títulos. *Left 4 Dead* [91] utiliza um sistema chamado *Diretor*. O papel dele é gerenciar a dificuldade durante uma partida. Para tal, ele atua principalmente em decidir quando ocorrerá uma horda de inimigos e quais tipos irão compor ela para atacar os jogadores. Além disso, pode modificar o layout do mapa, acrescentar armas, itens, entre outras possíveis ações. Ele realiza tais intervenções baseado no número de recursos que os jogadores possuem (vida, armas, itens, etc.). Outro game que utiliza algum ajuste de dificuldade é a famosa série de jogos de corrida Mario Kart [69]. Nele, existem alguns artifícios para manter a corrida balanceada. Primeiro, a chance de se conseguir itens mais valiosos durante a corrida depende da posição. Um competidor que estiver na primeira posição terá uma chance menor do que os últimos colocados. Outra técnica é chamada de *Rubber-Band*. NPCs terão vantagens ou desvantagens dependendo da distância do jogador. Caso o jogador esteja com muita vantagem, os oponentes controlados pelo computador terão um aumento na velocidade. No outro cenário no qual o jogador está muito atrás, eles terão velocidade reduzida e colidirão mais. Com isso, o objetivo é tentar manter a corrida interessante e competitiva.

### 3.3.2 Modelos Probabilísticos

Um dos primeiros estudos em DDA [44] propõe um método aplicado ao Half life. Este, nomeado Hamlet, utiliza a distribuição de dano recebida pelo jogador para tentar prever estados futuros. Caso ele encontre algum indesejado e evitável, ele intervê. Su et. al. [104] utilizou um grafo probabilístico de modo a modelar a probabilidade de um jogador vencer a fase, perder ou desistir do jogo completamente no Candy Crush [45]. Os nós representam esses três estados para uma determinada fase. Dessa forma, maximizar o engajamento é equivalente a maximizar o número de transições nesse grafo, visto que o estado terminal de desistência é indesejado. Com essas probabilidades, sementes aleatórias que geram as fases são classificadas quanto à dificuldade e escolhidas para maximizar o entretenimento. [35] modelou o problema para atribuir fases do jogo que apresentam um grau de dificuldade compatível ao jogador. Para tal, o método usa Otimização Bayesiana para fazer a melhor seleção. Para o trabalho, o logaritmo do tempo total necessário para resolver a fase é avaliado. O objetivo do otimizador é aproximar a variável para uma ideal de tempo determinado pelos autores. Similarmente, Miguel et. al. [34] utilizou o

mesmo modelo Bayesiano com uma forma de otimização rápida. Os autores propuseram o processo completo desde a geração dos mapas, modelagem do jogador via Árvores de Busca Monte Carlo e ajuste de dificuldade.

### 3.3.3 Heurísticas

Silva [88] desenvolveu um mecanismo de ajuste dinâmico de dificuldade para jogos MOBA. O método atua alternando entre três níveis de dificuldade (fácil, médio e difícil) definidos por comportamentos pré-programados e fixos. Baseado em fatores do jogo (como número de mortes, nível e torres derrotadas), o desempenho atual é comparado ao obtido pelo oponente e o nível de dificuldade é ajustado se a diferença for maior que o hiper parâmetro definido $\beta$. Knorr e Vaz [49] apresentaram um método que ajusta parâmetros de inimigos e nos itens a serem apresentados para o jogador no jogo *Half-life*. Os autores descreveram um conjunto de 155 regras que podem ser ativadas. Essas modificam parâmetros distintos do jogo baseados em estimativas locais e globais de desempenho. O uso das regras é controlado por um conjunto de pesos, e esses são atualizados toda vez que é verificado se essa teve impacto positivo ou negativo no ajuste de dificuldade. Sutoyo et. al. Miguel et. al. [94] aplicou um sistema DDA em um jogo de *Tower Defense*. De modo a realizar as atualizações necessárias, os autores observaram três parâmetros diferentes que definem o desempenho: vida do jogador e dos inimigos, assim como habilidades adquiridas durante uma partida. Ao final da partida, baseado no resultado observado, o conjunto de multiplicadores é atualizado. Esses definem os parâmetros do jogo, como tempo de surgimento de inimigos, geração de ouro, entre outros, que modificam a dificuldade experienciada pelo jogador.

### 3.3.4 Aprendizado Supervisionado

Em [79], o artigo propôs criar um modelo que aprende com as ações do jogador, para realizar tomadas de decisões parecidas ao longo do tempo. Com isso, o modelo apresenta um nível de proficiência similar ao jogador em questão. O sistema foi aplicado para o jogo *Aion*, estilo *Massively multiplayer online role-playing game (MMORPG)*, focado na parte de catacumbas. Ben Or et al. [74] utilizou aprendizado supervisionado e clusterização para desenvolver um sistema DDA. Esse conseguia otimizar automaticamente a experiência do

usuário considerando outros jogadores e restrições globais impostas pelo jogo. Para tal, ele define uma função de perda com dois objetivos. O primeiro é garantir que a dificuldade se ajuste ao perfil do jogador, obtida pelo erro entre a dificuldade predita e a real estipulada no treino. A segunda parte define que jogadores semelhantes devem ter dificuldades semelhantes, calculadas a partir da variância do cluster daquele jogador.

### 3.3.5 Métodos Evolutivos

Jacob et. al. [72] utilizou um método baseado em NEAT para criar um agente de DDA em tempo real para jogos RTS. NEAT é um método para construir automaticamente redes neurais por meio do método evolutivo TWEANNs. A evolução é realizada nos pesos e na topologia, permitindo alterar esses fatores em busca da rede ótima. Para a métrica de desafio, foram usadas (i) Número de guerreiros; (ii) Agressividade; (iii) Infraestrutura; (iv) Infraestrutura defensiva; (v) Média de treino de guerreiros; (vi) Número de trabalhadores. Para validar essas métricas, foram testados os dois primeiros parâmetros. Os agentes foram treinados apenas com o respectivo parâmetro e testados com o comportamento já programado no jogo. Com isso, foi possível verificar a validade de cada parâmetro separadamente e sua contribuição para o nível de desafio. Em [103], os autores aplicaram um algoritmo genético para um jogo de tabuleiro cuja condição de vitória é dada pela coleta de recursos e construção de bases. O sistema age alterando fatores do jogo, dessa forma um cromossomo foi modelado como essas características. O gene é codificado como a probabilidade de um determinado jogador conseguir coletar um tipo de recurso. Quanto menor a probabilidade, mais difícil o jogo. O fitness é definido pela diferença do score do jogador contra o pior. Cada um dos usuários possui uma população que, ao longo do jogo, é classificada pela função fitness. Quando todas são avaliadas, ocorre o processo de crossover e mutação. O objetivo final é reduzir a diferença entre as pontuações dos jogadores, modificando as probabilidades. Um jogo completamente balanceado ocorre quando não existe diferença entre os scores.

### 3.3.6 Aprendizado por Reforço

Diversos trabalhos que aplicaram aprendizado por reforço em tarefas de ajuste dinâmico de dificuldade utilizaram métodos de *Monte Carlo tree search* (MCTS). Sha et

al. [87] construíram um método de ajustar a dificuldade limitando o tempo total de busca, assim reduzindo a precisão e desempenho do agente. Em [22], criou variações do método para gerar um agente que apresente habilidade relativa similar ao jogador em um jogo de luta. Para tal, duas variações foram propostas: (i) Gera a árvore de busca normalmente visando alcançar a vitória, mas seleciona as ações que levam à diferença de vida próximas a zero. (ii) Similarmente, gera a árvore de busca normalmente, porém qualquer ação que leve a uma diferença de vida próxima a um intervalo envolto no zero são consideradas igualmente. Além disso, versões de ambas que expandem a árvore baseadas nas suas heurísticas. Kazuhisa [31] modificou o método AlphaZero para balancear um jogo em vez de ganhá-lo. O autor testou três modificações: (i) Número de expansões baseado no valor do estado determinado pela rede neural. (ii) Uso de *dropout* para danificar a rede. O valor da probabilidade é alterado conforme o valor do estado gerado pela rede sem *dropout*. (iii) MCTS modifica a busca baseada na distância de habilidade do jogador comparado a uma crença de sua habilidade média.

Para além de métodos de busca, Grau-Moya et al. [37] expandiu o framework de aprendizado por reforço para jogos estocásticos. O agente recebe um conjunto de restrições que permitem modificar o seu comportamento. Modelado como um jogo de dois jogadores, soma-se à função objetivo essas restrições. De modo a controlar o comportamento, cada uma dessas possui multiplicadores lagrangianos que controlam a distância da política atual a uma política completamente racional e com desempenho ótimo. Além disso, através do sinal desses multiplicadores é possível determinar o tipo de jogo, sendo possível ser competitivo ou cooperativo.

Um dos trabalhos que foi base deste é o Noblega et al. [70]. Os autores propuseram uma função de recompensa que guia o agente a manter o jogo balanceado. Para tal, o sistema leva em consideração a diferença de habilidade $\Delta Skill$ do jogador e do personagem controlado pelo agente DDA. Os autores introduziram o conceito de intervalo de balanceamento $I_{BC} \in [0, BC]$, em que $BC$ é uma constante. Esse define os limites mínimo e máximo que a métrica $\Delta Skill$ pode obter e o sistema ainda considera o jogo balanceado. Note que em $I_{BC}$ o limite inferior é 0, incentivando que o sistema apresente um desafio levemente maior que a habilidade do jogador, mantendo a dificuldade mais alta, porém limitada. Assim, é possível enumerar três distintos estados do jogo. O primeiro é quando o jogo está muito fácil, com $\Delta Skill < 0$. O segundo é quando $\Delta Skill > BC$, demonstrando o jogo muito difícil para o jogador. Por fim, o último estado é quando o jogo apresenta um nível de dificuldade ideal com $0 < \Delta Skill \leq BC$. Os autores apresentaram uma função de recompensa que direciona o agente a manter a métrica dentro do intervalo desejado. Para tal, o agente recebe, a cada passo, uma recompensa positiva quando o jogo está balanceado e negativa caso contrário. Além disso, no estado balanceado, a recompensa aumenta linearmente conforme o valor de $\Delta Skill$ se aproxima do limite superior do intervalo. Assim, incentivando o sistema a manter uma dificuldade avaliada próxima a $BC$. A

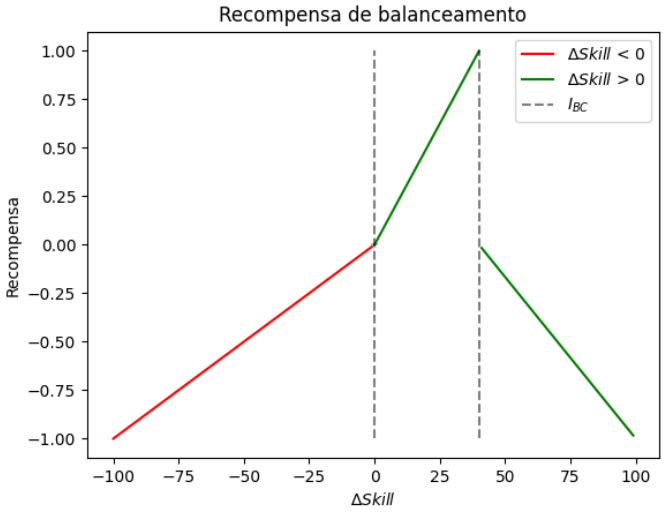



Figura 3.2: Função de recompensa proposta em [70] normalizada pelo máximo de vida. O intervalo dentro da linha pontilhada é considerado balanceado.

Figura 3.2 mostra de forma gráfica essa função de recompensa. O método foi testado em um jogo simples desenvolvido pelos autores. Verificou-se que o agente conseguia manter a diferença de habilidade relativa dentro do intervalo estipulado.

## 3.4 Discussão

O Ajuste Dinâmico de Dificuldade (DDA), apesar de ser uma área relativamente recente na pesquisa, já conta com diversos estudos que investigam os múltiplos aspectos envolvidos em sua implementação. Esses trabalhos propõem diferentes formas de estimar a dificuldade percebida pelo jogador. Entre elas, destacam-se abordagens baseadas em sinais fisiológicos, capturados por equipamentos especializados ou por meio da análise de expressões faciais; métodos que avaliam diretamente os estados do jogo para manter o jogador em cenários consideradas ideais; e métricas baseadas na proficiência demonstrada pelo jogador ao longo da interação com o sistema.

Além disso, as soluções variam quanto à forma de intervenção no jogo. Algumas modificam diretamente parâmetros como vida ou dano dos oponentes. Outras ajustam a dificuldade por meio de alterações no próprio cenário, tornando certas fases mais fáceis ou desafiadoras conforme necessário. Há ainda abordagens que alteram o nível de habilidade dos oponentes enfrentados, apresentando inimigos mais competentes à medida que o jogador demonstra maior domínio do jogo.

Complementando os dois principais componentes de um sistema DDA, a medição

da dificuldade percebida e o mecanismo de ajuste, a literatura também explora diferentes métodos para mapear essa métrica em ações concretas. Como discutido anteriormente, foram testadas técnicas de diversas áreas, que vão desde regras simples até abordagens baseadas em aprendizado profundo. No entanto, apesar do sucesso do Aprendizado por Reforço (RL), especialmente quando combinado com redes neurais profundas em diferentes domínios, seu uso ainda é pouco explorado no contexto de DDA.

Dessa forma, este trabalho tem como objetivo investigar formas de aplicação do Aprendizado por Reforço no contexto de Ajuste Dinâmico de Dificuldade (DDA). Para isso, utiliza-se o Aprendizado Profundo em conjunto, visando viabilizar sua aplicação em cenários reais e complexos. Além disso, busca-se desenvolver uma solução robusta e generalizável, capaz de se adaptar a diferentes perfis de jogadores, com foco na capacidade de generalização do modelo.

# Capítulo 4

# Jogos de Luta - Plataforma FightingICE

Jogos de luta clássicos são confrontos diretos entre dois jogadores. Ambos iniciam a partida com uma quantidade limitada de vida e têm como objetivo reduzir a vida do oponente por meio de ataques. A luta pode terminar de duas formas: quando o tempo total da rodada se esgota ou quando a vida de um dos jogadores chega a zero. O vencedor da rodada é aquele que possuir mais vida ao final do tempo. Geralmente, uma luta completa é composta por múltiplas rodadas. Cada jogador tem acesso a um conjunto de golpes que variam conforme o personagem escolhido. Esses ataques possuem diferentes características, influenciando sua velocidade, alcance e dano. Uma classificação comum em jogos do gênero separa os golpes em altos, médios e baixos, determinando a região do corpo do adversário que será atingida. Da mesma forma, a defesa segue essa estrutura: um jogador pode bloquear um golpe alto, mas permanecer vulnerável a ataques baixos.

Além dos ataques básicos, muitos jogos incluem golpes com grande alcance ou projéteis, permitindo ataques à distância. Também podem existir movimentos especiais que causam dano elevado, mas exigem o consumo de recursos ou o cumprimento de condições específicas. Para executar esses golpes mais elaborados, os jogadores precisam realizar sequências específicas de comandos. Outro aspecto estratégico dos jogos de luta são os efeitos causados ao ser atingido por um golpe. Dependendo do ataque sofrido, o jogador pode ser momentaneamente imobilizado, empurrado para outra posição ou impedido de agir por um curto período de tempo.

Com a ampla variedade de golpes disponíveis, há múltiplas formas de vencer uma luta. Alguns jogadores adotam uma abordagem agressiva, permanecendo próximos ao oponente e buscando o confronto direto. Outros preferem estratégias defensivas, reagindo às investidas adversárias e mantendo uma distância segura. No entanto, o estilo de luta não é definido apenas pela preferência do jogador, mas também como uma resposta adaptativa ao comportamento do oponente. Em geral, há uma relação de dominância cíclica entre os estilos de combate. Por exemplo, um jogador agressivo tende a ter dificuldades contra um adversário defensivo, que, por sua vez, pode ser vulnerável a um oponente que utiliza ataques à distância. Esse padrão reflete a não transitividade, uma propriedade

comum nesse tipo de jogo. Em jogos não transitivos, se um agente com estratégia $x$ vence outro com estratégia $y$, e $y$ vence um terceiro jogador com estratégia $z$, isso não implica que $x$ vencerá $z$. Essa característica impede a existência de uma única estratégia dominante, resultando em um ciclo de dominância. O exemplo clássico dessa propriedade é o jogo Pedra-Papel-Tesoura. Além disso, jogos de luta exigem que o jogador tome decisões em tempo real dentro de um vasto espaço de estados, tornando-os um ambiente desafiador e interessante para o desenvolvimento de métodos DDA. Esse contexto também permite explorar a diversidade de estratégias e desenvolver agentes DDA que sejam invariantes tanto à habilidade relativa do jogador quanto ao seu estilo de jogo.

DareFightingICE (antigo FightingICE - FTG) [1] é um jogo de luta desenvolvido pelo Ritsumeikan University Intelligent Computer Entertainment Lab (ICE Lab) [59] para fins de pesquisa. O laboratório realiza competições anuais desde 2013 investigando se é possível desenvolver um método de inteligência artificial para jogos de luta que seja forte contra qualquer oponente, seja esse humano ou outro agente, em qualquer modo de jogo utilizando qualquer personagem (e seus respectivos parâmetros). Para as competições, qualquer pessoa pode enviar seu método. Diversos laboratórios, pesquisadores e desenvolvedores independentes enviaram seus modelos ao longo dos anos. O simulador suporta as linguagens de programação Java e Python. Além disso, especificamente para Python, é ofertada uma interface Gym [14], facilitando a utilização de métodos de aprendizado por reforço, visto que a interação com o simulador é formatada no estilo de Processo de decisão de Markov, como discutido na seção 2. FightingICE (FTG) foi concebido com atraso de informações para simular o tempo de reação de um ser humano e introduzir risco a comportamentos defensivos. O simulador permite acesso à informação do jogo por meio de estruturas de dados (informações do jogador, oponente, personagem entre outros) ou por dados visuais (imagem da tela). Uma imagem do simulador é apresentada na Figura 4.1.

O simulador foi desenvolvido inspirado em grandes clássicos do gênero de luta e compartilha diversas características desses jogos. Um desses fatores é o recurso para utilização de habilidades. Denominado de energia, o jogador recebe conforme sofre dano ou acerta golpes. Habilidades especiais, usualmente mais fortes e com efeitos únicos, gastam essa energia. Um aspecto específico do FTG é a classificação de golpes. Além dos padrões de baixo, médio e alto, soma-se o tipo *Throw*, que causam um deslocamento do oponente. Além disso, existe outra diferenciação de golpes que separa em aéreos e no chão. Assim, existe um subconjunto de ações que só podem ser realizadas enquanto o personagem está no ar e, similarmente, outros que são válidos enquanto está no chão. Cada golpe realizado pode ser bloqueado ou desviado dependendo do tipo de defesa sendo executado pelo oponente. Essa dinâmica de defesa é apresentada na Figura 4.1.

Com um longo histórico de competições, o simulador oferece uma grande base

[1] O site oficial do simulador https://www.ice.ci.ritsumei.ac.jp/~ftgaic/index.htm


Figura 4.1: Imagem do simulador.

| | Guardas | | |
|---|---|---|---|
| **Tipo de ataque** | Em pé | Agachado | No ar |
| Alto | bloqueia | bloqueia | bloqueia |
| Médio | bloqueia | acerta | bloqueia |
| Baixo | acerta | bloqueia | acerta |
| Arremessar | acerta | acerta | erra |

Tabela 4.1: Tabela de dinâmica de defesa do simulador utilizado.

de testes com oponentes variados e diferentes níveis de habilidade. Neste estudo, em que se almeja um agente DDA robusto contra qualquer jogador, tanto as habilidades relativas quanto estilo são fatores de interesse para a avaliação do modelo. A versão 5.2 do simulador foi utilizada neste trabalho. Foram selecionados todos os competidores dos torneios do período de 2018 a 2021 e filtramos apenas os criados na linguagem Java. Além disso, alguns não puderam ser utilizados por questões de compatibilidade com a versão escolhida, totalizando 25 métodos. Para compreender melhor os resultados e ter uma base de comparação, um torneio foi realizado entre os competidores coletados para avaliar a habilidade relativa deles. O ranqueamento criado e apresentado na Figura 4.2 foi feito baseado no número total de vitórias, assim como feito em competições oficiais do FTG. Empates não são contabilizados. Cada competidor foi testado contra todos os outros em 100 partidas.

