Entrenamiento de Agentes Inteligentes mediante Aprendizaje por Refuerzo en ViZDoom

Proyecto Final
Teoría de Aprendizaje de Máquina (TAM)

Universidad Nacional de Colombia
Sede Manizales – 2025

Integrantes:
Juan Jerónimo Castaño Rivera
Oscar Andrés Gutiérrez Estepa
Wilmer Sebastián Pérez Cuastumal

Profesor: Andrés Marino Álvarez Meza

Índice

1. Motivación del Proyecto

El avance de los sistemas inteligentes ha generado un creciente interés por el desarrollo de agentes autónomos capaces de aprender a interactuar con entornos complejos sin supervisión directa. En este contexto, el *aprendizaje por refuerzo* (RL, por sus siglas en inglés) se ha posicionado como un paradigma fundamental dentro del aprendizaje automático, al permitir que un agente mejore su comportamiento mediante la interacción continua con el entorno y la retroalimentación basada en recompensas.

El problema general que se busca abordar es el entrenamiento efectivo de agentes inteligentes en entornos parcialmente observables y de alta complejidad visual, donde no existe una supervisión directa ni una señal clara de cuál debe ser la acción óptima en cada instante. Este tipo de problema es representativo de situaciones reales como la navegación autónoma, el control de robots, los videojuegos, la planificación de tareas en ambientes dinámicos, entre otros.

La plataforma ViZDoom, basada en el motor del videojuego Doom, ofrece un entorno de simulación en primera persona que emula condiciones realistas como percepción visual en tiempo real, toma de decisiones secuenciales, presencia de enemigos, recursos limitados y entornos laberínticos. Estas características lo convierten en un excelente banco de pruebas para algoritmos de aprendizaje por refuerzo profundo (Deep RL), permitiendo explorar arquitecturas modernas como PPO (Proximal Policy Optimization) o DQN (Deep Q-Networks).

La aplicación de técnicas de RL en ViZDoom resulta relevante desde múltiples perspectivas. Técnicamente, permite evaluar la capacidad de los modelos para aprender estrategias óptimas en entornos de alta dimensionalidad y retardo en las recompensas. Científicamente, facilita la comprensión del proceso de exploración-explotación y la transferencia de políticas aprendidas. Desde un enfoque social y económico, la investigación en RL contribuye al desarrollo de tecnologías aplicables en seguridad, automatización industrial, videojuegos educativos y asistencia autónoma en entornos peligrosos o inaccesibles para humanos.

Por estas razones, este proyecto propone el diseño, entrenamiento y evaluación de agentes inteligentes en escenarios del entorno ViZDoom, con el objetivo de analizar su desempeño y extraer conclusiones sobre la eficacia y limitaciones de las técnicas modernas de aprendizaje por refuerzo profundo.

2. Planteamiento del Problema

El presente proyecto se enfoca en resolver el problema específico de entrenar agentes inteligentes que logren aprender políticas efectivas en entornos tridimensionales visuales, con observaciones en primera persona, recompensas escasas y dinámica en tiempo real. Particularmente, se busca que los agentes aprendan a sobrevivir y maximizar la recompensa acumulada en escenarios del entorno ViZDoom mediante técnicas de aprendizaje por refuerzo profundo.

La pregunta de investigación que guía este trabajo puede formularse de la siguiente manera:

> **¿Qué tan efectivas son las políticas aprendidas por agentes entrenados mediante algoritmos de aprendizaje por refuerzo profundo, como PPO o DQN, para resolver tareas de combate, navegación o evasión en entornos visuales parcialmente observables como los que ofrece ViZDoom?**

El proyecto aborda este reto computacional a través del entrenamiento de agentes en dos escenarios:

- **Escenario *basic*:** un entorno de combate simple incluido por defecto en ViZDoom, donde el agente debe eliminar a un enemigo frente a él disparando con precisión y evitando permanecer inactivo.
- **Escenario *custom*:** un entorno diseñado manualmente que simula un escenario real del juego Doom, que incorpora enemigos múltiples, obstáculos y una estructura tipo laberinto, incrementando la complejidad de la tarea y los desafíos de exploración. En este entorno se pretende evaluar la capacidad del agente para generalizar, planificar rutas y tomar decisiones bajo incertidumbre.

Ambos escenarios proveen imágenes visuales como observaciones (frames de 160×120 píxeles, pudiendo ser tanto en RGB como en escala de grises), y la recompensa está determinada por eventos como daño infligido al enemigo, frags logrados, tiempo de supervivencia, penalización por recibir daño o por inactividad, etc.

La solución se evalúa dentro del dominio del control de agentes en videojuegos 3D, enmarcados en la investigación aplicada de aprendizaje por refuerzo en entornos complejos, sin acceso directo a señales de supervisión externas ni modelos explícitos del entorno.

3. Estado del Arte

El uso de aprendizaje por refuerzo profundo (Deep Reinforcement Learning, DRL) en videojuegos se ha convertido en un campo fértil de experimentación para el desarrollo de agentes inteligentes. En particular, ViZDoom ha sido adoptado como un entorno de referencia para probar agentes que deben aprender desde observaciones visuales en entornos 3D complejos. A continuación, se presentan trabajos relacionados:

Trabajo 1: "Training an AI-Powered Doomguy" (Boyi Xiao, 2023)

En este estudio, el autor entrenó un agente inteligente usando el algoritmo PPO (Proximal Policy Optimization) en distintos escenarios de ViZDoom, incluyendo "basic", "defend

the center" y "deathmatch". El trabajo destaca la importancia de estrategias como *reward shaping*, *frame skipping* y *curriculum learning* para mejorar la eficiencia del entrenamiento. También introduce una arquitectura convolucional personalizada con reducción de parámetros innecesarios y uso de imágenes borrosas para reducir el sobreajuste. Una contribución notable fue la definición de un espacio de acciones discretas inteligentes que evita combinaciones irrelevantes. Aunque se lograron buenos resultados contra bots, el agente presentó dificultades frente a tareas de exploración estratégica más complejas.

