UNIVERSIDAD ANDRÉS BELLO

FACULTAD DE INGENIERÍA

INGENIERÍA CIVIL INFORMÁTICA

Proyecto de Sistema Anti-Trampas para CS2 con Deep Learning

Alejandro Andrés Matus Silva

Profesor guía: Pablo Hernan Schwarzenberg Riveros

SANTIAGO – CHILE

Junio, 2025

Índice general

Índice de figuras

Índice de tablas

Resumen

Capítulo 1

Introducción

1.1. Contexto

El género de los videojuegos multijugador de disparos en primera persona (FPS), y en particular la franquicia Counter-Strike, ha visto un crecimiento exponencial en su base de jugadores y en la profesionalización de sus competiciones. En CS:GO (Counter-Strike: Global Offensive), por ejemplo, se estima que entre un 5 % y un 10 % de las partidas incluyen al menos un usuario que emplea software de trampa, lo que aumenta la probabilidad de encontrarse con un tramposo hasta un 60 % en cada partida competitiva. A medida que los torneos de e-sports ofrecen premios millonarios y seducen a audiencias globales, como el caso de Nikhil "forsaken"Kumawat exjugador de OpTic India que durante el evento en LAN de la ESL India Premiership en 2018 fue descubierto utilizando programas indebidos de terceros, el cual estaba llamado "Word.exe"para pasar desapercibido, siendo las consecuencias de esto fue la descalificación del equipo en el torneo y la suspensión del jugador para competiciones profesionales.

1.2. Problema

A pesar de los esfuerzos iniciales, los métodos de detección de trampas tradicionales —basados en firmas (p. ej., VAC de valve) o en mecanismos cliente-lado que monitorean el espacio de memoria del juego— han demostrado ser insuficientes frente a la sofisticación de los cheats actuales. Entre los tipos de trampas más comunes se encuentran:

Aimbot algoritmos que apuntan automáticamente al oponente, eliminando la necesidad de habilidad manual y ofreciendo una precisión inhumana.

Wallhack Alteran la representación grafica para permitir ver a través de paredes u obstáculos, otorgando información privilegiada sobre la posición de los enemigos.

Triggerbot Disparan automáticamente cuando el apuntador se posiciona sobre un objetivo valido, reduciendo drásticamente el tiempo de reacción.

Spinbot Modifican la rotación del modelo del jugador de forma errática para que sea difícil el apuntado en su persona y usar aimbot en personas que están detrás de el.

Speedhack Modifican la velocidad del jugador, normally utilizan un método donde hacen uso de "bunny hop"perfectos, provenientes de una mecánica dentro del juego o modifican

parámetros para aumentar la velocidad.

1.3. Importancia

La presencia continua de trampas no solo altera los resultados de las partidas, sino que tiene un impacto profundo en la experiencia del jugador y en la salud de las comunidades de e-sports. Desde la perspectiva psicológica, Smith et al. (2024) demuestran que la exposición repetida a cheats genera frustración, abandono del juego y una percepción de injusticia que puede disuadir la participación a largo plazo. A nivel social, la sensación de "juego amañado.[er]osiona la confianza en el sistema de emparejamiento y disminuye el valor de marca de los organizadores de torneos.

Por otro lado, la inteligencia artificial ha emergido como una alternativa prometedora:

- Modelos basados en series temporales de entradas de teclado y ratón, como redes CNN que analizan patrones de movimiento.
- Visión por computador, como VADnet (Nie & Ma, 2024), inspecciona directamente los frames de juego para detectar superposiciones ilegales sin interferir en el software del cliente.
- Redes neuronales clásicas (Maberry, 2016) han demostrado capacidad para identificar asistencia de tiro o .[a]im assistance.[a] partir de análisis de trayectoria del puntero, capturando comportamientos de puntería inhumanamente precisos.