Algumas modificações foram realizadas no simulador [2] de modo a otimizar o treinamento ou introduzir funcionalidades para aprimorar o agente resultante. O simulador permite a visualização gráfica e reprodução de efeitos sonoros nas partidas. Embora sejam fatores primordiais para jogos, durante o treinamento, apenas informações numéricas são utilizadas, e qualquer informação visual é desnecessária. O FTG permite desabilitar

[2] O repositório disponibilizado com as modificações realizadas https://github.com/TiagoNO/FightingICE

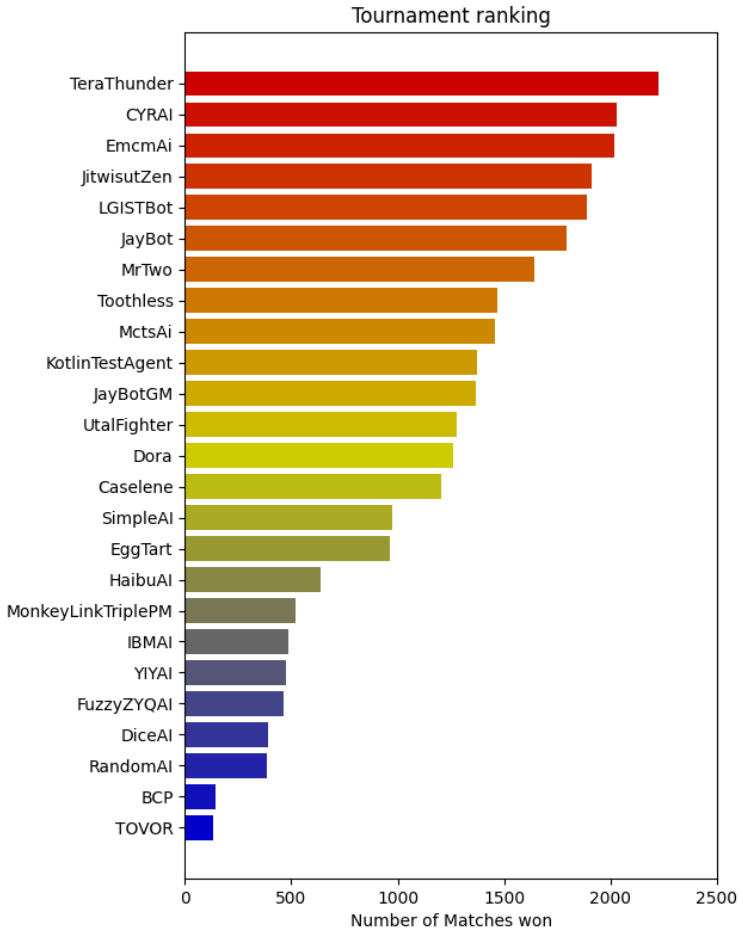



Figura 4.2: Resultado do ranqueamento feito para esse estudo. A pontuação é o número de vitórias total. Cada competidor jogou 100 partidas contra todos os outros do torneio. O ranqueamento é realizado no número de vitórias. Empates não são contabilizados. As cores codificam oponentes em fortes (vermelho), médios (amarelo) e fracos (azul).

a janela de visualização, mas o jogo ainda é renderizado e os áudios ainda são carregados. Para tal, o sistema utiliza desnecessariamente memória e tempo de processamento, dificultando a utilização de múltiplos ambientes. Dessa forma, a versão criada remove qualquer renderização ou efeito sonoro quando o comando de desabilitar a tela é ativado. Outra modificação é o posicionamento ao início de cada luta. Assim como é o padrão em jogos clássicos de luta, um jogador é posicionado sempre na esquerda e outro na direita, e entre lutas esse padrão se mantém. Entretanto, apresentar ao agente apenas uma posição inicial pode resultar em uma política que não consiga ser eficiente quando posicionada no outro lado. Assim, a modificação altera a cada luta a posição inicial dos jogadores.

## 4.1 Representação do Estado

O ambiente Gym permite duas representações de estado possíveis. A primeira são imagens da tela. A segunda forma é um vetor de valores que representam informações que poderiam ser obtidas observando a tela do jogo. Para este trabalho, a segunda forma foi escolhida. Além disso, o número de informações foi expandido. O vetor total é a agregação de cinco tipos de informações: dados do personagem de ambos jogadores, dados da ação atual que cada um está executando, dados dos projéteis em cena, dados do jogo e, por fim, dados do comportamento do oponente. Os valores foram normalizados entre $[0, 1]$. A descrição resumida das informações usadas e tipo estão descritas na Tabela 4.2.

O ambiente Gym oferece diferentes estratégias para determinar quando o agente recebe informações e executa ações. A abordagem mais simples consiste em fornecer dados ao agente a cada frame, exigindo que ele tome uma decisão continuamente. Dessa forma, toda a partida é observada. No entanto, esse método apresenta desafios, pois algumas ações podem deixar o personagem temporariamente sem controle de seus movimentos. Isso pode ocorrer quando uma ação demora mais que um frame para ser concluída, mantendo o personagem preso até o término da animação, ou quando ele recebe um golpe que o incapacita por um período. Embora essa abordagem forneça um maior volume de informações, muitas ações tomadas pelo agente não terão efeito, o que pode prejudicar o treinamento e retardar a convergência da política. A alternativa oferecida pelo simulador é permitir que o ambiente apenas envie informações e espere que o agente tome uma decisão quando o personagem estiver em pleno controle e puder executar um comando. Nessa configuração, o agente recebe dados antes e depois de concluir uma ação, tornando a atribuição de valor mais direta e simplificada. O custo dessa abordagem é a perda de informações durante a execução de ações. Este trabalho adota essa segunda estratégia para facilitar o aprendizado do agente.

| Informação do Personagem | | | |
|---|---|---|---|
| Nome | Tipo | Tamanho | Descrição |
| Vida | float | 1 | Vida total atual |
| Energia | float | 1 | Energia disponível para habilidades |
| Direção | bool | 1 | Direção personagem está orientado |
| Posição | float | 2 | Ponto (x,y) do centro do personagem |
| Colisor | float | 4 | AABB responsável para determinar colisões |
| Acertou | bool | 1 | Booleano determina se acertou golpe |
| Combo | int | 1 | Número de golpes consecutivos |
| Recuperação | int | 1 | Número de frame até recuperar de um golpe |
| Velocidade | float | 2 | Velocidade total |
| Controle | bool | 1 | Booleano determina se personagem pode executar ação |
| Encurralado | Bool | 1 | Booleano que verifica se está em preso em um canto |
| Estado | One-Hot | 4 | Estado de recuperação de um golpe |
| Informação da Última Ação | | | |
| Posição | float | 2 | Posição central do golpe |
| Colisor | float | 4 | AABB responsável para determinar colisão |
| Dano | float | 1 | Dano total (sem defesa) |
| Recuperação | int | 1 | Frames necessários para recuperar do golpe |
| Impacto | float | 2 | Força do impacto |
| Velocidade | float | 2 | Velocidade do golpe |
| Energia | float | 3 | Energia atribuída ao ser atingido |
| Contundente | float | 1 | Booleano determina se gera contusão |
| Tipo | One-Hot | 4 | Determina o tipo de ação para sistema de defesa |
| Ação | One-Hot | 42 | Identificador da ação |
| Informação dos Projéteis | | | |
| Posição | float | 2 | Posição atual do projétil |
| Colisor | float | 4 | AABB responsável para determinar colisão |
| Dano | float | 1 | Dano total (sem defesa) |
| Impacto | float | 2 | Força do impacto |
| Velocidade | float | 2 | Velocidade do projétil |
| Energia | float | 3 | Energia atribuída ao ser atingido |
| Contundente | float | 1 | Booleano determina se gera contusão |
| Informação do Jogo | | | |
| Distância | float | 2 | Distância atual entre personagens |
| Tempo | float | 1 | Tempo restante da partida |
| Informação do Oponente | | | |
| Distância | float | 2 | Distância média entre personagens |
| DPF | float | 1 | Dano por frame |
| EPF | float | 1 | Energia gasta por frame |
| Projéteis | float | 1 | Número de projéteis utilizado |
| Aérea | float | 1 | Número de ações aéreas utilizado |
| Posições | float | 10 | Últimas 5 posições |
| Histograma | float | 42 | Distribuição das ações realizadas |

Tabela 4.2: Definição do estado juntamente com cada informação obtida pelo agente e uma breve descrição.

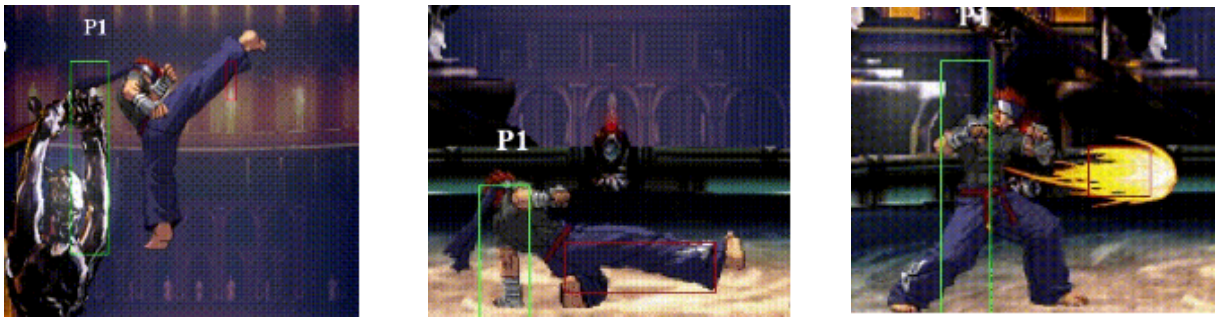



Figura 4.3: Exemplos de golpes presentes no simulador. A figura da esquerda é um chute aéreo, a do meio é uma rasteira e a da direita cria um projétil de energia.

## 4.2 Representação das Ações

Cada personagem possui uma gama de possíveis golpes a serem realizados. Alguns simples, como socos e chutes, que podem ser realizados com apenas um botão, enquanto outros mais fortes requerem que o jogador pressione uma sequência específica de teclas para a sua realização. A Figura 4.3 apresenta um exemplo de diferentes golpes. Treinar um agente com a representação de teclas prolonga o treinamento, visto que o agente não apenas precisa descobrir como ganhar o jogo, mas quais sequências de botões realizam cada golpe. Além disso, requer informações dos passos anteriores. Para solucionar tal problema, o simulador oferece tanto a interface de ação por teclas como por *tags*. Essas representam quais ações o agente quer realizar e um conversor gera a sequência de teclas necessárias para a sua execução. Este trabalho utiliza a segunda interface.

O simulador considera um total de 56 ações disponíveis para o personagem. Dentro dessas, existem algumas que são utilizadas apenas para o simulador e não têm efeito prático para o agente. Dentre elas, estão incluídas, por exemplo, *down*, significando que o agente está caído e sem ação, e *recovery*, que mostra que ele está recuperando de um golpe e está voltando ao estado padrão. De modo a simplificar o treinamento, essas foram retiradas do conjunto de ações e atribuídas às observações. Assim, o agente percebe quando o oponente está caído e evita ações sem efeito. Além disso, foi adicionada a ação *NEUTRAL* que não faz nada. Em trabalhos como [71], foi observado que uma ação desse tipo pode ter diversos benefícios em jogos de luta, por evitarem que o agente perca o controle por estar executando outra ação não ótima em momentos críticos. Com isso, o total de ações finais após o filtro é 42, listados e brevemente explicados na tabela 4.3.

| Ação | Descrição | Ação | Descrição |
|---|---|---|---|
| NEUTRAL | NO OP | STAND_FA | Soco em pé forte |
| FORWARD_WALK | Andar para frente | STAND_FB | Chute em pé forte |
| DASH | Corrida para frente | CROUCH_FA | *Uppercut* fraco |
| BACK_STEP | Movimento para trás | CROUCH_FB | Rasteira |
| CROUCH | Agachar | AIR_FA | Soco aéreo médio |
| JUMP | Pulo | AIR_FB | Chute aéreo médio |
| FOR_JUMP | Pulo para frente | AIR_UA | Chute aéreo forte |
| BACK_JUMP | Pulo para trás | AIR_UB | Chute aéreo forte |
| STAND_GUARD | Defesa em pe | STAND_D_DF_FA | Projétil fraco |
| CROUCH_GUARD | Defesa agachado | STAND_D_DF_FB | Projétil forte |
| AIR_GUARD | Defesa aéreo | STAND_F_D_DFA | *Uppercut* médio |
| THROW_A | Arremeso fraco do oponente | STAND_F_D_DFB | *Uppercut* forte |
| THROW_B | Arremeso forte do oponente | STAND_D_DB_BA | Pulo para frente com golpe |
| STAND_A | Soco em pé | STAND_D_DB_BB | Rasteira com avanço |
| STAND_B | Chute em pé | AIR_D_DF_FA | Projétil fraco aéreo |
| CROUCH_A | Soco agachado | AIR_D_DF_FB | Projétil forte aéreo |
| CROUCH_B | Chute agachado | AIR_F_D_DFA | Soco aéreo fraco com avanço |
| AIR_A | Soco aéreo | AIR_F_D_DFB | Chute aéreo fraco com avanço |
| AIR_B | Chute aéreo | AIR_D_DB_BA | Soco aéreo forte com avanço |
| AIR_DA | Soco aéreo com descida ao chão | AIR_D_DB_BB | Chute aéreo Forte com avanço |
| AIR_DB | Chute aéreo com descida ao chão | STAND_D_DF_FC | Projétil especial |

Tabela 4.3: Tabela de todas as ações utilizadas do simulador com uma breve descrição.

## 4.3 Função de recompensa

Em jogos de luta, cada jogador tem uma quantidade de vida. Um ganhador é definido ao final da partida se a sua vida for maior que a do oponente. Assim, uma ação que cause dano deve ser recompensada positivamente. Similarmente, ao receber dano, o agente deve ser punido. A função de recompensa padrão do ambiente Gym disponibilizada pelo simulador FTG é definida pela equação (4.1), onde $HP^t_{OP}$ e $HP^t_P$ representam a vida do oponente e jogador, respectivamente, no período $t$. Essa equação verifica a diferença do dano recebido e causado pelo jogador entre dois períodos de avaliação. Como no jogo em específico não existem formas de restaurar a vida, essa função retornará positiva toda vez que o jogador causa mais dano do que recebe e negativamente caso contrário.

$$r^t_{HP} = (HP^t_{OP} - HP^{t-1}_{OP}) - (HP^t_P - HP^{t-1}_P) \tag{4.1}$$

A recompensa é normalizada entre $[-1, 1]$ e avaliada a cada período de decisão. Ao final de uma partida, o agente recebe uma recompensa especial distinta da descrita previamente. O valor retornado pela função é dependente do agente ser ou não vitorioso. Caso a sua vida seja maior que a do oponente, o jogador recebe o valor de 1 e, caso contrário, $-1$.

# Capítulo 5

# Aprendizado por Reforço para DDA

Uma forma de tratar a tarefa de ajuste dinâmico de dificuldade é abordá-la como um problema de decisão sequencial. Assim, o sistema deve decidir a cada momento qual ação tomar, de forma que o jogo se mantenha em um estado balanceado. Com essa estruturação, o aprendizado por reforço torna-se uma modelagem direta para o problema. Embora o aprendizado por reforço apresente resultados promissores em diversas aplicações, sua utilização neste cenário traz dois desafios principais. O primeiro é definição do problema a ser solucionado. Assim como discutido na seção 3.1, definir uma métrica precisa de percepção de dificuldade é uma tarefa complexa. Além de uma métrica que capture a dificuldade experienciada pelo jogador, é necessária a definição de uma função de recompensa que direcione o agente a um comportamento que balanceie o jogo. O segundo desafio está relacionado a um problema de aprendizado de máquina, a generalização do modelo. Para aplicações em cenários reais, a política gerada deve ser robusta para diferentes cenários. Neste caso, o agente deve conseguir balancear o jogo para uma gama de diferentes jogadores, que diferem no estilo e proficiência no jogo em si. Dessa forma, fazem-se necessários mecanismos para promover a generalização do agente para cenários não encontrados durante o treinamento. As seções a seguir apresentam a formalização do problema de DDA com RL proposta nessa dissertação e como gerar uma política robusta.

## 5.1 Definição do Problema

O sistema DDA deve tomar ações que direcionem o jogo a apresentar uma dificuldade comparável à proficiência do usuário. Como discutido na seção 3.1, quantificar a percepção de dificuldade por meio de uma métrica única é um desafio complexo e multifatorial. Uma abordagem amplamente utilizada é aproximar essa percepção por meio de métricas baseadas em elementos do jogo, como condições de vitória, estados do jogo ou pontuação obtida. A escolha da métrica mais adequada depende do tipo de jogo em que o sistema está sendo aplicado. O objetivo do método DDA, portanto, é ajustar os

parâmetros do jogo de modo a levar uma métrica de dificuldade $Diff$ a um nível ótimo, compatível com a habilidade do jogador.

Este trabalho formaliza de forma genérica a métrica de dificuldade apresentada no trabalho de Noblega et al. [70] e discutido na seção 3.3.6. Assim, define-se a métrica de dificuldade alvo $B(s) = \frac{Diff_{jogador}(s) - Diff_{alvo}(s)}{Diff_{MAX}} \in [-1, 1]$, em que $s$ é o estado atual do jogo. Essa representa a distância normalizada entre a dificuldade atual aproximada $Diff_{jogador}$ e um grau de dificuldade alvo $Diff_{alvo}$. Assim, a cada estado do jogo é calculado $B(s)$ baseado nas métricas de dificuldade. Ambos $Diff_{jogador}$ e $Diff_{alvo}$ são métricas dependentes do jogo sendo aplicado. Alguns exemplos seriam a pontuação do jogador e uma média baseada na sua habilidade e distância da condição de vitória entre jogadores. $Diff_{MAX}$ é uma constante que normaliza os valores. Nessa escala, valores negativos indicam que o jogo está excessivamente fácil, enquanto valores positivos refletem uma dificuldade superior à proficiência do jogador. O alvo de balanceamento pode variar conforme o tipo de jogo ou com as decisões dos desenvolvedores. Um exemplo seria, em cenários com múltiplos jogadores, a diferença entre o desempenho do jogador e a média de outros participantes. Outra abordagem possível é comparar a performance do usuário com um valor esperado para seu nível, como o tempo médio necessário para completar uma fase ou o número de derrotas. Com essa formulação, a métrica pode indicar três estados distintos ao longo do processo de tomada de decisão.

- **Desafio insuficiente** $B(s) \approx -1$: O jogador percebe o jogo como muito fácil. Nesse caso, o sistema deve aumentar a dificuldade para evitar o tédio.
- **Desafio excessivo** $B(s) \approx 1$: O jogo apresenta um nível de dificuldade muito acima da proficiência do jogador, podendo gerar frustração. Assim, o sistema deve reduzir a dificuldade.
- **Desafio balanceado** $B(s) \approx 0$: A dificuldade do jogo está ajustada à habilidade do jogador, proporcionando uma experiência equilibrada.

Assim, o objetivo do sistema DDA é ajustar os elementos do jogo, de modo a manter a métrica $B(s)$ próxima de zero e maximizar o envolvimento do jogador.

Em experimentos iniciais realizados com esse método, o agente resultante conseguia manter o valor $B(s)$ dentro do intervalo de balanceamento. Entretanto, dois efeitos foram observados:

- Ao avaliar a recompensa a cada passo, o agente eventualmente atribui valores incorretos às ações. Um exemplo é quando o jogo alcança um estado balanceado, assim qualquer ação recebe um valor positivo, mesmo que essa não tenha impacto algum no balanceamento. Isso gerou diversos comportamentos indesejados. Ao avaliar o

método no jogo de luta, um comportamento fugitivo emergiu. Uma vez que o estado balanceado é alcançado, o agente evita novos confrontos com o jogador para se manter recebendo recompensas positivas.

- Pela definição da recompensa e do intervalo, a taxa de sucesso do jogador contra o agente, ao final da partida, é muito baixa ou quase nula. Dessa forma, embora o jogo pareça balanceado dentro do intervalo $I_{BC}$, na prática, ele impede que o usuário vença. No caso específico dos jogos de luta, a taxa de vitória do agente DDA contra os oponentes se distancia significativamente do ideal de 50%, frequentemente se aproximando de 100%.

Com isso, considerando esses efeitos, foram propostas duas abordagens para gerar um agente baseado em aprendizado por reforço no contexto de DDA. A primeira são modificações na função de recompensa apresentada, para evitar comportamentos indesejados. A segunda é tratar o problema de ajuste de dificuldade com dois objetivos distintos: Balanceamento e Competitividade. Balanceamento neste trabalho se refere à dificuldade percebida pelo jogador ao longo de uma sessão de jogo. Ao longo de toda a interação com o sistema, o jogador deve se sentir desafiado apropriadamente. Em contrapartida, a competitividade está relacionada ao resultado. Embora uma sessão de jogo balanceada seja desejável, o resultado da tarefa tem um importante impacto na percepção de entretenimento e realização do jogador. Um jogo que é desafiador, mas que o usuário consistentemente perde reduz a motivação do mesmo. Dessa forma, foi proposta uma função de recompensa modificada cujo objetivo é balancear o jogo ao longo da interação com o usuário e um mecanismo de penalidade para controlar o grau de competitividade. Assim, os dois conceitos serão tratados individualmente por duas funções de recompensa distintas.

