Entrenamiento de un Agente Deep Q-Learning para Atari Seaquest usando Gymnasium y Stable-Baselines3

André Chávez Contreras.
Universidad de Xalapa
Ingeniería en Inteligencia Artificial
Email: ux23ii263@ux.edu.com

***Resumen*—Este proyecto presenta el desarrollo de un agente inteligente basado en el algoritmo Deep Q-Learning (DQN) para el videojuego *Seaquest* del conjunto Atari, utilizando la librería Gymnasium [2] y Stable-Baselines3 [1]. El objetivo principal es entrenar un agente capaz de aprender una política óptima para maximizar la recompensa del entorno a partir de observaciones visuales en crudo (píxeles).**

El trabajo incluye la implementación de un pipeline de entrenamiento dividido en fases, con ajustes estratégicos en el *replay buffer*, la tasa de exploración y la duración del entrenamiento, con el fin de estabilizar el aprendizaje y mejorar la convergencia de la función $Q(s, a)$. Además, se incorpora la evaluación determinista del modelo y métricas cuantitativas para analizar el comportamiento del agente y detectar episodios de estancamiento o políticas subóptimas.

El reporte también contextualiza los fundamentos del aprendizaje por refuerzo profundo, desde Q-Learning tradicional hasta mejoras modernas como redes objetivo, *experience replay* y entrenamiento con múltiples entornos paralelos, siguiendo las recomendaciones de Mnih et al. [4]. Se incluyen referencias conceptuales de RL-Baselines3-Zoo [5] y Hugging Face [3], garantizando reproducibilidad y buenas prácticas experimentales.

Los resultados preliminares indican que el enfoque permite que el agente desarrolle conductas clave de juego, incluyendo la eliminación de enemigos, rescate de buzos y gestión de oxígeno, estableciendo una base sólida para investigaciones futuras en entornos visuales complejos y el desarrollo de agentes de aprendizaje por refuerzo más robustos y generalizables.

***Index Terms*—Deep Q-Learning, Aprendizaje por Refuerzo, Atari, Seaquest, Stable-Baselines3, Gymnasium.**

I. Introducción

El aprendizaje por refuerzo (RL, por sus siglas en inglés) es un paradigma fundamental dentro de la inteligencia artificial en el cual un agente aprende a tomar decisiones mediante la interacción continua con un entorno, con el objetivo de maximizar una señal de recompensa acumulada. A diferencia de otros enfoques del aprendizaje automático, el aprendizaje por refuerzo se basa en el descubrimiento autónomo de políticas óptimas a través de prueba y error, lo que lo convierte en una herramienta poderosa para problemas secuenciales y de control.

En los últimos años, el aprendizaje por refuerzo profundo (Deep Reinforcement Learning, DRL) ha logrado avances significativos gracias a la integración de modelos neuronales profundos capaces de aproximar funciones de valor y políticas complejas. Entre los hitos más relevantes se encuentra el trabajo de Mnih et al. [4], quienes introdujeron el algoritmo *Deep Q-Network* (DQN). Este enfoque permitió por primera vez que un agente aprendiera directamente a partir de representaciones visuales en bruto (imágenes), logrando desempeños comparables o superiores a los de jugadores humanos en diversos videojuegos de Atari.

El entorno *Seaquest*, perteneciente a la suite Atari Learning Environment (ALE), representa un desafío importante para los agentes de RL debido a su combinación de múltiples objetivos: navegación submarina, combate, rescate de buzos y administración limitada de oxígeno. Estas características exigen que el agente logre un equilibrio entre exploración, supervivencia y acumulación de puntos a largo plazo, lo que lo convierte en un excelente caso de estudio para evaluar la robustez y estabilidad del algoritmo DQN.

Este proyecto tiene como objetivo entrenar, ajustar y evaluar un agente basado en DQN para el entorno Seaquest utilizando herramientas contemporáneas como Gymnasium [2], ALE-py, Torch y Stable-Baselines3 [1]. Para ello, se implementó un pipeline de entrenamiento dividido en dos fases: una primera etapa enfocada en estabilizar la política con un *replay buffer* reducido, y una segunda etapa diseñada para mejorar la representación del valor con un buffer ampliado y una estrategia de re-exploración controlada.

