UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DENYSON JURGEN MENDES GRELLERT

Análise sobre proteção contra trapaças em jogos online com o uso de inteligência artificial

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Dante Augusto Couto Barone
Co-orientador: Dr. Leonardo Filipe Batista Silva de Carvalho

Porto Alegre
2023

“I never said it would be easy,
I only said it would be worth it.”
— Mae West

AGRADECIMENTOS

Gostaria de agradecer ao meu orientador Dante Augusto Couto Barone por aceitar a minha proposta de trabalho, me aconselhar e guiar nas diversas dificuldades encontradas durante o desenvolvimento. Também gostaria de agradecer ao meu co-orientador Leonardo Filipe Batista Silva de Carvalho por estar sempre presente, pelo suporte e ajuda, sem as quais o trabalho não teria avançado como deveria.

Gostaria de agradecer a minha namorada, Ketlen Pereira, que durante todo esse tempo foi paciente, esteve sempre presente me apoiando e encorajando. Também gostaria de agradecer aos meus pais Delmo e Evelyn, e minha irmã Deborah pelo suporte e carinho sempre presente. Por fim gostaria de agradecer aos meus amigos que sempre estiveram comigo e que foram essenciais nesta minha caminhada, em especial ao Daniel Trindade e Carlos Henrique Cardoso que me ajudaram a gerar os dados necessários para este trabalho.

RESUMO

As trapaças em jogos eletrônicos são um problema de longo tempo, quase tão velho quanto o próprio surgimento dos mesmos. Combater esse tipo de comportamento é benéfico para a comunidade de jogadores como um todo, assim como para a indústria bilionária dos jogos. O objetivo deste trabalho é desenvolver uma rede neural que seja capaz de detectar jogadores desonestos em jogos de tiro em primeira pessoa. Dentre esses, o jogo Counter Strike: Global Offensive foi escolhido para ser usado na geração dos dados necessários para o treinamento da rede neste trabalho. Para alcançar este objetivo, foi escolhido usar uma rede neural recorrente implementada em *Python* com a biblioteca *Keras*. O trabalho também pretende analisar o desempenho da rede considerando a precisão dos acertos e o número de falsos positivos como métricas. Seis modelos com topologias diferentes foram criados e avaliados, onde o melhor modelo obteve uma precisão de 63% no conjunto de treinamento e 53% nos testes.

Palavras-chave: Trapaças em jogos. Jogos eletrônicos. Anti-trapaças. Inteligência Artificial.

Analysis of anti-cheat in online games using artificial intelligence

ABSTRACT

Cheating in video games is an old problem, so old as the creation of video games itself. Fighting this behavior is beneficial for the whole gamer community, as well as to the billionaire industry of games. The main objective of this work is to develop a neural network that is capable of detecting unfair players in first person shooters games, in which the game Counter-Strike: Global Offensive was chosen to be used on needed data generation for the network training on this work. To achieve this objective, a recurrent neural network was chosen to be implemented in Python with the Keras library. This work also aims to assess network performance through the utilization of precision and false positive metrics. Six models with different topologies were created and evaluated, where the best model obtained an accuracy of 63% in the training set and 53% in the tests.

Keywords: Cheats on games. Videogames. Anti-cheats. Artificial Intelligence.

LISTA DE FIGURAS

Figura 2.1 Exemplos de curvas sub-ajustadas, bem ajustadas e sobre-ajustadas, respectivamente 15
Figura 2.2 Ilustração de uma possível disposição de nodos em camadas em uma rede neural artificial 16
Figura 2.3 Ilustração da computação de um nodo, elemento básico da rede neural 17
Figura 2.4 Grafo de uma rede neural recorrente 18
Figura 2.5 Um nodo LSTM desdobrado 19

Figura 3.1 Taxonomia de arquitetura de jogos por categoria 21
Figura 3.2 Captura de um frame do jogo Wolfstein 22
Figura 3.3 Captura de um frame do jogo Battlefield 2042 23
Figura 3.4 Captura de um frame durante uma partida de CS:GO 24

Figura 4.1 Exemplo de *hard cheat* com *aimbot* incluído 26

Figura 5.1 O layout típico da memória de uma aplicação com destaque na seção do código 29
Figura 5.2 O logo do programa *Easy Anti-Cheat* 31

Figura 7.1 Eixos de rotação para movimentação do jogador no jogo 35
Figura 7.2 Estrutura de uma linha do *log* gerado pelo *script* 35
Figura 7.3 Exemplo de *log* gerado pelo *script* 35
Figura 7.4 Topologia do modelo com janela de tamanho cinco e 100 nodos 37
Figura 7.5 Topologia do modelo com janela de tamanho dez e 100 nodos 37
Figura 7.6 Topologia do modelo com janela de tamanho 20 e 100 nodos 38

Figura 8.1 Gráfico de precisão do modelo 1 40
Figura 8.2 Gráfico de precisão do modelo 2 40
Figura 8.3 Gráfico de precisão do modelo 3 41
Figura 8.4 Topologia do modelo com janela de tamanho cinco e 20 nodos 41
Figura 8.5 Topologia do modelo com janela de tamanho cinco e 50 nodos 42
Figura 8.6 Topologia do modelo com janela de tamanho cinco e 100 nodos em duas camadas 43

LISTA DE TABELAS

Tabela 7.1 Versões 33

Tabela 8.1 Resultados das amostras de teste 39
Tabela 8.2 Resultados das amostras de teste para os modelos com 100 nodos 42
Tabela 8.3 Resultados das amostras de teste por jogador 44

LISTA DE ABREVIATURAS E SIGLAS

ADAM	Adaptative Moment Estimation
ARPG	Action Role Playing Game
CS:GO	Counter Strike: Global Offensive
DLL	Dynamic-Link Library
DNN	Deep Neural Network
FPS	First Person Shooter
IA	Inteligência Artificial
LSTM	Long Short Term Memory
ML	Machine Learning
MMO	Massive Multiplayer Online
MOBA	Multiplayer Online Battle Arena
P2P	Peer to Peer
RAM	Random Access Memory
RNA	Rede Neural Artificial
RNN	Recurrent Neural Network
RNR	Rede Neural Recorrente
RPG	Role Playing Game
WSL	Windows Subsystem for Linux

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Objetivos 12
1.2 Organização do Trabalho 12
2 INTELIGÊNCIA ARTIFICIAL 14
2.1 Aprendizado de Máquina 14
2.2 Aprendizado Supervisionado 15
2.3 Redes Neurais Artificiais 16
2.4 Redes Neurais Recorrentes 17
3 JOGOS ELETRÔNICOS 20
3.1 Conceitos 20
3.2 Gêneros 21
3.3 Estudo de caso, Counter Strike: Global Offensive 23
4 TRAPAÇAS EM JOGOS ELETRÔNICOS 25
4.1 Classificação de trapaças 25
4.2 Trapaças de uso comum 26
5 ANTI-CHEAT 28
5.1 Classificação 28
5.2 Estado da Arte 30
5.2.1 VAC 30
5.2.2 EAC 30
5.2.3 FairFight 31
6 TRABALHOS RELACIONADOS 32
7 DESENVOLVIMENTO 33
7.1 Ambiente de desenvolvimento 33
7.2 Coleta de Dados 34
7.3 Construção do Modelo de Rede Neural 36
7.4 Treinamento 37
8 RESULTADOS 39
8.1 Modelos de Rede Neural 39
9 CONCLUSÃO E TRABALHOS FUTUROS 46
REFERÊNCIAS 48
APÊNDICE A — CÓDIGOS FONTE 53

1 INTRODUÇÃO

A indústria de jogos tem atingido números de acessos diário e faturamento exorbitantes nos últimos anos. Os jogos tem feito, cada dia mais, parte do cotidiano de um número crescente de pessoas ao redor do globo. Os jogos atingem recordes de acesso diário (CHARTS, 2023) nos últimos anos. E com tantos acessos, o faturamento dessa indústria também atinge valores bilionários todos anos. Tendo a empresa *Activision Blizzard* reportado um faturamento de 7.5 e 8.8 bilhões de dólares nos anos de 2022 e 2021, respectivamente (BLIZZARD, 2022). Outra forma de movimentar essa economia acontece através da competição entre os jogadores, com prêmios para os vencedores de torneios e até mesmo apostas nos campeonatos.

Até mesmo no campo da pesquisa os jogos se mostram populares. Muitas áreas diferentes estudam os jogos eletrônicos por diversos pontos de vista. Pesquisas como a de Giuseppe (2018) e de Bitencourt (2017) aparecem no campo da música, mostrando a diversidade que os jogos possuem. Trabalhos na área da comunicação (PAZ, 2022) (NASCIMENTO, 2023), educação (SILVA, 2021), linguagens (DERETTI, 2017) (DUARTE, 2017), economia (BRAZIL, 2021) e computação (PITTOL, 2019) (FREITAS, 2021) mostram os avanços em diversos aspectos que os jogos podem trazer e evidenciam o interesse da comunidade acadêmica sobre os jogos e seus impactos na sociedade.

Com tamanha popularidade e dinheiro movimentado, é possível esperar que algumas pessoas tentem trapacear para obter vantagens nos jogos. As trapaças em jogos podem ser definidas como o uso não planejado, pelos desenvolvedores, de mecânicas dos jogos, uso de falhas, ou auxílio de programas externos nos mesmos para obter vantagens sobre outros jogadores. Uma visão mais aprofundada sobre o tema será apresentada no capítulo 4. Apesar de existir uma dificuldade maior em trapacear em campeonatos presenciais, esses jogadores desonestos estão muito presentes nos jogos onlines, como na comunidade *UnknownCheats* (UNKOWNCHEATS, 2023). Mesmo que as trapaças sejam uma prática proibida pelas empresas que publicam os jogos, é preciso monitorar os jogos com algoritmos para detecção de trapaças com a finalidade de garantir que a experiência dos jogadores honestos não será arruinada pelos trapaceiros.

Detectar e punir os jogadores desonestos tem se mostrado uma tarefa não trivial ao longo dos anos. Mesmo décadas após o lançamento do primeiro programa para proteção contra trapaças ser lançado, *PunkBuster* lançando em 2000 para jogos de FPS (LEHTONEN, 2020), os jogos ainda precisam lutar constantemente contra esse tipo de jogador.

Uma das trapaças que tem se destacado em jogos de tiro é chamada de *aimbot*. Os jogadores que utilizam um *aimbot* fazem o uso de algum programa externo ao jogo que move o cursor do mouse automaticamente, facilitando mirar em um oponente visto que o tempo de resposta de um programa é muito menor do que o reflexo de uma pessoa. Carter (2022) publicou um vídeo onde ele construiu um robô ao redor do mouse e o treinou para reconhecer alvos, mirar e atirar automaticamente. Kanervisto (2022) publicou um artigo onde uma inteligência artificial foi treinada para imitar o comportamento humano ao mirar, ao mesmo tempo em que melhora a performance do jogador e se mantém indetectável. Contudo são poucas as pesquisas no campo com o objetivo de combater essas trapaças.

1.1 Objetivos

Dado a motivação apresentada, o objetivo desse trabalho foi desenvolver um programa que, a partir do uso de inteligência artificial, seja capaz de detectar jogadores que utilizam a trapaça de *aimbot* em jogos de tiro. Portanto a contribuição do trabalho pode ser sumarizada em:

- Gerar dados para treinar e testar uma inteligência artificial na detecção de jogadores que utilizam *aimbot*
- Criar um modelo genérico de detecção de trapaças do tipo *aimbot* em jogos de tiro de primeira pessoa
- Treinar a inteligência artificial com diferentes parâmetros
- Avaliar o desempenho do sistema de acordo com os parâmetros utilizados
- Fornecer uma abordagem diferente daquelas das pesquisas atuais para contribuir com a pesquisa de proteção contra trapaças de *aimbot*

1.2 Organização do Trabalho

O capítulo 2 introduzirá conceitos pertinentes ao trabalho no campo da Inteligência Artificial. No capítulo 3 serão introduzidos alguns conceitos de jogos eletrônicos, assim como será apresentado o jogo escolhido para gerar os dados que servirão de treinamento e teste da rede neural. No capítulo 4 serão apresentados os conceitos de trapaças

e em seguida, no capítulo 5, os conceitos e o estado da arte dos programas de proteção contra trapaças. No capítulo 6 serão apresentadas pesquisas relacionadas ao tema deste trabalho. O capítulo 7 apresentará o desenvolvimento e a metodologia utilizada. Por fim a apresentação dos resultados será realizada no capítulo 8, enquanto que no capítulo 9 serão discutidas as conclusões acerca do trabalho.