### 5.1.1 Recompensa de balanceamento

O balanceamento refere-se à dificuldade apresentada ao usuário ao longo da interação com o sistema. Um jogo é considerado balanceado quando o nível de desafio é compatível com a proficiência do jogador. Para garantir esse equilíbrio, o objetivo do agente deve ser manter a métrica de dificuldade relativa $B(s) \approx 0$. Assim, a função de recompensa deve refletir esse princípio, atribuindo seu valor máximo quando a dificuldade está devidamente ajustada ao jogador. Além disso, desconsiderando o fator de competitividade, a recompensa deve ser simétrica, em que ambos os valores $B(s)$ e $-B(s)$ devem ser avaliados da mesma forma, visto que estão igualmente distantes do objetivo. Com apenas

o balanceamento em perspectiva, um novo intervalo é considerado $I_{BC} = [-BC, BC]$, em que $BC \in [0, 1]$ é a constante, definida pelo desenvolvedor, que limita o mínimo e máximo que $B(s)$ pode obter e a partida ainda é considerada balanceada. $I_{BC}$ é simétrico, centrado no zero e permite que o agente se apresente um pouco mais forte que o jogador (quando o valor for positivo) ou um pouco mais fraco (quando negativo), porém essa diferença é limitada pelo valor de $BC$. A função de recompensa proposta é definida na equação (5.1).

$$Recompensa(B(s), BC) = \begin{cases} \frac{BC+B(s)}{1-BC} & if BS(s) < -BC \\ \frac{BC^2-B(s)^2}{BC^2} & if -BC \leq BS(s) \leq BC \\ \frac{BC-B(s)}{1-BC} & if BS(s) > BC \end{cases} \tag{5.1}$$

A recompensa (5.1) é uma função por partes, correspondentes a cada um dos possíveis estados que o jogo pode se encontrar. Quando o valor $B(s)$ está no intervalo $I_{BC}$, o jogo é considerado balanceado e o agente recebe uma recompensa positiva. Neste estado, uma função quadrática simétrica com máximo no 0 é utilizada. Dessa forma, o agente é incentivado a manter a dificuldade medida no intervalo e próxima ao objetivo. Conforme o desafio atual se distancia do valor alvo em direção aos limites do intervalo, a função retorna valores decrescentes, mas ainda positivos. Quando a função quadrática é avaliada em estados desbalanceados, os valores explodem negativamente. Dessa forma, para garantir recompensas com magnitudes comparáveis, funções lineares são utilizadas quando o agente se apresenta muito forte ou muito fraco. Nesses casos, os valores retornados são sempre negativos, representando um desbalanceamento do jogo. Conforme o desafio se equipare à proficiência do usuário, a recompensa aumenta. A Figura 5.1 mostra de forma gráfica a função de recompensa proposta.

Um fator importante para definir uma função de recompensa é o período de avaliação. No trabalho de Noblega et. al. [70], o agente é recompensado a cada período de decisão conforme o estado atual do jogo. Dessa forma, a função incentiva comportamentos que mantenham o jogo no estado balanceado pelo maior tempo possível. Embora aparente ser uma consequência positiva, ações que não têm efeito ou até mesmo prejudiquem o balanceamento podem ser recompensadas positivamente. Dessa forma, existe erro na atribuição de valor às ações. Um resultado indesejado observado nos testes iniciais no simulador ocasionado por esse efeito é o comportamento de fuga. O agente aprende que, uma vez alcançado o estado balanceado, evitar novos confrontos com o adversário previne que a métrica de balanceamento mude. Assim, o mesmo recebe recompensas positivas, evitando a interação com o jogador. Para lidar com esse problema, faz-se necessário uma forma de calcular o efeito que a ação escolhida tem sobre o balanceamento atual. Dessa forma, o agente recebe a recompensa definida em (5.1) quando ocorre variação em alguma

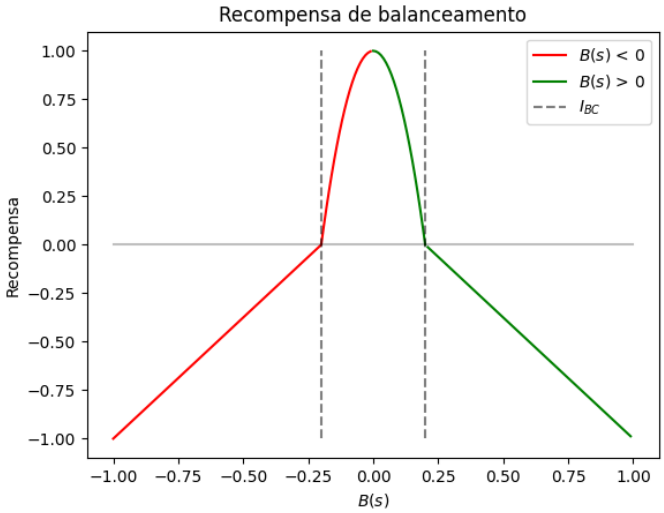



Figura 5.1: Função de recompensa proposta com intervalo $I_{BC} = [-0.2, 0.2]$.

métrica de dificuldade entre dois períodos de avaliação, e um pequeno valor negativo caso contrário. Essa pequena penalidade incentiva o sistema a adaptar rapidamente o jogo para o nível de balanceamento desejado. A condição de atribuição de recompensa pode ser verificada quando $Diff_{jogador}(s_t) \neq Diff_{jogador}(s_{t-1})$ ou $Diff_{alvo}(s_t) \neq Diff_{alvo}(s_{t-1})$ em que $s_t$ e $s_{t-1}$ são dois estados subsequentes distintos nos períodos $t$ e $t-1$ respectivamente. Com isso, apenas ações que têm impacto no balanceamento da partida serão recompensadas. No caso de jogos de luta, essa mudança incentiva o agente a manter a interação com o jogador, eliminando o comportamento fugitivo.

Assim como discutido previamente, calcular a dificuldade é uma tarefa complexa. Trabalhos na área de DDA aproximam essa métrica por meio de características do jogo. Heurísticas são usualmente definidas baseadas em condições de vitória. No caso de jogos competitivos, em que múltiplos jogadores disputam entre si, uma forma de verificar o desafio percebido por um jogador é avaliando sua habilidade relativa em comparação a outros. Existem diversas formas de avaliar habilidade relativa. Uma delas é o sistema de Elo [28], muito utilizado em xadrez. Outra forma é avaliar cada jogador com uma função de desempenho. Em diversos jogos, existe um sistema de pontuação que pode ser utilizado. Outros casos, a distância para a condição de vitória pode ser aplicada. Especificamente para jogos de luta, dois competidores estão equipados com diferentes habilidades e ataques para causar dano em seu oponente. No modo de jogo padrão, cada jogador possui um total limitado de vida e a partida termina quando o tempo total acaba ou quando a vida de algum jogador chega a zero. O ganhador é quem tem a maior vida ao final da partida. Assim como feito em outros trabalhos [22, 70], a diferença de vida entre os jogadores foi utilizada para aproximar a habilidade relativa e, consequentemente, a dificuldade. Como a condição de vitória nesse tipo de videogame é diretamente ligada aos pontos de vida, essa métrica captura eficientemente a atual diferença de habilidade

entre os jogadores. Dessa forma, para a aplicação realizada neste trabalho, essa métrica será utilizada para aproximar $B(s)$, normalizada pelo máximo de vida. A dificuldade alvo que o sistema DDA deve alcançar é $Diff_{alvo} = HP_{jogador}$, em que $HP_{jogador}$ é a vida do jogador no período avaliado. Similarmente, a dificuldade aproximada atual é dada por $Diff_{jogador} = HP_{DDA}$, em que $HP_{DDA}$ é a vida do personagem controlado pelo método apresentado.

### 5.1.2 Penalidade de competitividade

Obter uma partida desafiadora equiparável à proficiência do jogador é o objetivo desejado pelo sistema DDA. Entretanto, apenas o balanceamento não é suficiente para garantir um bom grau de entretenimento. Isso porque um fator de extrema importância é o resultado da tarefa. Malone [61] identificou em seu estudo que o sucesso em tarefas difíceis aprimora o engajamento, enquanto o fracasso tem o efeito oposto. Vang [100] desenvolveu um sistema DDA para jogos de luta com diferentes objetivos. Desses, existiam três versões com diferentes taxas de vitórias quando testadas contra jogadores reais, com 51%, 46% e 41% respectivamente. O agente com 46% foi apontado pelos usuários como mais realista e divertido de se jogar contra. O autor discute que os jogadores preferem adversários desafiadores, mas que sejam derrotados no final. Entretanto, ao ter uma grande taxa de sucesso, como o caso da versão com taxa de vitória mais baixa, o engajamento decai.

Com isso, esses resultados apontam que garantir um controle sobre o resultado da partida auxilia na percepção de dificuldade e entretenimento geral. Assim, define-se o conceito de Competitividade. Esse conceito está relacionado à probabilidade do jogador obter um resultado favorável no jogo. Em geral, essa chance deve ter um nível equilibrado, causando ao jogador a sensação de realização, mas que, ao mesmo tempo, não seja uma tarefa trivial. Assim, o sistema deve alterar fatores do jogo para garantir uma competitividade justa. Especificamente em jogos competitivos, como o gênero de luta, essa probabilidade está diretamente relacionada à habilidade relativa dos jogadores. Neste cenário, a competitividade se refere à probabilidade $P(j_1, j_2)$ de um jogador $j_1$ conseguir um resultado favorável contra outro jogador $j_2$ em uma partida. Um oponente com nenhuma chance de vencer é considerado não competitivo. Em contrapartida, um oponente com uma alta probabilidade de ganhar representa um desafio excessivo para o jogador. Em uma perspectiva de ajuste dinâmico de dificuldade, para uma partida com nível de dificuldade ideal, o jogador e seu oponente devem ter chances similares de ter o resultado positivo quando se enfrentam.

A fim de alcançar esse objetivo, uma penalidade foi proposta. Essa tem como

função controlar a probabilidade de sucesso do agente na partida quando enfrentando o jogador. Considerando que, dependendo do usuário ou estilo do jogo, pode-se almejar jogos mais difíceis ou fáceis de serem vencidos, a penalidade pode tomar diferentes alvos. Para tal, um novo intervalo é definido: $I_C = [c_i, c_j]$, em que $c_i, c_j \in [0, 1], c_i < c_j$ são constantes que delimitam o mínimo e máximo $P(j_1, j_2)$ que pode alcançar. A função é definida na equação (5.2).

$$Penalty(P(j_1, j_2), B(s)) = \begin{cases} -1 & if P(j_1, j_2) < c_i \ \ \& \ \ B(s) < 0 \\ -1 & if P(j_1, j_2) > c_j \ \ \& \ \ B(s) > 0 \\ 0 & otherwise \end{cases} \tag{5.2}$$

A penalidade descrita é determinada por duas variáveis: a probabilidade de vitória do agente na partida, $P(j_1, j_2)$, e a dificuldade relativa calculada, $B(s)$. Seu objetivo é ajustar dinamicamente a diferença de habilidade entre o agente e o jogador, garantindo uma competição equilibrada. De forma semelhante à função de recompensa de balanceamento, a penalidade segue um modelo por partes que classifica a partida em três estados possíveis. Quando $P(j_1, j_2) < c_i$, o agente não apresenta competitividade suficiente contra o jogador $j_2$, exigindo um aumento em sua habilidade relativa. Nessa situação, uma penalidade negativa é aplicada quando $B(s) < 0$. Por outro lado, se $P(j_1, j_2) > c_j$, o agente $j_1$ se torna um desafio excessivo para $j_2$, resultando em uma penalidade para valores de $B(s) > 0$, incentivando a redução da dificuldade. Já no intervalo $c_i \leq P(j_1, j_2) \leq c_j$, a competitividade é considerada adequada, e nenhuma penalização é imposta. Ao ser combinada com a função de balanceamento, essa abordagem permite que o agente ajuste sua dificuldade dentro dos limites estabelecidos por $I_{BC}$, até que a probabilidade de vitória esteja equilibrada. A Figura 5.2 apresenta uma representação gráfica da função proposta.

No contexto de jogos de luta, uma forma direta de aproximar $P(j_1, j_2)$ é utilizar a taxa de vitória do jogador $j_1$ sobre $j_2$. Entretanto, utilizar essa métrica apresenta alguns desafios. Durante o treinamento, o impacto de uma única partida é diluído no cálculo da probabilidade total, visto que um maior número de partidas é considerado para determinar a taxa de vitória. Isso causa dois problemas: variações lentas e valores limitados observados pelo agente. Com variação cada vez menor, principalmente no final do treino, o agente tem dificuldade de aprender a tarefa de manter a competitividade. Isso porque, independente do resultado de uma partida, o valor da taxa de vitória altera muito pouco. Assim, apenas um intervalo muito pequeno de valores será observado, resultando em baixa generalização para todos os possíveis valores de taxa de vitória. Além disso, outro problema é que a redução do impacto de uma partida não captura mudanças da política durante o treino. Conforme o agente aprende com o ambiente, sua competitividade perante o oponente muda e a taxa de vitória, por utilizar todas as partidas, não captura tal mudança. Para suprir essa limitação, apenas resultados das últimas 20 partidas mais recentes são utilizados para o cálculo da taxa de vitória. Com

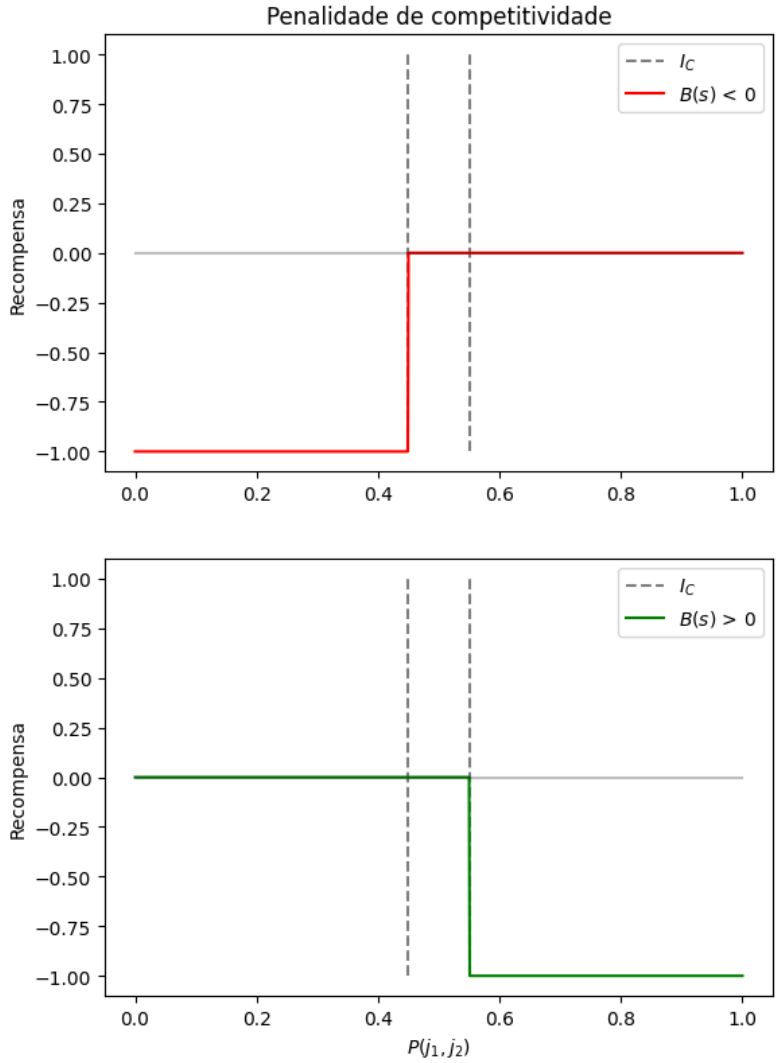



Figura 5.2: Penalidade de competitividade com $I_C = [0.45, 0.55]$.

isso, cada partida possui um efeito constante na probabilidade total. Dessa forma, uma maior variabilidade de valores de taxa de vitória serão apresentados ao método. Além disso, considerar apenas os resultados mais recentes captura de forma mais precisa a atual competitividade do agente contra o jogador.

## 5.2 Generalização em Aprendizado por Reforço

Um desafio central de aprendizado de máquina é gerar um preditor que tenha um desempenho bom para dados que não foram vistos durante o treinamento [36]. Essa capacidade é chamada de generalização. Em aprendizado supervisionado, generalização é um tópico amplamente estudado. Usualmente, um conjunto de dados $D_{train}$ é utilizado para treinar o modelo definido pelos parâmetros $\theta$ para minimizar o erro de treino $E_{train}$. Juntamente com esse objetivo, almeja-se minimizar o erro de generalização $E_{test}$, definido como o valor esperado do erro de novas entradas. Assume-se que as amostragens utilizadas durante o processo de treinamento e as que futuramente serão apresentadas ao modelo são independentes e ambas são igualmente distribuídas, amostradas de uma mesma distribuição. Essa propriedade permite que o modelo possa ter previsões sobre novas entradas. Além disso, torna-se viável comparar as relações entre erro de treino e teste. Para tal, tipicamente o conjunto total de dados é dividido em $D_{train}$, em que o modelo é otimizado, e $D_{test}$ no qual o modelo é avaliado. Ao calcular o desempenho do modelo em $D_{test}$, podemos ter uma métrica aproximada do erro de generalização.

Durante o treino, amostramos dados do conjunto $D_{train}$ para escolher parâmetros que reduzam o erro de treino. Depois, utilizamos o modelo para realizar previsões no conjunto de teste. Sobre esse processo, temos que o valor esperado do erro de treino é menor ou igual ao valor esperado do erro de teste. Com isso, três possíveis cenários existem. (i) O modelo apresenta ambos os valores esperados baixos, significando um bom desempenho. (ii) O valor esperado do treinamento é alto, indicando que o modelo é incapaz de aprender uma boa representação dos dados. Esse efeito é chamado de *underfitting*. (iii) O valor esperado do treinamento é baixo, porém o de teste é alto. Esse efeito, denominado *overfitting*, demostra que o modelo se especializou nos exemplos de treino, mas não apresenta bom desempenho em outro conjunto de dados.

Os efeitos descritos estão diretamente relacionados à complexidade do modelo, informalmente chamada de *capacidade*. Essa determina as possíveis relações que o modelo pode aprender. Modelos com baixa capacidade podem não conseguir aprender corretamente os padrões que um determinado tipo de dado possa ter. Similarmente, com alta capacidade, o modelo pode memorizar os padrões nos dados de treino que podem não ser representativos para outros elementos da mesma distribuição. Existem diversas formas de controlar a complexidade de um modelo. Em uma rede neural, por exemplo, pode-se definir uma arquitetura com mais neurônios, aumentando a capacidade e o risco de *overfitting* ou reduzindo o tamanho da rede, diminuindo sua capacidade e evitando *overfitting*, porém com risco de *underfitting*. Quando alteramos a complexidade do modelo, modifica-se o conjunto de possíveis funções que o algoritmo de aprendizado pode escolher como solução para o problema. Esse conjunto é chamado de espaço de hipótese. Outra forma de

controlar a capacidade do modelo que não seja alterando o espaço de hipótese é adicionar ao algoritmo preferência sobre funções. Isso é feito adicionando à função objetivo uma penalidade chamada de regularizador. O objetivo da regularização é reduzir o erro de generalização.

No contexto de aprendizado por reforço, generalização não possui uma definição tão direta quanto em aprendizado supervisionado. Usualmente em RL, o desempenho do agente é avaliado no mesmo ambiente do treino. Neste cenário, generalização pode ser avaliada quanto à robustez para estados que não foram vistos durante o treinamento. Entretanto, aplicações em cenários reais são dinâmicas, complexas e estão a todo momento mudando [48]. Dessa forma, os métodos devem ser capazes de lidar com diferentes ambientes e transferir conhecimento para cenários similares não vistos durante o treinamento. Neste caso, existem variações no próprio ambiente no qual o agente está inserido. Essas variações podem ser oriundas de uma mesma ou de diferentes distribuições. A próxima seção discute os arcabouços teóricos para abordar generalização em aprendizado por reforço e como desenvolver um treinamento cujo objetivo seja gerar um agente robusto.

### 5.2.1 Processo de Decisão de Markov Contextual

Originalmente, o conceito de Processo de Decisão de Markov Contextual (CMDP) foi proposta por Hallak et. al. [39] para modelar as variáveis latentes de usuários interagindo em uma página na web. Kirk et. al. [48] transportou esse conceito para o problema de generalização. Os autores tinham como objetivo criar mecanismos para racionalizar sobre coleções de tarefas. Para tal, define-se um CMDP $\mathcal{M}$ como um Processo de Decisão de Markov Parcialmente Observável (POMDP) cujos estados $s$ podem ser decompostos pela tupla $s = (c, s') \in S$ tal que $s' \in S'$ é o estado do ambiente, e $c \in C$ o contexto. Formalmente, $\mathcal{M}$ é uma tupla $(S, A, O, R, T, C, \phi : S' \times C \to O, p(s'|c), p(c))$. A diferença em relação a um POMDP tradicional é a composição dos elementos. Como $c$ faz parte do estado, observa-se que a função de recompensa $R : S' \times C \to \mathbf{R}$, função de transição $T((s, c), a)$ e outros componentes definidos a partir do estado também são dependentes do contexto. Além disso, a distribuição de probabilidade do estado inicial em um CMDP é definida como $p(s) = p((c, s')) = p(c) * p(s'|c)$, em que $p(c)$ é a distribuição do contexto. Essa determina os ambientes a serem utilizados. Com isso, a escolha de $c$ afeta todos os aspectos do ambiente gerado. A exceção é o espaço de ações, que se assume fixo entre tarefas. O contexto pode ser representado por um identificador, vetor de parâmetros ou uma semente aleatória. Um ponto importante é que o valor de $c$ permanece constante durante um episódio e difere apenas entre execuções. O CMDP não especifica se o con-

texto é observável pelo agente. Dessa forma, se o espaço de observação for $O = O' \times C$ e $\phi((s', c)) = (\phi'(s), c)$ é dito que o contexto é observável. Neste caso, o CMDP pode ser considerado um Processo de Decisão de Markov tradicional definido por um contexto.