Además, se busca analizar el comportamiento del agente a lo largo del entrenamiento, interpretar las fluctuaciones en las curvas de recompensa, estudiar fenómenos como los óptimos locales y evaluar su desempeño bajo una política determinista (*greedy*). Finalmente, este trabajo documenta el proceso completo siguiendo el formato IEEE, contribuyendo a la reproducibilidad y rigurosidad experimental exigida en la práctica académica del aprendizaje por refuerzo.

II. Antecedentes y Fundamentos Teóricos

El aprendizaje por refuerzo (RL) constituye una de las ramas más dinámicas del aprendizaje automático, al centrarse en la interacción secuencial entre un agente y un entorno. El agente aprende a maximizar la recompensa acumulada mediante prueba y error, desarrollando políticas que determinan qué acción ejecutar en cada estado. La complejidad de este proceso aumenta cuando los entornos poseen dinámicas no triviales, recompensas escasas o espacios de observación de

alta dimensión, como ocurre en los videojuegos de Atari. En este contexto, métodos tradicionales basados en tablas, como Q-Learning, se vuelven impracticables, dando paso a algoritmos de aprendizaje profundo capaces de aproximar funciones de valor con precisión mediante redes neuronales.

II-A. Aprendizaje por Refuerzo y Q-Learning

Q-Learning es un algoritmo libre de modelo (*model-free*) que busca aprender una función de acción-valor $Q(s, a)$ que represente la utilidad esperada de ejecutar una acción a en un estado s y seguir una política óptima posteriormente. Su regla de actualización clásica está dada por:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right], \quad (1)$$

donde α es la tasa de aprendizaje, γ el factor de descuento, r la recompensa recibida y s' el estado sucesor. A pesar de su solidez teórica, Q-Learning presenta dificultades cuando enfrenta espacios de observación enormes, continuos o basados directamente en imágenes. En tales casos, mantener o explorar una tabla Q se vuelve imposible, abriendo paso al uso de aproximadores de funciones como redes neuronales profundas.

II-B. Deep Q-Network (DQN)

Con el trabajo seminal de Mnih et al. [4], DeepMind introdujo Deep Q-Network (DQN), un algoritmo que combina Q-Learning con redes neuronales convolucionales para aproximar la función $Q(s, a)$ a partir de imágenes de píxeles crudas. DQN marcó un hito al lograr desempeño comparable al humano en múltiples juegos Atari sin ingeniería manual de características.

Para estabilizar el entrenamiento, DQN incorporó tres mecanismos clave:

- **Experience Replay:** Las transiciones experimentadas por el agente se almacenan en un buffer y se muestrean de manera aleatoria durante el entrenamiento. Esto rompe la correlación temporal entre muestras y mejora la eficiencia en el uso de datos.
- **Target Network:** Una segunda red congelada Q_{target} se actualiza cada ciertos pasos, evitando que la red principal se entrene con objetivos inestables y reduciendo el riesgo de divergencia.
- **Frame Stacking:** Al apilar varios frames consecutivos, el agente obtiene información implícita de movimiento, permitiendo inferir velocidad y dirección, elementos que no pueden observarse a partir de un solo frame estático.

La combinación de estas estrategias permite que DQN logre estabilidad y capacidad de generalización en entornos visuales complejos como Seaquest.

II-C. Entornos Atari y Preprocesamiento

Los entornos Atari han sido ampliamente utilizados como benchmark para algoritmos de RL debido a su diversidad, dificultad y estructura de recompensas. Gymnasium [2] ofrece implementaciones estandarizadas de Atari, junto con un conjunto de transformaciones recomendadas:

- reducción de resolución a 84×84 píxeles,
- conversión a escala de grises para disminuir la dimensionalidad,
- salto de frames (*frame skip*),
- inicialización con acciones aleatorias (*no-op reset*),
- wrappers optimizados para DQN como *Max-and-Skip*.

En este proyecto se emplea el entorno `SeaquestNoFrameskip-v4`, variante ampliamente usada en la comunidad y recomendada por Stable-Baselines3 por su estabilidad en agentes basados en valores.

II-D. RL-Baselines3-Zoo

RL-Baselines3-Zoo [5] es un conjunto de herramientas que facilita reproducir experimentos de RL mediante configuraciones de hiperparámetros estandarizadas y scripts de entrenamiento automatizados. Entre sus aportes destacan:

- archivos YAML con configuraciones reproducibles,
- entrenamiento en múltiples ambientes paralelos para acelerar la generación de experiencias,
- integración directa con TensorBoard para análisis visual,
- soporte extendido por la comunidad y guías oficiales de Hugging Face [3].