Enlace: `https://doi.org/10.2991/978-94-6463-370-2_26`

Trabajo 2: "ViZDoom Competitions" (Wydmuch et al., IEEE ToG 2018)

Este artículo documenta las ediciones 2016 y 2017 de la Visual Doom AI Competition (VDAIC), donde equipos presentaron agentes DRL capaces de jugar *deathmatch* multijugador desde píxeles. Los mejores agentes utilizaron arquitecturas como A3C, DRQN o Direct Future Prediction (DFP), combinadas con técnicas de entrenamiento como *curriculum learning* o aprendizaje jerárquico. Aunque algunos bots superaron a los enemigos controlados por el motor del juego, la mayoría no logró acercarse al rendimiento humano, especialmente en mapas no vistos o escenarios que requerían memoria, planificación a largo plazo o evasión estratégica.

Enlace: `https://arxiv.org/abs/1809.03470`

Trabajo 3: "Arnold y IntelAct" – Agentes del VDAIC

Entre los agentes destacados de la VDAIC se encuentra **Arnold**, que integró módulos independientes de navegación (DQN) y de puntería (DRQN), utilizando un clasificador binario para decidir cuál red debía actuar. Por su parte, **IntelAct** aplicó el algoritmo DFP, que predice medidas futuras del estado (como salud o munición) en lugar de la recompensa directamente, lo que permite ajustar el comportamiento del agente sin reentrenamiento.

4. Objetivo del Proyecto

Diseñar, implementar y evaluar agentes inteligentes entrenados mediante técnicas de aprendizaje por refuerzo profundo (Deep Reinforcement Learning, DRL) en escenarios visuales tridimensionales del entorno ViZDoom. Se busca que los agentes aprendan a interactuar eficazmente con el entorno utilizando únicamente información visual, tomando decisiones secuenciales que maximicen la recompensa acumulada en tareas de combate y navegación.

Objetivos específicos

- Entrenar un agente en el escenario *basic* de ViZDoom utilizando el algoritmo PPO (Proximal Policy Optimization), evaluando su capacidad de aprender una política eficiente para eliminar enemigos con precisión y rapidez.
- Diseñar un escenario *custom* en ViZDoom con mayor complejidad estructural (como múltiples enemigos, obstáculos y mapas laberínticos), con el fin de analizar la capacidad de generalización y exploración del agente en entornos no triviales.
- Comparar el desempeño del agente en ambos escenarios en función de métricas como número de frags, tasa de precisión, recompensa media por episodio y tiempo de supervivencia, discutiendo fortalezas y limitaciones de la estrategia utilizada.

5. Fundamentos Teóricos

5.1. Aprendizaje por Refuerzo (RL)

El aprendizaje por refuerzo (*Reinforcement Learning*, RL) es un paradigma del aprendizaje automático en el cual un agente aprende a tomar decisiones mediante interacción con un entorno, recibiendo recompensas o penalizaciones. El objetivo del agente es maximizar la recompensa acumulada a largo plazo.

Este proceso se modela formalmente mediante un **proceso de decisión de Markov** (MDP), definido como el cuarteto $(\mathcal{S}, \mathcal{A}, P, R)$ donde:

- $\mathcal{S}$ es el conjunto de estados.
- $\mathcal{A}$ es el conjunto de acciones posibles.
- $P(s'|s, a)$ es la probabilidad de transición al estado s' dado el estado actual s y la acción a.
- $R(s, a)$ es la recompensa esperada al tomar la acción a en el estado s.

El comportamiento del agente está determinado por una política $\pi(a|s)$, que define la probabilidad de tomar una acción a en un estado s. El objetivo del agente es encontrar una política óptima π^* que maximice el retorno esperado a largo plazo:

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

donde $\gamma \in [0, 1]$ es el factor de descuento.

5.2. Deep Q-Network (DQN)

DQN es una combinación del algoritmo clásico Q-learning con redes neuronales profundas para aproximar la función de acción-valor $Q(s, a)$:

$$Q^{\pi}(s, a) = E\left[G_t \,|\, s_t = s, a_t = a, \pi\right]$$

En Q-learning, la actualización de la función Q se realiza con la siguiente regla de Bellman:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

En DQN, $Q(s, a; \theta)$ se aproxima con una red neuronal con parámetros θ. La red es entrenada minimizando el error cuadrático entre la predicción y el valor objetivo y:

$$L(\theta) = E_{(s,a,r,s')}\left[(y - Q(s, a; \theta))^2\right]$$
$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

donde θ^- son los parámetros de una red objetivo congelada, que se sincroniza periódicamente con la red principal para estabilizar el entrenamiento.

5.3. Dueling DQN

Dueling DQN introduce una arquitectura en la que la red se divide en dos ramas internas después de la parte convolucional:

- Una que estima el **valor del estado** $V(s)$.
- Otra que estima la **ventaja** relativa de cada acción $A(s, a)$.

La función Q se reconstruye a partir de estas dos estimaciones:

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a')\right)$$

Esta descomposición mejora la estabilidad y eficiencia del entrenamiento, especialmente en entornos donde algunas acciones tienen poca influencia en el valor del estado.