O problema de generalização em aprendizado por reforço é um fator importante quando o modelo está sendo aplicado para uma coleção de diferentes tarefas [48]. Com o Processo de Decisão de Markov Contextual descrito, podemos formalmente definir o problema. Dado um CMDP $\mathcal{M}$ e $\mathcal{C}$ o conjunto de possíveis contextos, pode-se escolher um subconjunto $\mathcal{C}' \subseteq \mathcal{C}$ e criar outro CDMP $\mathcal{M}|_{\mathcal{C}'}$ desde que se realize os ajustes na distribuição de probabilidade de contextos. Essa definição permite a divisão da coleção de todos os MDPs contextuais em subconjuntos menores. Além disso, considerando uma política $\pi$, define-se o valor esperado do retorno como $\mathcal{R}(\pi, \mathcal{M}) := E_{c \sim p(\mathcal{C})}[\mathcal{R}(\pi, \mathcal{M}|_c)]$. Com isso, dado um CMDP $\mathcal{M}$ com conjunto de contextos $\mathcal{C}$ e uma escolha de dois subconjuntos $\mathcal{C}_{train}, \mathcal{C}_{test} \subseteq \mathcal{C}$, o objetivo é encontrar uma política $\pi$ que maximize o retorno esperado no conjunto de teste $\mathcal{M}|_{\mathcal{C}_{test}}$ definido por $J(\pi) := \mathcal{R}(\pi, \mathcal{M}|_{\mathcal{C}_{test}})$. Para tal, a política é treinada por um número arbitrário de passos utilizando no CDMP $\mathcal{M}|_{\mathcal{C}_{train}}$ e seu desempenho é avaliada no teste $\mathcal{M}|_{\mathcal{C}_{test}}$. Analogamente ao aprendizado supervisionado, a avaliação no conjunto de teste é uma estimativa do erro de generalização. Pode-se utilizar a diferença de performance do treino e teste como uma métrica para avaliar a generalização $GenGap(\pi) := \mathcal{R}(\pi, \mathcal{M}|_{C_{train}}) - \mathcal{R}(\pi, \mathcal{M}|_{C_{test}})$.

A formalização do Processo de Decisão de Markov Contextual captura classes de problemas de generalização. Dessa forma, nenhuma premissa sobre estruturas compartilhadas entre os CMDP de treino e teste são feitas. Em algumas tarefas em específico, faz-se necessário alguma premissa deste tipo para o aprendizado poder ocorrer. Uma propriedade comumente assumida é em relação aos conjuntos $\mathcal{M}|_{\mathcal{C}_{train}}$ e $\mathcal{M}|_{\mathcal{C}_{test}}$. Assume-se que, embora não idênticos, os elementos que formam ambos foram amostrados de uma mesma distribuição. Outro exemplo de premissa é a de Generalização de Domínio, que não assume a propriedade iid, mas que ambos possuem similaridades entre si. Um exemplo disso é a aplicação de robôs em cenários reais, no qual o mesmo foi treinado em simulações.

Em ambientes de múltiplos agentes (do inglês *Multi-Agent Reinforcement Learning* - MARL), especialmente em jogos de dois jogadores, a dinâmica do ambiente depende tanto das ações do agente quanto das decisões do oponente. Uma abordagem para lidar com essa complexidade é tratar cada agente de forma independente, considerando os demais jogadores como parte do ambiente. Essa estratégia reduz o custo de comunicação e melhora a escalabilidade do treinamento. No entanto, embora seja eficaz em ambientes parcialmente observáveis (POMDP), essa formulação pode tornar o problema não estacionário [75]. No caso dos jogos de luta, as regras do jogo permanecem constantes, independentemente dos jogadores envolvidos. Dessa forma, do ponto de vista do agente, o principal fator latente que diferencia uma tarefa da outra é o comportamento do oponente,

representado pela sequência de ações que ele emprega. Isso permite modelar o problema como um conjunto de tarefas dentro de um Contextual Markov Decision Process (CMDP), onde cada tarefa corresponde a um MDP definido pelo comportamento adversário. Assim, o contexto do problema pode ser identificado pela estratégia do oponente enfrentado pelo agente.

O principal objetivo deste trabalho é gerar um agente de ajuste dinâmico de dificuldade com a capacidade de generalização para diferentes tipos de jogadores. Um usuário pode se diferenciar de outro por uma variedade de fatores. Como discutido na seção 3.1, categorizar tipos de jogadores é uma tarefa complexa em que diversos fatores têm influência nessa definição. Neste trabalho, iremos focar em dois principais: habilidade relativa e estilo. Habilidade relativa significa a proficiência que um jogador tem sobre o jogo em relação a outro. Essa determina o seu entendimento e capacidade de executar a tarefa quando comparado ao seu oponente. Estilo está relacionado à estratégia específica que um jogador executa para alcançar o seu objetivo no jogo. Embora estratégia seja algo muito variado, usualmente é possível identificar um conjunto de estratégias que se encaixam em um determinado estilo. Assim, para gerar o agente desejado, faz-se necessário uma forma de torná-lo robusto a variações tanto em estilo de jogo quanto em habilidade relativa. Isso significa que ele deve aprender a balancear o jogo independente da proficiência e tipo de estratégia que o oponente irá realizar. Considerando esse problema como um CMDP, em que o contexto define o comportamento do oponente, podemos avaliar o desempenho esperado do agente em cenários não encontrados durante o treino. Para tal, inicialmente precisa-se definir dois conjuntos distintos: treino $\mathcal{M}|_{\mathcal{C}_{train}}$ e teste $\mathcal{M}|_{\mathcal{C}_{test}}$.

Ambos, $\mathcal{M}|_{\mathcal{C}_{train}}$ e $\mathcal{M}|_{\mathcal{C}_{test}}$, são subconjuntos de todos os possíveis comportamentos que o oponente pode utilizar. Na prática, é impossível que esses cubram todo o espaço de estratégias, mas é desejável que esses conjuntos sejam bem representativos. Com essa propriedade, durante o treinamento, o agente enfrenta uma variedade de estilos e habilidades relativas diferentes, tornando-o robusto a variações em $c$. Similarmente ao aprendizado supervisionado, espera-se que, durante o período de teste, $GenGap(\pi)$ seja baixo, indicando uma política robusta. O grande desafio é gerar os conjuntos de treino e teste com pluralidade estratégica e de habilidades relativas distintas.

O simulador utilizado promove competições anuais desde 2013. Assim, diversos agentes foram implementados para o jogo utilizando uma grande variedade de algoritmos ao longo dos anos. Além da variedade de soluções, as competições permitem a avaliação de habilidade relativa entre eles. Com isso, esse é um promissor conjunto de teste. Em relação ao treinamento, esse trabalho visa gerar $\mathcal{M}|_{\mathcal{C}_{train}}$ com as propriedades desejadas. Uma forma de resolver o problema para o caso específico de jogos de luta seria criar manualmente comportamentos pré-definidos. Embora seja uma solução que garante controle sobre as estratégias utilizadas no conjunto, requer o cuidadoso design de comportamentos e inúmeros testes, de modo a adequar as variações necessárias do conjunto. Com isso,

um método para gerar o conjunto de treino é proposto. A seção 5.3 a seguir discute essa solução.

## 5.3 Construção do conjunto de treinamento

A fim de gerar um agente que mantenha o balanceamento e competitividade contra diferentes oponentes, faz-se necessário torná-lo robusto às variações de contexto. Para tal, durante o treinamento, o agente deve observar ambientes com tais variações e, assim, possibilitar aprender uma política boa para qualquer contexto $c$. Como discutido, o contexto se refere a um oponente, mais especificamente sua habilidade relativa e estilo. Dessa forma, são necessários métodos para obter políticas com essas variações. Gerar de forma procedural oponentes variados que cumpram os requisitos é uma tarefa complexa. O grande desafio é torná-los o mais representativo possível. Dessa forma, cada oponente no conjunto deve representar alguma variação em habilidade relativa ou estilo. Além disso, deseja-se que o processo de criação dos elementos do conjunto não necessite de intervenção humana, assim evitando a necessidade de oponentes pré-programados, reduzindo o custo de desenvolvimento do processo. Para tal, um método baseado em aprendizado por reforço será utilizado de modo a gerar o conjunto de políticas diversas. RL tem capacidade de gerar agentes com alto desempenho mesmo em ambientes com alta complexidade. A próxima seção descreve o treinamento de vários agentes e como gerar diferentes estratégias para ganhar o jogo utilizando a função de recompensa padrão descrita em **??**.

### 5.3.1 Treinamento Baseado em População

Considerando a capacidade que métodos de treinamento em *self-play* demonstraram em trabalhos prévios discutidos na seção 2.2 e, dado que ao longo do processo as versões históricas possuem diversas habilidades relativas, este se torna uma boa solução para gerar o conjunto de treinamento descrito na seção 5.2.1. Para tal, define-se o esquema de treinamento seguindo o framework descrito em [41]. Um total de $n$ agentes diferentes serão otimizados simultaneamente, todos utilizando uma mesma função de recompensa dada na equação (4.1). Com isso, diferentes soluções serão exploradas, cobrindo uma maior diversidade no espaço de estratégias. O zoológico $\pi^o$ será compartilhado entre os processos. A função $G$ armazena em $\pi^o$ os parâmetros de cada agente a cada $h$ passos. No

início do episódio, uma política $\pi_i$ é amostrada do zoológico seguindo uma distribuição $\Omega$ e será utilizada pelo oponente. Similar ao trabalho feito em [71], divide-se o conjunto em dois grupos, as $k$ políticas mais recentes de todos os agentes e as versões mais antigas de cada um. Assim, amostram-se uniformemente as versões mais recentes com probabilidade $p$ e com $1-p$ as versões históricas com chances uniformes. Maiores valores de $p$ resultam em mais rápida adaptação contra os atuais agentes, enquanto valores menores reduzem o efeito de *catastrophic forgetting*, estabilizando o treinamento. Dessa forma, ao longo do processo, o valor $p$ é reduzido gradativamente. Assim, inicialmente, o agente enfrenta as políticas mais recentes com o foco de aumentar a proficiência na tarefa. Conforme o progresso do treinamento, $p$ reduz de magnitude e a probabilidade de amostragem de políticas históricas aumenta, assim priorizando a robustez do modelo. A Figura 5.3 provê um diagrama e uma visão de alto nível do treinamento proposto.

Ao longo do processo de treinamento, os agentes aprendem a dinâmica do ambiente. Conforme a proficiência aprimora, outros na população são incentivados a desenvolver novas habilidades para vencê-lo. Com a amostragem de versões históricas, as políticas geradas devem ser robustas para enfrentar diversos comportamentos. Ao final do processo, o zoológico contém naturalmente políticas com diferentes habilidades relativas, visto que possuem distintos tempos de treinamento.

Embora apresentem resultados promissores, métodos atuais baseados em *self-play* usualmente visam maximizar a taxa de vitória contra suas versões históricas, resultando em estilos e estratégias limitadas, além da tendência a encontrar ótimos locais [47]. Mesmo com o caso de múltiplos agentes sendo otimizados, a diversidade estratégica ocorre por inicialização aleatória, exploração ou melhor resposta contra os oponentes e, muitas vezes, resulta em soluções parecidas e pouco variadas.

Uma solução para esse problema é incentivar a diversidade de estratégias na população de forma explícita. Oh et. al. [71] treinou simultaneamente três agentes baseados em aprendizado por reforço em um jogo de luta. Usualmente, jogos deste gênero possuem estilos muito determinados e com funcionamento de pedra-papel-tesoura, em que um tipo de estilo ganha de outro. Para gerar oponentes robustos, cada um dos agentes no treinamento otimiza uma função de recompensa distinta, definida por um conjunto de parâmetros. Essa técnica é chamada de modelagem de recompensa (do inglês - *Reward Shapping*) que modifica ou adiciona outros valores à recompensa para guiar o agente à solução mais rapidamente. Os autores utilizaram essa técnica para gerar diferentes estilos no jogo (agressivo, defensivo e balanceado). O resultado é uma política que consegue ganhar contra distintos estilos. Dessa forma, ao atribuir explicitamente mecanismos para aprimorar a diversidade estratégica, aprimora-se a generalização do agente resultante e, consequentemente, o desempenho geral contra um novo oponente aleatório.

Embora tenha apresentado bons resultados, a técnica aplicada para incentivar a diversidade é dependente da tarefa e requer o design de novas funções e testes para garantir

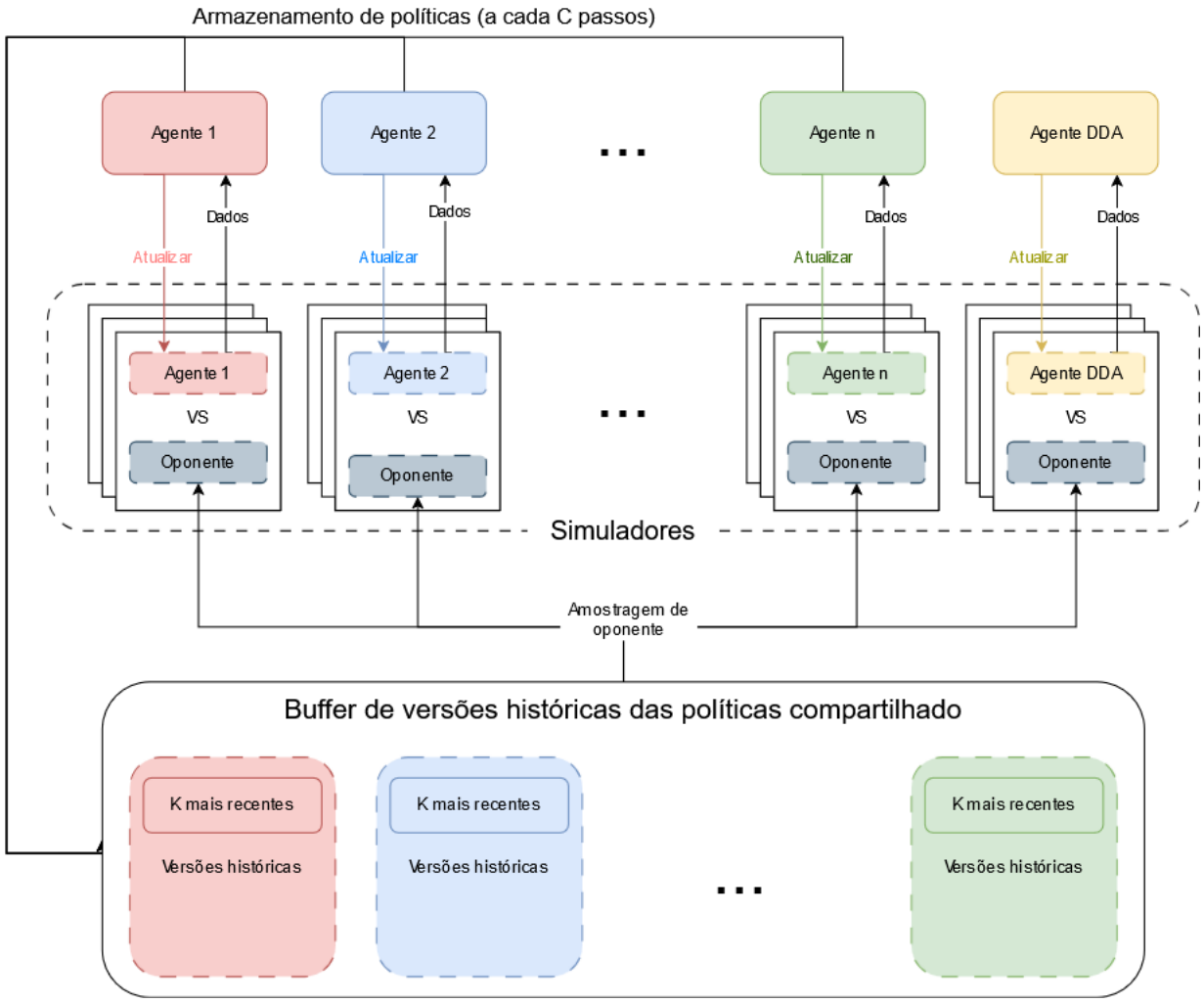



Figura 5.3: Esquema do treinamento proposto. Os agentes enumerados são os com objetivo de vencer o jogos. O agente DDA é treinado separadamente do processo e apenas recupera políticas do zoológico. Cada processo tem múltiplos ambientes em paralelo. Ao início de uma nova partida, cada ambiente amostra do zoológico compartilhado uma política para servir de oponente. Cada agente otimiza a recompensa obtida e a métrica de diversidade. A cada $C$ passos, os agentes enumerados armazenam no zoológico uma cópia da sua política congelada.

que o comportamento seja o desejado. Outra forma de alcançar essa propriedade é utilizar uma função de similaridade. Essa é utilizada para calcular a diversidade estratégica dos agentes. Ao atribuir essa métrica ao processo de otimização, a população encontrará de forma automática diferentes soluções. A próxima seção a seguir descreve o método utilizado.

#### 5.3.1.1 Função de diversidade

Como discutido na seção 2.2.1, para obter um agente DDA robusto, o conjunto de treinamento deve ser variado e com uma gama de distintas estratégias, assim como diferentes níveis de habilidade relativa. Para conseguir essas propriedades, deve-se gerar oponentes que cubram diferentes locais no espaço de estratégias. O treinamento baseado em população gradualmente gera políticas mais fortes ao longo do processo, entretanto não garante diversidade de comportamentos. A atribuição de uma função de similaridade ao processo de otimização auxilia nessa tarefa. Zhao et. al. [107] incorporou essa métrica adicionando um novo objetivo ao MDP. Para tal, os autores adicionaram uma nova função de recompensa na qual os agentes são recompensados quanto maior for a pluralidade estratégica na população. Essa solução incentiva a geração de políticas diferentes, entretanto altera o ambiente sendo otimizado. Outra solução seria atribuir a função de diversidade ao processo de otimização dos parâmetros $\theta$. Como discutido na seção 2.1, uma política $\pi_\theta$ visa encontrar um conjunto de parâmetros $\theta$ para maximizar o valor esperado do somatório das recompensas $J(\pi_\theta) = \mathbf{E}_{\tau\sim\pi_\theta}[R(\tau)]$. Considerando um conjunto de $n$ políticas $(\pi_{\theta_1}, ..., \pi_{\theta_n})$ e uma métrica qualquer de diversidade $D_\tau(\pi_1, ..., \pi_n)$ definida na trajetória $\tau$, pode-se aprimorar esse objetivo adicionando um novo termo apresentado na equação (5.3).

$$J(\pi_\theta) = \mathbf{E}_{\tau\sim\pi_\theta}[R(\tau)] + \mu\mathbf{E}_{\tau\sim\pi_\theta}[D_\tau(\pi_{\theta_1}, ..., \pi_{\theta_n})] \tag{5.3}$$

O termo de similaridade incentiva o processo de otimização a encontrar um conjunto de parâmetros que maximize tanto a recompensa quanto a diversidade. Neste cenário, a tarefa que se deseja otimizar se mantém a mesma. De modo a obter o conjunto de treinamento com maior pluralidade estratégica, a diversidade baseada no par estado-ação se encaixa melhor. Neste trabalho, existem $n$ distintos agentes otimizando a própria função de recompensa simultaneamente, onde cada um contem $k$ versões históricas. Deseja-se que esses $n$ agentes apresentem distintas formas de solucionar o problema. Com isso, a diversidade será avaliada em relação ao estilo de um agente comparado ao restante. Considerando apenas as versões mais recentes de cada estilo $(\pi_{\theta_1}, ..., \pi_{\theta_n})$,

define-se na equação (5.4) a média da população $\pi'_i(\tau)$ comparando o estilo $i$ com o restante.

$$\pi'_i(\tau) = \frac{1}{n-1} \sum_{j=1|j\neq i}^{n} \pi_{\theta_j}(\tau) \tag{5.4}$$

Ao longo do processo de treinamento, novas políticas serão adicionadas ao zoológico. O cálculo apresentado considera apenas a versão mais recente de cada estilo. Assim, incentivando que cada agente explore diferentes direções no espaço de estratégias. Com essa formulação, a diversidade compara duas distintas políticas, o agente $\pi_i$ e a média $\pi'_i$ do zoológico. Essas são consideradas divergentes quando mapeiam diferentes distribuições em uma mesma trajetória $\tau$. Existem diversas métricas estatísticas que capturam a distância entre duas distribuições de probabilidade. Similar ao que foi feito em [60], a métrica aplicada neste trabalho é a Divergência de Jensen-Shannon (do inglês *Jensen-Shannon Divergence* - JSD). Essa métrica apresenta propriedades interessantes para esse cenário. JSD é definida em todos os pontos e é suave. Além disso, a função é simétrica, facilitando a interpretabilidade dos valores. Somado a isso, a métrica é limitada, permitindo a definição de maximização. A equação (5.5) define a divergência de Jensen-Shannon no caso de duas políticas:

$$\begin{aligned} JSD(\pi_i||\pi'_i) &= \frac{1}{2}KL(\pi_i||M) + \frac{1}{2}KL(\pi'_i||M) \\ M &= \frac{1}{2}(\pi_i + \pi'_i) \\ KL(\pi_i||M) &= \sum_{x\in\pi_i} \pi_i(x) * log(\frac{\pi_i(x)}{M(x)}) \end{aligned} \tag{5.5}$$

Em que $M$ é chamada de distribuição de mistura e $KL$ é a divergência de *Kullback-Leibler*. A trajetória foi omitida por motivos de simplicidade. Um fator importante a ser considerado na otimização de diversidade é o *trade-off* com a performance. A equação (5.3) apresenta o fator de pluralidade juntamente com um escalar $\mu$. Esse definido no intervalo $[0, 1]$ determina a magnitude que a métrica $D_\tau$ tem sobre o objetivo. Intuitivamente, no limite, a população cobre todas as estratégias possíveis, maximizando a diversidade. Dentre esses comportamentos, existem aqueles que não são ótimos em relação à função de recompensa. Com isso, alguns trabalhos utilizam o fator $\mu$ que controla a importância entre maximizar a recompensa ou a diversidade.

