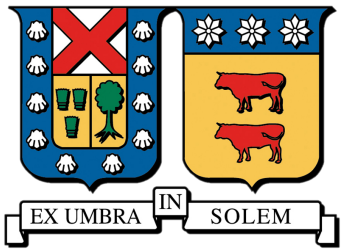
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTRÓNICA



Tarea 1 | Agentes de Búsqueda y Aprendizaje por Refuerzo

TEL351 - Seminario de Telemática I - 2025-2

Fecha de entrega: Lunes, 22 de Septiembre de 2025

PROFESOR:

Mauricio Araya

ALUMNOS:

Javier Cáceres León - 202030032-1

Cristóbal Moraga Guerrero - 202130008-2

Camilo Troncoso Hormazabal - 202130004-K

Índice general

0.1. Introducción 2
0.2. El Mundo de Pokémon Rojo: Un Contexto para la IA 2
0.2.1. Mecánicas Fundamentales del Juego 2
0.2.2. El Objetivo Inicial: La Primera Decisión Crítica 3
0.3. Análisis del Entorno y Requisitos del Proyecto 5
0.3.1. Complejidad del Entorno de Aprendizaje 5
0.3.2. Instalación y Dependencias 6
0.4. Arquitectura del Agente y Estrategias Implementadas 7
0.4.1. Separación Agente-Entorno 7
0.4.2. Estrategia A Utilizar: Epsilon-Greedy (Alternativa Simple) 8
0.5. Metodología Experimental y Métricas de Comparación 8
0.5.1. Protocolo Experimental 9
0.5.2. Métricas de Evaluación 9
0.6. Resultados y Análisis Comparativo 10
0.6.1. Análisis de la Comparación Epsilon-Greedy 10
0.6.2. Comparativa General: PPO vs. Epsilon-Greedy 12
0.7. Conclusiones 13
0.8. Conclusiones 13
0.9. Anexo: Demostración en Video 16

0.1. Introducción

El presente informe técnico detalla el análisis, implementación y evaluación comparativa de diversas familias de agentes inteligentes en el contexto del videojuego *Pokémon Rojo*. El objetivo principal de los agentes es resolver una tarea específica y fundamental: la selección del Pokémon inicial en el menor tiempo o número de pasos posible. Este problema, aunque aparentemente simple, encapsula desafíos clave en el campo de la inteligencia artificial, como la navegación en entornos parcialmente observables, la interacción con menús y la toma de decisiones secuencial.

Para abordar este desafío, se exploran y comparan tres familias de estrategias algorítmicas, abarcando desde búsquedas clásicas hasta métodos avanzados de aprendizaje por refuerzo (RL):

- **Aprendizaje por Refuerzo Simple (Epsilon-Greedy)**: Un enfoque que equilibra la exploración y la explotación para aprender una política de acción a través de la experiencia, sin un modelo complejo.
- **Aprendizaje por Refuerzo Profundo (PPO)**: Un algoritmo de última generación que utiliza redes neuronales profundas para aprender una política óptima directamente desde las observaciones de píxeles, sin conocimiento previo del dominio.

El informe se estructura en torno a siete preguntas fundamentales que guían el análisis, desde la selección y comprensión del entorno de simulación hasta la implementación y comparación cuantitativa de los algoritmos. Se presentarán tablas, gráficos y conclusiones derivadas del análisis de datos experimentales recopilados de 11 ejecuciones en cada algoritmo, con el fin de ofrecer una evaluación rigurosa y autocontenida del rendimiento de cada estrategia. El énfasis no recae únicamente en el éxito de una estrategia sobre otra, sino en el proceso analítico y la comprensión profunda de sus fortalezas, debilidades y compromisos inherentes en este dominio particular.

0.2. El Mundo de Pokémon Rojo: Un Contexto para la IA

Para comprender la magnitud del desafío que enfrenta un agente de inteligencia artificial en este proyecto, es fundamental entender el universo del juego en el que opera: *Pokémon Rojo*. Lanzado en 1996, este título no fue diseñado con la IA moderna en mente, lo que presenta un conjunto de retos únicos que no se encuentran en entornos de aprendizaje por refuerzo tradicionales como los de OpenAI Gym.