5.4. Proximal Policy Optimization (PPO)

PPO es un algoritmo basado en la familia actor-crítico. Consiste en dos redes:

- El **actor** estima la política $\pi_\theta(a|s)$.
- El **crítico** estima el valor del estado $V_w(s)$.

El objetivo del actor es maximizar la siguiente función de ventaja:

$$L^{CLIP}(\theta) = E_t \left[\text{mín} \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

donde $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ es el ratio de probabilidad entre la política nueva y la anterior, y $\hat{A}_t$ es la estimación de la ventaja generalizada (GAE):

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

PPO incorpora también un término de entropía para fomentar la exploración y un límite de actualización para garantizar que la nueva política no se aleje demasiado de la anterior, lo que mejora la estabilidad.

6. Configuración Experimental

El presente proyecto tiene como objetivo entrenar y evaluar agentes de aprendizaje por refuerzo profundo en entornos tridimensionales de tipo videojuego, utilizando la plataforma **ViZDoom**. Este entorno permite simular tareas visuales complejas en tiempo real, a partir de imágenes en primera persona derivadas del motor clásico de Doom.

Se seleccionaron escenarios diseñados específicamente para tareas de control y combate simples, con recompensas bien definidas. Particularmente, se utilizó el escenario `Basic` como punto de partida para validar los modelos implementados, debido a su naturaleza controlada y estructura clara de recompensas.

Los experimentos se ejecutaron en un entorno de cómputo basado en notebooks de Kaggle, con acceso a una GPU (GPU Tesla P100), lo que permitió realizar múltiples millones de pasos de entrenamiento. Todo el código fue desarrollado en Python 3, utilizando PyTorch como framework principal para definir y entrenar los modelos de redes neuronales, debido a su flexibilidad y su compatibilidad con CUDA (GPU).

Durante la ejecución, se realizaron evaluaciones periódicas del desempeño del agente, mediante grabaciones de video y almacenamiento de checkpoints del modelo para facilitar análisis posteriores.

Además, se incorporaron estrategias de *reward shaping*, selección discreta de acciones y apilamiento de frames para mejorar el aprendizaje a partir de señales visuales ruidosas. Las observaciones del entorno fueron preprocesadas y se usaron como entrada a modelos convolucionales tanto para algoritmos tipo DQN como para PPO.

7. Metodología Propuesta

7.1. Escenario 1: `VizdoomBasic-v0`

Figura 1: Escenario Basic

El primer entorno utilizado para entrenar y evaluar al agente es el escenario predefinido `VizdoomBasic-v0`, uno de los más simples disponibles en la plataforma ViZDoom. Este escenario tiene como propósito principal validar si es factible entrenar un agente de inteligencia artificial en un entorno 3D usando aprendizaje por refuerzo profundo.

Descripción del escenario

El mapa consiste en una sala rectangular con paredes, piso y techo grises. El jugador aparece centrado en una de las paredes largas, mientras que un enemigo (un monstruo rojo de forma circular) aparece aleatoriamente sobre la pared opuesta. El agente puede realizar únicamente tres acciones: moverse a la izquierda, moverse a la derecha y disparar. Un solo disparo acertado es suficiente para eliminar al enemigo. El episodio termina cuando el enemigo es eliminado o se alcanza un límite de tiempo de 300 *tics* (pasos).

Recompensas y penalizaciones

- **+106** puntos por eliminar al enemigo.
- **-5** puntos por cada disparo realizado (sin importar si acierta).
- **+1** punto por cada *tic* que el agente permanece con vida.

Condición de término del episodio: eliminación del enemigo o tiempo agotado (300 tics).

Configuración técnica del entorno

- **Acciones disponibles:** mover a la izquierda, mover a la derecha, disparar.
- **Variable de juego disponible:** cantidad de munición.
- **Formato de observación:** imagen RGB de tamaño 240×320, posteriormente convertida a escala de grises y redimensionada a 84×84.
- **Motor de entorno:** `VizdoomBasic-v0` con wrapper personalizado en `Gymnasium`.

Estrategia de entrenamiento

Para el entrenamiento se utilizó una red Dueling DQN (Dueling Deep Q-Network), implementada en PyTorch. Esta arquitectura permite descomponer la estimación del valor-acción $Q(s, a)$ en dos componentes: el valor del estado $V(s)$ y la ventaja relativa $A(s, a)$.

Especificaciones del modelo:

- Tres capas convolucionales seguidas por dos ramas densas independientes (valor y ventaja).
- Entrada: pila de 4 frames preprocesados ($4 \times 84 \times 84$).
- Salida: valores Q para cada acción.

Configuración experimental

- **Plataforma de entrenamiento:** Kaggle Notebooks con GPU NVIDIA Tesla P100.
- **Librerías:** PyTorch, OpenCV, VizDoom v1.2.4, Gymnasium v0.28.1.

- **Exploración:** estrategia ϵ-greedy con decaimiento exponencial ($\epsilon_0 = 1{,}0$, $\epsilon_{min} = 0{,}1$, decaimiento = 10,000 pasos).
- **Tamaño del batch:** 32.
- **Memoria de experiencia:** buffer circular de 50,000 transiciones.
- **Red objetivo:** actualizada cada 1,000 pasos.
- **Pérdida:** error cuadrático medio (MSE) entre Q evaluado y objetivo.
- **Optimizador:** Adam, tasa de aprendizaje 1×10^{-4}.
- **Factor de descuento γ:** 0.99.
- **Episodios de entrenamiento:** 500.

Evaluación y visualización

Tras el entrenamiento, se evaluó el agente ejecutando un episodio de prueba en modo determinista, generando un video con el desempeño y las decisiones tomadas en tiempo real. Además, se trazó la curva de recompensa acumulada por episodio para observar la evolución del aprendizaje.