## 5.4 Treinamento do Agente DDA

O treinamento do agente DDA possui duas etapas. A primeira é similar ao processo apresentado em 5.3.1 e pode ser realizado simultaneamente. Os oponentes que serão amostrados são um subconjunto do zoológico. Esse subconjunto cresce conforme o progresso do treinamento, apresentando gradualmente desafios maiores ao oponente. Utilizando o fato de que políticas de um mesmo estilo se diferenciam por tempo de treino e, consequentemente, habilidade relativa, o conjunto de treinamento do agente DDA é composto por todas as políticas históricas com o mesmo número de passos de treinamento ou inferior ao atual. A probabilidade de seleção desse conjunto é igual à discutida anteriormente, com probabilidade $p$ de selecionar os $k$ mais recentes e $1 - p$ de selecionar o restante. Essa probabilidade decresce com o decorrer do processo. Dessa forma, o intuito dessa primeira etapa é gerar um agente capaz de realizar a tarefa de balanceamento. Entretanto, como discutido anteriormente, treinamento baseado em população sofre do efeito de *catastrophic forgetting*.

Ao decorrer do treinamento, a probabilidade de selecionar uma política antiga reduz com a entrada de novas ao zoológico e priorização das mais recentes. Com isso, a distribuição dos dados muda ao longo do processo, possibilitando que a política esqueça comportamentos previamente aprendidos. No caso da geração do conjunto $\mathcal{M}|_{\mathcal{C}_{train}}$ esse efeito não impossibilita a realização da tarefa, visto que o objetivo do agente é ganhar uma partida, independente de como. Assim, esquecer uma estratégia que previamente ganhava de um oponente qualquer por outra que também ganhe não faz diferença no objetivo final. Para o ajuste de dificuldade dinâmica, o objetivo é muito mais restrito, dessa forma, desaprender a balancear o jogo contra um oponente pode resultar no desvio da tarefa. Com isso em vista, a segunda parte do treino é realizada após a primeira ser concluída. Essa segunda visa apresentar oponentes de diferentes estilos e habilidades relativas com a mesma probabilidade. Assim, espera-se que o comportamento final balanceie o jogo para todos os espectros apresentados.

Além de diferenças no treinamento, o espaço de estados foi aumentado com algumas informações adicionais para permitir ao modelo balancear a partida. Esses dados estão descritos na tabela 5.1. Em relação ao balanceamento, são adicionados três valores que indicam qual estado a partida se encontra para ambas competitividade e balanceamento baseado nos intervalos $I_{BC}$ e $I_c$. Além disso, adiciona-se a distribuição das ações do oponente a fim de aprimorar a adaptação do modelo contra diferentes estratégias.

| Informação do Oponente | | |
|---|---|---|
| Nome | Tipo | Tamanho |
| Distribuição das Ações | float | 42 |
| Informação de Competitividade | | |
| Nome | Tipo | Tamanho |
| Estado de Competitividade | One-Hot | 3 |
| Penalidade aplicada | Bool | 1 |
| Informação de Balanceamento | | |
| Sinal da Diferença | float | 1 |
| Magnitude da Diferença de Vida | float | 1 |
| Estado de Balanceamento | One-Hot | 3 |

Tabela 5.1: Valores adicionais ao estado no treinamento do agente DDA.

# Capítulo 6

# Experimentos

A fim de avaliar o método proposto, dois conjuntos de experimentos foram realizados. O primeiro tem como foco avaliar a função de recompensa proposta e como a variação nos intervalos afeta o comportamento e as métricas de balanceamento. O segundo conjunto tem como foco gerar um agente capaz de balancear o jogo para jogadores com diferentes habilidades relativas e estilos. Para tal, a generalização do modelo para oponentes não vistos durante o treino foi investigada. Inicialmente, métodos de regularização foram testados, para verificar o efeito que o *overfitting* tem sobre o comportamento final. Resultados apontaram que apenas regularização não é suficiente para resolver o problema de generalização. Assim, um modelo de treinamento baseado em *self-play* é apresentado. O objetivo é gerar diferentes comportamentos com habilidades relativas e estilos distintos. Assim, o agente DDA será treinado com uma ampla variedade de estratégias se tornando mais robusto contra diferentes oponentes não vistos durante o treino.

O método utilizado neste trabalho foi o PPO [85]. A implementação base foi providenciada pela biblioteca Stable Baselines 3 [80]. Algumas modificações foram realizadas para acrescentar a função de similaridade. O código com as mudanças e todos os experimentos realizados estão disponibilizados no repositório público [1]. O método permite o uso de ambientes paralelos para acelerar o treinamento. Todos os experimentos utilizaram redes neurais completamente conectadas. Os hiperparâmetros da rede utilizada estão listados na tabela 6.1. Os específicos do método PPO estão listados em cada conjunto de experimentos.

| Hiperparâmetro | Valor |
|---|---|
| Neurons per layer | 256 |
| Number of layers | 3 |
| Type of layer | Fully connected |

Tabela 6.1: Hiperparâmetros da rede neural utilizada nos experimentos.

[1] O repositório com os ambientes Gym, PPO modificado e experimentos https://github.com/TiagoNO/ThanosBot

## 6.1 Estudo da Função de recompensa

Para conduzir os experimentos em relação à função de recompensa, os oponentes foram selecionados baseado em suas performances no torneio realizado, apresentados na Figura 4.2. Três foram selecionados para representar diferentes níveis de habilidade. Com isso, foi possível verificar o efeito de cada estudo perante níveis de proficiência distintos. Os escolhidos foram TeraThunder (Forte - 1 lugar), UtalFighter (Médio - 12 lugar) e BCP (Fraco - 24 lugar). Neste conjunto de experimentos, os treinamentos foram realizados contra um oponente único e replicados para cada nível de dificuldade.

Inicialmente, uma comparação do método apresentado em [70] (BC) e o proposto por esse trabalho (BC quadrático) foi realizada. Os resultados obtidos estão representados na Figura 6.1. A melhor parametrização para o intervalo de balanceamento citada no artigo original é com $BC = 0.3$ do valor máximo de vida. Assim, para comparação, os intervalos $I_{BC} = [0, 0.3]$ e $I_{BC} = [-0.15, 0.15]$ foram utilizados respectivamente para os métodos comparados. Especificamente para o BC quadrático, o intervalo de competitividade foi definido como $I_C = [0.45, 0.55]$, para gerar um agente igualmente competitivo comparado ao oponente. Observando os resultados, ao final do treinamento, o método BC original consistentemente alcançava uma taxa de vitória de 75% considerando apenas as últimas 20 partidas quando enfrentando o oponente médio e difícil. Contra o considerado fraco, os pesos aleatórios no início do treinamento são suficientes para apresentar uma alta taxa de vitória e o comportamento resultante aprende a ganhar todas as partidas. Em contrapartida, o BC quadrático proposto neste trabalho, ao final do treinamento, convergiu para o intervalo de competitividade $I_C$ (entre 45% e 55%) considerando a taxa de vitória nas últimas 20 partidas mais recentes. Em relação à interação com o oponente, outra métrica observada foi o número médio de combates por partida. Esse pode ser avaliado toda vez que a vida de algum jogador é reduzida. Em média, o modelo proposto resultou em 86% da partida representando combates em comparação a 18% para o BC. Isso significa que o BC quadrático interage mais com o jogador ao longo da partida sem comprometer o balanceamento e competitividade, levando a experiências mais dinâmicas e engajantes.

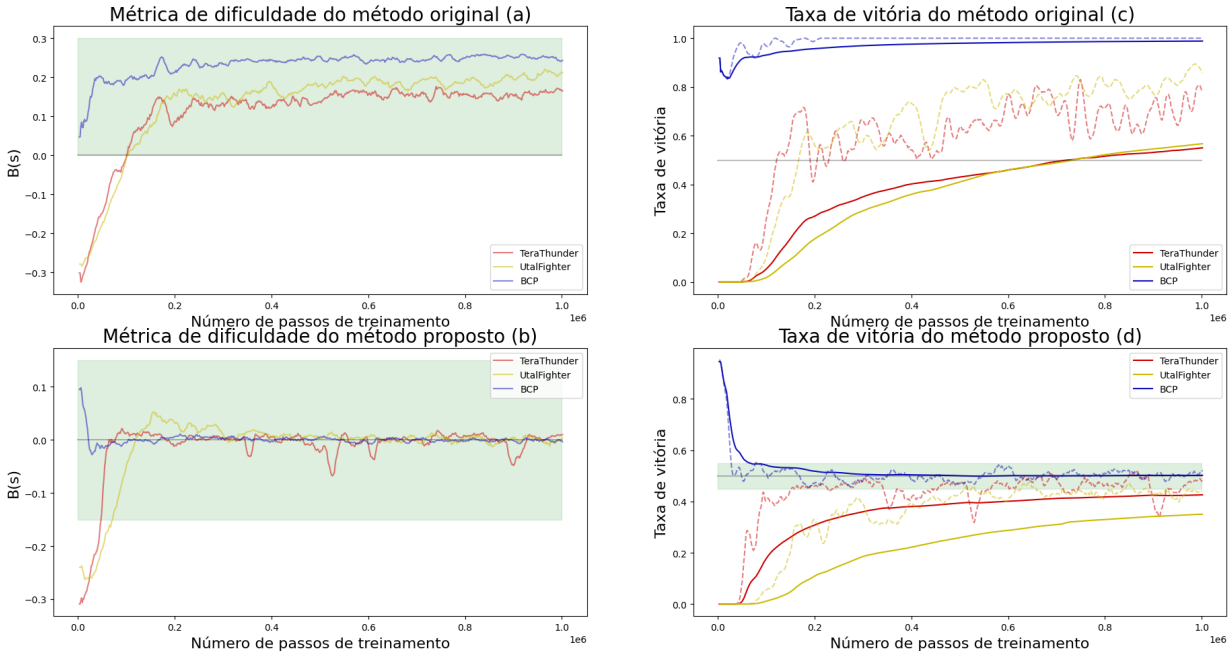



Figura 6.1: Métrica de dificuldade e taxa de vitória do método original (a) e (c) e o proposto (b) e (d) respectivamente. As Linhas pontilhadas representam a taxa de vitória das últimas 20 partidas mais recentes. A linha contínua é a taxa de vitória ao longo de todo o treinamento. Faixas verdes indicam o intervalo considerado balanceado em cada métrica.

### 6.1.1 Estudo do Intervalo de Balanceamento

O objetivo do agente é apresentar uma habilidade relativa próxima ao jogador durante a partida. O modelo proposto alcança isso definindo o intervalo de balanceamento simétrico e centrado em 0. A fim de compreender mais profundamente a função de recompensa proposta, foram realizados experimentos variando o tamanho de $I_{BC}$. Os tamanhos escolhidos foram similares às variações do parâmetro BC testados no artigo do método BC. Assim, os valores testados são $I_{BC} = [-0.1, 0.1]$ e $I_{BC} = [-0.2, 0.2]$.

A Figura 6.2 apresenta a média de vida e taxa de vitória ao longo do treinamento para os diferentes intervalos e oponentes. Nota-se que, independente de ambos os níveis de proficiência e valor de $BC$, o agente converge para diferença de vida próximas a zero. Assim, o agente consegue manter o balanceamento em todos os cenários. Com isso, observa-se que o tamanho do intervalo tem impacto mínimo na performance em geral durante a partida. Isso decorre do fato de que alterar o valor de $BC$ não altera o valor máximo da função de recompensa. Entretanto, essa variação afeta a competitividade. Intervalos com tamanhos menores, em todos os casos avaliados, apresentam uma convergência mais lenta para a competitividade desejada. Esse efeito pode ocorrer porque, com $I_{BC}$ menores, o objetivo se torna mais restrito, apresentando um desafio maior ao

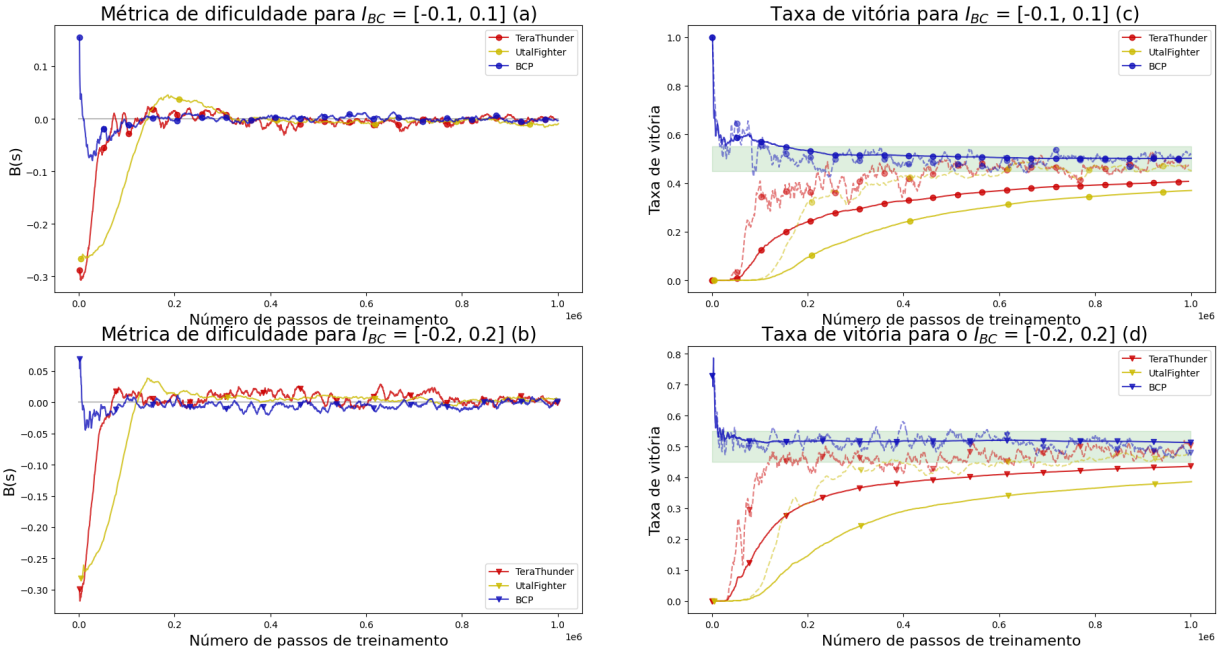



Figura 6.2: Diferença de vida (a) e (b) e taxa de vitória (c) e (d) para os intervalos de balanceamento $I_{BC} = [-0.1, 0.1]$ e $I_{BC} = [-0.2, 0.2]$ respectivamente. Linha pontilhada representa a taxa de vitória das últimas 20 partidas. Linha contínua representa a taxa de vitória do treinamento inteiro. Faixas verdes indicam o intervalo considerado balanceado em cada métrica.

agente. Assim, o tempo necessário para alcançar o nível desejado de competitividade aumenta.

### 6.1.2 Estudo do Intervalo de Competitividade

Trabalhos prévios em DDA tinham como objetivo criar um agente com uma taxa de vitória próxima a 50% [22]. Em um jogo verdadeiramente balanceado, ambos os jogadores têm uma igual probabilidade de vencer uma partida. Usualmente, um sistema DDA lida com diferentes tipos de jogadores, cada um com suas preferências e expectativas perante o jogo. Assim, alguns podem preferir oponentes mais competitivos, enquanto outros podem preferir enfrentar inimigos mais fracos. O método proposto neste trabalho permite o controle da competitividade para melhor se ajustar ao desejado. Dessa forma, experimentos foram conduzidos de modo a testar como a variação do intervalo $I_C$ afeta o agente resultante. Neste experimento, os valores $I_C = [0.2, 0.3]$ e $I_C = [0.7, 0.8]$ foram escolhidos. Dessa forma, ambos são disjuntos do intervalo padrão previamente testado e representam, respectivamente, um oponente com pouca e alta competitividade. Além disso, para o balanceamento foi utilizado o intervalo $I_{BC} = [-0.15, 0.15]$.

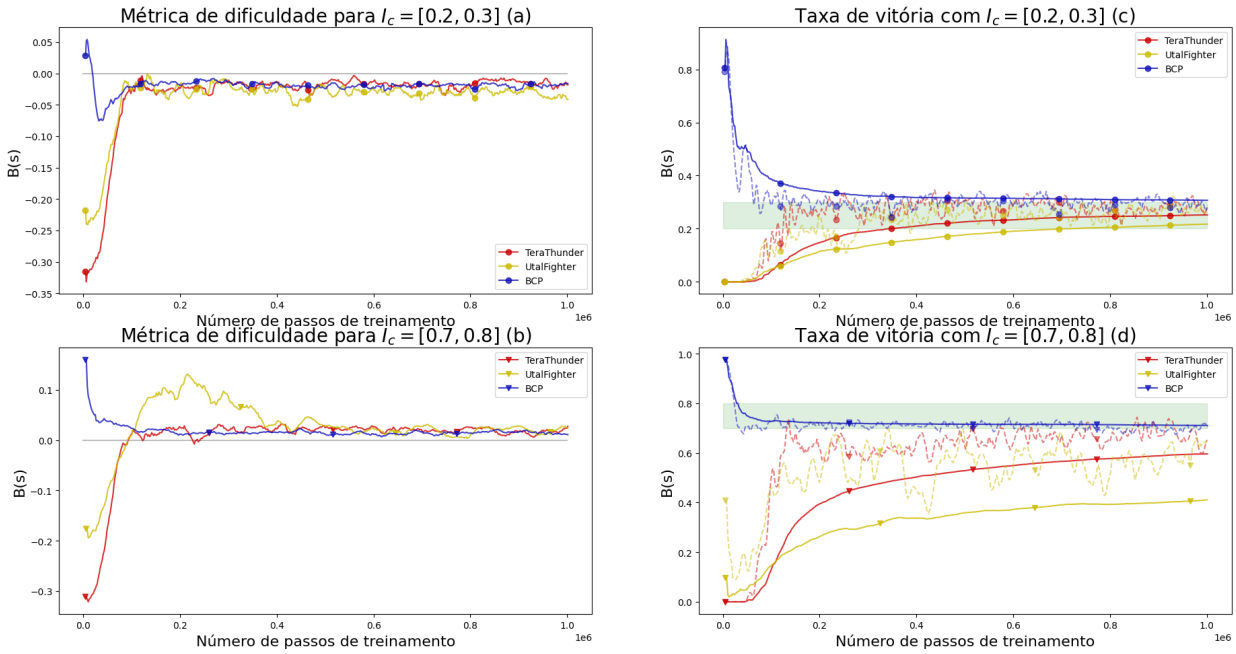



Figura 6.3: Diferença de vida (a) e taxa de vitória (b) para os distintos intervalos de competitividade. (b) As Linhas pontilhadas representam a taxa de vitória das últimas 20 partidas mais recentes. A linha contínua é a taxa de vitória ao longo de todo o treinamento. Faixas verdes indicam o intervalo considerado balanceado em cada métrica.

Os resultados estão presentes na Figura 6.3. Quando a competitividade é definida para perder mais regularmente (com $I_c = [0.2, 0.3]$), a taxa de vitória das últimas 20 partidas mais recentes converge para o especificado. No caso, quando a competitividade do agente é escolhida para ganhar mais ($I_c = [0.7, 0.8]$), contra os oponentes fácil e difícil os valores convergiram para o ideal. Em contrapartida, na dificuldade média (UtalFighter) os resultados exibiram uma convergência mais lenta em todos os experimentos para competitividade. Isso pode ser devido ao estilo de luta mais defensivo do UtalFighter. Esse tende a usar ações de longo alcance e se manter a uma certa distância mais frequentemente que os outros.

## 6.2 Estudo sobre Generalização

O primeiro conjunto de experimentos teve como objetivo entender como as diferentes partes da função de recompensa afetam o comportamento gerado. Entretanto, o objetivo final do trabalho é conseguir gerar um agente DDA capaz de balancear uma partida para um conjunto diverso de jogadores. Inicialmente, foi investigado se apenas variações na habilidade relativa e utilização de regularizadores poderiam obter o agente

desejado. Em seguida, testes com treinamento descrito na seção 5.4 foram feitos de modo a aprimorar a generalização e robustez do método final.