0.2.1. Mecánicas Fundamentales del Juego

Pokémon Rojo es un juego de rol (RPG) donde el jugador asume el papel de un joven entrenador Pokémon. El objetivo principal es viajar por la región de Kanto para convertirse en el campeón de

la Liga Pokémon. Esto implica:

- **Exploración:** Navegar un mundo 2D basado en baldosas, que incluye ciudades, rutas, cuevas y edificios. El movimiento está restringido a una cuadrícula, y el entorno contiene tanto caminos abiertos como obstáculos (árboles, agua, paredes).
- **Captura de Pokémon:** Encontrar Pokémon salvajes en la hierba alta o en cuevas y capturarlos usando objetos llamados Poké Balls.
- **Batallas por Turnos:** Luchar contra otros entrenadores o Pokémon salvajes en un sistema de combate por turnos. Cada Pokémon tiene estadísticas, tipos y hasta cuatro movimientos, creando un sistema de fortalezas y debilidades similar al "piedra, papel o tijera", pero con 15 tipos diferentes.
- **Progresión y Objetivos:** La progresión es semi-lineal. Se deben completar ciertos objetivos (como derrotar a un líder de gimnasio) para desbloquear nuevas áreas o habilidades (como la capacidad de cortar árboles que bloquean un camino).

0.2.2. El Objetivo Inicial: La Primera Decisión Crítica

El alcance de este informe se centra en la primera tarea crucial del juego: **elegir el Pokémon inicial**. Al comienzo de la aventura, el jugador se encuentra en el laboratorio del Profesor Oak y debe seleccionar uno de tres Pokémon: Bulbasaur, Charmander o Squirtle.