Código: `https://github.com/juacastanori/TAM/blob/main/ViZDoom_Project/ViZDoom_BasicV0_ph1.ipynb`

Problemas identificados y segundo entrenamiento

Durante el análisis del primer entrenamiento, se identificó que el agente presentaba dificultades notorias cuando el enemigo era generado en posiciones cercanas al borde derecho del mapa. En estos casos, el agente tenía baja probabilidad de girar y disparar con precisión a tiempo. Esto sugiere una política sesgada o con capacidad limitada de exploración.

Para abordar esta limitación, se implementó una segunda fase de entrenamiento, partiendo del modelo previamente entrenado. Esta nueva etapa utilizó el mismo entorno, arquitectura y configuración base, pero extendió el número de episodios a 1000 y reutilizó los pesos aprendidos como punto de partida. La red fue cargada desde un archivo '.pth', y se continuó el entrenamiento con una política ϵ-greedy que permitió una mejor exploración del espacio de acciones.

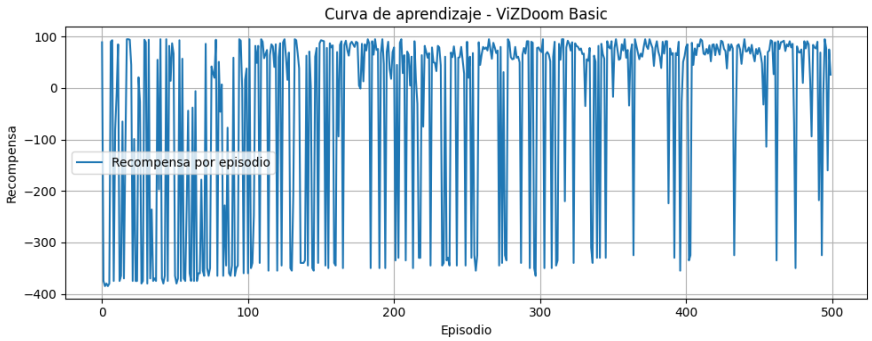

Figura 2: Curva de recompensa por episodio – Entrenamiento inicial en escenario `basic`.

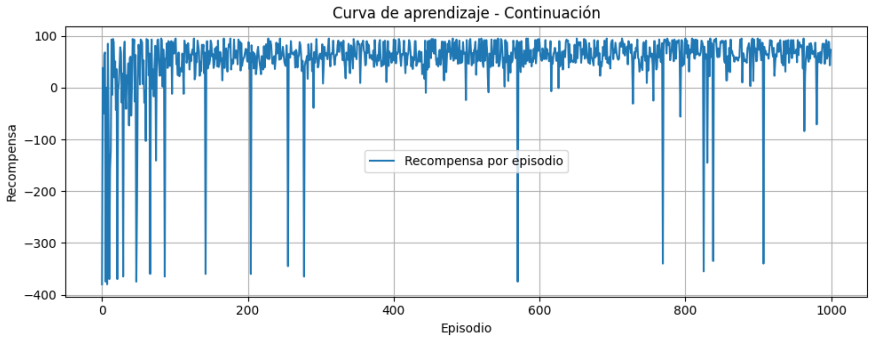

Figura 3: Curva de recompensa por episodio – Segunda fase de entrenamiento con modelo precargado.

Los resultados muestran una mejora significativa en la estabilidad del entrenamiento y en la recompensa media acumulada. En la fase inicial, los episodios exitosos eran irregulares y presentaban alta varianza; mientras que en la segunda fase, se observó una tendencia creciente más estable, indicando que el agente logró superar las deficiencias en situaciones laterales.

Repositorio y demostración

El código completo de ambas fases de entrenamiento, así como el modelo y video evaluación, se encuentra disponible en el siguiente enlace:

- **Código:** `https://github.com/juacastanori/TAM/blob/main/ViZDoom_Project/ViZDoom_BasicV0_ph2.ipynb`
- **Video de demostración:** `https://youtu.be/Xyk-PK5OxaQ`

7.2. Escenario 2: `VizdoomDefendLine-v0`

Figura 4: Escenario Defend The Line

Este escenario corresponde a la versión conocida como “Defend The Line” en ViZDoom. Es un escenario defensivo donde el agente permanece estático y debe eliminar enemigos que aparecen a lo lejos, antes de que logren infligirle daño. Se trata de un entorno clásico de visión en primera persona con tareas de disparo rápido y priorización de objetivos

Acciones disponibles

El espacio de acciones es discreto y corresponde a lo que define ViZDoom al crear el entorno, con botones como:

- `ATTACK`
- `TURN_LEFT`
- `TURN_RIGHT`

Observaciones: Captura y Preprocesamiento de Imágenes

El entorno `VizdoomDefendLine-v0` se inicializa con el modo RGB, lo cual permite capturar imágenes RGB de cada frame del juego.

Estas imágenes se preprocesan mediante un wrapper personalizado llamado `GrayResize`, definido como una subclase de `gymnasium.ObservationWrapper`. El procesamiento consta de los siguientes pasos:

1. Se accede al frame.

2. Se convierte a escala de grises.
3. Se redimensiona a tamaño fijo de 84×84 píxeles mediante interpolación bilineal.

Posteriormente, se aplica el wrapper `FrameStack` con $k = 4$, lo que permite apilar los últimos 4 frames consecutivos a lo largo del eje de canales. El agente recibe así una entrada de forma $(4, 84, 84)$, que representa la dinámica del entorno en el tiempo (como una imagen en movimiento).