### 6.2.1 Métodos de Regularização

Similarmente ao feito no trabalho [55], diferentes regularizadores foram avaliados a fim de compreender o efeito sobre a capacidade do agente resultante em generalizar para diferentes variações de contexto. Novamente, no caso desse trabalho, o objetivo é aprimorar a capacidade do agente em balancear o jogo para jogadores com distintas habilidades relativas para manter um nível de competitividade desejado. Para este experimento, o treinamento será feito contra os três oponentes (TeraThunder, UtalFighter e BCP) simultaneamente com ambientes paralelos, assim apresentando ao agente variações de contexto com diferentes habilidades relativas. A política resultante foi testada contra todos os competidores do torneio realizado (ver Figura 4.2). Várias técnicas de regularização foram comparadas com o *baseline*, incluindo penalidade de entropia, *L2* e *dropout*.

Os resultados resumidos são disponibilizados na Tabela 6.2. Estes apresentam tanto a taxa de vitória nos 100 jogos realizados, assim como a média de vida ao longo das partidas. O *baseline*, sem nenhum método de regularização, apresentou resultados bons para oponentes com habilidade relativa baixa se observado o balanceamento. Entretanto, a competitividade é maior do que a especificada para alguns desses oponentes fáceis. Para oponentes mais fortes no ranqueamento realizado, ambas as métricas apresentam resultados distantes do desejado. As técnicas de entropia e dropout apresentam resultados similares e ambas aprimoram sobre a performance do *baseline*. Nesses, a diferença de vida em quase todos os oponentes se aproximou do objetivo e, na maioria, são consideradas balanceadas. Entretanto, deve-se notar que a taxa de vitória não se aproximou do especificado para parte dos competidores. Por fim, a técnica L2 apresentou resultados diferentes. Em casos específicos, ocorreu uma melhora se comparada ao método sem regularização, porém, contra a maioria dos competidores, o método prejudicou a performance. Além disso, durante o treinamento, o agente não convergiu para os oponentes médio e difícil.

Os resultados prévios mostram que o agente falha em alcançar o balanceamento e competitividade contra alguns oponentes específicos do torneio. Embora esse experimento tenha como principal foco a diferença de habilidade relativa entre oponentes, um fator importante que influencia o tipo de jogador é o estilo. Ele se refere à estratégia específica que um jogador utiliza para alcançar seus objetivos no jogo. Para compreender melhor os resultados e a incapacidade do agente de convergir para certos casos de teste, foi inves-

Tabela 6.2: Comparação entre diferentes tipos de regularização avaliando taxa de vitória (Wr) e diferença média de vida (ΔHP) ao longo de 100 partidas por oponentes. Valores em negrito são considerados balanceados para $I_{BC} = [-0.15, 0.15]$ e $I_C = [0.45, 0.55]$.

| | Baseline | | Entropy | | Dropout | | L2 | |
|---|---|---|---|---|---|---|---|---|
| Oponente | Wr (%) | ΔHP | Wr (%) | ΔHP | Wr (%) | ΔHP | Wr (%) | ΔHP |
| TeraThunder | 43.50 | **8.51** | **45.00** | **6.31** | **54.00** | **-8.81** | 0.0 | −122.71 |
| CYR_AI | 5.00 | −69.52 | 1.50 | −78.32 | 25.0 | **-20.82** | 44.00 | **49.15** |
| EmcmAi | 7.50 | **-51.94** | 6.50 | −77.02 | 0.00 | −62.12 | 0.0 | −124.94 |
| Jitwisut_Zen | 0.0 | −104.71 | 19.00 | **-46.46** | 11.00 | **-55.78** | 1.00 | −118.72 |
| LGIST_Bot | 3.00 | −77.23 | 1.00 | −108.77 | 3.00 | −76.69 | 0.0 | −124.70 |
| JayBot | 6.00 | −82.57 | 1.00 | −102.31 | 15.00 | −60.33 | 0.0 | −129.93 |
| MrTwo | 10.00 | **-54.15** | 42.50 | **-3.19** | 29.00 | **-0.14** | 4.00 | **-40.91** |
| Toothless | 0.0 | −125.33 | 0.0 | **-52.34** | 0.00 | −63.76 | **50.0** | **5.89** |
| MctsAi | 9.00 | **-53.74** | 39.00 | **-3.14** | 31.50 | **-2.39** | 9.00 | **-46.27** |
| KotlinTestAgent | 1.00 | −121.18 | 0.0 | **24.54** | 0.0 | −132.96 | 100.0 | 91.44 |
| JayBot_GM | 1.00 | −72.44 | 8.00 | −72.58 | 21.00 | **-12.70** | 0.0 | −98.62 |
| UtalFighter | 34.00 | **-7.21** | **46.00** | **-5.32** | 35.00 | **-1.56** | 0.0 | −120.73 |
| Dora | 1.00 | −121.18 | 0.0 | **24.54** | 0.0 | −132.96 | 100.0 | 91.44 |
| Caselene | 11.00 | **-35.42** | 43.50 | **-2.83** | 37.5 | **10.84** | 12.00 | **-22.80** |
| SimpleAI | 21.50 | **-21.16** | 4.50 | −127.96 | 28.00 | **-32.26** | 0.0 | −183.09 |
| HaibuAI | 41.50 | **-11.51** | 38.50 | **-12.41** | **44.50** | **-4.26** | 9.50 | **-35.36** |
| MonkeyLink_TriplePM | 23.50 | **-16.33** | 17.00 | **-12.13** | 0.0 | **-18.38** | 0.0 | **-41.45** |
| IBM_AI | **52.00** | **5.17** | **49.50** | **-2.18** | **49.00** | **4.49** | 13.00 | **-26.60** |
| YIYAI | **52.50** | **5.15** | **49.00** | **-6.02** | **47.50** | **0.39** | 70.00 | **40.27** |
| Fuzzy_ZYQAI | **51.00** | **-3.82** | **50.0** | **-14.46** | **47.50** | **0.41** | 2.00 | −88.56 |
| DiceAI | 58.50 | **6.36** | **55.00** | **2.77** | **51.00** | **5.13** | 19.50 | **-16.26** |
| RandomAI | 62.00 | **7.31** | 57.00 | **8.20** | **51.00** | **2.20** | **48.00** | **-0.56** |
| BCP | 57.00 | **-7.68** | **52.00** | **0.43** | **46.50** | **3.88** | **50.0** | **12.24** |
| TOVOR | 63.00 | **4.17** | 56.50 | **7.30** | 57.00 | **6.54** | 71.00 | **21.82** |

tigado se estilos de jogo não vistos durante o treino influenciaram a falha em balancear. A fim de caracterizar e comparar estilos, foi utilizada a distribuição de probabilidade sobre as possíveis ações que um jogador realizou durante uma partida. A ideia principal é que jogadores com diferentes estilos irão exibir padrões na escolha de ações distintos. A Divergência de Kullback-Leibler (KL) foi empregada para quantificar a similaridade entre o comportamento de dois jogadores. KL mede a diferença entre duas distribuições de probabilidade, calculando o quão distante ou perto elas estão uma da outra.

Para essa análise, o agente resultante do treinamento utilizando entropia apresentado na Tabela 6.2 foi utilizado. Com a finalidade de investigar o impacto da divergência de estilos no balanceamento do jogo, foi coletada a distribuição de probabilidade das ações dos três competidores usados no treino (TeraThunder, UtalFighter e BCP). Essa coleta ocorreu durante o teste, em que 100 partidas foram realizadas e as ações contra o nosso agente foram registradas. Além disso, uma média entre esses três estilos foi calculada. Por fim, esses estilos foram comparados contra todos os competidores do torneio realizado. As distâncias entre os estilos de treino e os competidores foram computadas usando a métrica de divergência de KL, calculando, assim, a diferença de estilo entre competidores de treino e teste. As métricas calculadas estão apresentadas na Figura 6.4

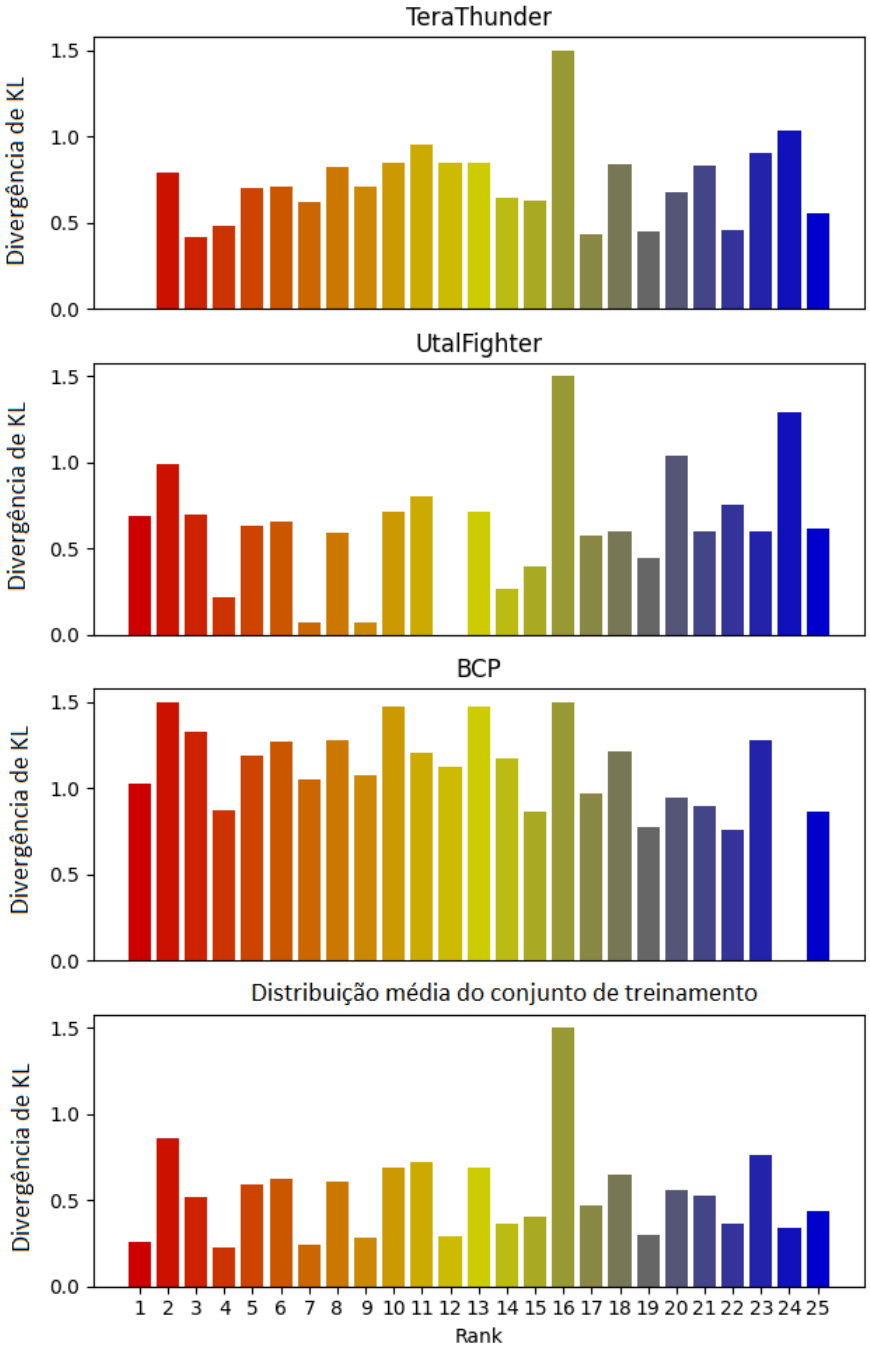



Figura 6.4: Distâncias de estilos dos oponentes de treino em relação aos adversários do torneio. O eixo $x$ representa o competidor do ranqueamento realizado. O eixo $y$ representa a distância de estilo. Primeiros três gráficos representam a distância de estilo dos competidores TeraThunder, UtalFighter e BCP respetivamente contra todos os outros do ranqueamento. O último gráfico representa a média de estilo do conjunto de treino comparada ao teste.

A fim de explorar mais profundamente a relação entre a divergência de estilo e balanceamento do jogo, um gráfico de dispersão (Figura 6.5) foi criado. Este relaciona a divergência de estilo do conjunto de treino (TeraThunder, UtalFighter e BCP) com o teste (outros competidores do torneio) com a distância da taxa de vitória obtida no teste com a ideal 50%. Dessa forma, deseja-se verificar se existe relação entre a incapacidade do agente de apresentar uma competitividade correta e a divergência em estilo observada no treino. Para o gráfico, apenas os pontos com distância à taxa de vitória negativa que não são considerados balanceados são utilizados. Esse filtro nos dados foi aplicado pelo fato de que existem poucos pontos positivos e eles estão muito próximos ou até mesmo no intervalo de competitividade. Além dos pontos, foi realizada uma regressão para verificar o comportamento dos pontos conforme a distância do ideal. O gráfico de dispersão revela uma tendência onde pontos com maior distância do nível de competitividade ideal exibem maior divergência de estilo comparado ao treino. Isso sugere que diferenças significativas de estilo podem influenciar na falha do agente em alcançar o balanceamento e competitividade ideal contra certos oponentes. Indica que o agente pode ter dificuldade em adaptar e encontrar um comportamento ideal de balanceamento quando enfrentando oponentes com estilos de jogo únicos e não vistos durante o treinamento.

### 6.2.2 Agentes com Estilos

Em todos os experimentos realizados para gerar o conjunto de treinamento, foram utilizados os mesmos hiperparâmetros. Os relacionados ao método PPO utilizado estão descritos na Tabela 6.3. Em relação ao treinamento, foram realizados três processos simultâneos em diferentes máquinas. A cada passo de otimização, esses eram sincronizados para equalizar diferença entre *hardwares*. Em relação à seleção de oponentes, foi considerada uma janela de $k = 3$ com as políticas mais recentes de cada estilo. A probabilidade de seleção de elementos desse conjunto foi de $p = 0.8$ e decaiu linearmente para $p = 0.5$ ao longo do treinamento. Os experimentos realizados nesta sessão tiveram como objetivo avaliar o efeito da função de diversidade, assim como determinar o valor ideal para o parâmetro $\mu$ que otimize o *tradeoff* entre maximizar função de recompensa e variabilidade estratégica.

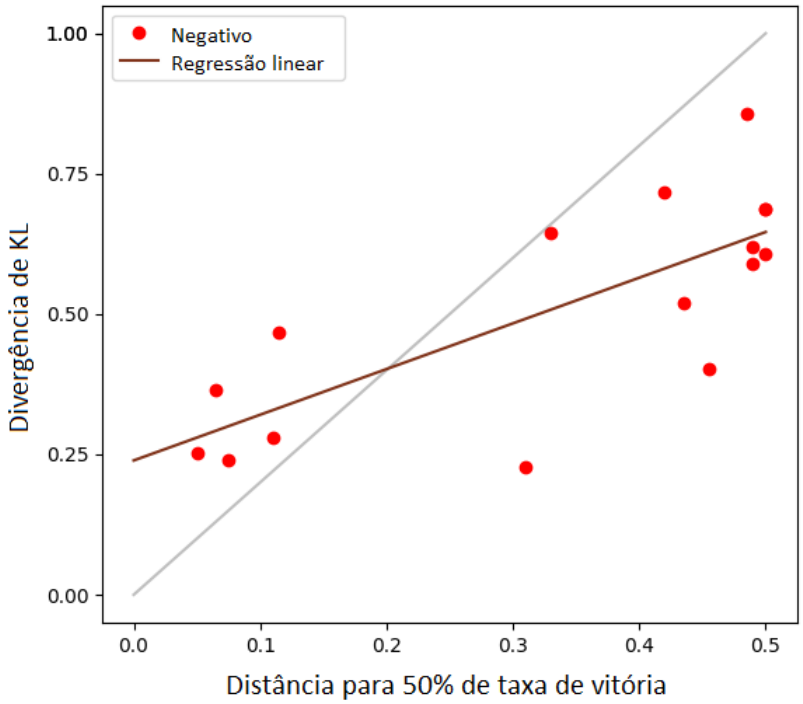



Figura 6.5: Correlação entre a distância de estilos e distância da taxa de vitória alvo (50%). Valores próximos da origem no eixo X representam oponentes que estão mais próximos do objetivo de balanceamento. Valores mais próximos da origem no eixo Y representam que o oponente possui maior similaridade de estilo em relação ao conjunto de treino.

| Hiperparâmetro | Valor |
|---|---|
| Learning rate | 0.0003 |
| Steps per env | 2048 |
| Batch size | 2048 |
| Epochs | 10 |
| Discount factor | 0.99 |
| Gae Lambda | 0.95 |
| Clip range | 0.2 |
| Clip range Value Function | 0.1 |
| Normalize Advantage | True |
| Entropy coeficient | 0.01 |
| Value function coeficient | 0.5 |
| Maximum Gradient Norm | 0.5 |

Tabela 6.3: Hiperparâmetros do método PPO utilizados na geração do conjunto de treinamento.

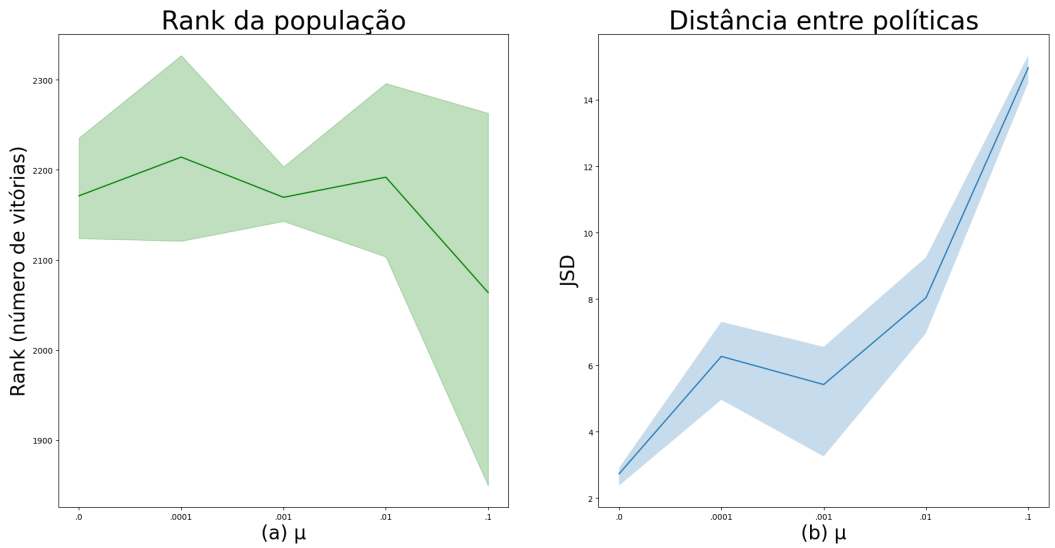
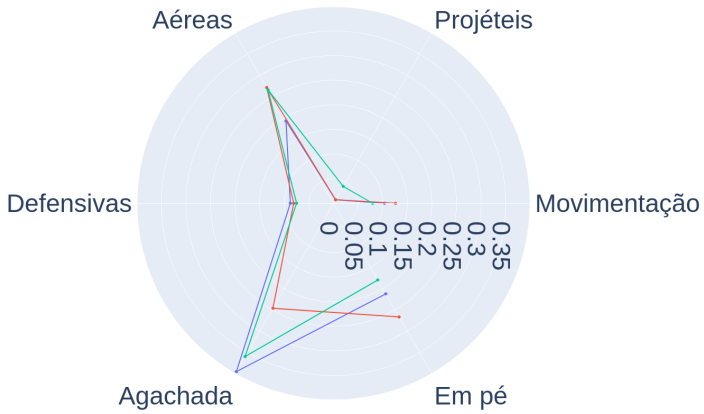



Figura 6.6: (a) Rank e (b) diferença entre políticas médio, mínimo e máximo com a variação de $\mu$.

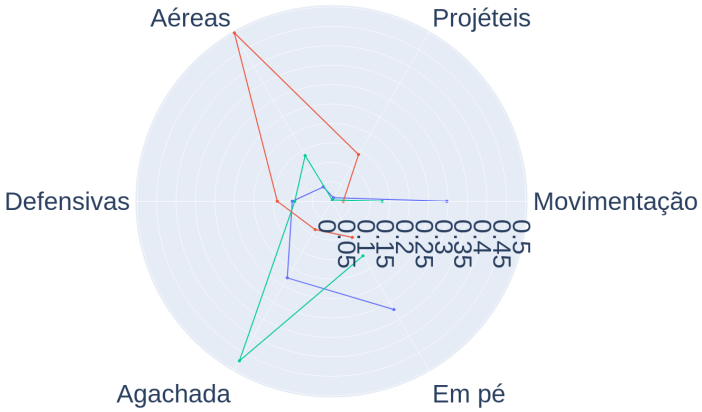
#### 6.2.2.1 Estudo da função de Diversidade

Os experimentos anteriores demonstraram que apenas considerar habilidade relativa não é suficiente para generalizar o comportamento contra uma diversidade de oponentes. Além disso, através do estudo de regularização, foram apontados que uma possível causa da baixa capacidade de balanceamento contra oponentes do torneio é a falta de diversidade de estratégias no treinamento. Dessa forma, faz-se necessário a criação de um conjunto $\mathcal{M}|_{\mathcal{C}_{train}}$ com essa propriedade. Assim, foram realizados experimentos com o treinamento proposto na seção 5.3.1 com o intuito de gerar tal conjunto. Idealmente, deseja-se maximizar ambas as performances e variedade estratégica no zoológico final. Entretanto, como discutido na seção 5.3.1.1, existe um *tradeoff* entre maximizar a recompensa e diferença entre políticas. Dessa forma, foram realizados experimentos variando o parâmetro $\mu$ que controla a função de similaridade a fim de escolher a melhor combinação possível. A Figura 6.6 mostra o *rank* final dos agentes treinados com seus respectivos valores de $\mu$, assim como a divergência calculada entre os agentes em relação ao zoológico. Observa-se que conforme $\mu$ aumenta, o valor máximo de *rank* não se altera muito, entretanto, ao avaliar o mínimo, existe uma queda grande quando testado o $\mu = 0.1$. Com valores altos do parâmetro, o otimizador prioriza aumentar a divergência entre políticas em detrimento da função de recompensa. Considerando a imagem, foi escolhido o valor $\mu = .01$, que apresenta um bom equilíbrio entre habilidade relativa e variabilidade estratégica.