Capítulo 2

Revisión Bibliográfica

Los enfoques tradicionales para combatir este problema se han basado en cuatro pilares fundamentales: la detección por firmas mediante escaneo de memoria, el análisis estadístico de comportamiento inusual, los sistemas de revisión comunitaria como Overwatch, y la aplicación de restricciones internas del motor del juego (Daniel, 2019). Estos métodos han demostrado eficacia frente a trampas simples, pero presentan importantes limitaciones. Por ejemplo, los sistemas basados en firmas requieren actualizaciones constantes y no detectan variantes nuevas, mientras que los métodos estadísticos pueden generar falsos positivos y suelen ser poco eficaces frente a trampas sutiles.

Frente a estas limitaciones, en los últimos años se han desarrollado sistemas más adaptativos y proactivos, utilizando técnicas de inteligencia artificial. Un caso destacado es VACNet, desarrollado por Valve, que emplea redes neuronales profundas para analizar telemetría de partidas y detectar comportamientos anómalos asociados a trampas, como movimientos de puntería inhumanamente precisos o patrones de disparo automatizados (Daniel, 2019). Este enfoque ha permitido automatizar parte del proceso de revisión que antes dependía exclusivamente de usuarios humanos, mejorando la eficiencia y la escala de la detección.

Varios estudios académicos han contribuido con modelos basados en aprendizaje profundo aplicados a la detección de trampas. Pinto et al. (2021) propusieron un sistema que transforma series de eventos de entrada (teclado y ratón) en secuencias multivariadas, que son luego procesadas por redes neuronales convolucionales (CNN). Este modelo logró detectar aimbots y triggerbots en escenarios de CS:GO con una precisión superior al 98 %. En una línea similar, Nie y Ma (2024) desarrollaron VADNet, una red neuronal basada en visión por computador capaz de analizar directamente las imágenes del juego y detectar superposiciones ilegales, lo que representa una ventaja frente a trampas visuales que no alteran el cliente ni acceden a la memoria del sistema.

Otros enfoques han explorado el uso de redes neuronales recurrentes (RNN y LSTM) para analizar series temporales del comportamiento de los jugadores, identificando patrones que podrían corresponder a trampas. Liu et al. (2017) demostraron que este tipo de modelos logra detectar trampas clásicas y nuevas variantes basadas en visión artificial, aunque enfrentan limitaciones cuando se introducen trampas evasivas más sofisticadas.

Desde una perspectiva de seguridad, Karkallis y Blasco (2025) han abordado los ataques a los sistemas anticheat desde un ángulo técnico. En su investigación, se analizan las debilidades de los anticheats que operan únicamente del lado del cliente, destacando cómo estos pueden

ser vulnerables a técnicas como la introspección de máquinas virtuales (VIC), que permite ejecutar trampas sin ser detectadas por software de seguridad tradicional. Este tipo de trampas basadas en hardware o virtualización plantea nuevos retos incluso para los modelos basados en IA (Chen, 2024).

Complementariamente, diversos estudios destacan la necesidad de que estos sistemas sean no solo eficaces, sino también éticos y transparentes. El uso de anticheats a nivel kernel ha sido duramente criticado por su potencial invasivo y sus riesgos de privacidad (Collins et al., 2024). En este sentido, los sistemas que combinan IA con datos públicos y modelos supervisados no invasivos representan una alternativa más aceptable para las comunidades de jugadores.

Finalmente, se reconoce también el impacto psicológico y social del cheating en videojuegos. Investigaciones como la de Smith et al. (2024) demuestran que los jugadores expuestos frecuentemente a trampas muestran mayor frustración, menor adherencia al juego y una percepción deteriorada de la justicia del sistema. Esto subraya la urgencia de implementar mecanismos efectivos y fiables que restauren la confianza en el entorno competitivo.