Reward Shaping aplicado

El entorno incluye una capa de reward shaping mediante un wrapper personalizado que modifica la recompensa devuelta en cada paso con base en variables internas del juego.

Las variables clave utilizadas son:

- `HEALTH` – Nivel de salud del agente.
- `FRAGCOUNT` – Número de enemigos eliminados por el agente.

El cálculo de la recompensa ajustada es el siguiente:

$$r' = r + k_{\text{hp}} \cdot \Delta\text{HP} + \text{penalización por disparo fallido} \tag{1}$$

Donde:

- r es la recompensa original del entorno.
- ΔHP representa el daño recibido.
- k_{hp} es el peso del castigo por daño (por defecto $-0{,}1$).
- Si el agente presiona el botón `ATTACK` pero no incrementa su `FRAGCOUNT`, se le aplica una penalización adicional k_{shot} (por defecto $-0{,}05$).

Esta estrategia promueve que el agente aprenda a disparar con precisión y a evitar ser dañado, mejorando la eficiencia del entrenamiento.

Arquitectura del Modelo y Proceso de Entrenamiento

La arquitectura implementada es una red neuronal convolucional (CNN) empleada dentro del marco de un Deep Q-Network (DQN). Esta red está definida en la clase `DoomDQN` y se compone de dos bloques principales:

- **Extractor convolucional de características**:
  - `Conv2d(4, 32, kernel_size=8, stride=4)` seguido de `ReLU`
  - `Conv2d(32, 64, kernel_size=4, stride=2)` seguido de `ReLU`
  - `Conv2d(64, 64, kernel_size=3, stride=1)` seguido de `ReLU`
- **Cabezal completamente conectado**:
  - Un vector aplanado de tamaño $7 \times 7 \times 64 = 3136$ se conecta a una capa densa con activación `ReLU`
  - Finalmente, se conecta a una capa de salida que devuelve los valores $Q(s, a)$ para cada acción disponible.

El agente se entrena usando el algoritmo clásico de **Deep Q-Learning** con repetición de experiencias y una red objetivo. El procedimiento sigue los siguientes pasos:

- **Memoria de repetición (Replay Buffer)**: Se almacenan transiciones en un buffer circular de tamaño 100,000. Se extraen mini-batches de tamaño 32.
- **Función de pérdida**: Se emplea la pérdida Huber (`Smooth L1`) entre los valores Q predichos y los valores objetivo.
- **Actualización del valor objetivo (Double DQN)**:

$$y = r + \gamma \cdot Q_{\text{target}}(s', \arg\max_{a} Q_{\text{online}}(s', a))$$

- **Optimizador**: Se usa Adam con tasa de aprendizaje 1×10^{-4}.
- **Factor de descuento**: $\gamma = 0{,}99$
- **Sincronización de red objetivo**: La red objetivo se sincroniza cada 10,000 pasos.

Se utiliza una estrategia de exploración ϵ-greedy con el siguiente calendario de decaimiento:

- Valor inicial: $\epsilon = 1{,}0$
- Valor mínimo: $\epsilon = 0{,}05$

- Decrecimiento lineal a lo largo de 1,000,000 pasos.

Cada 500,000 pasos, se guarda el modelo actual y se realiza una evaluación determinista del agente, grabando un episodio en formato MP4 mediante la función `record_checkpoint()`. Esta evaluación se realiza en un entorno aislado y se imprime la recompensa total obtenida.

Resumen de Hiperparámetros

Parámetro	**Valor**
Pasos totales de entrenamiento	5,000,000
Tamaño del replay buffer	100,000
Tamaño del mini-batch	32
Inicio del aprendizaje	10,000 pasos
Factor de descuento γ	0.99
Tasa de aprendizaje	1×10^{-4}
Frecuencia de sincronización	10,000 pasos
ϵ inicial	1.0
ϵ final	0.05
Pasos de decaimiento de ϵ	1,000,000
Forma de entrada	$(4, 84, 84)$
Norma máxima de gradiente	10

Cuadro 1: Resumen de hiperparámetros del entrenamiento.

Se obtuvieron las recompensas:

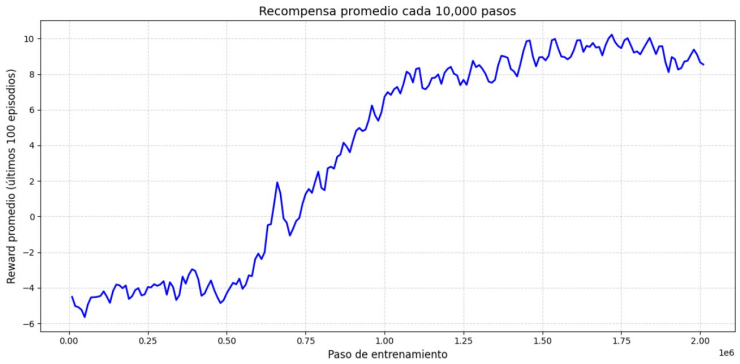

Figura 5: Curva por recompensa por episodio, escenario DefendTheLine

- **Código:** `https://github.com/juacastanori/TAM/blob/main/ViZDoom_Project/ViZDoom_Defendtheline.ipynb`
- **Video de demostración:** `https://youtu.be/onLg1Ou6Nks`

7.3. Escenario 3: Entorno personalizado

Para evaluar el rendimiento de un agente PPO entrenado desde cero, se utilizó un entorno personalizado derivado del trabajo de Kieliger [4]. El escenario `bots_deathmatch_multimaps.wad` consiste en un mapa tipo deathmatch con múltiples salas interconectadas, armas y pickups. Para esta implementación, primeramente se eliminaron todos los enemigos del entorno y se enfocó únicamente en recompensar al agente por explorar y recolectar objetos.