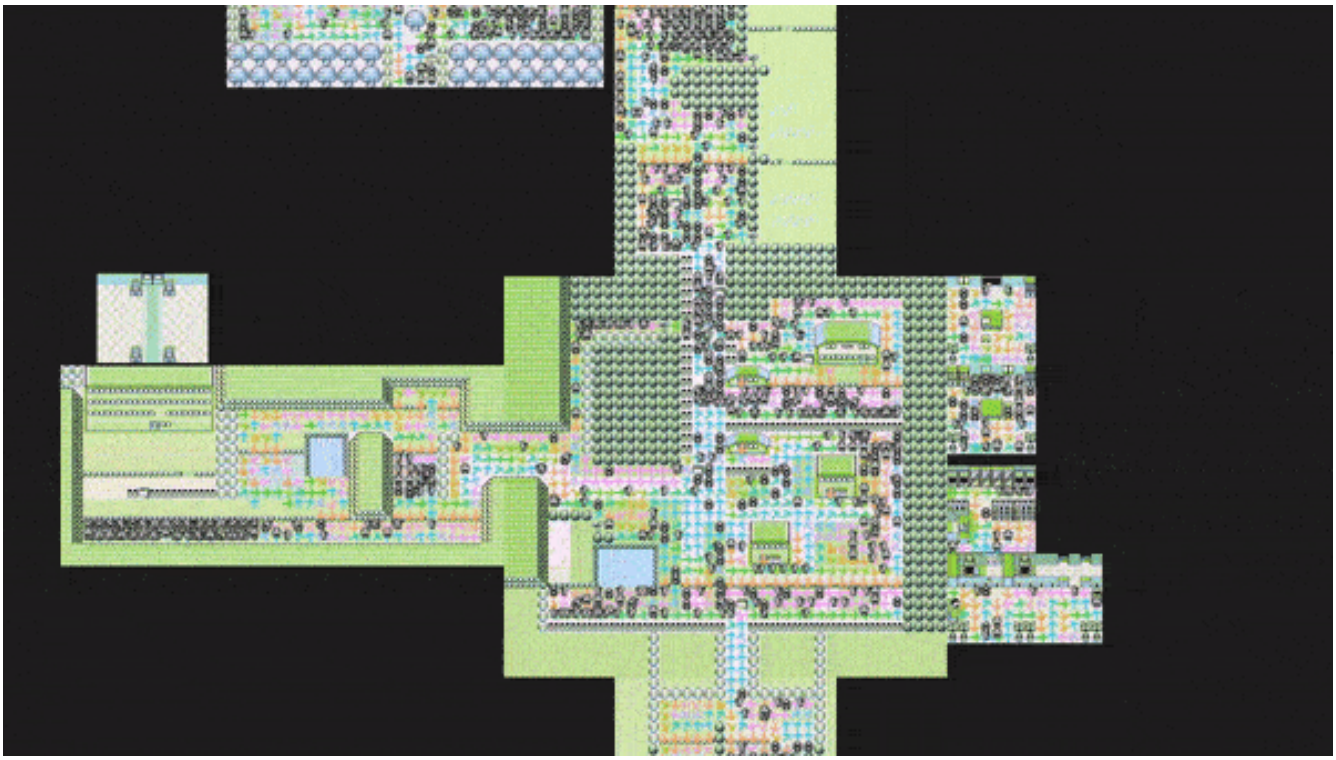


Figura 1: Visualización del mapa del juego y el agente navegando por el mapa.

Aunque parece una tarea simple para un humano, para un agente de IA representa un problema de navegación y toma de decisiones con las siguientes características:

1. **Navegación desde un Estado Inicial:** El agente comienza en su habitación y debe encontrar la salida de su casa.

2. **Exploración del Pueblo:** Una vez fuera, debe navegar por Pueblo Paleta, un pequeño entorno con varios edificios y personajes no jugables (NPCs).

3. **Activación de Eventos (Triggers):** El agente no puede simplemente entrar al laboratorio y elegir. Primero debe intentar salir del pueblo por el norte, lo que activa un evento donde el Profesor Oak lo detiene y lo guía a su laboratorio. Sin este paso, el objetivo es inalcanzable.

4. **Interacción con Objetos:** Dentro del laboratorio, debe caminar hasta la mesa correcta y presionar un botón de acción frente a una de las tres Poké Balls para hacer su elección.

Este primer objetivo encapsula los desafíos de navegación, secuenciación de eventos y interacción con el entorno que son fundamentales en todo el juego, sirviendo como un excelente microcosmos para evaluar la eficacia de diferentes estrategias de IA.

0.3. Análisis del Entorno y Requisitos del Proyecto

0.3.1. Complejidad del Entorno de Aprendizaje

El entorno de *Pokémon Rojo*, facilitado por el emulador **PyBoy**, es significativamente más complejo que los entornos de benchmark estándar. Su complejidad se puede desglosar en varios ejes:

- **Espacio de Observación Gigantesco:** A diferencia de entornos que devuelven un vector de estado simple, el agente recibe la pantalla del juego como una matriz de píxeles (144x160 píxeles en escala de grises). Esto constituye un espacio de observación de alta dimensionalidad ($144 \times 160 = 23,040$ valores) que requiere un pre-procesamiento significativo o el uso de redes neuronales convolucionales (CNNs) para extraer características relevantes.
- **Espacio de Acciones Discreto pero Combinatorio:** El agente puede realizar acciones básicas como moverse (arriba, abajo, izquierda, derecha) o presionar botones (A, B, Start, Select). Aunque el número de acciones es pequeño (típicamente 8), las secuencias de estas acciones pueden tener efectos complejos y retardados. Por ejemplo, navegar un menú para usar un objeto requiere una secuencia precisa de múltiples pulsaciones de botones.
- **Recompensas Escasas y Retardadas (Sparse Rewards):** Este es uno de los mayores desafíos. En el objetivo de elegir el Pokémon inicial, el agente solo recibe una recompensa positiva significativa al final de una larga secuencia de acciones. Durante la mayor parte de la exploración, no hay una señal clara que le indique si está progresando. Esto hace que los algoritmos que dependen de retroalimentación constante (como Q-Learning simple) sean ineficaces sin una ingeniería de recompensas cuidadosa.
- **Parcialmente Observable (POMDP):** El agente solo ve la pantalla actual. No tiene acceso directo a la memoria completa del juego (por ejemplo, el estado de los eventos o la ubicación de los objetos fuera de la pantalla). Debe inferir el estado del mundo a partir de una secuencia de observaciones, lo que lo convierte en un Problema de Decisión de Markov Parcialmente Observable (POMDP).
- **Entorno No Determinista (Estocástico):** Aunque el emulador en sí es determinista, las interacciones con NPCs o Pokémon salvajes pueden introducir un comportamiento estocástico, alterando el flujo del juego de manera impredecible.