En conjunto, el estado del arte muestra que la implementación de sistemas anticheat basados en inteligencia artificial es una tendencia creciente, efectiva y necesaria frente a la evolución de las trampas. La literatura apoya la necesidad de soluciones híbridas, que integren técnicas tradicionales con modelos inteligentes, adaptativos y éticos. Esto fundamenta el desarrollo del presente proyecto, que busca construir un sistema anticheat basado en IA, con foco específico en el entorno competitivo de Counter-Strike.

Capítulo 3

Objetivos del Trabajo

3.1. Objetivo General

Desarrollar un sistema anticheat basado en inteligencia artificial para detectar trampas en Counter-Strike, contribuyendo a una experiencia de juego justa y segura, mediante la aplicación de modelos de aprendizaje automático e implementación en una arquitectura cliente-servidor funcional.

3.2. Objetivos Específicos

- Analizar el estado actual de los sistemas anticheat aplicados en Counter-Strike, identificando sus limitaciones técnicas y riesgos asociados a la privacidad del usuario, para fundamentar el diseño del sistema.
- Diseñar y entrenar un modelo de inteligencia artificial basado en aprendizaje automático supervisado, capaz de detectar patrones anómalos asociados a trampas (cheats) mediante el uso de datos simulados o reales, con un rendimiento mínimo esperado de un 90 % de precisión en entornos controlados.
- Implementar un prototipo funcional del sistema en una arquitectura cliente-servidor, que permita la detección de trampas en tiempo real y cuya interfaz cumpla con criterios de usabilidad, así como con las normativas ISO/IEC 27001, 27002 y 27701, en el plazo máximo de dos semestres.

Capítulo 4

Metodología

4.1. Dataset

Para realizar el experimento fue utilizado el dataset de HuggingFace llamado Çounter-Strike 2 Cheat Detection", este conjunto de datos proviene de partidas de Counter-Strike 2, con el objetivo de detectar comportamientos de tipo trampa a partir de las señales de movimiento, disparo y estado del jugador en cada tick del juego. Cada registro corresponde a un instante concreto para un jugador, e incluye información como posición en el mapa, velocidades, apuntado, estado del arma y estadísticas de combate. Al leer y preprocesar el dataset, se filtran únicamente las columnas mas relevantes para el análisis y modelado. A continuación, se describen brevemente cada una de ellas.

X, Y, Z Posición tridimensional del jugador en el mapa.

tick Numero de tick o iteración dentro de la partida, correspondiendo a un instante breve.

steamid Identificador anónimo del jugador.

velocity_X, velocity_Y, velocity_Z Componentes de la velocidad del jugador en cada eje.

is_airborne, is_walking Indicadores si el jugador esta en el aire o caminando en ese tick.

yaw, pitch Angulos de orientación del jugador, siendo yaw un indicador donde mira en rotación horizontal y pitch sobre la rotación vertical.

usercmd_mouse_dx, usercmd_mouse_dy Entrada del ratón en ese tick.

active_weapon Identificador numérico o índice de arma que el jugador tiene activa en ese momento.

active_weapon_ammo, total_ammo_left Cantidad de munición restante en el arma actual y munición total restante para esa arma.

shots_fired, kills_total, deaths_total Estadísticas acumuladas por jugador en la partida.

headshot_kills_total Número acumulado de muertes por disparo a la cabeza.

health, armor_value, is_alive Con esta columnas se comprueba si el jugador recibe daño correctamente al ser disparado.