A continuación se muestra una imagen del mapa utilizado:

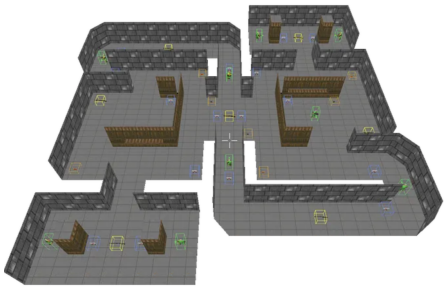

Figura 6: Mapa del entorno personalizado

El entorno fue implementado como un wrapper personalizado compatible con `Gym`, con observaciones de tipo imagen RGB de tamaño 160×120, y recompensas diseñadas cuidadosamente (reward shaping). Las recompensas contemplan eventos como: cambio de arma, recolección y uso de munición, daño infligido, mejoras en salud y armadura, desplazamiento efectivo en el mapa, y enemigos eliminados, para ser reutilizado para hacer el mismo entrenamiento, pero ahora agregando enemigos.

Para diseñar un entorno personalizado con técnicas de aprendizaje por refuerzo más efectivas, se consideraron tanto las señales del entorno (variables internas del juego) como un conjunto discreto de acciones posibles para el agente. A continuación, se describen las variables empleadas y las acciones disponibles.

Variables del Juego

El cuadro 2 presenta las variables del entorno ViZDoom que se monitorean en cada paso de simulación, utilizadas tanto para observar el estado del entorno como para calcular la recompensa modelada.

Variable	Descripción
HEALTH	Salud actual del agente. Valor entre 0 y 100.
ARMOR	Nivel actual de armadura del agente.
POSITION_X, POSITION_Y	Coordenadas espaciales del agente en el mapa.
DAMAGECOUNT	Daño total infligido por el agente.
KILLCOUNT	Número de enemigos eliminados.
SELECTED_WEAPON	ID del arma actualmente seleccionada.
AMMOi	Munición disponible para cada arma (índice $i \in [0, 9]$).
WEAPONi	Armas disponibles (poseídas) por el agente (índice $i \in [0, 9]$).

Cuadro 2: Variables de juego utilizadas en el entorno DoomShapedEnv.

Acciones Disponibles

El espacio de acción se definió como un conjunto discreto de 14 combinaciones específicas sobre seis botones de control: ATTACK, MOVE_FORWARD, TURN_LEFT, TURN_RIGHT, MOVE_LEFT y MOVE_RIGHT. Cada acción es un vector binario de tamaño 6.

#	Vector de acción	Descripción
0	[1, 0, 0, 0, 0, 0]	Atacar
1	[0, 1, 0, 0, 0, 0]	Avanzar
2	[0, 0, 1, 0, 0, 0]	Girar a la izquierda
3	[0, 0, 0, 1, 0, 0]	Girar a la derecha
4	[0, 0, 0, 0, 1, 0]	Moverse a la izquierda
5	[0, 0, 0, 0, 0, 1]	Moverse a la derecha
6	[0, 1, 1, 0, 0, 0]	Avanzar + Girar a la izquierda
7	[0, 1, 0, 1, 0, 0]	Avanzar + Girar a la derecha
8	[1, 0, 1, 0, 0, 0]	Atacar + Girar a la izquierda
9	[1, 0, 0, 1, 0, 0]	Atacar + Girar a la derecha
10	[1, 1, 0, 0, 0, 0]	Atacar + Avanzar
11	[1, 0, 0, 0, 1, 0]	Atacar + Moverse a la izquierda
12	[1, 0, 0, 0, 0, 1]	Atacar + Moverse a la derecha
13	[0, 0, 0, 0, 0, 0]	Ninguna acción (inactividad)

Cuadro 3: Combinaciones de acciones posibles para el agente.

Reward Shaping aplicado

El shaping de recompensa consiste en añadir señales densas para guiar mejor el aprendizaje. La recompensa final se calcula como la suma de la recompensa base del entorno y los términos modelados, como se muestra en la Ecuación (1). En la Tabla **??** se definen las recompensas:

Señal	**Coeficiente**	**Descripción**
Daño infligido al enemigo	+0,01	Recompensa proporcional al daño causado (variable `DAMAGECOUNT`).
Munición usada	−0,01	Penalización por gastar munición del arma activa.
Munición recogida	+0,02	Recompensa al recolectar munición compatible.
Salud ganada	+0,02	Recompensa por aumento de salud respecto al paso previo.
Salud perdida	−0,01	Penalización por pérdida de salud.
Armadura obtenida	+0,01	Recompensa por adquirir armadura.
Movimiento efectivo	+0,0005	Recompensa si se desplazó más de 3 unidades.
Falta de movimiento	−0,0025	Penalización por permanecer quieto.
Cambio de arma	+0,05	Pequeña recompensa por cambiar de arma.
Eliminación de enemigo (Kill)	+1,00	Recompensa fuerte por eliminar un oponente.

Cuadro 4: Señales de Reward Shaping utilizadas en el entorno `DoomShapedEnv`.

Este esquema de shaping permite reducir la varianza en la señal de recompensa y guiar al agente hacia políticas más eficientes, evitando comportamientos triviales como disparar sin estrategia o quedarse estático. Además, promueve una mayor exploración del entorno y una mejor gestión de los recursos disponibles.

Arquitectura del Agente y Proceso de Entrenamiento

Para este proyecto se implementó un agente basado en el algoritmo **Proximal Policy Optimization (PPO)**, una técnica de aprendizaje por refuerzo de tipo actor-crítico con políticas estocásticas y actualización confiable por clipping. PPO ha demostrado un desempeño robusto en entornos visuales complejos como ViZDoom debido a su balance entre estabilidad y eficiencia computacional.