El uso de estas herramientas permite realizar experimentos estandarizados, comparables y más consistentes con las buenas prácticas de investigación en RL.

III. METODOLOGÍA

El desarrollo del agente de Aprendizaje por Refuerzo Profundo (DRL) para el videojuego *Seaquest* en su versión SeaquestNoFrameskip-v4 se realizó utilizando el algoritmo Deep Q-Network (DQN) implementado con la librería Stable-Baselines3. El procedimiento metodológico se estructuró en tres fases principales: entrenamiento inicial, continuación y optimización del modelo, y evaluación final mediante scripts especializados. Esta organización permitió garantizar reproducibilidad, control del flujo de entrenamiento y un análisis detallado del rendimiento del agente.

Instalación de Dependencias

Para garantizar la reproducibilidad del entorno utilizado en el proyecto, se recomienda instalar las dependencias con sus versiones exactas. El proyecto fue desarrollado con las siguientes versiones:

- **Python**: 3.12.7
- **Gymnasium**: 1.2.2
- **ALE-py**: 0.9.0
- **Stable-Baselines3**: 2.7.0
- **PyTorch**: 2.9.1+cu130
- **OpenCV**: 4.10.0

Para instalar todas las dependencias necesarias en un entorno limpio, ejecutar:

```
pip install gymnasium==1.2.2
pip install ale-py==0.9.0
pip install stable-baselines3==2.7.0
pip install torch==2.9.1+cu130 torchvision torchau
```

```
pip install opencv-python==4.10.0.84
```

También es recomendable instalar el entorno Atari y aceptar la licencia correspondiente:

```
pip install "gymnasium[atari]" "gymnasium[accept-rom-licenses]"
```

Con lo anterior, el proyecto queda listo para ejecutarse sin conflictos de compatibilidad.

III-A. Fase I: Entrenamiento Inicial (Entrenamiento_inicial.py)

La primera fase se enfocó en establecer una línea base de rendimiento y generar el primer checkpoint del modelo. El script `Entrenamiento_inicial.py` configuró un entorno vectorizado con cuatro instancias paralelas (`n_envs=4`) y aplicó el preprocesamiento estándar recomendado para Atari: conversión a escala de grises, redimensionamiento a 84×84 píxeles y apilamiento de cuatro frames consecutivos mediante `VecFrameStack`. Se utilizó una política basada en redes neuronales convolucionales (CNN) con una tasa de aprendizaje de 1e-4, un factor de descuento $\gamma = 0{,}99$ y un *Replay Buffer* de 100,000 pasos. El entrenamiento abarcó dos millones de *timesteps*, con el objetivo de llenar adecuadamente el buffer y lograr una primera convergencia del valor Q. Además, se empleó un `CheckpointCallback` para guardar el modelo cada 100,000 pasos.

III-B. Fase II: Continuación y Optimización (ContinuarEntrenamiento.py)

Debido a que los entornos Atari requieren una gran cantidad de muestras y presentan inestabilidad cuando se utilizan buffers pequeños o modelos parcialmente entrenados, esta fase se dedicó a retomar el entrenamiento desde un checkpoint previo y optimizarlo. El script `ContinuarEntrenamiento.py` realizó varias funciones críticas:

- **Carga y transferencia del checkpoint:** Se localizó automáticamente el archivo con mayor avance y se cargaron sus pesos en un nuevo modelo DQN.
- **Corrección de forma de observaciones:** Se integró el `VecTransposeImage` para ajustar la estructura del espacio de observación, resolviendo errores de compatibilidad entre el modelo cargado y el *Replay Buffer*.
- **Optimización del Replay Buffer:** Se reemplazó el buffer existente por uno nuevo con capacidad de 100,000 pasos, garantizando diversidad y estabilidad durante el entrenamiento prolongado.
- **Ajuste de hiperparámetros:** El entrenamiento continuó con una tasa de aprendizaje refinada (0,00015), un valor de descuento $\gamma = 0{,}995$, y un esquema de exploración fijo de $\epsilon = 0{,}05$.

El modelo ejecutó 3,570,752 pasos adicionales, acumulando un total cercano a 10 millones de *timesteps*, permitiendo que el agente alcanzara una política más estable y generalizable.