Capítulo 5

Desarrollo

5.1. Análisis de datos

5.1.1. Distribución de clases (`is_cheater`)

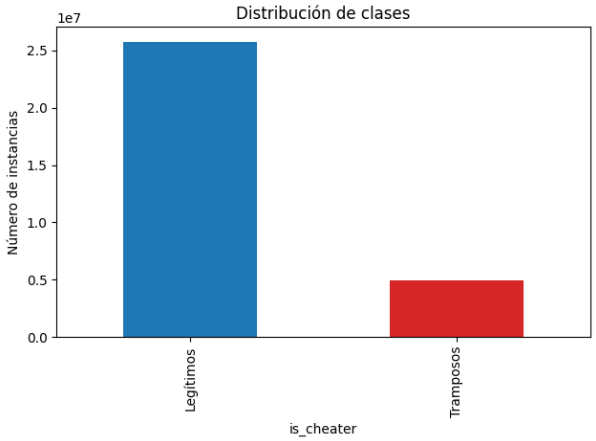

Figura 5.1: Distribución de clases: Legítimos vs. Tramposos

- **Legítimos (clase 0):** aproximadamente 25.7 millones de filas (84 %).
- **Tramposos (clase 1):** aproximadamente 4.96 millones de filas (16 %).

Hay un claro desbalance: la mayoría de los ticks corresponden a jugadores legítimos, lo cual penaliza modelos que no compensen la clase minoritaria.

5.1.2. Distribución de velocidad (magnitud)

La mayoría de los registros tienen velocidad cercana a 0 (jugador quieto o movimientos muy lentos). Aparece una cola larga que llega hasta valores > 300 (movimientos muy rápidos), probablemente debidos a saltos, caídas o teletransportes (o artefactos de registro). Conviene acotar el rango (por ejemplo, $[-5, 5]$ m/s) si queremos modelar comportamiento "normalz tratar los outliers por separado.

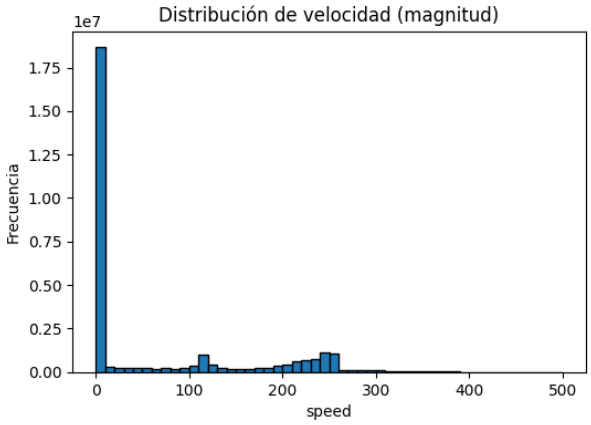

Figura 5.2: Distribución de la magnitud de velocidad

5.1.3. Boxplot de `headshot_kills_total` por clase

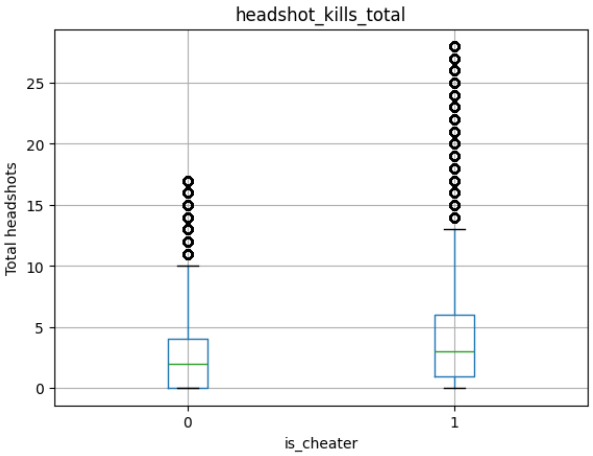

Figura 5.3: Boxplot de headshot_kills_total por clase

Mediana de headshots acumulados por tick:

- Legítimos: 2 headshots.
- Tramposos: 3 headshots.

Rango intercuartílico (IQR): mayor dispersión en tramposos (IQR más ancho). Atípicos: existen casos extremos con más de 15–20 headshots en un solo tick (muy improbable en juego normal).