2 INTELIGÊNCIA ARTIFICIAL

Desde a primeira aparição do termo, a Inteligência Artificial, ou simplesmente IA, começou a ser alvo maior de pesquisas na década de 50, pelo time de pesquisa do Instituto de Tecnologia de Massachusetts. Desde então, a área da IA se desenvolveu muito e o próprio termo ganhou definições um pouco diferentes da definição inicial (CAGLE, 2019). Da década de 50 até os anos atuais, a pesquisa na área já passou pelo que foi chamado de inverno da IA, onde se viu uma falta de evolução ou até mesmo diminuição de pesquisas na área, período no qual se destacam principalmente na década de 80 (HENDLER, 2008) e anos anteriores (CAGLE, 2019). Reconhecidamente, parte do problema era a falta de poder computacional para processar os algoritmos que estavam sendo propostos. Contudo nos últimos anos temos visto o que pode ser chamado de explosão da IA, com inúmeras pesquisas publicadas, e também com várias aplicações comerciais sendo lançadas, como o ChatGPT (OPENAI, 2022), um modelo de processamento de linguagem natural lançado em 2022 que se tornou muito popular.

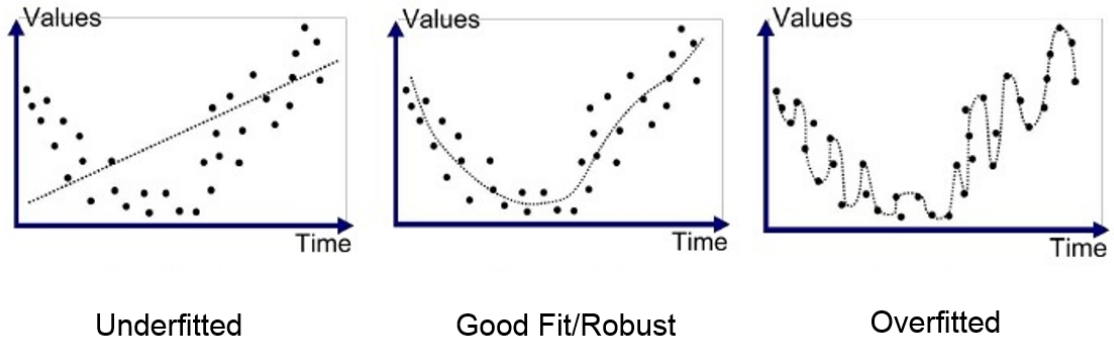
2.1 Aprendizado de Máquina

O aprendizado de máquina, chamado de ML ou *Machine Learning* em inglês, é uma subárea da IA com o objetivo de estudar algoritmos de aprendizado. O aprendizado na forma computacional nada mais é do que a experiência convertida em dados. Quando o algoritmo é alimentado com a experiência em dados, o modelo pode fazer previsões sobre novas observações (ZHOU, 2021). Esta subárea tem diversas aplicações, como por exemplo a classificação do risco de motoristas ao contratarem um plano de seguro. No caso, uma seguradora pode utilizar um modelo previamente treinado para classificar um novo cliente e entender melhor o quanto precisa cobrar desse cliente para assegurar o seu lucro.

Um dos problemas encontrados de forma recorrente nessa área é o que chamamos de *overfitting* e *underfitting*, traduzindo em português podemos chamar de sobre-ajustado e sub-ajustado, respectivamente. O primeiro problema ocorre quando o modelo aprende demais sobre os dados de forma que só consegue obter bons resultados para os dados de treino. Seria "como se o modelo tivesse apenas decorado os dados de treino e não fosse capaz de generalizar para outros dados nunca vistos antes."(BRANCO, 2023). O segundo problema ocorre quando o modelo não aprende o suficiente sobre os dados apresentados,

de forma que ele também não consegue gerar resultados satisfatórios, tanto nos dados de treino quanto nos dados de teste. Na Figura 2.1 podemos ver um exemplo de *underfitting* e *overfitting* representados em gráficos.

Figura 2.1 – Exemplos de curvas sub-ajustadas, bem ajustadas e sobre-ajustadas, respectivamente



Fonte: (BRANCO, 2023)

Os algoritmos de aprendizado de máquina podem ser separados em três classificações diferentes. São elas o Aprendizado Supervisionado, Aprendizado Não-Supervisionado e o Aprendizado por Reforço. O primeiro é utilizado em circunstâncias onde os dados são previamente separados ou classificados de acordo com o resultado esperado. No aprendizado não-supervisionado os dados não possuem uma classificação prévia e é objetivo do algoritmo encontrar semelhanças e padrões. Por fim os algoritmos de aprendizado por reforço tem o objetivo de resolver problemas em que um agente precisa aprender um comportamento por tentativa e erro a partir do ambiente em que está inserido (WILLMAN, 2020). Visto que o problema tratado neste trabalho pode ser visto como um problema de classificação de jogadores, com dados conhecidos e, portanto, previamente classificados, a categoria de aprendizado supervisionado foi escolhida como abordagem.

2.2 Aprendizado Supervisionado

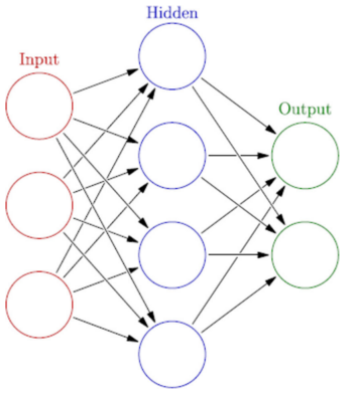
O aprendizado supervisionado é um algoritmo que pode ser usado quando se tem conhecimento do resultado esperado para um grande número de amostras. Dessa forma o algoritmo pode aprender com os dados de maneira supervisionada, visto que os resultados esperados no treinamento já são conhecidos. Um exemplo de uso desse algoritmo é a classificação de imagens, em que é possível treinar um modelo para reconhecer um objeto predeterminado nas imagens que forem apresentadas a ele (WILLMAN, 2020). Um

modelo de aprendizado supervisionado pode usar diferentes algoritmos, como as Redes Neurais Artificiais cujos conceitos e implementações são abordados na próxima seção. O algoritmo de Redes Neurais Artificiais foi escolhido como abordagem neste trabalho devido ao levantamento do referencial teórico feito, onde o mesmo se mostrou uma abordagem comum de outros trabalhos.

2.3 Redes Neurais Artificiais

Com o avanço da pesquisa na área da Inteligência Artificial, não se pode deixar de comparar a IA com o cérebro humano. A partir do modelo do cérebro humano, foram criadas as Redes Neuras Artificiais. Uma rede neural artificial baseia seu funcionamento naquele de um cérebro humano, sendo formada por vários nodos que estão conectados mutualmente e que podem ser comparados a neurônios (WU; FENG, 2017). Esses nodos seguem uma organização em camadas, de forma que a saída resultante de um nodo pode ser alimentada como entrada para os nodos da próxima camada, como mostra a Figura 2.2.

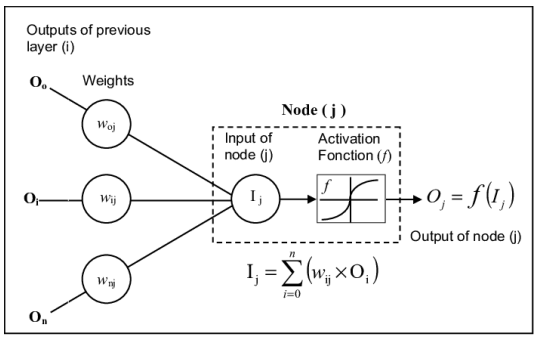
Figura 2.2 – Ilustração de uma possível disposição de nodos em camadas em uma rede neural artificial



Fonte: (WU; FENG, 2017)

Formalmente, cada nodo (j) da rede neural recebe como dados de entrada (I_j) as saídas dos nodos da camada anterior (O_i) multiplicados pelo seu peso (w_{ij}), como mostra a Equação 2.1. A entrada (I_j) é então passada para uma função de ativação (f) para produzir a saída do nodo (O_j). A função de ativação pode ser uma sigmoide, tangente hiperbólica ou a Unidade Retificada Linear (GHEDIRA; BERNIER, 2004). A Figura 2.3 ilustra um nodo básico de uma rede neural de acordo com a descrição apresentada.

Figura 2.3 – Ilustração da computação de um nodo, elemento básico da rede neural



Fonte: (GHEDIRA; BERNIER, 2004)

$$I_j = \sum_{i=0}^{n} (w_{ij} \times O_i) \tag{2.1}$$

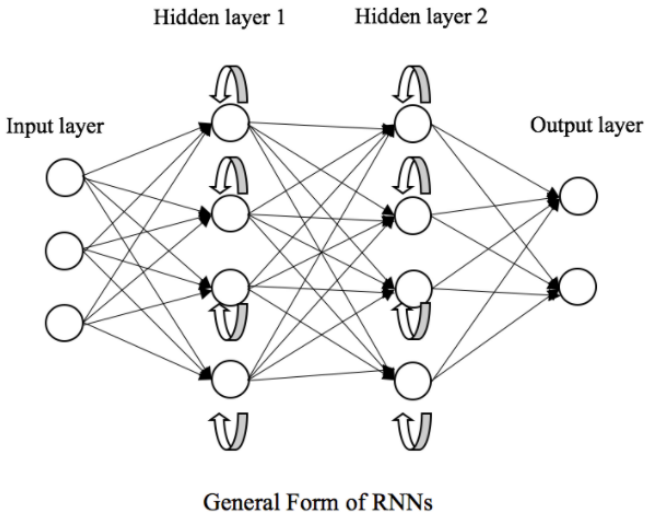
Já o que se chama de treinamento de uma rede neural consiste em ajustar cada um dos pesos w_{ij}, que geralmente são iniciados com valores aleatórios, de forma a tentar minimizar o erro na saída. Existem diversas funções otimizadoras que podem ser escolhidas para tal processo. Uma das primeiras funções utilizadas nas pesquisas se chama gradiente descendente. Outras funções foram estudadas e desenvolvidas com o tempo. Uma função chamada Adam foi apresentada em 2015. O nome é derivado do termo *Adaptative moment estimation* e "utiliza estimativas do primeiro e segundo momentos do gradiente para adaptar a taxa de aprendizagem para cada peso da rede neural"(BUSHAEV, 2018). Contudo o nodo, como apresentado anteriormente, não tem visão de observações passadas, e não mantém histórico algum dos outros registros. Por isso, esse algoritmo não é recomendado para predições em dados que possuem uma série temporal ou sequenciais (DANCKER, 2022). Para esse propósito, foram criados algoritmos chamados de Redes Neurais Recorrentes ou RNN (*Recurrent Neural Network*), que serão abordados a seguir.

2.4 Redes Neurais Recorrentes

Nas redes neurais tradicionais a informação se move em um único sentido, da entrada para a saída. Logo, o algoritmo não possui informação sobre estados anteriores. A RNN tem o objetivo de resolver este problema. A parte recorrente do algoritmo RNN é o que o diferencia dos algoritmos tradicionais e torna possível a obtenção de informações sobre estados anteriores. Isto porque, do ponto de vista da informação, a direção do fluxo

não é mais única, visto que agora existem *loops* de *feedback* nos nodos que lhes concedem a habilidade de memorizar um estado anterior (LIU, 2020). O fluxo da informação em uma RNN pode ser visualizado na Figura 2.4.