III-C. Fase III: Evaluación del Modelo (Visualizar_modelo.py)

Una vez finalizado el proceso de entrenamiento, el script `Visualizar_modelo.py` se utilizó para analizar de manera visual y cuantitativa el desempeño del agente. Este script permite:

- cargar el modelo final o el checkpoint con mayor recompensa promedio (`ep_rew_mean`),
- ejecutar episodios con política determinista ($\epsilon = 0$),
- generar videos o visualizar el comportamiento del agente en tiempo real,
- evaluar la recompensa promedio utilizando la función `evaluate_policy` de Stable-Baselines3.

Este proceso proporciona una métrica objetiva del desempeño alcanzado y permite comparar diferentes configuraciones del agente.

IV. Análisis de Resultados

El entrenamiento completo culminó con la obtención del modelo final `dqn_seaquest_run9_finished_11429248_steps`, entrenado a lo largo de dos fases diferenciadas. El comportamiento del agente se evaluó mediante análisis cuantitativos, métricas internas de entrenamiento y observación directa del desempeño en el entorno Atari *Seaquest*. A partir de estas evaluaciones, se verificó que el agente logró capturar los patrones fundamentales del juego: eliminar enemigos, rescatar buzos atrapados y regular su nivel de oxígeno ascendiendo a la superficie cuando fuera necesario.

IV-A. Comportamiento del Agente Durante el Entrenamiento

Durante la Fase I (primeros 2 millones de pasos), el agente mostró un aprendizaje inicial lento pero estable, limitado principalmente por el tamaño reducido del *Replay Buffer*. Esto provocó una actualización del estimador de $Q(s, a)$ basado en un número limitado de experiencias, lo cual derivó en oscilaciones temporales y una estrategia conservadora.

En fases intermedias se identificaron errores comunes, tales como:

- Permanecer demasiado tiempo en zonas profundas, ignorando el nivel de oxígeno.
- Colisiones frecuentes con enemigos cercanos por falta de evasión.
- Falta de iniciativa para rescatar humanos, favoreciendo comportamientos de bajo riesgo.

Con la transición a un *Replay Buffer* de mayor capacidad y la incorporación de una tasa de exploración constante ($\epsilon = 0{,}05$), la Fase II permitió al agente escapar de varios óptimos locales detectados en la política inicial. Se observó que la re-exploración controlada incentivó al agente a descubrir secuencias beneficiosas como la combinación óptima entre ataque, rescate y administración de oxígeno.

IV-B. Evaluación Determinista del Modelo

Dado que la exploración estocástica introduce ruido en las métricas de recompensa, la evaluación final se realizó bajo una política determinista (`deterministic=True`). Esta evaluación reveló el desempeño real del modelo, libre de acciones aleatorias. El agente alcanzó una recompensa promedio cercana a $1{,}152$ puntos, consistente con resultados reportados para agentes DQN clásicos en este entorno.

La política final refleja decisiones más maduras, tales como:

- Movimientos laterales anticipados para evitar colisiones.
- Disparo continuo en trayectorias óptimas.
- Ascensos estratégicos en el momento exacto antes de agotar oxígeno.
- Rescate de buzos sin comprometer la supervivencia del submarino.

IV-C. Historial de Recompensas por Fase

La Tabla I resume los valores clave registrados a lo largo del entrenamiento. Estos resultados muestran la relación entre los ajustes implementados (como el tamaño del *Replay Buffer* y los parámetros de exploración) y la evolución de la política aprendida.

Cuadro I
RESUMEN HISTÓRICO DEL RENDIMIENTO DEL AGENTE DQN DURANTE EL ENTRENAMIENTO.

Hito	Pasos (Timesteps)	Recompensa Promedio
Fin Fase I	2,000,000	350
Pico Inestable	6,429,248	1,270
Re-Estabilización	7,000,416	758
Estancamiento Estable	9,195,456	741
Modelo Final (Greedy)	**11,429,248**	**1,152**

La dinámica observada en la tabla muestra un pico prematuro de rendimiento alrededor de los 6.4 millones de pasos, típico de un *DQN* expuesto temporalmente a patrones favorables del entorno. La posterior caída y estabilización reflejan el reajuste de la política tras ampliar la capacidad del *Replay Buffer*. Finalmente, el rendimiento bajo política determinista confirma la consolidación de una política más estable, generalizable y eficaz.

IV-D. Síntesis de los Resultados

En conjunto, los resultados demuestran que:

- El agente aprendió comportamientos esenciales del juego y alcanzó desempeño competitivo.
- La inestabilidad inicial fue mitigada mediante una reconfiguración del proceso de entrenamiento.
- La evaluación determinista es crucial para obtener métricas reales del rendimiento del modelo.
- El uso de un buffer grande y una exploración residual mejora significativamente la estabilidad del aprendizaje.