5.1.4. Mapa de calor de correlaciones

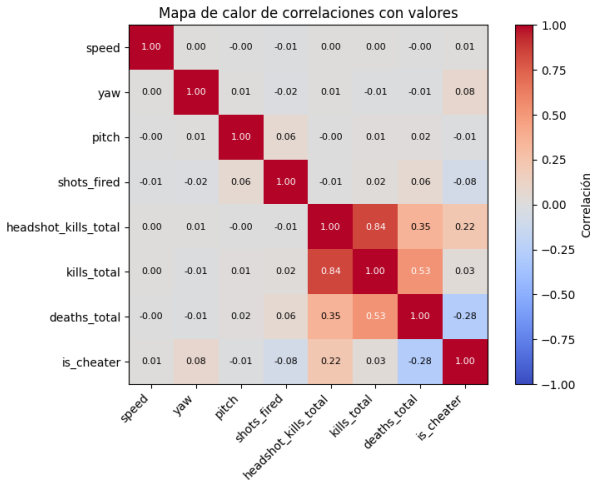

Figura 5.4: Mapa de calor de correlaciones entre variables

- `headshot_kills_total` y `kills_total` están fuertemente correlacionados (color rojo intenso).
- `kills_total` también correlaciona positivamente con `shots_fired` y en menor medida con `speed`.
- `is_cheater` muestra ligera correlación positiva con `kills_total` y `headshot_kills_to` (0.2–0.3), y pequeña correlación negativa con `deaths_total` (–0.2), indicando que los tramposos tienden a morir menos.
- Variables como `yaw`, `pitch` o `speed` apenas se correlacionan directamente con la etiqueta (valores cercanos a 0), lo que sugiere construir características más específicas (por ejemplo, saltos bruscos en ángulo).

5.1.5. Distribución de Δ yaw por tick

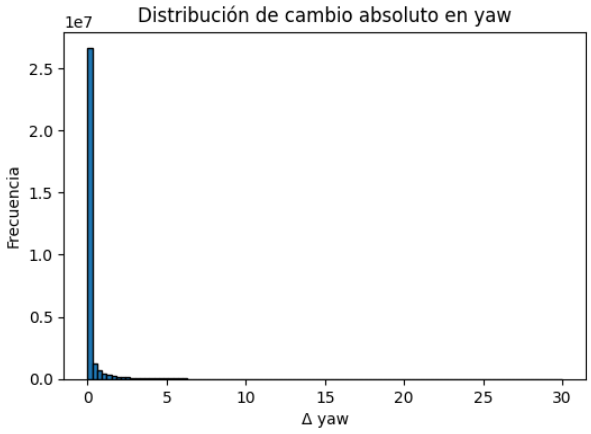

Figura 5.5: Distribución de los cambios absolutos de yaw por tick

La mayoría de cambios absolutos de yaw son muy pequeños (<1°) por tick. Aparecen outliers con Δ yaw de hasta 30° o más, probablemente debido a "snaps"de aimbot o

correcciones de cámara.

5.1.6. Boxplots de yaw y pitch por clase

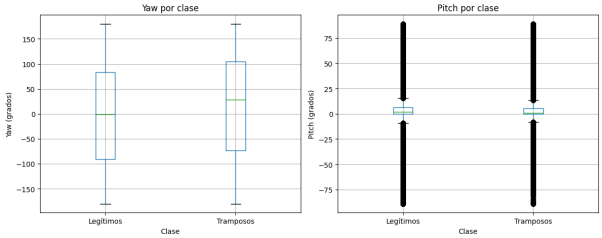

Figura 5.6: Boxplot de yaw por clase

- **Yaw:**
  - Tramposos: mediana positiva (+30°) frente a medianas negativas para legítimos (–5°).
  - IQR de tramposos desplazado hacia ángulos mayores, indicando giros más orientados al objetivo.
- **Pitch:**
  - Ambas clases tienen medianas muy cercanas (+2° a +3°), pero tramposos muestran IQR más ancho y más outliers extremos (±90°), reflejando "snaps"verticales bruscos.