Se diseñó una red convolucional (`CustomCNN`) adaptada a las entradas visuales del entorno, que consisten en stacks de 4 imágenes RGB (resolución 160×120). Esta red reemplaza al extractor de características por defecto de `Stable-Baselines3`, y tiene la siguiente arquitectura:

- **Entrada:** Tensor de forma (4, 120, 160), normalizado en el rango [0, 1].
- **Convolucional 1:** 32 filtros, kernel 8×8, stride 4 + ReLU.
- **Convolucional 2:** 64 filtros, kernel 4×4, stride 2 + ReLU.
- **Convolucional 3:** 64 filtros, kernel 3×3, stride 1 + ReLU.
- **Flatten** y capa lineal con 512 unidades + ReLU.

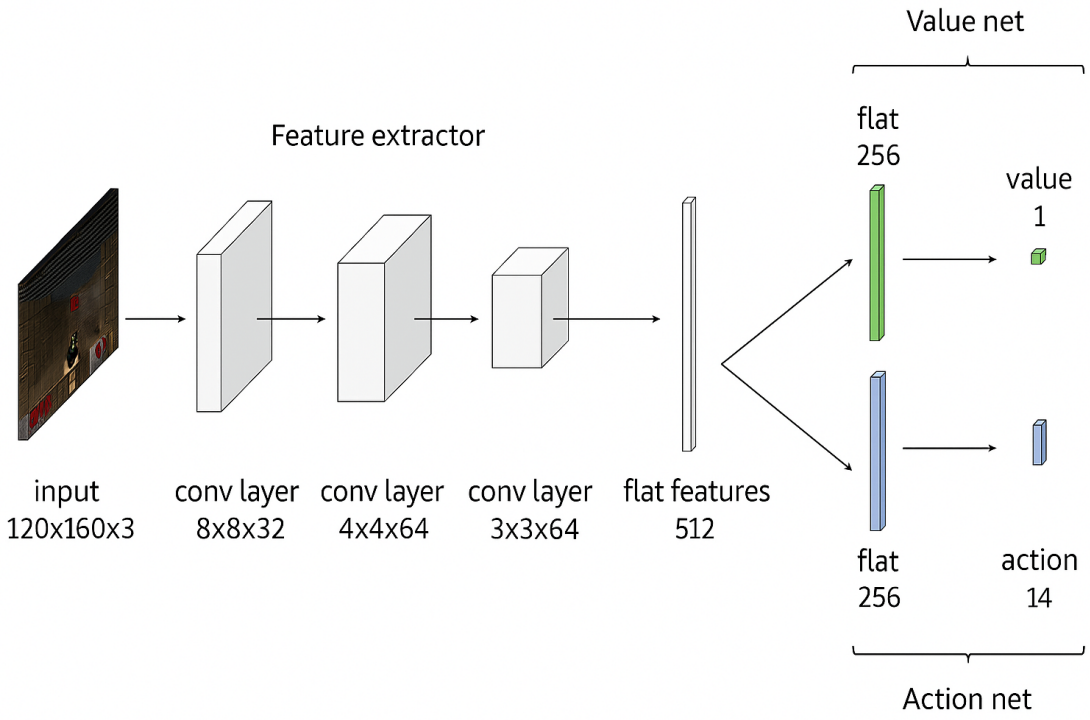

Figura 7: Arquitectura del modelo

Esta red extrae una representación compacta del entorno que se pasa al actor y al crítico del modelo PPO.

Para acelerar el entrenamiento y reducir la varianza del gradiente, se empleó un esquema de entornos paralelos usando `SubprocVecEnv`, con **8 entornos simultáneos**. Cada uno fue envuelto con `VecFrameStack` para proporcionar stacks de 4 frames como entrada (con `channels_first`).

Los hiperparámetros seleccionados para el entrenamiento se resumen en el cuadro 5.

Parámetro	Valor
learning_rate	1×10^{-4}
n_steps	1024
batch_size	1024
n_epochs	4
ent_coef	0.02
total_timesteps	10,000,000
frame_stack	4
num_envs	8
features_dim	512

Cuadro 5: Parámetros usados para entrenamiento del agente PPO.

El entrenamiento se ejecutó por 10 millones de pasos de interacción agente-entorno, utilizando una política tipo `CnnPolicy` enriquecida con la red `CustomCNN`. La entropía del actor se monitoreó como indicio de la exploración, y se aplicaron clipping y batch updates conforme al esquema PPO. La gráfica de recompensas fue:

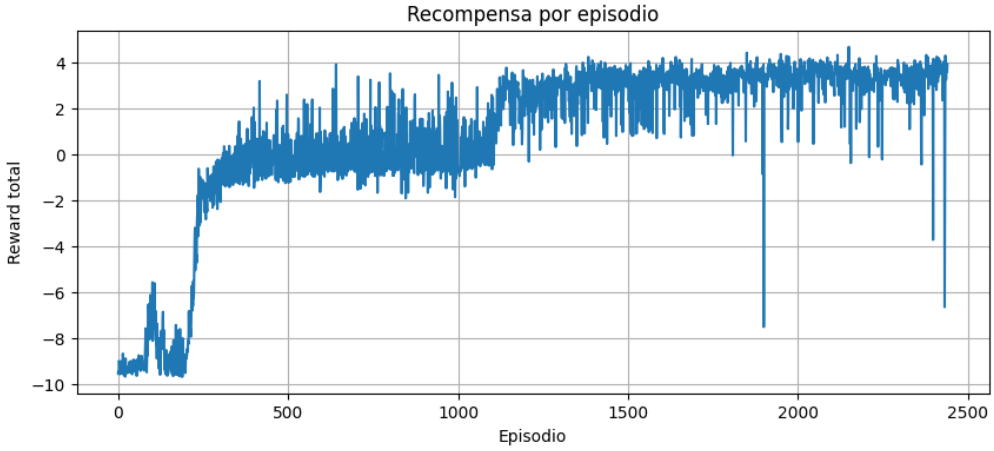

Figura 8: Recompensas entorno personalizado sin enemigos

El código del escenario puede consultarse en el repositorio original: `https://github.com/lkiel/rl-doom/tree/develop/scenarios`