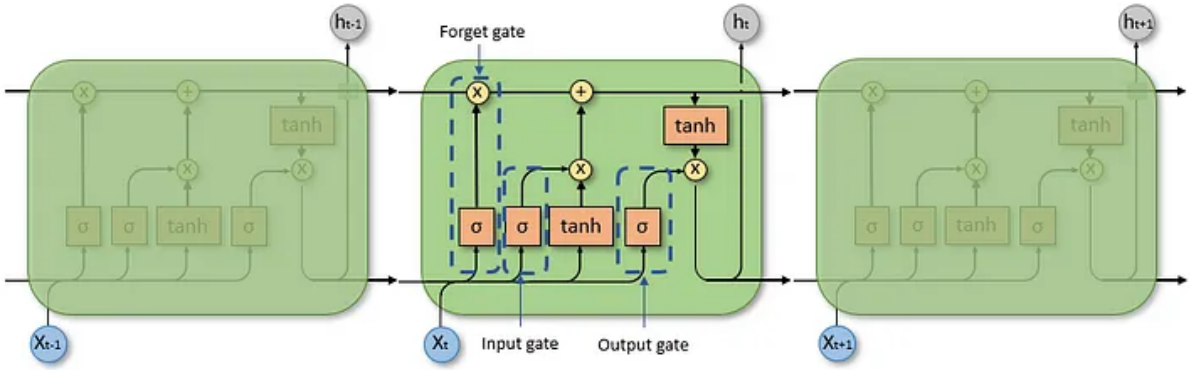
Figura 2.4 – Grafo de uma rede neural recorrente



Fonte: (LIU, 2020)

Em uma RNN, para que o nodo possa receber um *feedback* de um estado anterior é necessário que o nodo em questão seja ajustado. Uma abordagem de RNN que se propõe a isso é o LSTM (*Long-Short-Term-Memory*) que utiliza o conceito de portões ou torneiras para controlar as informações. Em particular, o algoritmo estabelece que há três portões que funcionam como filtros, determinando qual informação deve ser mantida ou descartada. Estes são os chamados portões do esquecimento, de entrada e de saída (DANCKER, 2022). Na Figura 2.5 podemos ver um nodo LSTM desdobrado em t-1, t e t+1.

Figura 2.5 – Um nodo LSTM desdobrado



Fonte: (DANCKER, 2022)

3 JOGOS ELETRÔNICOS

Neste capítulo é apresentada uma breve introdução de conceitos de jogos que são relevantes para o entendimento deste trabalho.

3.1 Conceitos

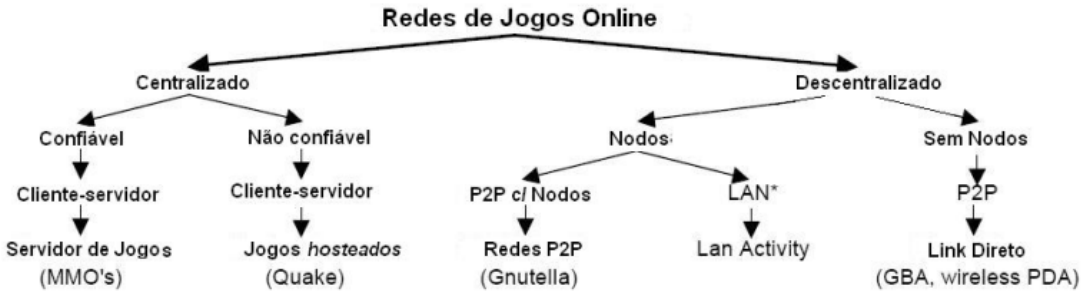
Um jogo eletrônico, ou videogame, é um software com elementos pensados para entreter o público alvo de forma interativa. Geralmente os jogos utilizam elementos que criam uma atmosfera artificial a partir de recursos gráficos, sonoros, narrativos e de um conjunto de desafios (SCHUYTEMA, 2008).

Os jogos podem ter diferentes números de jogadores. Jogos para apenas um jogador, são chamados de *singleplayer* em inglês, enquanto que para vários jogadores são chamados de *multiplayer* em inglês. Além disso, os jogos podem ter também uma quantidade massiva de jogadores, os jogos chamados de MMO, *massive multiplayer online* (CASTRO, 2019). Apesar de jogos *singleplayer* também sofrerem com jogadores que recorrem ao uso de trapaças, as mesmas causam dano mínimo para a experiência de jogo, com consequências apenas para o próprio jogador. Jogos *multiplayer*, em que mais de uma pessoa joga ao mesmo tempo, são mais suscetíveis a trapaças, visto que um jogador pode querer obter vantagens sobre os outros. Logo, para os jogadores honestos a experiência pode ser arruinada por aqueles que escolheram utilizar algum *cheat* [1], trapaça em português, como forma de obter vantagens.

Outro aspecto a observar, além da quantidade de jogadores, é a arquitetura sob a qual o jogo foi construído. Nos jogos onlines uma arquitetura que pode ser utilizada é a de cliente-servidor. Neste tipo de arquitetura, um componente do sistema (servidor) atende as requisições dos jogadores, em seus dispositivos (clientes). No modelo cliente servidor, dispositivos clientes não possuem comunicação direta, logo as informações de outros jogadores só são conhecidas quando o servidor permite. Nesta arquitetura, o servidor pode ser confiável, hospedado pela empresa produtora do jogo, ou não confiável, comumente com servidores hospedados por um jogador. Outra arquitetura bastante utilizada é a chamada P2P, *Peer-to-Peer*, no qual não existe um controle centralizado, com todos os jogadores podendo ser considerados um servidor e trocando informações de forma direta. Essas arquiteturas estão demonstradas na Figura 3.1 com mais detalhes e alguns exemplos

[1] Obter vantagem de forma ilegal em jogos eletrônicos, aprofundado no capítulo 4

Figura 3.1 – Taxonomia de arquitetura de jogos por categoria



Fonte: (GASPARETO, 2008)

de jogos que as utilizam. Com o tempo e o crescimento do interesse por jogos competitivos, o modelo cliente-servidor foi crescendo em detrimento dos demais, visto a maior confiabilidade que oferece e, como será demonstrado na seção 5.1, por permitir a proteção contra trapaças nos dois lados do sistema. Possibilidade que não se pode alcançar, por exemplo, na arquitetura P2P, que permite o software de proteção apenas no lado do cliente, que pode ser manipulado.

3.2 Gêneros

Os jogos podem ser classificados de diversas maneiras considerando os diferentes aspectos possíveis de serem introduzidos nos mesmos. Uma forma muito comum de classificar jogos é baseado no gênero do jogo, usando elementos que estão presentes dentro do mesmo, como mecânicas de jogos que se assemelham. A classificação por gênero, ainda que muito utilizada e útil para os usuários, não traz muitas informações para uma análise científica. O fato dos gêneros não serem mutuamente exclusivos também torna essa classificação difícil. Um exemplo disso é o subgênero de RPG de ação, ARPG, em que os jogos tem se mostrado cada vez mais próximos do gênero MMORPG sem haver definições muito claras do que os diferencia. Contudo, a classificação por gênero ainda é uma das mais utilizadas e alguns gêneros principais possuem elementos bem definidos (SILVA, 2022).

Uma das categorias que possui elementos bem definidos é a de jogos de Tiro em Primeira Pessoa, ou FPS (*First Person Shooter*). Os jogos de FPS são desenvolvidos de forma que o jogador não vê o corpo do personagem que controla, mas sim a perspectiva do próprio personagem, perspectiva essa chamada de "primeira pessoa". O gênero surgiu

primeiramente em jogos *singleplayer* em que o objetivo era atirar em alvos, geralmente NPC's, *non-playable-character*, i.e., personagens não controlados por pessoas. Posteriormente com o surgimento de jogos online o gênero evoluiu para também abranger o modelo *multiplayer* no qual em geral há partidas em que há o confronto de um grupo de jogadores contra outro (HITCHENS, 2011). Um dos primeiros jogos do gênero foi o jogo *Wolfstein 3D*, com um exemplo na Figura 3.2. Outro jogo mais recente, *Battlefield 2042*, lançado em 2021, conta com o modo *multiplayer*, exemplificado na Figura 3.3

Figura 3.2 – Captura de um frame do jogo Wolfstein

Fonte: (SOFTWARE, 1992)

Entre os gêneros mais jogados estão MOBA (*Multiplayer Online Battle Arena*), Battle Royale e FPS. Estes gêneros possuem modos competitivos em que os jogadores ganham pontos a cada partida vencida ou de acordo com a classificação, no caso do Battle Royale. Um dos exemplos da popularidade de alguns jogos desses gêneros pode ser visto através dos diversos campeonatos organizados, tanto pela comunidade de jogadores, como campeonatos oficiais organizados pelas próprias desenvolvedoras dos jogos. Comumente esses campeonatos oferecem prêmios, que podem variar desde conteúdo dentro do jogo, por exemplo itens, texturas e outros, até milhares de dólares, onde muitas vezes o dinheiro vem de patrocinadores interessados na popularidade que a quantidade de jogadores pode oferecer.

Dentre os jogos com campeonatos com premiações em dinheiro, aquele que mais

Figura 3.3 – Captura de um frame do jogo Battlefield 2042

Fonte: (KRSTIć, 2023)

movimentou dinheiro em campeonatos foi Dota2 (de tipo MOBA) com um total de 335 milhões de dólares desde que foi lançado em 2013, seguido por Fortnite (de tipo Battle Royale) com 163 milhões de dólares, desde 2017, e CS:GO (de tipo FPS) com 157 milhões de dólares, desde 2012 (EARNINGS, 2023c). O país de origem dos jogadores que mais ganharam campeonatos é a China, com um total de 253 milhões de dólares em dinheiro arrecadado em prêmios. O Brasil aparece na lista em sexto lugar com um total de 51 milhões de dólares ganhos em competições (EARNINGS, 2023b). Importante notar que esses dados provenientes do site *Esports Earnings* se limitam a dados públicos e de campeonatos, desconsiderando o dinheiro que os jogos movimentam internamente, com a venda de conteúdo dentro do jogo, como as *skins* (texturas especiais para armas ou personagens), ou o dinheiro movimentado por organizações que contratam jogadores, sites de apostas, entre outros.

3.3 Estudo de caso, Counter Strike: Global Offensive

O jogo Counter Strike: Global Offensive, conhecido também pela sigla CS:GO, é um jogo da categoria FPS. Lançado em 21 de agosto de 2012 (STEAM, 2023a), com onze anos de mercado o jogo chegou ao seu maior pico de jogadores este ano no mês de maio, atingindo cerca de um milhão e oitocentos mil jogadores simultâneos (CHARTS,

2023). Ainda, em 2023 o jogo manteve média mensal de acessos simultâneos sempre superior a setecentos mil jogadores, sendo este um dos jogos mais populares do mundo, principalmente em sua categoria.

O jogo conta com diversos modos, mas o mais jogado em torneios é o modo em que os jogadores são separados em dois grupos onde um grupo é encarregado de plantar uma bomba, enquanto o outro grupo é encarregado de impedir este acontecimento ou, depois de armada, desarmar a bomba. O primeiro grupo é chamado "terrorista"enquanto o segundo grupo é nomeado de "contra-terrorista". O jogo conta com diferentes mapas. Em uma partida, um mapa é escolhido e então os jogadores começam em um dos grupos. Após um número de rodadas definido os times trocam de papéis, o que estava encarregado de armar a bomba agora deve impedir o outro de armar a mesma. Na Figura 3.4 podemos ver uma captura de tela durante uma partida.

Figura 3.4 – Captura de um frame durante uma partida de CS:GO

Fonte: Imagem capturada pelo autor

Historicamente CS:GO é um jogo com vários casos de jogadores banidos pelo uso de trapaças, inclusive em campeonatos oficiais (LAHTI, 2014) (LIMA, 2018). Com mais de 6500 torneios e 157 milhões de dólares em prêmios (EARNINGS, 2023a) desde que foi lançado, pode-se entender a preocupação para combater o uso de trapaças que podem alterar o destino das premiações e apostas. Dado a relevância do jogo, facilidade em coletar dados e certa experiência do autor com o mesmo, o jogo foi escolhido como exemplo para teste do algoritmo construído neste trabalho.

4 TRAPAÇAS EM JOGOS ELETRÔNICOS

As trapaças, também chamadas de *cheats* em inglês, são um problema antigo dos jogos e surgiram praticamente com a criação dos mesmos. Uma trapaça em jogo pode ser definida como qualquer ação ou comportamento que visa obter uma vantagem injusta ou desleal, comprometendo a integridade, equilíbrio ou a experiência dos outros jogadores. Normalmente todo tipo de trapaça vai contra as regras e termos dos jogos sendo passível de banimento ou outras penalidades. Contudo nem sempre os jogadores que usam trapaças são identificados pelos mecanismos de proteção e, portanto, durante o tempo que os mesmos estão trapaceando outros jogadores são lesados, podendo, em alguns casos, abandonar o jogo por conta disso.

4.1 Classificação de trapaças

Assim como os jogos, as trapaças podem ser classificadas de diferentes formas. Uma forma de classificação de trapaças proposta por Lehtonen (2020) separa as mesmas em *soft* e *hard cheat*. *Soft cheats* são métodos de trapaças que se utilizam das próprias mecânicas do jogo, burlando as regras do mesmo como forma de conseguir vantagem sobre os outros jogadores. No geral isso se faz explorando falhas do jogo ou mesmo utilizando alguma de suas mecânicas de forma inesperada, que não foi prevista pelos desenvolvedores, e que pode ser considerada injusta com outros jogadores. Esse tipo de trapaça não é facilmente capturado pelos *anti-cheats*, visto que não utilizam programas externos, mas sim o próprio jogo. Portanto, o mais comum nesses casos é alterar o jogo, consertando as falhas ou alterando as mecânicas, depois de serem reportadas. Já os *hard cheats* são métodos de trapaças que no geral envolvem algum programa externo. As formas de explorar as brechas com ferramentas externas variam bastante.