5.1.7. Estadísticos descriptivos

	mean	50 %	std	min	max
X	-103.	-152.	1170.	-2652.	3619.
Y	504.	529.	1418.	-2603.	3912.
Z	-54.	-63.	152.	-368.	701.
tick	50733	42596	38137	1	194737
speed	90.9	0	1958.8	0	303373.5

Tabla 5.1: Estadísticos descriptivos de variables seleccionadas.

Media de velocidad (91 unidades) no es muy informativa por la cola de outliers; la mediana (0) indica que el jugador está parado la mayoría del tiempo. Coordenadas (X,Y,Z) tienen media cerca de cero (centro del mapa) pero gran desviación estándar (jugadores repartidos por todo el terreno). El `tick` llega hasta 200000, evidenciando la duración de cada partida en instantes de juego.

5.1.8. Estadísticas medias por clase

is_cheater	speed	headshot_kills_total	shots_fired
0	86.21	2.62	0.51
1	114.89	4.60	0.04

Tabla 5.2: Estadísticas medias de velocidad, headshots y disparos por clase.

Tramposos se mueven algo más rápido (media 115 vs 86) y hacen más headshots (4.6 vs 2.6). Curiosamente, disparan menos (shots_fired 0.04 vs 0.51), lo que podría indicar ráfagas muy rápidas concentradas en headshots con intervalos sin input de disparo.

5.1.9. Tasa de tramposos por mapa

map	Tasa promedio de is_cheater
de_edin	0.90
de_train	0.62
de_basalt	0.50
cs_office	0.43
de_dust2	0.09
de_inferno	0.08
de_mirage	0.07
de_anubis	0.00

Tabla 5.3: Proporción media de tramposos por mapa.

Los mapas `de_edin` y `de_train` muestran las tasas de tramposos más elevadas, con aproximadamente 90 % y 62 % respectivamente. Mapas muy populares como `de_dust2` presentan una incidencia relativamente baja de tramposos (9 %), mientras que otros escenarios como `de_basalt` y `cs_office` tienen valores intermedios. El caso de `de_anubis` refleja ausencia de trampas en los datos muestreados (0 %), aunque puede tratarse de un artefacto de menor número de partidas cargadas. Estos resultados sugieren que es recomendable segmentar los modelos de detección según mapa o desplegar contramedidas específicas en servidores donde la incidencia de cheats sea especialmente alta.

5.2. Diseño de experimentos

Para evaluar y comparar distintos enfoques de detección, planteamos dos líneas de experimentación:

1. Experimentos con algoritmos clásicos

- **Partición de datos:** hold-out 75 % entrenamiento / 25 % prueba y validación cruzada de 5 folds en el conjunto de entrenamiento.

- **Modelos candidatos:** Random Forest, SVM (kernel RBF), XGBoost y MLP clásico.
- **Selección de hiperparámetros:** grid search sobre parámetros clave (n_estimators, profundidad en RF; C y gamma en SVM; learning rate y capas en MLP).
- **Técnicas de balanceo:** uso de pesos de clase en la función de costo y submuestreo / sobremuestreo de la clase minoritaria.

2. Experimentos con Deep Learning

- **Arquitecturas propuestas:**
  - *Redes Neuronales Recurrentes:* LSTM o GRU de 2–3 capas para capturar secuencias temporales de ticks.
  - *Redes Convolucionales 1D:* extraer patrones locales en la serie temporal de telemetría.
  - *Modelos híbridos (CNN+LSTM):* usar capas convolucionales para reducción de dimensionalidad seguida de LSTM para dependencia temporal.
- **Entrenamiento:**
  - Optimización con Adam y tasa de aprendizaje inicial de $1e^{-3}$, reducción automática de lr con *learning rate scheduler* y *early stopping* en validación.
  - Uso de *batch normalization* y *dropout* (0.3–0.5) para regularización.
  - Entrenamiento en GPU, con tamaño de batch entre 64 y 256.
- **Balanceo e imbalanced loss:** entrenamiento con *weighted cross-entropy* o *focal loss* para mitigar desbalance de clases.
- **Validación:** split hold-out y validación cruzada estratificada para evaluar robustez.