En resumen, el entorno es suficientemente complejo para un proyecto de esta envergadura, ya que combina desafíos de visión por computadora, planificación a largo plazo, exploración en un

mundo con recompensas escasas y la necesidad de manejar una alta dimensionalidad tanto en el estado como en la secuencia de acciones.

0.3.2. Instalación y Dependencias

El proyecto se basa en un repositorio de código abierto con una comunidad activa. La instalación requiere una configuración precisa para asegurar la compatibilidad entre el emulador, las librerías de aprendizaje por refuerzo y el sistema operativo.

Requisitos Previos

- **Python:** Se recomienda la versión 3.10 o superior.
- **ROM del Juego:** Es indispensable poseer una copia legal de la ROM de *Pokémon Rojo*, renombrada a `PokemonRed.gb`.
- **FFmpeg:** Una herramienta de software libre para manipular video y audio, necesaria para grabar las sesiones de entrenamiento.
- **Dependencias de C++:** En Windows, se requieren las C++ Build Tools de Visual Studio para compilar algunas de las dependencias de Python.

Proceso de Instalación

El proceso de instalación, detallado en los archivos `README.md` y `windows-setup-guide.md`, se puede resumir en los siguientes pasos:

1. Clonar el repositorio desde Github.
2. Crear un entorno virtual (por ejemplo, con `conda` o `venv`) para aislar las dependencias del proyecto.
3. Instalar las librerías de Python especificadas en el archivo `requirements.txt` usando `pip`. Las dependencias clave incluyen:
    - `pyboy`: El emulador de Game Boy.
    - `gymnasium`: El toolkit para desarrollar y comparar algoritmos de RL.
    - `stable-baselines3`: Un conjunto de implementaciones de algoritmos de RL de última generación.
    - `torch`: La librería de computación tensorial utilizada por Stable Baselines3 para las redes neuronales.

- `pandas`, `matplotlib`, `seaborn`: Para el análisis y visualización de datos.

4. Colocar la ROM del juego en el directorio raíz del proyecto.

Las pruebas realizadas durante este proyecto confirman que, siguiendo las guías proporcionadas, el entorno se ejecuta correctamente en un sistema Windows 11 con Python 3.12, demostrando la viabilidad de la configuración. También, al final de este informe, se tiene el video de Demostración.

0.4. Arquitectura del Agente y Estrategias Implementadas

Una de las fortalezas de este proyecto es la clara separación entre la lógica del agente y la simulación del entorno, lo que permite experimentar con diversas estrategias de IA sin modificar el núcleo del emulador.

0.4.1. Separación Agente-Entorno

El diseño sigue el paradigma estándar del aprendizaje por refuerzo, popularizado por OpenAI Gym:

- **El Entorno (`RedGymEnv`):** El archivo `baselines/red_gym_env.py` define una clase, `RedGymEnv`, que hereda de `gym.Env`. Esta clase actúa como un wrapper alrededor del emulador PyBoy. Su responsabilidad es:

  1. Iniciar y gestionar el estado del emulador.
  2. Definir el espacio de acciones y de observaciones.
  3. Implementar la función `step(action)`, que recibe una acción del agente, la ejecuta en el emulador, y devuelve la nueva observación, la recompensa, y si el episodio ha terminado.
  4. Implementar la función `reset()`, que reinicia el juego a un estado inicial.