Os *hard cheats* podem ser separados em grupos menores com características em comum. Lehtonen (2020) cita quatro grupos, sendo eles os *bots* ou automações, aqueles que utilizam dados secretos, os que modificam pacotes do jogo e os de *spoofing*, em que o atacante se passa por outro jogador. O tipo de trapaça abordada neste trabalho se enquadra no grupo de automações, visto que é um programa que modifica a entrada do jogador, em específico o mouse, automatizando parte do processo de mirar no oponente. A Figura 4.1 mostra um exemplo de um *hard cheat* disponível para o jogo Valorant, outro jogo do gênero FPS.

Figura 4.1 – Exemplo de *hard cheat* com *aimbot* incluído

Fonte: (MO1RA, 2021)

4.2 Trapaças de uso comum

Como discutido na seção 3.1, jogos online com vários jogadores simultâneos são mais suscetíveis a trapaças. Segundo Reeves (2009), jogos de FPS como *Counter Strike* exigem diversas habilidades para que o jogador atinja altos níveis competitivos, de forma que masterizar um jogo assim é um desafio que muitos podem não conseguir, levando algumas dessas pessoas a recorrerem a programas de trapaças que os auxiliem.

Neste sentido, é de se esperar que os jogos mais populares sejam os com maior incidência de trapaceiros. Segundo o site Surfshark (SURFSHARK, 2021), Fortnite seria o jogo com mais *cheaters* no ano de 2021, seguido por *Overwatch* e CS:GO. Contudo a pesquisa foi feita com base em vídeos no *YouTube* por termos como *aimbot* e *wallhack* que são mais utilizados em jogos de FPS e Battle Royale. Logo, para resultados mais acurados seriam necessários dados diretamente das fontes de venda ou de download desses *cheats* ou das empresas dos jogos. Contudo não é do interesse dessas empresas compartilhar publicamente esses números, visto que, caso sejam altos, podem dar a impressão de que o jogo possui muitas falhas e levar os jogadores honestos a pararem de jogar.

Como forma de ter uma ideia geral, alguns sites de *cheats* fornecem os números de *downloads* das trapaças para cada jogo. O site Unknowncheats, por exemplo, tem uma lista de arquivos mais populares assim como uma lista de jogos e diferentes *cheats* ou

programas relacionados para cada jogo (UNKOWNCHEATS, 2023). Nestas listas, por exemplo, um programa de trapaça para CS:GO tem mais de 600 mil downloads, estando em terceiro lugar dos arquivos mais baixados na plataforma. Já o jogo CS:GO é o jogo com mais arquivos de trapaça disponível, contando com mais de três mil opções. O site também menciona que o número de downloads total já ultrapassa os 47 milhões de arquivos para os mais de 28 mil que se encontram disponíveis na plataforma, divididos ao longo de 138 categorias diferentes.

5 ANTI-CHEAT

Softwares anti-trapaça ou *anti-cheat* são projetados para detectar, prevenir e combater trapaças ou qualquer comportamento fraudulento. O objetivo deles é prover uma experiência de jogo justa para todos os jogadores, protegendo os jogadores honestos contra qualquer outro que tente ganhar alguma vantagem de forma injusta. Assim como os vírus de computador e os anti-vírus, ou mesmo as bactérias e o nosso sistema imunológico, o sistema de proteção dos jogos evoluiu para combater o atacante. O que leva o invasor a desenvolver novas habilidades para burlar essa evolução (LEHTONEN, 2020).

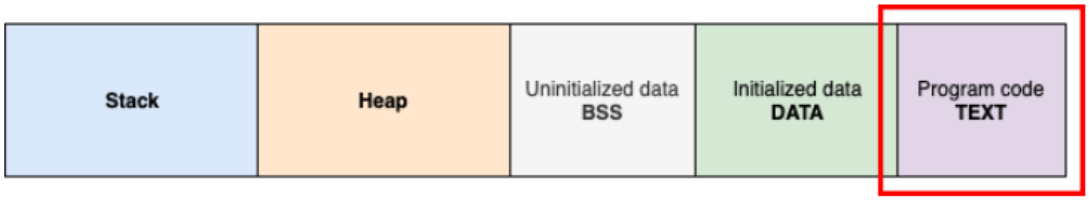
5.1 Classificação

Os programas de proteção dos jogos utilizam diversas técnicas para identificar e combater os atacantes. Essas técnicas podem ser classificadas de diversas formas. Uma classificação básica diferencia os programas de proteção voltados ao servidor ou voltados ao cliente (LEHTONEN, 2020) (SILVA, 2022) (KANERVISTO; KINNUNEN; HAUTAMÄKI, 2022) (KHALIFA, 2016). Os métodos anti-trapaça do lado de servidor costumam ser menos invasivos do que os métodos no lado do cliente (GASPARETO, 2008). Contudo nem todos os algoritmos de proteção contra trapaças instalados no cliente são invasivos.

Uma forma de proteção que pode ser utilizada é a verificação de arquivos por *hashing*, assegurando que os mesmos não foram alterados. A técnica de *hashing* que consiste em gerar um código único a partir do conteúdo do arquivo. Quando o programa de proteção do jogo começa a monitoração, é feita uma requisição ao servidor previamente configurado enviando o *hash* (código único) dos arquivos. O servidor verifica o *hash* e, caso o arquivo possua algum byte alterado, o *hash* será incompatível e, portanto, o servidor pode desconectar o cliente ou aplicar outra punição. Este método costuma ser eficiente contra trapaças que alteram o jogo visualmente, como as que permitem ao jogador ver através de paredes a partir da alteração dos arquivos de texturas das mesmas (LEHTONEN, 2020).

Outra técnica de *anti-cheat* é a utilização da encriptação do código. A Figura 5.1 mostra uma aplicação, que tipicamente carrega o seu código em memória. Caso o mesmo não esteja encriptado ele pode ser lido por outro programa e ser analisado em busca de falhas. Na técnica de encriptação, o código se torna incompreensível. Contudo para a

Figura 5.1 – O layout típico da memória de uma aplicação com destaque na seção do código



Fonte: (LEHTONEN, 2020)

execução do programa, o código precisa ser restaurado, por isso o cliente deve ter a chave de encriptação e desencriptação. O momento em que o código será desencriptado pode ser durante o seu carregamento em memória ou na hora da execução, quando o cliente deve contar com um ponto de entrada e com o descriptografador já carregado, sem encriptação. Uma das desvantagens desse método é o aumento do tempo de execução.

Por fim outros dois métodos utilizados para proteção no cliente são a detecção de trapaças conhecidas e a utilização de *drivers* a nível de *kernel*. O primeiro utiliza uma varredura na memória do cliente em busca de códigos de trapaças conhecidas. O segundo utiliza código a nível de *kernel* que possuí mais privilégios, podendo espionar requisições na área de memória do jogo. Dessa forma, ambos os métodos podem ser utilizados em conjunto para uma maior eficácia. Contudo, ambos os métodos são intrusivos e podem abrir novas brechas no cliente (LEHTONEN, 2020).

Os programas de proteção contra trapaças que rodam no servidor usam técnicas diferentes. Um método básico, mas em muitos casos bem efetivo, é chamado de "Não confie no cliente"e é baseado em verificar os dados que o cliente envia. Um exemplo desse método é não aceitar os resultados do cliente sem conferir se as entradas realmente geram aquele resultado.

Outro método muito utilizado é a análise estatística dos jogadores. Com base em parâmetros pré-estabelecidos, o software pode identificar clientes com dados discrepantes dos demais. A partir desse momento, os *softwares* podem ter abordagens diferentes. Uma abordagem é marcar estes clientes e fornecer evidências para análise humana, como o *Overwatch*, um programa do CS:GO, que fornece vídeos anonimizados, de forma que todas as informações que possam identificar qualquer jogador são retiradas do vídeo, para análise por outros jogadores. Quando o cliente é marcado como *cheater* pela grande maioria dos analistas, então as punições são aplicadas, como é explicado no site do programa (STRIKE, 2023). Outra abordagem é utilizar modelos estatísticos de forma a fazer todo

o processo de detecção automaticamente (KANERVISTO; KINNUNEN; HAUTAMÄKI, 2022). Em todas as abordagens, os *anti-cheats* baseados em métodos estatísticos respeitam a privacidade do usuário (KHALIFA, 2016).

5.2 Estado da Arte

Os softwares anti-cheat utilizados nos jogos em sua maioria não são de código aberto e as empresas tampouco liberam muitas informações sobre como funcionam, visto que isso poderia abrir brechas para atacantes. Contudo, existem informações superficiais sobre essas tecnologias e alguns dos programas que rodam no lado do cliente já foram analisados com engenharia reversa. A seguir serão abordados alguns dos principais softwares do mercado nessa área.

5.2.1 VAC

O software Valve Anti-Cheat (VAC) é um dos sistemas anti-trapaça mais conhecidos e utilizados. Desenvolvido pela empresa Valve, mesma empresa responsável pelo jogo CS:GO, mais de 100 jogos utilizam o sistema (STEAM, 2023b). Sites não oficiais divulgam que o número de banimento de contas pelo sistema em um único semestre pode ter chego perto de 200 mil (CARBONE, 2022). O sistema conta com um algoritmo de rede neural profunda, DNN, *Deep Neural Network* [1], além de métodos estatísticos e tradicionais (MCDONALD, 2018). O VAC tem um módulo instalado no cliente, assim como um que deve ser habilitado nos servidores dos jogos.

5.2.2 EAC

Easy Anti-Cheat é um software desenvolvido pela Kamu, mas que foi adquirido pela Epic Games. É o software utilizado em grandes jogos como Fortnite, Apex Legends e outros. O algoritmo do EAC usa métodos de monitoramento da memória, em conjunto com asserção de assinaturas dos *drivers* e estatística dos jogadores (KANERVISTO; KINNUNEN; HAUTAMÄKI, 2022) (SILVA, 2022). EAC é um software que

[1] "Uma DNN é uma RNA com múltiplas camadas ocultas que podem modelar relações não lineares e extrair novas características implícitas nos dados"(LOCA; RAUBER, 2019)

Figura 5.2 – O logo do programa *Easy Anti-Cheat*

Fonte: *Easy Anti-Cheat*

deve ser instalado no lado do cliente.

5.2.3 FairFight

FairFight é um software para proteção no lado do servidor. O seu algoritmo conta com métodos estatísticos de análise da performance do jogador. Com acesso a todos os dados do jogo, ele utiliza essas informações para detectar anomalias e caso alguma seja detectada, o caso é enviado para revisão. Contudo, o sistema se mostrou não muito efetivo em alguns casos (STANTON, 2016). O jogo *The Division*, que utiliza o *FairFight* como uma das formas de proteção, foi atacado por diversos jogadores desonestos, por tempo o suficiente para que muitos jogadores honestos desistissem de jogar. Importante notar que no caso deste jogo, houve a acusação do mesmo depositar muita confiança no cliente, guardando informações sensíveis, como a posição de todos jogadores, no cliente mesmo quando não seria necessário.

6 TRABALHOS RELACIONADOS

Algumas pesquisas na área de *anti-cheat* já foram feitas com abordagens diferentes. Em Willman (2020) uma Rede Neural Recorrente foi treinada para distinguir jogadores honestos de trapaceiros no jogo CS:GO. A pesquisa fez uso de trapaças de tipo *aimbot*, contudo os resultados não foram satisfatórios, com uma taxa de acerto de apenas 53%, o que o autor atribui a pequena quantidade de dados disponível para a rede neural.

Outra pesquisa interessante foi publicada em 2021 por um time da NVIDIA (JONNALAGADDA et al., 2021). Nesta pesquisa, uma rede neural profunda foi treinada com imagens de partidas de dois jogos. Na análise dos resultados foi percebido que o algoritmo marcou alguns quadros legítimos como trapaceiros, mas este comportamento da rede foi minimizado. Mesmo com bons resultados em certos cenários, vários casos não tiveram uma boa confiança, i.e., o algoritmo não marcou o quadro consistentemente na mesma categoria. Os quadros marcados como sem confiança obtiveram resultados piores, mas os quadros com um alto nível de confiança obtiveram bons resultados. O que não foi esclarecido pelo artigo é se os quadros não confiáveis são aleatórios ou se certos jogadores, partidas ou *cheats* tiveram seus quadros constantemente marcados como não confiáveis, o que faria esses jogadores não serem detectados.