Métricas de evaluación

Para todas las pruebas registraremos al menos cuatro métricas clave:

- **Accuracy:** $\frac{TP + FP}{TP + TN + FP + FN}$, proporción de predicciones correctas.
- **Precision:** $\frac{TP}{TP + FP}$, indica cuántos positivos predichos son verdaderos.
- **Recall:** $\frac{TP}{TP + FN}$, mide la capacidad de detectar todos los tramposos.
- **F1-score:** $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$, equilibrio entre precision y recall.

Capítulo 6

Resultados

Los resultados dan cuenta de la puesta en marcha de lo desarrollado durante el proyecto.

Capítulo 7

Discusión

Los resultados son analizados y se comparan con resultados previos en lo práctico o teórico (bibliografía).

Capítulo 8

Conclusiones

Realizar la conclusión referenciando el resultado de abordar cada objetivo del proyecto.

Capítulo 9

Limitaciones del trabajo y trabajos futuros

Bibliografía

[1] Collins, S., Poulopoulos, A., Muench, M., & Chothia, T. (2024). Anti-Cheat: Attacks and the Effectiveness of Client-Side Defences. *CheckMATE '24: Proceedings Of The 2024 Workshop On Research On Offensive And Defensive Techniques In The Context Of Man At The End (MATE) Attacks*, 30-43. https://doi.org/10.1145/3689934.3690816

[2] Karkallis, P., & Blasco, J. (2025, 17 febrero). VIC: Evasive Video Game Cheating via Virtual Machine Introspection. *arXiv.org*. https://arxiv.org/abs/2502.12322

[3] Smith, J., Ahmed, H., Burrough, M., Britland, I., & Mertkolu, B. (2024). The Psychology and Impact of Cheating. *DigitalCommons@Kennesaw State University*. https://digitalcommons.kennesaw.edu/engl1102/17/

[4] Chen, M. (2024). AI cheating versus AI anti-cheating: A technological battle in game. *Applied And Computational Engineering*, 73(1), 222-227. https://doi.org/10.54254/2755-2721/73/20240402

[5] Daniel, F. A. (2019, 1 marzo). Detección de trampas en el videojuego CS:GO utilizando técnicas de Inteligencia Artificial. https://e-archivo.uc3m.es/entities/publication/341c2df2-9f87-4dla-a4de-c19d90d24747

[6] Maberry, K. (2016). Using an Artificial Neural Network to detect aim assistance in Counter-Strike: Global Offensive. https://www.cs.nmt.edu/kmaberry/ann_fps_cheater.pdf

[7] Nie, B., & Ma, B. (2024). VADNet: Visual-Based Anti-Cheating Detection Network in FPS Games. *Traitement Du Signal*, 41(1), 431-440. https://doi.org/10.18280/ts.410137

[8] Pinto, J. P., Pimenta, A., & Novais, P. (2021). Deep learning and multivariate time series for cheat detection in video games. *Machine Learning*, 110(11-12), 3037-3057. https://doi.org/10.1007/s10994-021-06055-x

[9] Liu, D., Gao, X., Zhang, M., Wang, H., & Stavrou, A. (2017). Detecting Passive Cheats in Online Games via Performance-Skillfulness Inconsistency. *2017 47th Annual IEEE/IFIP International Conference On Dependable Systems And Networks (DSN)*, 615-626. https://doi.org/10.1109/dsn.2017.20

[10] Loo, M. M. Z., & Lužkov, G. (2025). CS2CD.Counter-Strike_2_Cheat_Detection (Revision 44e5129) [Conjunto de datos]. Hugging Face. https://doi.org/10.57967/hf/5654