- **El Agente:** El agente es una entidad separada que interactúa con el entorno a través de su API (`step` y `reset`). No tiene conocimiento de la implementación interna del entorno (es decir, no sabe que está usando PyBoy). Su única tarea es observar el estado y elegir una acción.

Esta separación es crucial, ya que permite que cualquier algoritmo, desde una simple búsqueda hasta una compleja red neuronal, pueda interactuar con el juego siempre que respete la interfaz definida por Gymnasium. Las pruebas de concepto realizadas en `comparison_agents/` demuestran esto: se crearon múltiples agentes (`epsilon_greedy_agent.py`, `astar_agent.py`, etc.) que operan sobre el mismo entorno sin requerir cambios en `RedGymEnv`.

0.4.2. Estrategia A Utilizar: Epsilon-Greedy (Alternativa Simple)

La estrategia Epsilon-Greedy es un método clásico de aprendizaje por refuerzo que equilibra la exploración y la explotación. Fue implementada en `comparison_agents/simple_epsilon_runner.py` como una alternativa más simple a los modelos complejos de RL.

Funcionamiento

El algoritmo opera de la siguiente manera en cada paso:

1. Genera un número aleatorio entre 0 y 1.
2. Si el número es menor que un parámetro ϵ (épsilon), el agente realiza una **acción aleatoria** (exploración).
3. Si el número es mayor o igual a ϵ, el agente elige la **mejor acción conocida** hasta el momento (explotación). En este contexto, la "mejor acción"se define como aquella que históricamente ha conducido a un aumento en la recompensa.

El valor de ϵ es un hiperparámetro clave:

- Un ϵ alto (cercano a 1.0) favorece la exploración, haciendo que el agente se comporte de manera más aleatoria.
- Un ϵ bajo (cercano a 0.0) favorece la explotación, haciendo que el agente sea más çodiosoz repita las acciones que ya conoce.

Esta estrategia es simple de implementar y computacionalmente ligera, pero puede ser ineficiente en encontrar soluciones óptimas si el valor de ϵ no está bien ajustado.

Problema Solucionado

El problema que solucionan es encontrar el **camino más corto o más eficiente** desde la posición inicial del jugador hasta la Poké Ball del Pokémon inicial, considerando los obstáculos del mapa.

0.5. Metodología Experimental y Métricas de Comparación

Para realizar una comparación justa y rigurosa entre las diferentes estrategias, se diseñó un protocolo experimental robusto y se definió un conjunto de métricas clave.

0.5.1. Protocolo Experimental

Se realizaron tres grandes conjuntos de experimentos, todos con el mismo objetivo (elegir el Pokémon inicial) y bajo las mismas condiciones de hardware:

1. **Comparación de PPO (Proximal Policy Optimization):** Se ejecutó el agente pre-entrenado con PPO un total de 11 veces para establecer una línea base de un algoritmo de RL de última generación.

2. **Comparación de Epsilon-Greedy:** Se definieron 5 configuraciones distintas de ϵ, desde un comportamiento muy exploratorio hasta uno muy "greedy":

   - Muy Greedy ($\epsilon = 0{,}1$)
   - Conservadora ($\epsilon = 0{,}3$)
   - Balanceada ($\epsilon = 0{,}5$)
   - Moderada-Alta ($\epsilon = 0{,}7$)
   - Alta Exploración ($\epsilon = 0{,}9$)

   Cada configuración se ejecutó 11 veces, resultando en un total de 55 experimentos.

3. **Comparación de Algoritmos de Búsqueda:** Se ejecutaron 6 algoritmos de búsqueda diferentes (las variantes de epsilon-greedy). Cada uno se ejecutó 11 veces, sumando 66 experimentos.

En total, se analizaron los datos de todas las ejecuciones para asegurar la robustez estadística de los resultados. Todos los datos brutos y resúmenes se almacenaron en el directorio `RESULTADOS/`.