Outra pesquisa na mesma área com uma abordagem diferente foi publicada por Silva (2022). A tese propõe uma análise comportamental baseada em estatísticas dos jogos no histórico do jogador. Silva utilizou uma base de dados pública do jogo CS:GO, disponibilizada por terceiros. Contudo a pesquisa não obteve resultados devido ao volume de dados, a falta de hardware e tempo para solucionar o problema.

Por fim uma outra pesquisa na área de *anti-cheat* foi desenvolvida por Gaspareto (2008). No seu trabalho foi criada e treinada uma rede neural para identificar outro tipo de trapaça, em que o jogador ultrapassa os limites de velocidade, chamada de *speed cheating*. Como resultado, duas redes foram treinadas, uma atingiu cerca de 60% de acertos, contudo levantou mais de 4% de falsos positivos, enquanto a rede que obteve somente 21% de acertos gerou menos de 1% de falsos positivos.

No geral, as pesquisas que utilizam inteligência artificial nesta área, em especial o aprendizado de máquina, mostram resultados positivos. Contudo, ainda há muito espaço para o incremento ou criação de novas abordagens para o problema.

7 DESENVOLVIMENTO

O presente capítulo apresenta a descrição da solução proposta para a proteção contra o uso de *aimbot* em jogos de FPS, utilizando o jogo CS:GO como exemplo. Para isto, são apresentados os ambientes de desenvolvimento do projeto, a forma de obtenção dos dados e uma descrição do modelo de redes neurais utilizado para a resolução do problema.

7.1 Ambiente de desenvolvimento

Com o objetivo de coletar os dados para treinamento e teste da rede neural, foi preciso definir um *script* para CS:GO de forma a capturar informações precisas e transcrevê-las em um arquivo. O *script* utilizado foi baseado no de Willman (2020), com algumas alterações. Este *script*, assim como todos aqueles voltados para a instalação em um servidor de CS:GO, precisa ser escrito em *SourcePawn*, que é derivado de *Pawn*, uma linguagem com sintaxe semelhante ao C.

Para o desenvolvimento da rede neural foi utilizado a linguagem *Python*, visto a vasta documentação disponível, em conjunto com as bibliotecas *Keras* e *Tensorflow* para a construção das redes neurais. As versões de cada linguagem e biblioteca utilizadas estão listadas na Tabela 7.1.

O computador que executou o treino dos modelos contém um processador *Ryzen 5 3600*, com seis núcleos físicos e 12 *threads*, uma placa de vídeo *RTX 3050* com oito *gigabytes* de memória dedicada GDDR5 e 16 *gigabytes* de memória RAM DDR4. O sistema operacional instalado no computador é o *Windows 11*, contudo para a biblioteca *Tensorflow* utilizar a placa de vídeo para os cálculos do treinamento da rede neural foi utilizado o Subsistema *Windows* para *Linux*, WSL, com a distribuição Ubuntu 22.04.3.

Tabela 7.1 – Versões

Nome	*Versão*
SourcePawn	1.7
Python	3.10.9
Keras	2.13.1
Tensorflow	2.13.0

Fonte: O Autor

7.2 Coleta de Dados

Foram coletados dados de três jogadores por 286 rodadas do jogo CS:GO. Para que a coleta de dados fosse possível, foi utilizado um computador, no qual foram instalados o *script* adaptado e o servidor de CS:GO. Como uma partida precisa de dez jogadores, os jogadores faltantes eram controlados pelo computador, chamados de *bots*. O *aimbot* utilizado foi obtido do site *UnknowCheats* para o jogo CS:GO (UNKOWNCHEATS, 2023). A escolha do *aimbot* foi precedida por diversas tentativas de uso de outras trapaças do mesmo tipo, mas que utilizavam o método de *DLL injection*, onde a proteção do *Windows* 11 contra o mesmo mostrou-se efetiva.

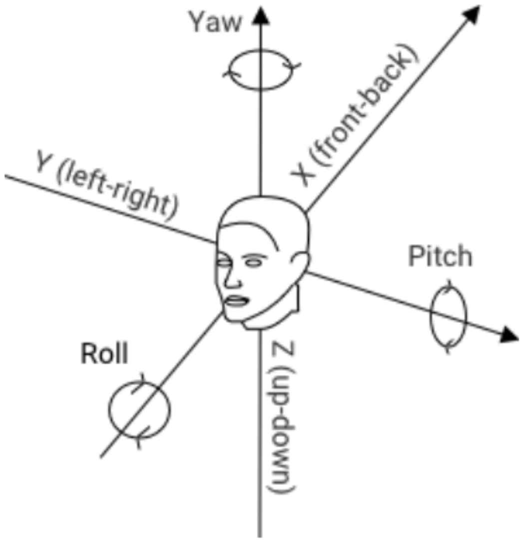
Como o jogo simula a visão de uma pessoa andando em um ambiente, o jogador pode se movimentar pelo mapa e olhar para diferentes ângulos. Visto que o *aimbot* altera os ângulos visando mirar em um oponente automaticamente, o foco está em capturar a alteração desses ângulos e em identificar quando a mira está sobre um oponente ou não. A Figura 7.1 mostra os ângulos existentes numa simulação, como a que foi sugerida, para um ambiente de três dimensões. Como o jogador controla apenas dois desses ângulos, o *Yaw* do eixo Z e o *Pitch* do eixo Y [1], estes serão os valores coletados. O ângulo *Roll* é sempre zero, e portanto pode ser excluído da coleta e análise. Dessa forma, a rede neural deve receber como entrada duas características, os ângulos *Yaw* e *Pitch*.

O servidor executa o *script* cerca de 64 vezes por segundo, de forma que se pode obter qualquer modificação nos ângulos do cliente a cada 16 milissegundos, desconsiderando os possíveis problemas de rede, como perda de pacotes. Na prática foram observadas algumas variações na quantidade de registros, geralmente não maior que 10%, o que se deve, possivelmente, pelo hardware limitado do servidor e/ou por problemas de conexão. Todos os registros do *log* contém a estrutura apresentada na Figura 7.2. O *log* registra a informação sobre se o cliente estava com a mira posicionada sobre um inimigo em "*AimingAtClient*", onde o número um representa o booleano verdadeiro, os ângulos dos eixos *Pitch* e *Yaw* e por fim o número do cliente e seu nome. Um exemplo desses registros pode ser verificado na Figura 7.3.

Para filtrar os dados de interesse dos arquivos de *logs* gerados a partir do *script*, foi criado o programa em *Python* disponível no Apêndice A, o qual recebe como parâmetro o número de sequências temporais que serão utilizadas para separar os dados. O Algoritmo 1 ilustra a forma de funcionamento do código.

[1] *Roll*, *Pitch* e *Yaw* é a nomenclatura de SourcePawn para os ângulos dos eixos X, Y e Z, respectivamente

Figura 7.1 – Eixos de rotação para movimentação do jogador no jogo



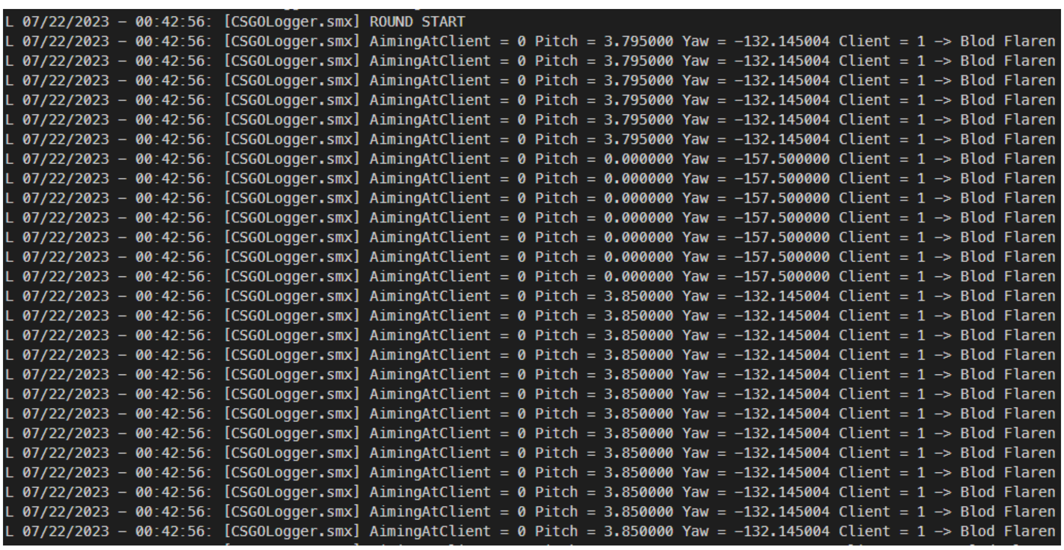
Fonte: (WILLMAN, 2020)

Figura 7.2 – Estrutura de uma linha do *log* gerado pelo *script*

L 08/15/2023 - 00:24:36: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 0.000000 Yaw = 0.000000 Client = 1 -> Player

Fonte: O Autor

Figura 7.3 – Exemplo de *log* gerado pelo *script*



```
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] ROUND START
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.795000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.795000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.795000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.795000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.795000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.795000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 0.000000 Yaw = -157.500000 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 0.000000 Yaw = -157.500000 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 0.000000 Yaw = -157.500000 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 0.000000 Yaw = -157.500000 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 0.000000 Yaw = -157.500000 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 0.000000 Yaw = -157.500000 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 0.000000 Yaw = -157.500000 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
L 07/22/2023 - 00:42:56: [CSGOLogger.smx] AimingAtClient = 0 Pitch = 3.850000 Yaw = -132.145004 Client = 1 -> Blod Flaren
```

Fonte: O Autor

Algoritmo 1: Filtrar os dados do log

```
Function getAimingAtClient(steps):
    i = 0
    while i menor que número de linhas do log do
        if line[i+steps] está mirando no oponente then
            adiciona Yaw e Pitch da linha i até i + steps
            i += steps
        i += 1
```

Como demonstra o algoritmo, os dados serão coletados em janelas de tempo definidas pelo parâmetro *steps*, que determina o tamanho da sequência temporal de entrada para a rede neural. O último registro da janela é o registro em que o jogador estava mirando no oponente. A ideia por trás desse filtro é que tanto o *aimbot* como qualquer jogador honesto terão reações diferentes ao ver um oponente e tentar mirar no mesmo. Portanto, é esperado que a rede neural seja capaz de distinguir essas diferenças analisando os registros anteriores àquele em que a mira já está sobre o oponente.

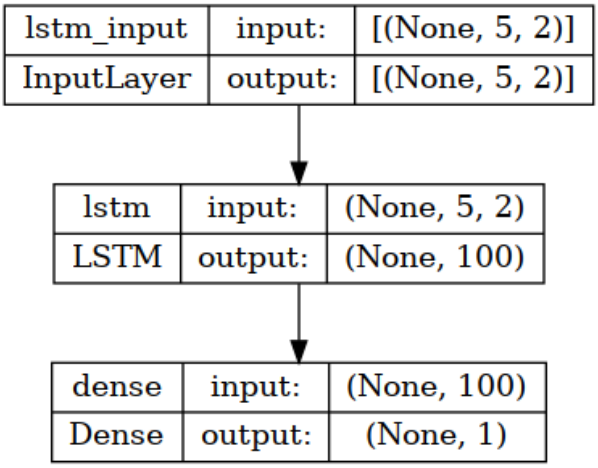
Para que os dados possam ser utilizados como entrada na rede neural, eles precisam ser normalizados de forma a ficar contidos na faixa de 0 a 1. Para avaliar o melhor tamanho da janela de tempo de entrada, foram arbitradas amostras com sequências de diferentes tamanhos, os quais podiam conter cinco, dez ou 20 registros cada. Para as sequências de cinco registros, o número de amostras foi 74.600, para as de dez registros foram 46.201 amostras e por fim para as sequências de 20 registros, o número de amostras foi 31.412.