Estos hallazgos permiten afirmar que la implementación desarrollada constituye una base sólida para la explotación de variantes modernas del algoritmo DQN y para la extensión del estudio a algoritmos más avanzados de aprendizaje por refuerzo profundo.

V. LIMITACIONES DEL ESTUDIO

A pesar de los resultados satisfactorios obtenidos, el presente proyecto presenta diversas limitaciones metodológicas, computacionales y experimentales que deben considerarse al interpretar el desempeño del agente DQN entrenado en el entorno *SeaquestNoFrameskip-v4*.

V-A. Limitaciones Computacionales

El entrenamiento de agentes basados en Deep Q-Network requiere una gran cantidad de recursos computacionales. En este proyecto, el tiempo total de entrenamiento estuvo condicionado por las capacidades del hardware disponible, limitando aspectos como:

- el número total de *timesteps* alcanzables,
- la posibilidad de entrenar múltiples semillas para análisis estadístico,
- la exploración exhaustiva de hiperparámetros,
- el uso de arquitecturas más profundas o costosas.

Estas restricciones influyen directamente en la estabilidad y convergencia del modelo, y pueden sesgar los resultados hacia configuraciones específicas.

V-B. Limitaciones del Algoritmo DQN

Aunque DQN es un método ampliamente utilizado, presenta limitaciones inherentes al diseño del algoritmo:

- es altamente sensible a los hiperparámetros;
- puede presentar oscilaciones debido a la correlación entre muestras;
- requiere un *Replay Buffer* grande para evitar sobreajuste;
- puede quedar atrapado en óptimos locales, especialmente en entornos con recompensas escasas o múltiples objetivos, como Seaquest.

Estas limitaciones explican fenómenos observados durante el entrenamiento, como picos inestables o periodos de estancamiento prolongado.

V-C. Limitaciones del Entorno de Atari

El entorno *Seaquest* presenta características que complican la generación de políticas óptimas:

- múltiples objetivos simultáneos (combate, rescate y oxígeno),
- recompensas relativamente densas pero con alta variabilidad,
- dependencia del *frame stacking* para inferir movimiento.

Estas propiedades incrementan la dificultad del aprendizaje, especialmente cuando se usan enfoques basados únicamente en imágenes en crudo.

V-D. Limitaciones de Evaluación

La evaluación determinista utilizada ofrece una medición fiable del desempeño real del agente; sin embargo, presenta sus propias limitaciones:

- no refleja la robustez del agente bajo condiciones estocásticas,
- no se realizaron pruebas cruzadas en distintas semillas,

- no se comparó contra otros algoritmos como Double DQN, PPO o A2C.

Estas restricciones limitan el alcance comparativo del estudio y dejan abierta la posibilidad de evaluar métodos alternativos en trabajos futuros.

En conjunto, estas limitaciones no invalidan los resultados, pero sí contextualizan el alcance real del trabajo y justifican la necesidad de explorar variantes modernas del algoritmo en estudios posteriores.

VI. Conclusiones y Trabajo Futuro

El presente proyecto desarrolló, entrenó y evaluó un agente basado en Deep Q-Network (DQN) para el entorno Atari *Seaquest*, explorando distintas configuraciones de entrenamiento y analizando su impacto en la estabilidad del aprendizaje y en la calidad de la política obtenida. El enfoque adoptado se estructuró en dos fases: una etapa inicial destinada a construir una política base utilizando un *replay buffer* reducido, y una segunda fase que extendió el entrenamiento con un buffer mayor y una reintroducción de exploración controlada.

Los resultados experimentales muestran que el agente logró un aprendizaje sólido y funcional, alcanzando un rendimiento promedio de aproximadamente $1{,}152$ puntos bajo una política *greedy*. Este desempeño es consistente con las evaluaciones reportadas para implementaciones clásicas de DQN en este juego. El agente desarrolló conductas clave como la eliminación eficiente de enemigos, el rescate de buzos y la administración del oxígeno, lo que indica que la función de valor aprendida captura adecuadamente los elementos críticos de la dinámica del entorno.

Durante el entrenamiento se identificaron episodios de estancamiento en políticas conservadoras o subóptimas. Sin embargo, estas situaciones fueron mitigadas mediante la incorporación de una tasa de exploración constante ($\epsilon = 0{,}05$) durante la segunda fase, lo que favoreció la reintroducción de estados novedosos y una mejora significativa en la política final. Asimismo, se observó que la evaluación determinista (*greedy*) es un indicador más fiel del desempeño del agente, ya que elimina la variabilidad introducida por la exploración.