De modo a avaliar se a função de divergência gera realmente diferentes estilos, avaliamos as ações tomadas pelos agentes finais com $\mu = 0.0$ e $\mu = 0.01$ no torneio




(a) $\mu = 0.0$




(b) $\mu = 0.01$

Figura 6.7: Tipos de ações escolhidas pelos agentes finais nos treinamentos sem a função de divergência (a) e com (b).

realizado. Para tal, dividimos entre seis diferentes grupos de ações: aéreas, projéteis, defensivas, movimentação, agachada e em pé. Algumas ações, como *defesa aérea* que defende golpes enquanto o agente está no ar, podem pertencer a mais de um grupo. Para lidar com essas sobreposições, quando essa ação é escolhida, ela é contabilizada em todos os grupos a que pertence. Foram contabilizadas todas as ações escolhidas pelos agentes para todos os oponentes do torneio e normalizadas pelo total de escolhas. O resultado dessa coleta de dados está presente na Figura 6.7. Observa-se que para $\mu = 0$, os tipos de ações escolhidas são similares entre todos os agentes do treinamento. Em contrapartida, no caso $\mu = 0.01$ existe uma divergência entre as escolhas. Assim, no segundo caso, os agentes realizam diferentes tipos de ações para ganhar o jogo, consequentemente, apresentam diferentes estilos de jogo.

Por fim, as figuras 6.8 e 6.9 apresentam as taxas de vitória dos agentes resultantes finais dos treinos com $\mu = 0.0$ e $\mu = 0.01$ respectivamente contra oponentes do torneio.

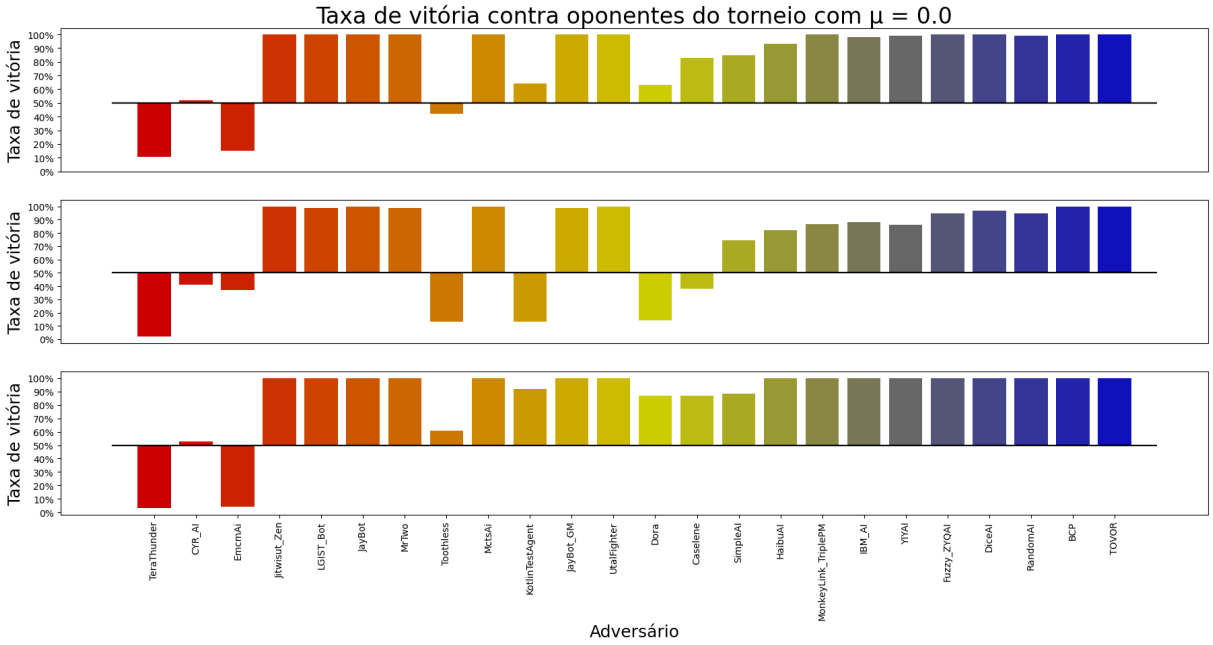



Figura 6.8: Taxa de vitória dos agentes finais contra oponentes do simulador com $\mu = 0.0$.

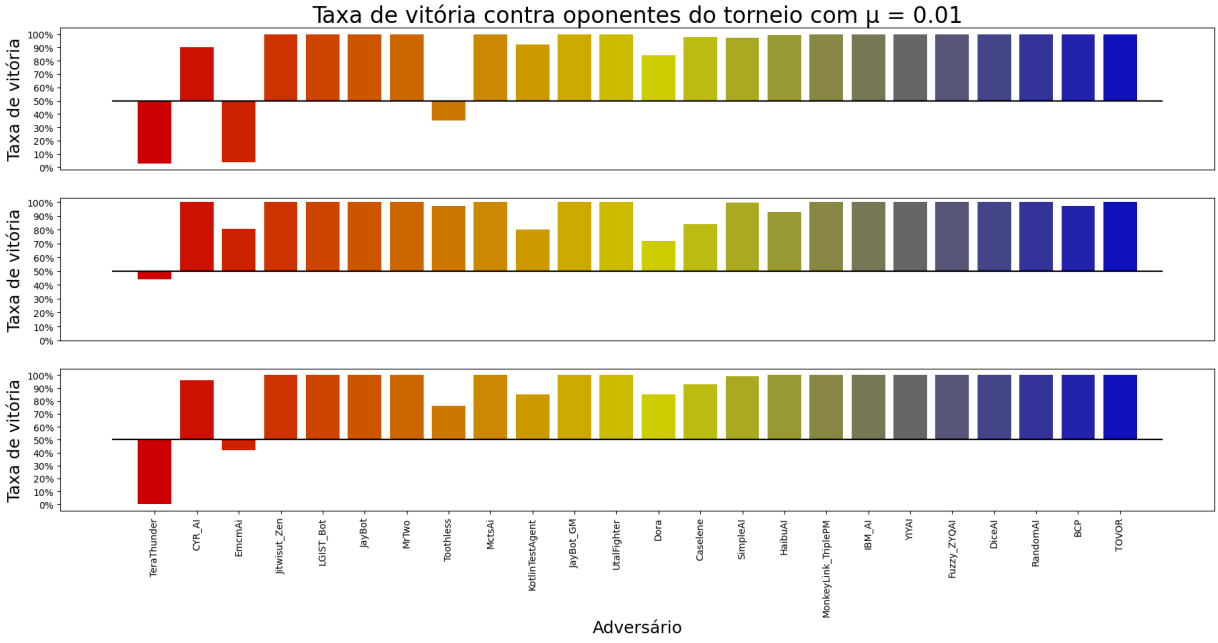



Figura 6.9: Taxa de vitória dos agentes finais contra oponentes do simulador com $\mu = 0.01$.

Observa-se que mesmo sem a função de diversidade, o treinamento com $\mu = 0.0$ apresenta um comportamento robusto, conseguindo vencer grande parte dos competidores do torneio. Entretanto, nota-se que existem alguns casos, principalmente nas dificuldades médias e altas, que alguns desses agentes são incapazes de vencer contra alguns oponentes. Em contrapartida, no caso de $\mu = 0.01$, os comportamentos gerados conseguem ganhar desses adversários. Com a função de diversidade, os agente observam diferentes estratégias e conseguem lidar com a variedade de estilos presentes nos competidores.

### 6.2.3 Agente DDA

Após a geração do conjunto $\mathcal{M}|_{\mathcal{C}_{train}}$, o treinamento do agente DDA pode ser realizado. Os hiperparâmetros relacionados ao método PPO estão na Tabela 6.4.

| Hiperparâmetros do método PPO para o agente DDA | |
|---|---|
| Learning rate | 0.0003 |
| Steps per env | 2048 |
| Batch size | 2048 |
| Epochs | 10 |
| Discount factor | 0.99 |
| Gae Lambda | 0.95 |
| Clip range | 0.2 |
| Clip range Value Function | 0.1 |
| Normalize Advantage | True |
| Entropy coeficient | 0.0 |
| Value function coeficient | 0.1 |
| Maximum Gradient Norm | 0.5 |

Tabela 6.4: Hiperparâmetros do método PPO utilizados na geração do agente DDA.

O treinamento utiliza quatro ambientes em paralelo. Cada um deles amostra um oponente com probabilidade correspondente ao número de passos e etapa de treinamento conforme descrito na seção 5.4. Para todos os experimentos realizados nessa seção, os intervalos de balanceamento escolhidos foram $I_{BC} = [-0.2, 0.2]$ e $I_C = [0.4, 0.6]$. A fim de avaliar o efeito que a diversidade no conjunto causa no método DDA, foram realizados dois treinamentos distintos, com os agentes resultantes $\mu = 0.0$ e $\mu = 0.01$. Os métodos DDA resultantes de ambos os treinamentos foram testados contra oponentes do torneio. As Figuras 6.11 e 6.12 apresentam o desempenho no teste dos agentes de balanceamento respectivamente. As faixas verdes demonstram os intervalos de balanceamento em cada uma das métricas avaliadas. Inicialmente, observando apenas a diferença de vida, observa-se que, em ambos os casos, os agentes conseguiram manter a partida balanceada para quase todos os oponentes testados. Em ambos os casos, o agente falhou em balancear na maior parte das partidas contra o primeiro adversário (*TeraThunder*). Em relação à competitividade, ambos apresentaram dificuldade para manter a taxa de vitória no intervalo para alguns oponentes específicos. Nota-se que, embora não consiga manter a competitividade para alguns oponentes, o treinamento $\mu = 0.01$ apresentou melhores resultados comparado ao sem a função de diversidade, além de conseguir balancear para mais oponentes.

Inicialmente, esperava-se que o treinamento sem a função de diversidade resultasse em um agente significativamente menos robusto em comparação com aquele treinado

utilizando essa função. No entanto, os resultados indicam que ambos os treinamentos apresentaram um bom desempenho em termos de generalização, com os agentes resultantes sendo capazes de balancear tanto $B(s)$ quanto a taxa de vitória contra grande parte dos oponentes. O pequeno aumento na generalização ao aplicar a função de diversidade pode ser atribuído a dois possíveis fatores: o método de treinamento adotado e o conjunto de teste utilizado. Primeiramente, como discutido na Seção 5.3.1, o treinamento baseado em população favorece a geração de comportamentos robustos, pois cada agente explora diferentes estratégias devido ao mecanismo de exploração. Além disso, ao enfrentarem uns aos outros, os agentes buscam estratégias para superar seus adversários dentro do conjunto, assim a busca da melhor resposta pode gerar diferentes soluções. Essa variabilidade pode ser suficiente, no ambiente aplicado, para produzir um agente DDA robusto.

Outra possível explicação está relacionada ao conjunto de teste. Neste estudo, assumiu-se que os oponentes coletados de competições anteriores do simulador possuem uma boa variação estratégica. No entanto, alguns desses métodos são evoluções de versões anteriores, modelos aprimorados de outros ou utilizam comportamentos pré-programados comuns considerados ideais em certas situações. Dessa forma, pode haver similaridade entre alguns oponentes, o que pode levar a uma superestimação da generalização do agente treinado. A fim de tentar avaliar essa possibilidade, um gráfico de estilo, similar ao feito na Seção 6.2.2.1, foi realizado com os integrantes do torneio. As ações realizadas por cada oponente contra o agente DDA $\mu = 0.01$ foram analisadas. A Figura 6.10 mostra a distribuição das ações realizadas, por grupo, dos competidores do torneio. Pode-se observar que existe grande concentração em alguns grupos e padrões de escolha similares. Em específico, comportamentos defensivos, aéreos e projéteis são pouco explorados. Com esses padrões de comportamento parecidos, a generalização calculada nesse conjunto de teste pode ser não tão representativa.

Apesar do ganho de desempenho ter sido pequeno, observa-se que o agente DDA treinando contra o conjunto $\mu = 0.01$, em relação à competitividade, consegue balancear para oponentes que o treinamento sem função de diversidade foi incapaz. Isso sugere que, mesmo com um impacto relativamente pequeno na generalização, a diversidade contribui para uma melhor adaptação do agente a diferentes estilos de jogo.

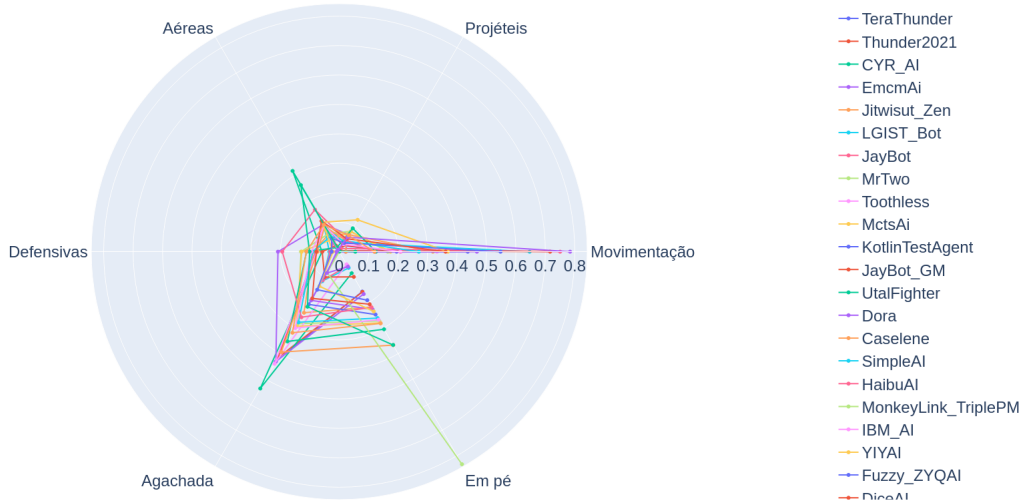



Figura 6.10: Tipos de ações escolhidas pelos competidores do torneio realizado contra o agente treinado.

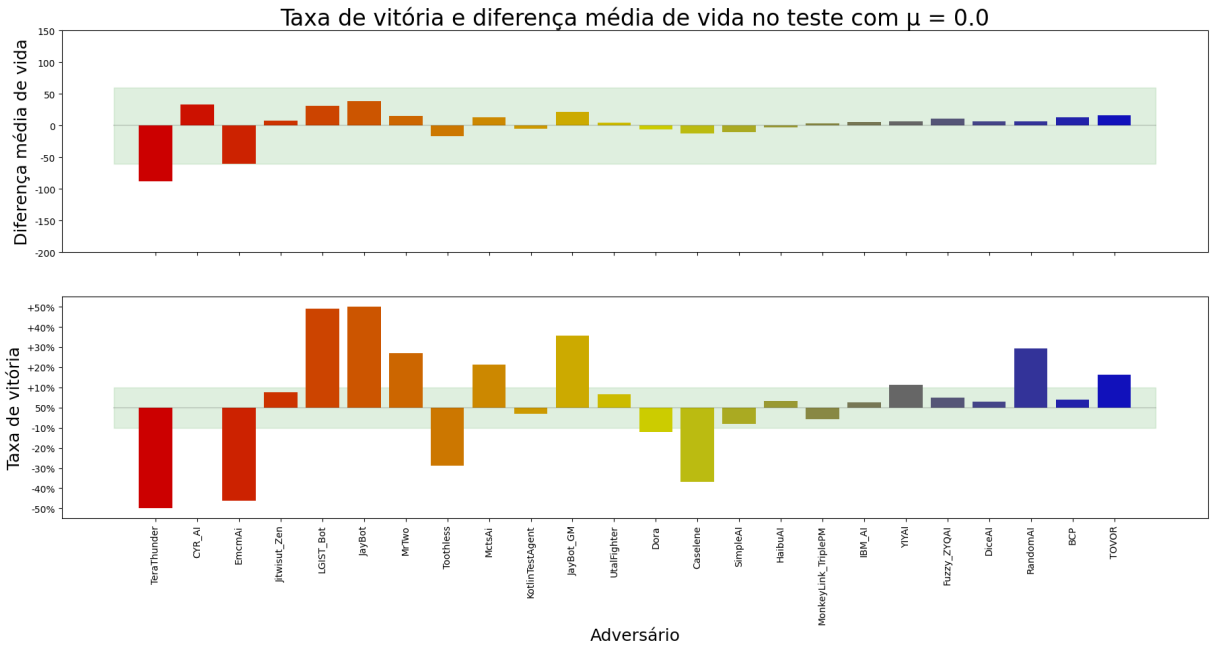



Figura 6.11: Taxa de vitória dos agentes finais contra oponentes do simulador com $\mu = 0.0$.

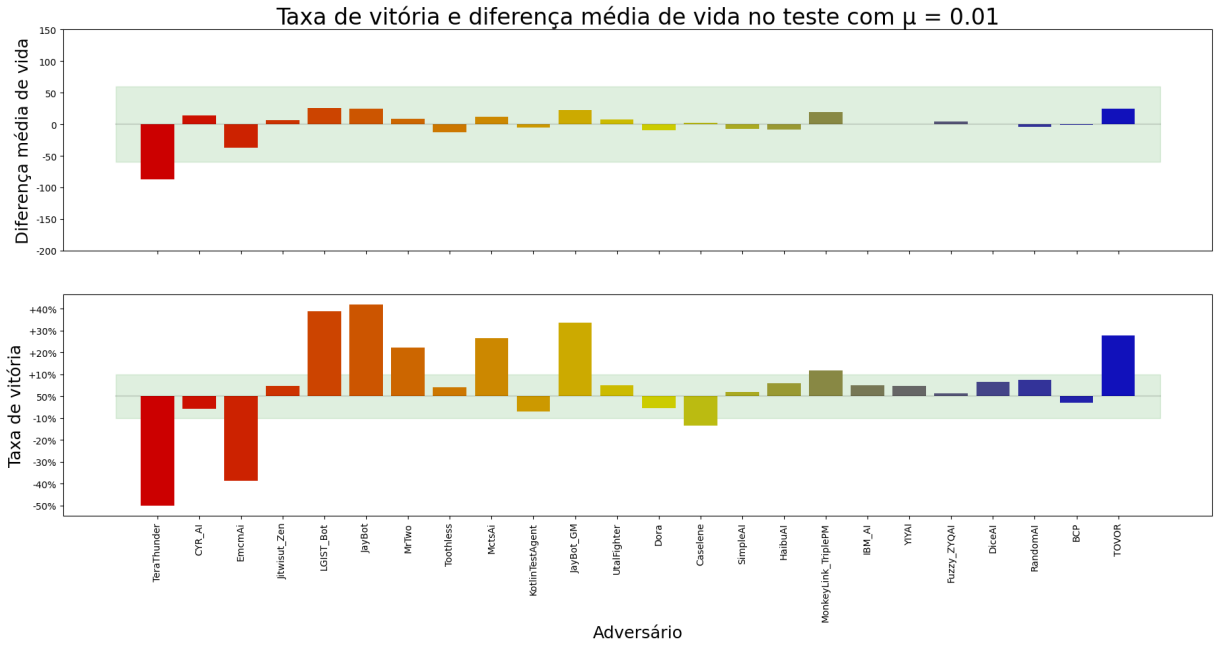



Figura 6.12: Taxa de vitória dos agentes finais contra oponentes do simulador com $\mu = 0.01$.

# Capítulo 7

# Conclusões

## 7.1 Visão geral

Neste trabalho foi apresentado um método de ajuste dinâmico de dificuldade baseado em aprendizado por reforço aplicado a um jogo de luta (FightingICE). O objetivo é conseguir proporcionar a um jogador qualquer uma experiência engajante, apresentando um grau de dificuldade ideal para a habilidade do usuário. O método alcança isso controlando dois fatores distintos do jogo: balanceamento e competitividade. Balanceamento se refere ao grau de dificuldade apresentado ao jogador durante a tarefa sendo executada. Especificamente para o jogo aplicado, mede-se essa percepção de dificuldade através da diferença de vida do agente e do jogador. Assim, o método busca manter essa diferença próxima a zero, consequentemente gerando partidas balanceadas. Competitividade se refere à probabilidade do jogador obter sucesso ao executar a tarefa. Promover partidas balanceadas não é suficiente para maximizar o entretenimento. Isso porque ter sucesso em tarefas difíceis aumenta o engajamento, enquanto perder tem o efeito oposto.