7.3 Construção do Modelo de Rede Neural

Com os dados coletados, filtrados e transformados, iniciou-se a construção do algoritmo de aprendizado de máquina que iria classificar uma amostra em honesta ou não. O algoritmo criado usa uma Rede Neural Recorrente, visto que os dados são séries temporais. O modelo contou com uma camada de nodos LSTM, onde foram utilizados diferentes números de nodos para testar a performance dos modelos. Por fim foi utilizada uma camada de saída, chamada de *Dense* na biblioteca *Keras*. A função de ativação utilizada foi a tangente hiperbólica, o padrão do LSTM na biblioteca *Keras*, e o otimizador escolhido foi a função Adam. Os valores padrões do mesmo foram utilizados, i.e., uma taxa de aprendizado igual a 0.001 (KERAS, 2023). O código completo se encontra no

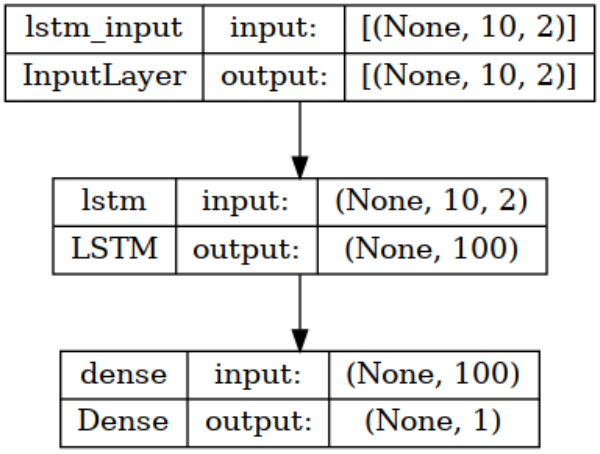
Apêndice A. A topologia resultante de cada um destes modelos iniciais se encontram nas figuras 7.4, 7.5 e 7.6.

Figura 7.4 – Topologia do modelo com janela de tamanho cinco e 100 nodos



Fonte: O Autor

Figura 7.5 – Topologia do modelo com janela de tamanho dez e 100 nodos

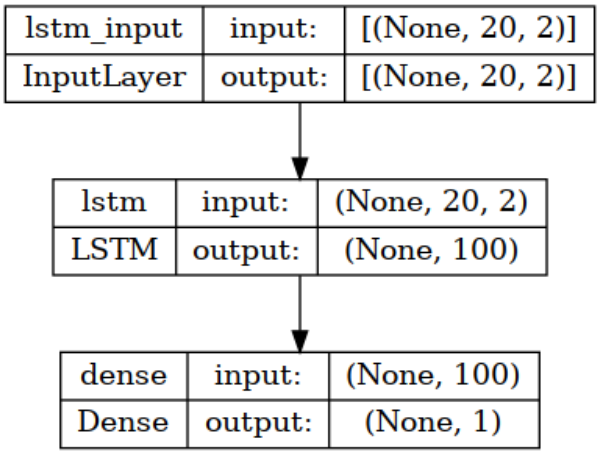


Fonte: O Autor

7.4 Treinamento

Depois do modelo ter sido construído, ele foi treinado. Esta etapa tem como objetivo agregar conhecimento à rede neural, atualizando os pesos das conexões entre os nodos de forma a minimizar o erro entre a saída do modelo e a saída pretendida. Para o treinamento é preciso informar ao modelo os dados de entrada, o resultado esperado e

Figura 7.6 – Topologia do modelo com janela de tamanho 20 e 100 nodos



Fonte: O Autor

o número de iterações, também chamado de épocas. Concluída esta etapa, é gerado um arquivo com os pesos resultantes da última iteração de treino, que representam o conhecimento adquirido da rede neural e que pode, posteriormente, ser carregado por um modelo com a mesma estrutura, de forma a utilizar o conhecimento persistido anteriormente.

Para o treinamento do algoritmo proposto foram utilizados oitenta por cento das amostras disponíveis das partidas de três jogadores honestos e quatro desonestos, utilizando os três intervalos mencionados na seção 7.2. Já os outros vinte por cento foram utilizados para avaliar o desempenho do treinamento, juntamente com as amostras de quatro jogadores honestos e dois desonestos. O *hardware* disponível demorou em média dez segundos para fazer uma iteração de treinamento na topologia inicialmente sugerida. Visto que 400 épocas foram utilizadas para uma avaliação inicial de cada modelo, o período de tempo requerido para o treinamento de cada modelo foi ligeiramente superior a uma hora. Após as avaliações iniciais outros modelos foram treinados, como será apresentado no capítulo 8.

8 RESULTADOS

Este capítulo apresentará os resultados e validações dos modelos desenvolvidos. No final os resultados serão interpretados, discutidos e comparados com os objetivos iniciais.

8.1 Modelos de Rede Neural

Após o treinamento, dos modelos apresentados anteriormente na seção 7.3, é retornado um histórico com as métricas resultantes do mesmo. Nas Figuras 8.1, 8.2 e 8.3 temos o histórico de precisão, que pode ser definido como a porcentagem de acerto do modelo, visualizados em um gráfico ao longo das iterações. Nestes gráficos, quanto mais próximo de 100% forem os acertos do algoritmo melhores os resultados.

Os valores máximos de precisão do algoritmo foram de 62.45%, 61.51% e 58.98% para os modelos 1, 2 e 3, respectivamente. É possível notar um pequeno decremento da precisão para os números de sequências temporais maiores, possivelmente resultado de uma necessidade de mais iterações e/ou nodos para estas janelas. Após o treinamento, os modelos foram testados com os conjuntos de amostras separadas. Na Tabela 8.1 é mostrado o resultado desses testes, que considera uma amostra como trapaceira se o modelo retorna um valor maior que 0.95. A escolha deste valor visa minimizar o número de falsos positivos, por classificar como fraudulento apenas amostras para as quais a rede retorna um grau de certeza alto, como explicado no trabalho do Gaspareto (2008).

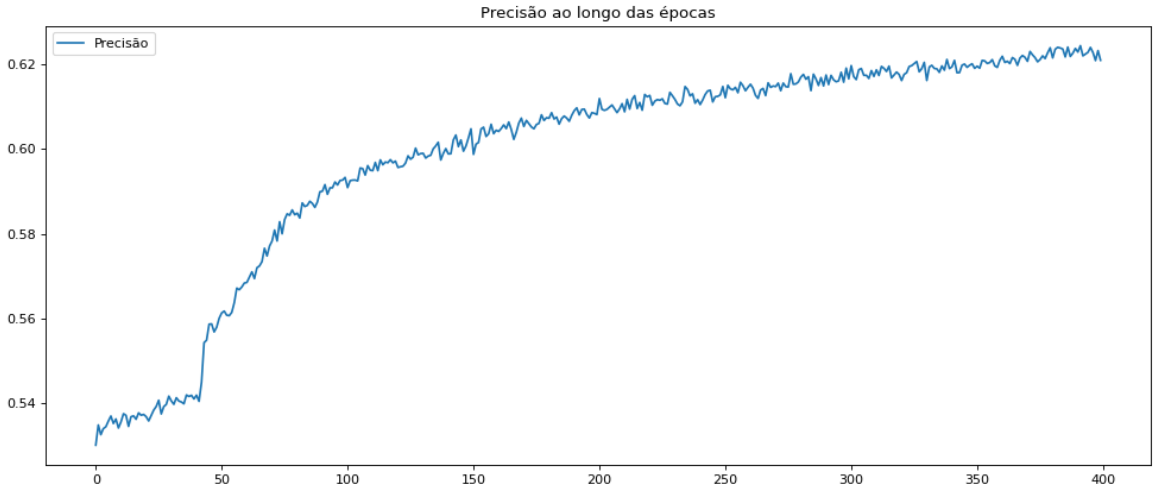
Tabela 8.1 – Resultados das amostras de teste

Modelo	Intervalo	Acertos	Erros	Precisão	Falsos Positivos	Cheaters
1	5	7546	7374	50.57%	2	639
2	10	4507	4734	48.77%	2	280
3	20	3050	3233	48.54%	0	156

Fonte: O Autor

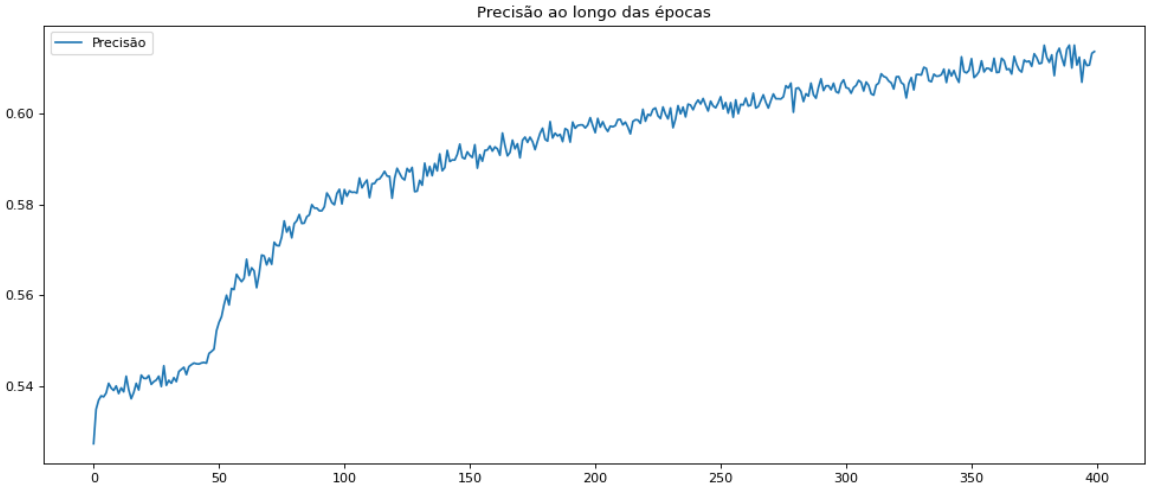
Visto que tanto no conjunto de treinamento como no de teste a maior precisão foi no modelo com a janela de cinco registros, outros modelos com topologias de cinco registros e duas características foram analisadas. Duas topologias com um número menor de nodos foram testadas, mantendo as mesmas camadas, como demonstram as Figuras 8.4 e 8.5. Contudo ambos os modelos apresentaram uma precisão menor do que o modelo inicial com 100 nodos, com 49.69% e 49.47% de acertos no conjunto de teste nos modelos

Figura 8.1 – Gráfico de precisão do modelo 1



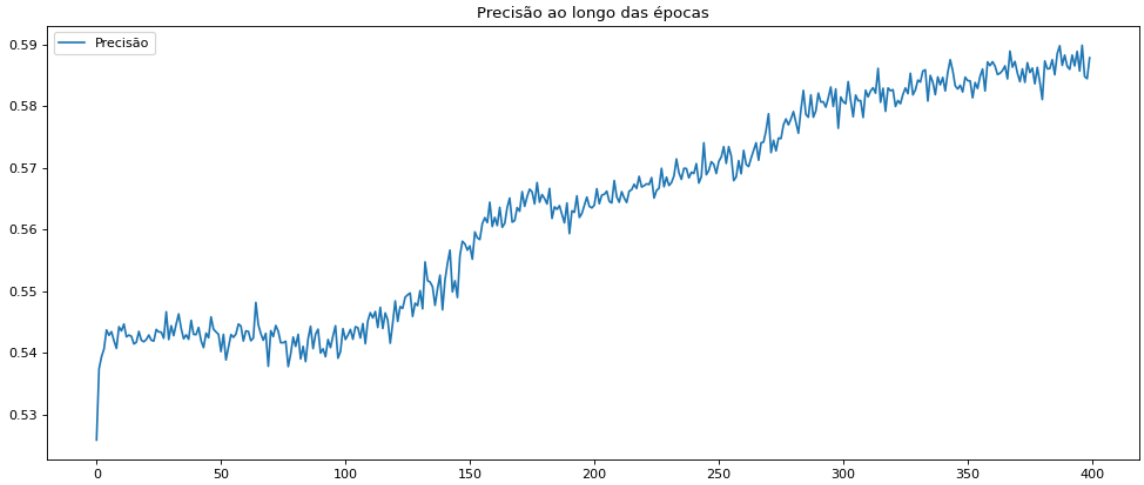
Fonte: O Autor

Figura 8.2 – Gráfico de precisão do modelo 2



Fonte: O Autor

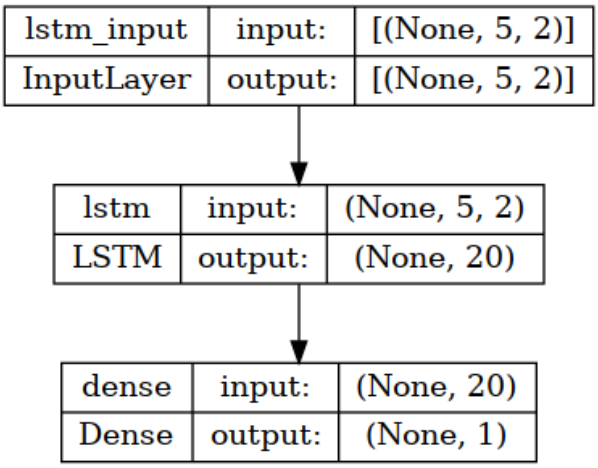
Figura 8.3 – Gráfico de precisão do modelo 3



Fonte: O Autor

de 20 e 50 nodos, respectivamente.