Un aspecto relevante es la robustez del pipeline empleado: la combinación de un *replay buffer* ampliado, fases de entrenamiento diferenciadas y control sobre la exploración permitió que la política aprendida fuera más estable, reduciera la incidencia de *overfitting* a secuencias temporales específicas y maximizara la generalización a situaciones no vistas durante el entrenamiento. Además, la integración de métricas cuantitativas y observación directa del comportamiento del agente facilitó la identificación de patrones de mejora y de limitaciones inherentes a DQN, como la sensibilidad a hiperparámetros y la dificultad para escapar de óptimos locales.

Si bien los resultados son prometedores, se identifican limitaciones importantes: la capacidad del agente sigue estando condicionada por el tamaño del *replay buffer*, la duración del entrenamiento y la arquitectura de la red, lo que podría restringir su desempeño frente a entornos más complejos o con dinámicas no lineales más pronunciadas. Estas observaciones destacan la necesidad de explorar variantes avanzadas de DQN, arquitecturas de redes más profundas y técnicas de *prioritized experience replay* para optimizar el aprendizaje en situaciones de alta dimensionalidad.

En conjunto, los resultados permiten concluir que el enfoque implementado fortalece la estabilidad y la convergencia del método DQN, mejora la estimación de la función $Q(s, a)$ y ofrece un marco reproducible para futuros experimentos. Este proyecto establece una base sólida para la investigación de agentes de aprendizaje por refuerzo profundo, proporcionando tanto lineamientos metodológicos como evidencias prácticas sobre cómo configurar y evaluar agentes en entornos visuales complejos.

VI-A. Trabajo Futuro

Existen diversas rutas de investigación y mejora que pueden profundizar y extender los resultados presentados:

- **Variantes avanzadas de DQN:** Integrar algoritmos como Double DQN, Dueling DQN, Prioritized Experience Replay (PER) o Noisy Networks con el fin de reducir el sesgo en la estimación de valor, mejorar la estabilidad y optimizar la eficiencia del muestreo.
- **Optimización del proceso de entrenamiento:** Realizar una búsqueda sistemática de hiperparámetros, evaluar el impacto del tamaño del *replay buffer*, probar estrategias alternativas de exploración y emplear técnicas de *checkpointing* para analizar trayectorias particulares de aprendizaje.
- **Métricas y evaluación avanzada:** Comparar el rendimiento con benchmarks oficiales de Atari, ejecutar múltiples semillas para estimar la varianza del desempeño y reportar curvas de aprendizaje normalizadas alineadas con buenas prácticas de investigación reproducible.
- **Interpretabilidad del agente:** Aplicar herramientas como *saliency maps* o métodos de atribución visual que permitan comprender qué regiones de la imagen son más relevantes en la toma de decisiones del agente.
- **Mejoras en la representación del estado:** Probar arquitecturas de redes más profundas, métodos modernos de preprocesamiento visual o diseños alternativos de *frame stacking* que capturen mejor la dinámica temporal.
- **Comparación con algoritmos basados en políticas:** Implementar métodos como PPO, A2C o Rainbow para evaluar diferencias entre enfoques basados en valor y en políticas, ampliando así el análisis comparativo del entorno.
- **Reproducibilidad y expansión del repositorio:** Consolidar un repositorio con scripts modulares, pesos preentrenados, documentación completa y experimentos replicables, siguiendo estándares académicos solicitados por rúbricas de evaluación formal.

En síntesis, este proyecto constituye un punto de partida robusto para el estudio de agentes de aprendizaje por refuerzo profundo en entornos Atari. Las técnicas exploradas y los resultados obtenidos abren la puerta a investigaciones más

avanzadas que permitan desarrollar agentes más competentes, estables y completamente reproducibles.

REFERENCIAS

[1] Stable-Baselines3 Documentation. Disponible en: https://stable-baselines3.readthedocs.io/

[2] Gymnasium Atari Environments. Disponible en: https://gymnasium.farama.org/environments/atari/

[3] Hugging Face Deep Reinforcement Learning Course. Disponible en: https://huggingface.co/learn/deep-rl

[4] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, 518(7540), 2015.

[5] RL-Baselines3-Zoo GitHub Repository. Disponible en: https://github.com/DLR-RM/rl-baselines3-zoo