- **Código Entrenamiento:** `https://github.com/juacastanori/TAM/blob/main/ViZDoom_Project/ViZDoom_Custom_map_nobot.ipynb`
- **Código Evaluación:** `https://github.com/juacastanori/TAM/blob/main/ViZDoom_Project/ViZDoom_Custom_nobot_video.ipynb`
- **Video de demostración:** `https://youtu.be/wakcjBQBmAU`

Evaluación en escenario con enemigos

Se puede reutilizar la misma estrategia de entrenamiento para el mismo escenario, pero aumentando la dificultad con enemigos. Las recompensas obtenidas fueron:

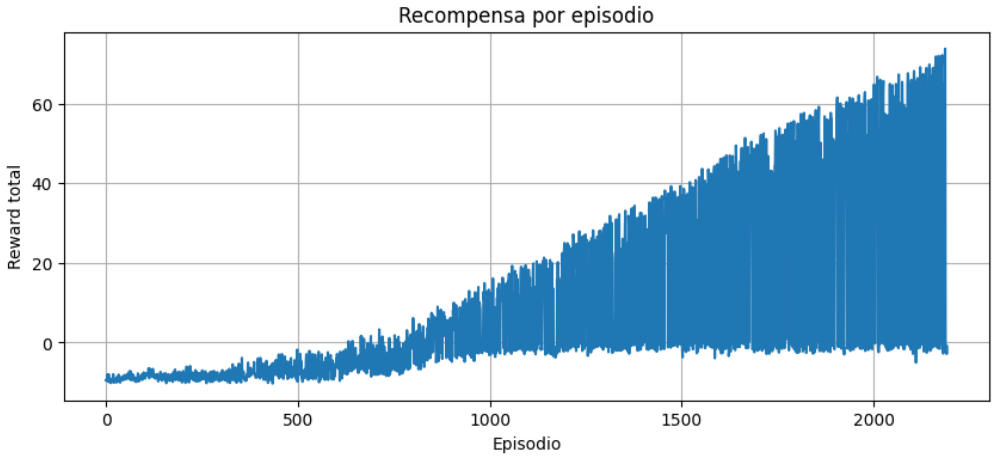

Figura 9: Recompensas entorno personalizado con enemigos

Se observa que las recompensas oscilan mucho, pero hay una tendencia de aumento. Se podría implementar esta estrategia, realizando varias fases de entrenamiento.

Código Entrenamiento: `https://github.com/juacastanori/TAM/blob/main/ViZDoom_Project/ViZDoom_Custom_map_bots_train.ipynb`

8. Conclusiones y Trabajo Futuro

Conclusiones

- El uso de aprendizaje por refuerzo profundo mediante diversos algoritmos permitió entrenar un agentes capaces de aprender estrategias efectivas para maximizar recompensas, en entornos 3D parcialmente observable como `VizdoomBasic-v0` y `VizdoomDefendLine-v0`.
- Tanto el preprocesamiento de la entrada de los modelos como el esquema de recompensas personalizadas (reward shaping) ayudaron a dirigir el aprendizaje hacia comportamientos más eficientes, demostrando la importancia del diseño de estos.
- El entrenamiento fue realizado exitosamente utilizando PyTorch, lo cual facilitó tanto la implementación del modelo como la depuración y monitoreo del entrenamiento en Kaggle.

- Se observaron mejoras significativas en la recompensa media del agente durante el entrenamiento, validando la efectividad del enfoque incluso con un modelo DQN básico. En trabajos futuros, se recomienda explorar variantes en PPO en escenarios más complejos para obtener políticas más robustas.

Trabajo Futuro

Queda pendiente la obtención de un modelo eficiente para un escenario más complejo, como el Entorno personalizado con enemigos. El modelo implementado para este escenario puede ser una base sólida de la que se puede partir para lograr este objetivo. Una posible mejora para este modelo sería implementar redes neuronales LSTM, que tienen conceptos de memoria.

9. Referencias

[1] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning," in *Proc. IEEE Conf. Computational Intelligence and Games (CIG)*, 2016. [Online]. Available: arXiv:1605.02097

[2] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: doi:10.1038/nature14236

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint*, 2017. [Online]. Available: arXiv:1707.06347

[4] L. Kieliger, "Playing Doom with Deep Reinforcement Learning – Part 3: Boosting Performance with Reward Shaping," *Medium*, 2021. [Online]. Available: Enlace

[5] Y. Zhang, T. Liu, M. Jiang, J. Li *et al.*, "ViZDoom Competitions: Playing Doom from Pixels," *arXiv preprint*, 2018. [Online]. Available: arXiv:1809.03470

[6] R. Huang, J. Xu, and B. Liu, "Training an AI-Powered Doomguy Leveraging Deep Reinforcement Learning," in *Proc. Int. Conf. Data, Artificial Intelligence and Information Technology (DAI)*, Atlantis Press, 2023. [Online]. Available: Enlace