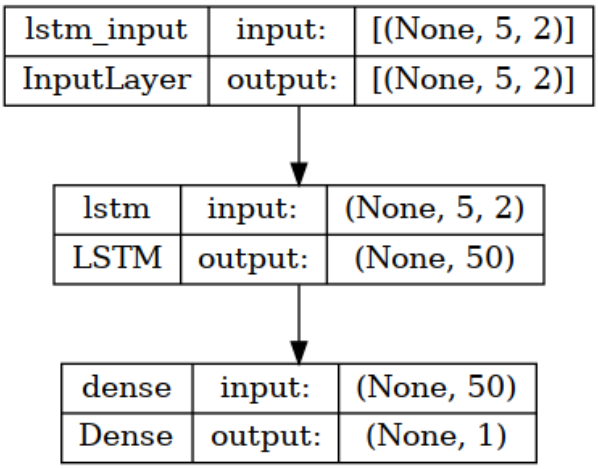
Figura 8.4 – Topologia do modelo com janela de tamanho cinco e 20 nodos



Fonte: O Autor

Portanto, estabeleceu-se o padrão de um modelo com no mínimo 100 nodos, dado os resultados apresentados. Posteriormente, em outro modelo foram adicionados mais camadas. Estas camadas foram uma camada adicional de nodos LSTM com o mesmo número de nodos e duas camadas *dropout*, um tipo de camada que tem o objetivo de diminuir o *overfitting* (BROWNLEE, 2020). A topologia resultante pode ser consultada na Figura 8.6. Em um primeiro momento o modelo foi treinado por 400 iterações, dado que a precisão do modelo melhorou em relação aos treinamentos anteriores, a mesma

Figura 8.5 – Topologia do modelo com janela de tamanho cinco e 50 nodos



Fonte: O Autor

topologia foi utilizada num treinamento de 1000 iterações. Os resultados de ambos esses treinamentos estão na Tabela 8.2. O treinamento desta nova topologia demorou um tempo maior do que o das topologias iniciais, visto que com mais camadas e nodos, o número de pesos também aumenta. Nesta topologia o tempo média de uma iteração durou cerca de 20 segundos. Consequentemente, o treinamento com 1000 iterações durou mais de cinco horas.

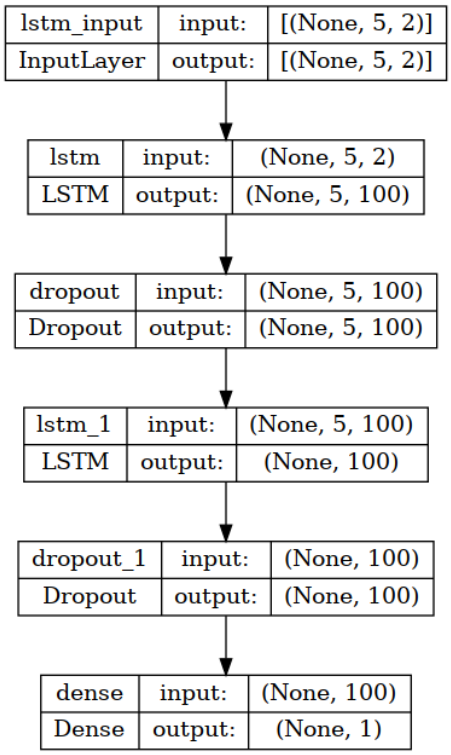
Tabela 8.2 – Resultados das amostras de teste para os modelos com 100 nodos

Iterações	N° Acertos	N° Erros	Precisão	N° Falsos Positivos	N° Cheaters
400	7986	6934	53.52%	3	1044
1000	7978	6942	53.47%	1	948

Fonte: O Autor

Em um primeiro momento, observando-se o percentual de acerto dos modelos, pode-se concluir que os resultados não foram satisfatórios, dado que a precisão está próxima a de um acerto aleatório 50/50. Contudo, é importante notar que a tabela permite inferir que o número de falsos positivos foi menor do que 0.02% nos dois modelos, percentual obtido através da divisão do número de falsos positivos pelo número total de amostras no conjunto de testes. Para que um jogador seja considerado fraudulento não é necessário que todas as amostras dele sejam fraudulentas. É possível estabelecer um limiar de forma a desconsiderar possíveis falsos positivos em um intervalo de rodadas. Ainda, no caso do acumulo de amostras classificadas como fraudulentas, com um grau de confiança acima do limiar estabelecido, o jogador pode ser considerado desonesto e estar sujeito as devidas punições. A Tabela 8.3 mostra um teste com o modelo de duas

Figura 8.6 – Topologia do modelo com janela de tamanho cinco e 100 nodos em duas camadas



Fonte: O Autor

camadas LSTM de 100 nodos, separado por jogador. Neste teste, a coluna que registra a precisão do modelo utiliza o mesmo cálculo da biblioteca *Keras* demonstrado no Algoritmo 2 (DOMMARAJU, 2020). O algoritmo demonstra que a precisão é calculada com o número de acertos dividido pelo total de amostras.

Algoritmo 2: Cálculo da precisão pelo *Keras*

```
Function precisao(saida, previsao):
    resultado = 0
    for i = 0; i menor que tamanho da saida; i += 1 do
        if saida[i] igual a previsao[i] then
            resultado += 1
    retorna resultado/tamanho(saida)
```

Não foi encontrado na literatura um valor para o ponto de corte que seja considerado ideal nos sistemas de proteção contra trapaças, visto que, como já foi mencionado, as pesquisas na áreas são fechadas por parte das empresas e os dados são de acesso limitado. Contudo, pesquisas como a publicada por Jonnalagadda (2021) consideram o número de falsos positivos como uma medida importante, dado que é "um tipo de erro que é incompa-

Tabela 8.3 – Resultados das amostras de teste por jogador

Jogador	N° Acertos	N° Erros	Precisão
Honesto 1	1114	208	84.26%
Honesto 2	674	28	96.01%
Honesto 3	253	380	39.97%
Honesto 4	348	41	31.96%
Honesto 5	104	276	27.36%
Honesto 6	273	320	46.04%
Honesto 7	377	910	29.29%
Desonesto 1	571	368	60.81%
Desonesto 2	179	190	48.51%
Desonesto 3	171	134	56.06%
Desonesto 4	536	292	64.73%
Desonesto 5	539	488	52.48%
Desonesto 6	122	493	19.84%

Fonte: O Autor

tível com a implantação prática de um sistema anti-trapaça eficaz."(JONNALAGADDA et al., 2021). Com os resultados obtidos, observa-se que o modelo não alcançou o desempenho esperado. Estabelecendo o limiar para detecção de fraude em 50% das amostras, seis jogadores seriam classificados corretamente. Contudo cinco jogadores seriam erroneamente classificados como desonestos, um alto número de falsos positivos. Ainda, dois jogadores desonestos não seriam detectados. Caso o limiar estabelecido fosse maior, de forma a diminuir o número de falsos positivos, haveria ainda mais jogadores desonestos não detectados. Em especial, para as amostras que foram utilizadas para testes, os jogadores honestos quatro ao sete, não tiveram uma boa precisão. Neste contexto, algumas hipóteses podem ser levantadas para explicar o desempenho dos modelos:

1. O número de amostras não foi grande e variado o suficiente.
2. A quantidade de características relevantes de entrada foi insuficiente.
3. As topologias utilizadas não foram adequadas.

Os resultados mostram que o modelo apresentou sinais de *overfitting*, mesmo com as camadas de *dropout*. Contudo, o histórico de precisão do treinamento não indica que um número menor de iterações seria a solução para o problema. Por estes motivos as hipóteses 1 e 2 foram formuladas. Sobretudo a hipótese número 2 pode ter maior correlação com os resultados obtidos, dado que outras características podem de fato indicar que um jogador é desonesto e está utilizando uma trapaça do tipo *aimbot*. Por exemplo, se após mirar no oponente um tiro foi disparado, em qual parte do corpo do oponente a mira estava posicionada, por quanto tempo o ponteiro esteve sobre o oponente, entre outras. Já

a hipótese 3 está fundada no fato de haver possibilidades não exploradas neste primeiro momento, quer sejam referentes as funções de ativação, no tipo de nodo LSTM em detrimento de outros, otimizadores, valor da taxa de aprendizado, função de desempenho e outros.

9 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foi possível verificar a importância do combate aos jogadores desonestos dentro dos jogos eletrônicos. Com o objetivo de contribuir na pesquisa da área de *anti-cheat*, foi proposto o desenvolvimento de uma rede neural recorrente treinada para detectar jogadores que utilizam a trapaça de *aimbot* em jogos de FPS. Depois de construídos os modelos, eles foram avaliados com as métricas de precisão e número de falsos positivos.

Uma das dificuldades encontradas foi a procura de uma trapaça que funcionasse dentro do jogo CS:GO, visto que a proteção do Windows 11 contra injeção de código na memória de programas se mostrou efetiva até certo ponto. A solução foi utilizar um *cheat* encontrada no fórum UnkwnonCheats, *cheat* este que é testado pela própria comunidade e possui código aberto, de forma que se mostram mais seguros. Outra dificuldade encontrada foi o tempo de treinamento dos modelos. No *hardware* disponível, uma iteração com as topologias iniciais da proposta leva em torno de dez segundos para ser completada. Portanto as 400 épocas de treinamento que foram utilizadas para cada um dos modelos levaram em torno de 4000 segundos para serem completadas, i.e., um pouco mais de 1 hora. Já a topologia mais complexa desenvolvida levou cerca de 20 segundos por iteração e o treinamento como um todo, em suas 1000 iterações, demorou mais de cinco horas.

Seis modelos foram desenvolvidos, treinados e avaliados. Mesmo que o modelo com melhores resultados tenha apresentado uma precisão de cerca de 63% no conjunto de treinamento e sido capaz de distinguir corretamente seis dos 13 jogadores, mais da metade dos sete jogadores honestos foram equivocadamente classificados como desonestos. Dessa forma os resultados não foram totalmente satisfatórios, pois, mesmo que não haja um valor de corte de precisão para a área que seja consenso entre as pesquisas, o número de falsos positivos deve ser mínimo. Contudo a contribuição para a pesquisa na área foi atingida, dado que novas propostas de pesquisa podem ser traçadas a partir dos erros e acertos aqui registrados.

Isto sugere que, em trabalhos futuros, o trabalho pode ser evoluído ao serem coletados mais dados para treinamento da rede neural. Ainda, outros pontos que podem ser evoluídos são:

- Geração de dados com diferentes *aimbots*, visto que cada um pode ter desempenho diferente, o que pode influenciar no resultado da sua detecção ou não, como o *aimbot* desenvolvido por Kanervisto (2022);

- Obtenção de dados de jogadores honestos mais habilidosos. Pode-se considerar que, em níveis profissionais, os jogadores podem ter performance tão elevada que a rede poderia detectar como um falso positivo;
- Capturar mais dados relevantes para entrada da rede neural, como a posição da mira no inimigo, se foram realizados disparos, em que momento o inimigo entrou no campo de visão do jogador, entre outros;
- Ampliar o número de avaliações das amostras de teste, de forma a entender se os resultados são constantes ou se há amostras que não são classificadas da mesma forma sempre.

REFERÊNCIAS

BITENCOURT, E. J. C. **Trilha musical para videogames: compondo para o jogo Tibia**. Monografia (Graduação) — Universidade Federal do Rio Grande do Sul, 2017. Available from Internet: <http://hdl.handle.net/10183/172667>.

BLIZZARD, A. **2022 Annual Report**. 2022. Investor Activision. Available from Internet: <https://investor.activision.com/static-files/01d1f04d-1c00-4a17-8743-4e6a20e17335>.

BRANCO, H. **Overfitting e underfitting em Machine Learning**. 2023. Associação Brasileira de Ciência de Dados. Available from Internet: <https://abracd.org/overfitting-e-underfitting-em-machine-learning/>.