O método proposto tem como objetivo controlar essa probabilidade para o nível desejado. Novamente, para o jogo de luta, foi controlada a taxa de vitória do jogador contra o agente. O método tentará ganhar ou perder as partidas, mantendo o balanceamento dentro do estipulado, para alcançar a competitividade ideal. Além do método proposto, foram investigadas formas de tornar o modelo robusto para distintos jogadores. Para tal, foi proposto um esquema de treinamento que gera oponentes com diferentes habilidades relativas e estilos de jogo. A fim de gerar tais oponentes, o *framework* de treinamento baseado em população foi aplicado. Esse tipo de treinamento mantém um conjunto de agentes treinando entre si simultaneamente com o intuito de aprimorar a performance coletiva. Para garantir a diversidade estratégica, foi aplicado um novo objetivo na atualização dos pesos das redes. Esse tem como não só maximizar as recompensas, como também maximizar a diversidade de estilos. O agente DDA é treinado contra os outros agentes, apresentando distintos estilos e versões históricas dos mesmos. Com isso, aumentando a generalização do modelo final e consequentemente tornando-o mais robusto

contra jogadores não vistos durante o treinamento.

Inicialmente, foram feitos experimentos contra oponentes fixos. Esses experimentos demonstraram o comportamento do método proposto com diferentes parâmetros. Observou-se que neste cenário o agente conseguia manter ambos balanceamento e competitividade nos níveis desejados. Além disso, o método proposto gerou comportamentos mais interativos comparado ao método original. Em relação à generalização, inicialmente foram testados métodos de regularização para verificar se os mesmos podiam trazer benefícios para a robustez do modelo. Os resultados apontaram que os métodos *entropy* e *dropout* tiveram aumento na performance no teste comparado ao *baseline*. Em contrapartida, o *L2* prejudicou o desempenho tanto no treino quando no teste. Por fim, os experimentos com a metodologia de treinamento proposta mostraram duas vantagens ao utilizar uma função de diversidade. A primeira é gerar agentes que tem objetivo de vencer a partida mais fortes e robustos. A segunda é o impacto positivo na generalização do agente DDA final, embora os ganhos de generalização foram poucos comparado ao treino sem a função de diversidade explícita. Com esse treinamento foi possível gerar um comportamento que consegue manter o balanceamento para todos os oponentes do torneio, com exceção do primeiro colocado. Além disso, o método DDA final mantêm a competitividade no nível desejado para 16 competidores de 25. Com isso, foi possível verificar que a abordagem adotada gerou resultados positivos, alcançando os objetivos propostos.

## 7.2 Trabalhos Futuros

Como trabalhos futuros, uma possibilidade seria aplicar o método em uma escala maior, aumentando o número de agentes envolvidos no treinamento e utilizando redes neurais mais complexas. Isso permitiria avaliar a escalabilidade e o desempenho do modelo proposto em cenários mais desafiadores e diversificados.

Outra direção interessante seria realizar testes com usuários reais. Como o objetivo principal do método é aprimorar o engajamento do jogador, investigar a percepção dos usuários sobre o agente DDA final seria importante. Esse experimento não apenas forneceria informações sobre a percepção e a experiência dos jogadores, mas também permitiria avaliar a robustez do método frente aos comportamentos dinâmicos e imprevisíveis dos jogadores humanos, oferecendo um passo significativo em direção à aplicação prática do modelo.

# Referências


[1] Newzoo: The global standard for games data., 2022.

[2] The science of gamer motivation., 2022.

[3] Oladayo S. Ajani and Rammohan Mallipeddi. Pareto-based dynamic difficulty adjustment of a competitive exergame for arm rehabilitation. *International Journal of Human-Computer Studies*, 178:103100, 2023.

[4] M. Taufik Akbar, M. Nasrul Ilmi, Imanuel V. Rumayar, Jurike Moniaga, Tin-Kai Chen, and Andry Chowanda. Enhancing game experience with facial expression recognition as dynamic balancing. *Procedia Computer Science*, 157:388–395, 2019. The 4th International Conference on Computer Science and Computational Intelligence (ICCSCI 2019) : Enabling Collaboration to Escalate Impact of Research Results for Society.

[5] Tomás Alves, Henrique Carvalho, and Daniel Simões Lopes. Winning compensations: Adaptable gaming approach for upper limb rehabilitation sessions based on compensatory movements. *Journal of Biomedical Informatics*, 108:103501, 2020.

[6] Zahra Amiri, Yoones A. Sekhavat, and Sakineh Goljaryan. Stepar: A personalized exergame for people with multiple sclerosis based on video-mapping. *Entertainment Computing*, 42:100487, 2022.

[7] Gustavo Andrade, Geber Ramalho, Hugo Santana, and Vincent Corruble. Extending Reinforcement Learning to Provide Dynamic Game Balancing. In *IJCAI 2005 Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 7–12, Edinburgh, United Kingdom, July 2005.

[8] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search, 2017.

[9] Atari. Breakout. Atari 2600, 1978.

[10] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition, 2018.

[11] Richard Bartle. Hearts, clubs, diamonds, spades: Players who suit muds. 06 1996.

[12] Riccardo Berta, Francesco Bellotti, Alessandro De Gloria, Danu Pranantha, and Carlotta Schatten. Electroencephalogram and physiological signal analysis for assessing flow in games. *Computational Intelligence and AI in Games, IEEE Transactions on*, 5:164–175, 06 2013.

[13] Matteo Bettini, Ajay Shankar, and Amanda Prorok. System neural diversity: Measuring behavioral heterogeneity in multi-agent learning, 2024.

[14] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[15] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[16] Konstantina Chrysafiadi, Margaritis Kamitsios, and Maria Virvou. Fuzzy-based dynamic difficulty adjustment of an educational 3d-game. *Multimedia Tools and Applications*, 82:1–25, 02 2023.

[17] Chris Crawford. The art of computer game design. 1984.

[18] Mihaly Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. 01 1990.

[19] Ali Darzi, Sean McCrea, and Vesna Novak. User experience with dynamic difficulty adjustment methods for an affective exergame: Comparative laboratory-based study. *JMIR Serious Games*, 9:e25771, 05 2021.

[20] Tiago Negrisoli De Oliveira and Luiz Chaimowicz. Investigating reinforcement learning for dynamic difficulty adjustment. In *Proceedings of the 22nd Brazilian Symposium on Games and Digital Entertainment*, SBGames '23, page 66–75, New York, NY, USA, 2024. Association for Computing Machinery.

[21] Simon Demediuk, Marco Tamassia, Xiaodong Li, and William Raffe. Challenging ai: Evaluating the effect of mcts-driven dynamic difficulty adjustment on player enjoyment. pages 1–7, 01 2019.

[22] Simon Demediuk, Marco Tamassia, William L. Raffe, Fabio Zambetta, Xiaodong Li, and Florian Mueller. Monte carlo tree search based algorithms for dynamic difficulty adjustment. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 53–59, 2017.

[23] Simon Demediuk, Marco Tamassia, William L. Raffe, Fabio Zambetta, Florian "Floyd"Mueller, and Xiaodong Li. Measuring player skill using dynamic difficulty adjustment. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '18, New York, NY, USA, 2018. Association for Computing Machinery.

[24] Alena Denisova and Paul Cairns. Adaptation in digital games: The effect of challenge adjustment on player performance and experience. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '15, page 97–101, New York, NY, USA, 2015. Association for Computing Machinery.

[25] Alena Denisova, Paul Cairns, Christian Guckelsberger, and David Zendle. Measuring perceived challenge in digital games: Development  validation of the challenge originating from recent gameplay interaction scale (corgis). *International Journal of Human-Computer Studies*, 137:102383, 2020.

[26] Anders Drachen, Lennart E. Nacke, Georgios Yannakakis, and Anja Lee Pedersen. Correlation between heart rate, electrodermal activity and player experience in first-person shooter games. In *Proceedings of the 5th ACM SIGGRAPH Symposium on Video Games*, Sandbox '10, page 49–54, New York, NY, USA, 2010. Association for Computing Machinery.

[27] Paulo Eduardo Battistella and Christiane Gresse von Wangenheim. Games for teaching computing in higher education – a systematic review. *IEEE Technology and Engineering Education*, 1:8–30, 03 2016.

[28] Arpad E. Elo. The rating of chessplayers, past and present. 1978.

[29] Blizzard Entertainment. Starcraft ii. Microsoft Windows, macOS, 2010.

[30] Julian Frommel, Fabian Fischbach, Katja Rogers, and Michael Weber. Emotion-based dynamic difficulty adjustment using parameterized difficulty and self-reports of emotion. pages 163–171, 10 2018.

[31] Kazuhisa Fujita. Alphadda: Strategies for adjusting the playing strength of a fully trained alphazero system to a suitable human training partner, 2022.

[32] Marta Garnelo, Wojciech Marian Czarnecki, Siqi Liu, Dhruva Tirumala, Junhyuk Oh, Gauthier Gidel, Hado van Hasselt, and David Balduzzi. Pick your battles: Interaction graphs as population-level objectives for strategic diversity, 2021.

[33] Robby Goetschalckx. Games with dynamic difficulty adjustment using pomdps. 2010.

[34] Miguel González-Duque, Rasmus Berg Palm, David Ha, and Sebastian Risi. Finding game levels with the right difficulty in a few trials through intelligent trial-and-error, 2020.

[35] Miguel González-Duque, Rasmus Berg Palm, and Sebastian Risi. Fast game content adaptation through bayesian-based player modelling, 2021.

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[37] Jordi Grau-Moya, Felix Leibfried, and Haitham Bou-Ammar. Balancing two-player stochastic games with soft q-learning, 2018.

[38] Renê Gusmão, Kennet Calixto, and Caetano Segundo. Dynamic difficulty adjustment through parameter manipulation for space shooter game. 09 2015.

[39] Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes, 2015.

[40] Juho Hamari and Lauri Keronen. Why do people play games? a meta-analysis. *International Journal of Information Management*, 37(3):125–141, 2017.

[41] Daniel Hernandez, Kevin Denamganai, Yuan Gao, Peter York, Sam Devlin, Spyridon Samothrakis, and James Walker. A generalized framework for self-play training. 07 2019.

[42] Tianyi Hu, Zhiqiang Pu, Xiaolin Ai, Tenghai Qiu, and Jianqiang Yi. Measuring policy distance for multi-agent reinforcement learning, 2024.

[43] Robin Hunicke. The case for dynamic difficulty adjustment in games. volume 265, pages 429–433, 06 2005.

[44] Robin Hunicke and Vernell Chapman. Ai for dynamic difficulty adjustment in games. *Challenges in game artificial intelligence AAAI workshop*, 2, 01 2004.

[45] King Games Inc. Candy crush saga. Facebook, iOS, Android, Windows Phone, Windows 10, Tizen, 2012.

[46] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, May 2019.

[47] Yuhua Jiang, Qihan Liu, Xiaoteng Ma, Chenghao Li, Yiqin Yang, Jun Yang, Bin Liang, and Qianchuan Zhao. Learning diverse risk preferences in population-based self-play, 2023.

[48] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, January 2023.

[49] Johann Knorr and Carlos Vaz de Carvalho. Using dynamic difficulty adjustment to improve the experience and train fps gamers. pages 195–200, 10 2021.

[50] Raphaël Koster and Will Wright. A theory of fun for game design. 2004.

[51] Carlos Lara-Alvarez, Hugo Mitre-Hernandez, Juan Flores, and Humberto Espinosa. Induction of emotional states in educational video games through a fuzzy control system. *IEEE Transactions on Affective Computing*, 1:1, 05 2018.

[52] Matthias Lehmann. The definitive guide to policy gradients in deep reinforcement learning: Theory, algorithms and implementations. *ArXiv*, abs/2401.13662, 2024.

[53] Jiayu Li, Hongyu Lu, Chenyang Wang, Weizhi Ma, Min Zhang, Xiangyu Zhao, Wei Qi, Yiqun Liu, and Shaoping Ma. A difficulty-aware framework for churn prediction and intervention in games. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 943–952, New York, NY, USA, 2021. Association for Computing Machinery.

[54] Changchun Liu, Pramila Agrawal, Nilanjan Sarkar, and Shuo Chen. Dynamic difficulty adjustment in computer games through real-time anxiety-based affective feedback. *Int. J. Hum. Comput. Interaction*, 25:506–529, 08 2009.

[55] Xiangyu Liu, Hangtian Jia, Ying Wen, Yaodong Yang, Yujing Hu, Yingfeng Chen, Changjie Fan, and Zhipeng Hu. Unifying behavioral and response diversity for open-ended learning in zero-sum games, 2021.

[56] Yuan Liu, Ruimin Shen, Miqing Li, Yingfeng Chen, Juan Zou, and Changjie Fan. Explicitly maintaining diverse playing styles in self-play, 2023.

[57] Zongkai Liu, Chao Yu, Yaodong Yang, Peng Sun, and Zifan Wu. A unified diversity measure for multiagent reinforcement learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc.

[58] Weifan Long, Taixian Hou, Xiaoyi Wei, Shichao Yan, Peng Zhai, and Lihua Zhang. A survey on population-based deep reinforcement learning. *Mathematics*, 11:2234, 05 2023.

[59] Feiyu Lu, Kaito Yamamoto, Luis H. Nomura, Syunsuke Mizuno, YoungMin Lee, and Ruck Thawonmas. Fighting game artificial intelligence competition platform. In *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, pages 320–323, 2013.

[60] Andrei Lupu, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, page 1593–1595, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems.

[61] Thomas W. Malone. What makes things fun to learn? a study of intrinsically motivating computer games. 1981.

[62] Kevin R. McKee, Joel Z. Leibo, Charlie Beattie, and Richard Everett. Quantifying the effects of environment and population diversity in multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 36(1), March 2022.

[63] Olana Missura and Thomas Gärtner. Player modeling for intelligent difficulty adjustment. volume 5808, pages 197–211, 10 2009.

[64] Jurike V. Moniaga, Andry Chowanda, Agus Prima, Oscar, and M. Dimas Tri Rizqi. Facial expression recognition as dynamic game balancing system. *Procedia Computer Science*, 135:361–368, 2018. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life.

[65] JaeYoung Moon, YouJin Choi, TaeHwa Park, JunDoo Choi, Jin-Hyuk Hong, and Kyung-Joong Kim. Diversifying dynamic difficulty adjustment agent by integrating player state models into monte-carlo tree search. *Expert Systems with Applications*, 205:117677, 06 2022.

[66] Paraschos Moschovitis and Alena Denisova. Keep calm and aim for the head: Biofeedback-controlled dynamic difficulty adjustment in a horror game. *IEEE Transactions on Games*, pages 1–1, 2022.

[67] Lennart E. Nacke, Chris Bateman, and Regan L. Mandryk. Brainhex: A neurobiological gamer typology survey. *Entertainment Computing*, 5(1):55–62, 2014.

[68] Nicolas Perez Nieves, Yaodong Yang, Oliver Slumbers, David Henry Mguni, Ying Wen, and Jun Wang. Modelling behavioural diversity for learning in open-ended games, 2021.

[69] Nintendo. Mario kart series. Microsoft Windows, 1992.

[70] Ashey Noblega., Aline Paes., and Esteban Clua. Towards adaptive deep reinforcement game balancing. In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,*, pages 693–700. INSTICC, SciTePress, 2019.

[71] Inseok Oh, Seungeun Rho, Sangbin Moon, Seongho Son, Hyoil Lee, and Jinyun Chung. Creating pro-level ai for a real-time fighting game using deep reinforcement learning, 2019.

[72] Jacob Kaae Olesen, Georgios N. Yannakakis, and John Hallam. Real-time challenge balance in an rts game using rtneat. In *2008 IEEE Symposium On Computational Intelligence and Games*, pages 87–94, 2008.

[73] OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning, 2019.

[74] Dvir Ben Or, Michael Kolomenkin, and Gil Shabat. Dl-dda – deep learning based dynamic difficulty adjustment with ux and gameplay constraints, 2021.

[75] Afshin OroojlooyJadid and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning, 2021.

[76] Fatih Ozkul, Yunus Palaska, Engin Masazade, and Duygun Erol-Barkana. Exploring dynamic difficulty adjustment mechanism for rehabilitation tasks using physiological measures and subjective ratings. *IET Signal Processing*, 13(3):378–386, 2019.

[77] Jack Parker-Holder, Aldo Pacchiano, Krzysztof Choromanski, and Stephen Roberts. Effective diversity in population based reinforcement learning, 2020.

[78] Xiaolan Peng, Chenyu Meng, Xurong Xie, Jin Huang, Hui Chen, and Hongan Wang. Detecting challenge from physiological signals: A primary study with a typical game scenario. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI EA '22, New York, NY, USA, 2022. Association for Computing Machinery.

[79] Johannes Pfau, Jan David Smeddinck, and Rainer Malaka. Enemy within: Long-term motivation effects of deep player behavior models for dynamic difficulty adjustment. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–10, New York, NY, USA, 2020. Association for Computing Machinery.

[80] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[81] Guillermo Romera Rodriguez. Player modeling for dynamic difficulty adjustment in top down action games. 2019.

[82] Marcos P. C. Rosa, Eduardo A. dos Santos, Iago L. R. de Moraes, Tiago B. P. e Silva, Mauricio M. Sarmet, Carla D. Castanho, and Ricardo P. Jacobi. Dynamic difficulty adjustment using performance and affective data in a platform game. In Constantine Stephanidis, Marcelo M. Soares, Elizabeth Rosenzweig, Aaron Marcus, Sakae Yamamoto, Hirohiko Mori, Pei-Luen Patrick Rau, Gabriele Meiselwitz, Xiaowen Fang, and Abbas Moallem, editors, *HCI International 2021 - Late Breaking Papers: Design and User Experience*, pages 367–386, Cham, 2021. Springer International Publishing.

[83] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[84] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[85] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[86] Yoones A. Sekhavat. Mprl: Multiple-periodic reinforcement learning for difficulty adjustment in rehabilitation games. In *2017 IEEE 5th International Conference on Serious Games and Applications for Health (SeGAH)*, pages 1–7, 2017.

[87] Lingdao Sha, Souju He, Junping Wang, Jiajian Yang, Yuan Gao, Yidan Zhang, and Xinrui Yu. Creating appropriate challenge level game opponent by the use of dynamic difficulty adjustment. In *2010 Sixth International Conference on Natural Computation*, volume 8, pages 3897–3901, 2010.

[88] Mirna Silva, Victor Silva, and Luiz Chaimowicz. Dynamic difficulty adjustment on moba games. *Entertainment Computing*, 18, 10 2016.

[89] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[90] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Driessche, Thore

Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 10 2017.

[91] Valve South. Left 4 dead. Microsoft Windows, 2008.

[92] Adi Stein, Yair Yotam, Rami Puzis, Guy Shani, and Meirav Taieb-Maimon. Eeg-triggered dynamic difficulty adjustment for multiplayer games. *Entertainment Computing*, 25:14–25, 2018.

[93] Matthew Stephenson and Jochen Renz. Agent-based adaptive level generation for dynamic difficulty adjustment in angry birds, 2019.

[94] Rhio Sutoyo, Davies Winata, Katherine Oliviani, and Dedy Supriyadi. Dynamic difficulty adjustment in tower defence. *Procedia Computer Science*, 59:435–444, 2015.

[95] R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998.

[96] Penelope Sweetser and Peta Wyeth. Gameflow: a model for evaluating player enjoyment in games. *Comput. Entertain.*, 3(3):3, July 2005.

[97] K.S. Tekinbas and E. Zimmerman. *Rules of Play: Game Design Fundamentals*. ITPro collection. MIT Press, 2003.

[98] Julian Togelius, Renzo De Nardi, and Simon M. M. Lucas. Making racing fun through player modeling and track evolution. 2006.

[99] Valve. Mario kart series. Windows, Linux, OS X, 2013.

[100] Chineng Vang. The impact of dynamic difficulty adjustment on player experience in video games. 09 2022.

[101] Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575, 11 2019.

[102] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James

Molloy, Tom Le Paine, Caglar Gulcehre, Ziyun Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575:350 – 354, 2019.

[103] Matheus Weber and Pollyana Notargiacomo. Dynamic difficulty adjustment in digital games using genetic algorithms. In *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 62–70, 2020.

[104] Su Xue, Meng Wu, John Kolen, Navid Aghdaie, and Kazi A. Zaman. Dynamic difficulty adjustment for maximized engagement in digital games. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, page 465–471, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[105] Georgios N. Yannakakis and John Hallam. Towards optimizing entertainment in computer games. *Applied Artificial Intelligence*, 21(10):933–971, 2007.

[106] Yaqian Zhang and Wooi-Boon Goh. Personalized task difficulty adaptation based on reinforcement learning. *User Modeling and User-Adapted Interaction*, 31:1–32, 09 2021.

[107] Rui Zhao, Jinming Song, Yufeng Yuan, Hu Haifeng, Yang Gao, Yi Wu, Zhongqian Sun, and Yang Wei. Maximum entropy population-based training for zero-shot human-ai coordination, 2022.

[108] Mohammad Zohaib. Dynamic difficulty adjustment (dda) in computer games: A review. *Advances in Human-Computer Interaction*, 2018:1–12, 11 2018.

[109] Alexander Zook and Mark Riedl. A temporal data-driven player model for dynamic difficulty adjustment. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 8(1):93–98, Jun. 2021.