BRAZIL, C. H. K. **Clonagem e falsificação na indústria de jogos digitais: uma análise da ambivalência da pirataria**. Monografia (Graduação) — Universidade Federal do Rio Grande do Sul, 2021. Available from Internet: <http://hdl.handle.net/10183/234991>.

BROWNLEE, J. **Dropout with LSTM Networks for Time Series Forecasting**. 2020. Available from Internet: <https://machinelearningmastery.com/use-dropout-lstm-networks-time-series-forecasting/>.

BUSHAEV, V. **Adam — latest trends in deep learning optimization.** 2018. Towards Data Science. [Accessed 19-08-2023]. Available from Internet: <https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>.

CAGLE, K. **What Is Artificial Intelligence?** 2019. Available from Internet: <https://www.forbes.com/sites/cognitiveworld/2019/08/20/what-is-artificial-intelligence/?sh=62e831da306f>.

CARBONE, F. **CS:GO: quase 200 mil contas foram banidas por trapaça no 1º semestre**. 2022. [Accessed 08-08-2023]. Available from Internet: <https://ge.globo.com/esports/csgo/noticia/2022/07/19/csgo-quase-200-mil-contas-foram-banidas-por-trapaca-no-1o-semestre.ghtml>.

CARTER, K. **Robot that makes me an Aiming Pro**. 2022. [Accessed 19-08-2023]. Available from Internet: <https://www.youtube.com/watch?v=ne9bmMX82iY>.

CASTRO, M. **Diferenças entre Single Player e Multiplayer**. 2019. [Accessed 17-08-2023]. Available from Internet: <https://www.linkedin.com/pulse/jogos-digitais-em-rede-moisÃl's-castro/?originalSubdomain=pt>.

CHARTS, S. **Counter-Strike: Global Offensive - Steam Charts**. 2023. [Accessed 03-08-2023]. Available from Internet: <https://steamcharts.com/app/730#1y>.

DANCKER, J. **A Brief Introduction to Recurrent Neural Networks**. 2022. [Accessed 19-08-2023]. Available from Internet: <https://towardsdatascience.com/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4>.

DERETTI, A. A. **O "ver e sentir"dos videogames: a importância da gameplay experience na localização de jogos**. Monografia (Graduação) — Universidade Federal do Rio Grande do Sul, 2017. Available from Internet: <http://hdl.handle.net/10183/179528>.

DOMMARAJU, G. **Keras' Accuracy Metrics**. 2020. Available from Internet: <https://towardsdatascience.com/keras-accuracy-metrics-8572eb479ec7>.

DUARTE, F. H. **Desafios da localização de jogos: o caso de Life is Strange**. Monografia (Graduação) — Universidade Federal do Rio Grande do Sul, 2017. Available from Internet: <http://hdl.handle.net/10183/171714>.

EARNINGS, E. **Counter-Strike: Global Offensive Prize Pools**. 2023. [Accessed 06-08-2023]. Available from Internet: <https://www.esportsearnings.com/games/245-counter-strike-global-offensive>.

EARNINGS, E. **Highest Earnings By Country**. 2023. [Accessed 05-08-2023]. Available from Internet: <https://www.esportsearnings.com/countries>.

EARNINGS, E. **Top Games Awarding Prize Money**. 2023. [Accessed 06-08-2023]. Available from Internet: <https://www.esportsearnings.com/games>.

FREITAS, V. M. R. d. **Procedural generation of cave-like maps for 2D top-down games**. Monografia (Graduação) — Universidade Federal do Rio Grande do Sul, 2021. Available from Internet: <http://hdl.handle.net/10183/224871>.

GASPARETO, O. B. **Redes neurais artificiais aplicadas ao reconhecimento de speed cheating em jogos online de computador**. Dissertation (mathesis) — UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, may 2008. Available from Internet: <https://www.lume.ufrgs.br/handle/10183/153317>.

GHEDIRA, H.; BERNIER, M. The effect of some internal neural network parameters on SAR texture classification performance. In: **IEEE International IEEE International IEEE International Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004**. [S.l.]: IEEE, 2004. v. 6, p. 3845 – 3848 vol.6. ISBN 0-7803-8742-2.

GIUSEPPE, M. d. O. **Composição de trilhas musicais para videogame**. Monografia (Graduação) — Universidade Federal do Rio Grande do Sul, 2018. Available from Internet: <http://hdl.handle.net/10183/189443>.

HENDLER, J. Avoiding another ai winter. **IEEE INTELLIGENT SYSTEMS**, v. 23, p. 2–4, 2008. ISSN 1541-1672. Available from Internet: <https://www.researchgate.net/profile/James-Hendler/publication/3454567_Avoiding_Another_AI_Winter/links/558833d808ae8c4f3406358f/Avoiding-Another-AI-Winter.pdf?ref=>.

HITCHENS, M. A survey of first-person shooters and their avatars. **Game Studies**, Game Studies, v. 11, n. 3, p. 96–120, 2011. Available from Internet: <https://gamestudies.org/1103/articles/michael_hitchens/>.

JONNALAGADDA, A. et al. Robust vision-based cheat detection in competitive gaming. **Proceedings of the ACM on Computer Graphics and Interactive Techniques**, Association for Computing Machinery, New York, NY, USA, v. 4, n. 1, p. 1–18, 4 2021. Available from Internet: <https://doi.org/10.1145/3451259>.

KANERVISTO, A.; KINNUNEN, T.; HAUTAMÄKI, V. Gan-aimbots: Using machine learning for cheating in first person shooters. **arXiv e-prints**, p. arXiv:2205.07060, may 2022. Available from Internet: <https://ui.adsabs.harvard.edu/abs/2022arXiv220507060K>.

KERAS. **Adam**. 2023. Available from Internet: <https://keras.io/api/optimizers/adam/>.

KHALIFA, S. **Machine learning and anti-cheating in fps games**. Dissertation (Master), sep. 2016. Available from Internet: <https://www.researchgate.net/profile/Salman-Alkhalifa/publication/308785899_Machine_Learning_and_Anti-Cheating_in_FPS_Games/links/57f1105808ae886b8978f2f1/Machine-Learning-and-Anti-Cheating-in-FPS-Games.pdf>.

KRSTIć, D. **Contagem atual de jogadores do Battlefield 2042 e o que isso significa para o futuro do jogo**. 2023. Available from Internet: <https://fpschampion.com/pt-br/contagem-atual-de-jogadores-do-battlefield-2042-e-o-que-isso-significa-para-o-futuro-do-jogo/>.

LAHTI, E. **CS:GO competitive scene in hacking scandal, 3 players banned**. 2014. [Accessed 16-08-2023]. Available from Internet: <https://www.pcgamer.com/csgo-competitive-scene-embroiled-in-hacking-scandal-as-three-players-are-banned/>.

LEHTONEN, S. J. Comparative study of anti-cheat methods in video games. **Helsingin yliopisto**, 3 2020. Available from Internet: <https://helda.helsinki.fi/items/b1141406-eb65-48a5-8922-d1b23d4cfe51>.

LIMA, L. F. **Hacks e cheats no CS:GO: 5 jogadores que foram banidos do jogo**. 2018. [Accessed 08-08-2023]. Available from Internet: <https://www.techtudo.com.br/listas/2018/07/hacks-e-cheats-no-csgo-5-jogadores-que-foram-banidos-do-jogo-esports.ghtml>.

LIU, Y. H. **Python Machine Learning by Example Build Intelligent Systems Using Python, TensorFlow 2, Pytorch, and Scikit-Learn, 3rd Edition**: Build intelligent systems using python, tensorflow 2, pytorch, and scikit-learn, 3rd edition. [S.l.]: Packt Publishing, Limited, 2020. ISBN 978-1800209718.

LOCA, A.; RAUBER, T. Uso de uma rede neural convolucional unidimensional para detecção de falhas em processos industriais. In: SBC. **Anais da XIX Escola Regional de Computação Bahia, Alagoas e Sergipe**. 2019. p. 42–47. Available from Internet: <https://sol.sbc.org.br/index.php/erbase/article/download/8952/8853/>.

MCDONALD, J. **Robocalypse Now: Using Deep Learning to Combat Cheating in Counter-Strike: Global Offensive**. 2018. [Accessed 08-08-2023]. Available from Internet: <https://www.youtube.com/watch?v=kTiP0zKF9bc>.

MO1RA. **Valorant Aproject Internal (Free Hack)**. 2021. Available from Internet: <https://www.unknowncheats.me/forum/3039936-post1.html>.

NASCIMENTO, Y. R. C. d. S. R. d. **Dinâmicas sociais do desenvolvimento das culturas dos videogames no Brasil: o caso do Counter-Strike**. Dissertation (Master) — Universidade Federal do Rio Grande do Sul, 2023. Available from Internet: <http://hdl.handle.net/10183/257994>.

OPENAI. **ChatGPT**. 2022. Available from Internet: <https://openai.com/blog/chatgpt>.

PAZ, S. **A espectralidade des/contínua do tempo no gameplay de Dark Souls:[Dark Souls] remastered e Dark Souls 3**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2022. Available from Internet: <http://hdl.handle.net/10183/249785>.

PITTOL, G. L. D. **A história e contribuição dos jogos e consoles de videogame para a sociedade e a computação**. Monografia (Graduação) — Universidade Federal do Rio Grande do Sul, 2019. Available from Internet: <http://hdl.handle.net/10183/198493>.

REEVES S., B. B. . L. E. Experts at play: Understanding skilled expertise. **Games and Culture**, Sage Publications Sage CA: Los Angeles, CA, v. 4, n. 3, p. 205–227, 2009. Available from Internet: <https://journals.sagepub.com/doi/abs/10.1177/1555412009339730>.

SCHUYTEMA, P. **Design de games: uma abordagem prática**. [S.l.]: Cengage Learning, 2008.

SILVA, B. D. d. **A gamificação a partir de suas críticas: uma leitura da relação games/aprendizagem pela arte/educação**. Dissertation (Master) — Universidade Federal do Rio Grande do Sul, 2021. Available from Internet: <http://hdl.handle.net/10183/219896>.

SILVA, J. N. **Towards Automated Server-side Video Game Cheat Detection**. Dissertation (Master) — Universidade do Porto, jul. 2022. Available from Internet: <https://repositorio-aberto.up.pt/bitstream/10216/142935/2/572983.pdf>.

SOFTWARE id. **Wolfstein 3D(1992)**. 1992. Available from Internet: <https://pt.wikipedia.org/wiki/Wolfenstein_3D>.

STANTON, R. **Have hackers and cheats ruined The Division on PC?** 2016. [Accessed 08-08-2023]. Available from Internet: <https://www.theguardian.com/technology/2016/apr/26/hackers-cheats-ruined-the-division-pc-ubisoft>.

STEAM. **Counter-Strike: Global Offensive on Steam**. 2023. [Accessed 05-08-2023]. Available from Internet: <https://store.steampowered.com/app/730/CounterStrike_Global_Offensive/?l=brazilian>.

STEAM. **Steam Search**. 2023. [Accessed 08-08-2023]. Available from Internet: <https://store.steampowered.com/search/?ignore_preferences=1&category1=998&category2=8&ndl=1>.

STRIKE, C. **Counter Strike: Global Offensive. Overwatch faq**. 2023. [Accessed 17-08-2023]. Available from Internet: <https://blog.counter-strike.net/index.php/overwatch/>.

SURFSHARK. **Cheating games - which online games have the most cheaters?** 2021. Surfshark. [Accessed 06-08-2023]. Available from Internet: <https://surfshark.com/hacking-wins>.

UNKOWNCHEATS. **UnKnoWnCheaTs - Multiplayer Game Hacking and Cheats**. 2023. [Accessed 05-08-2023]. Available from Internet: <https://www.unknowncheats.me/forum/downloads.php>.

WILLMAN, M. **Machine Learning to identify cheaters in online games**. Dissertation (mathesis) — Umeå University, may 2020. Available from Internet: <https://www.diva-portal.org/smash/record.jsf?pid=diva2:1431282&dswid=1708>.

WU, Y.-c.; FENG, J.-w. Development and application of artificial neural network. **Wireless Personal Communications**, Springer Science and Business Media LLC, v. 102, n. 2, p. 1645–1656, dec 2017.

ZHOU, Z.-H. **Machine learning**. [S.l.]: Springer Nature, 2021.

APÊNDICE A — CÓDIGOS FONTE

https://github.com/DenysonJ/data

https://github.com/DenysonJ/anticheat

https://github.com/DenysonJ/tccResults