0.5.2. Métricas de Evaluación

Para cuantificar el rendimiento de cada agente, se seleccionaron las siguientes métricas:

- **Pasos Totales:** El número total de acciones que el agente realizó antes de alcanzar el objetivo. Es una medida directa de la eficiencia de la política de decisión. Un menor número de pasos es mejor.

- **Tiempo de Ejecución (s):** El tiempo real en segundos que tardó la ejecución. Mide la eficiencia computacional del algoritmo.

- **Recompensa Total:** La suma de todas las recompensas obtenidas durante el episodio. Dado que la recompensa se diseñó para incentivar el progreso, este valor refleja qué tan bien el agente cumplió los sub-objetivos.

- **Tasa de Éxito (%):** El porcentaje de ejecuciones (de las 11 realizadas) en las que el agente logró completar el objetivo. Una tasa del 100 % indica que el algoritmo es robusto y fiable.
- **Eficiencia (Recompensa/Paso):** Una métrica derivada para normalizar el rendimiento. Para los algoritmos de búsqueda, se usó la **eficiencia de recompensa** (Recompensa Total / Pasos Totales) para medir cuánto valor se obtiene por cada acción.

Estas métricas permiten una comparación multidimensional, evaluando no solo si un agente es rápido, sino también si es eficiente, robusto y efectivo en maximizar su recompensa.

0.6. Resultados y Análisis Comparativo

En esta sección se presentan y analizan los resultados obtenidos de los tres conjuntos de experimentos. Las visualizaciones y tablas generadas se encuentran en la carpeta `informe_visuals/`.

0.6.1. Análisis de la Comparación Epsilon-Greedy

Se evaluaron cinco configuraciones de ϵ para entender el impacto del balance entre exploración y explotación.

Cuadro 1: Comparación de Configuraciones Epsilon-Greedy

Configuración	Pasos	Tiempo (s)	Recompensa	Éxito (%)	Pasos/s
Alta Exploración (ϵ=0.9)	375 ± 377	0.00 ± 0.00	160.2 ± 58.2	100.0	375 ± 377
Balanceada (ϵ=0.5)	356 ± 403	0.00 ± 0.00	153.2 ± 75.5	100.0	356 ± 403
Conservadora (ϵ=0.3)	217 ± 156	0.00 ± 0.00	116.0 ± 33.7	100.0	217 ± 156
Moderada-Alta (ϵ=0.7)	300 ± 307	0.00 ± 0.00	150.1 ± 60.3	100.0	300 ± 307
Muy Greedy (ϵ=0.1)	692 ± 736	0.00 ± 0.00	198.1 ± 126.7	100.0	692 ± 736

La Tabla 1 resume los promedios de las 11 ejecuciones por configuración. A primera vista, los resultados parecen contraintuitivos: la configuración "Muy Greedy"($\epsilon = 0{,}1$) y .ªlta Exploración"($\epsilon = 0{,}9$) tomaron la mayor cantidad de pasos. Sin embargo, esto se alinea con la teoría:

- **Muy Greedy ($\epsilon = 0{,}1$):** El agente se aferra rápidamente a las primeras acciones que le dieron una recompensa, aunque no sean parte de la ruta óptima. Le cuesta descubrir nuevos caminos, por lo que termina repitiendo secuencias sub-óptimas.
- **Alta Exploración ($\epsilon = 0{,}9$):** El agente actúa de forma casi aleatoria. Aunque eventualmente encuentra el objetivo, su camino es muy ineficiente y errático.

- **Valores Intermedios** ($\epsilon \in [0{,}3, 0{,}7]$)**:** Estos valores encontraron un mejor equilibrio. El agente fue capaz de explorar lo suficiente para no quedarse atascado, pero también de explotar el conocimiento adquirido para optimizar su ruta. La configuración **Conservadora** ($\epsilon = 0{,}3$) fue la más eficiente en términos de pasos.

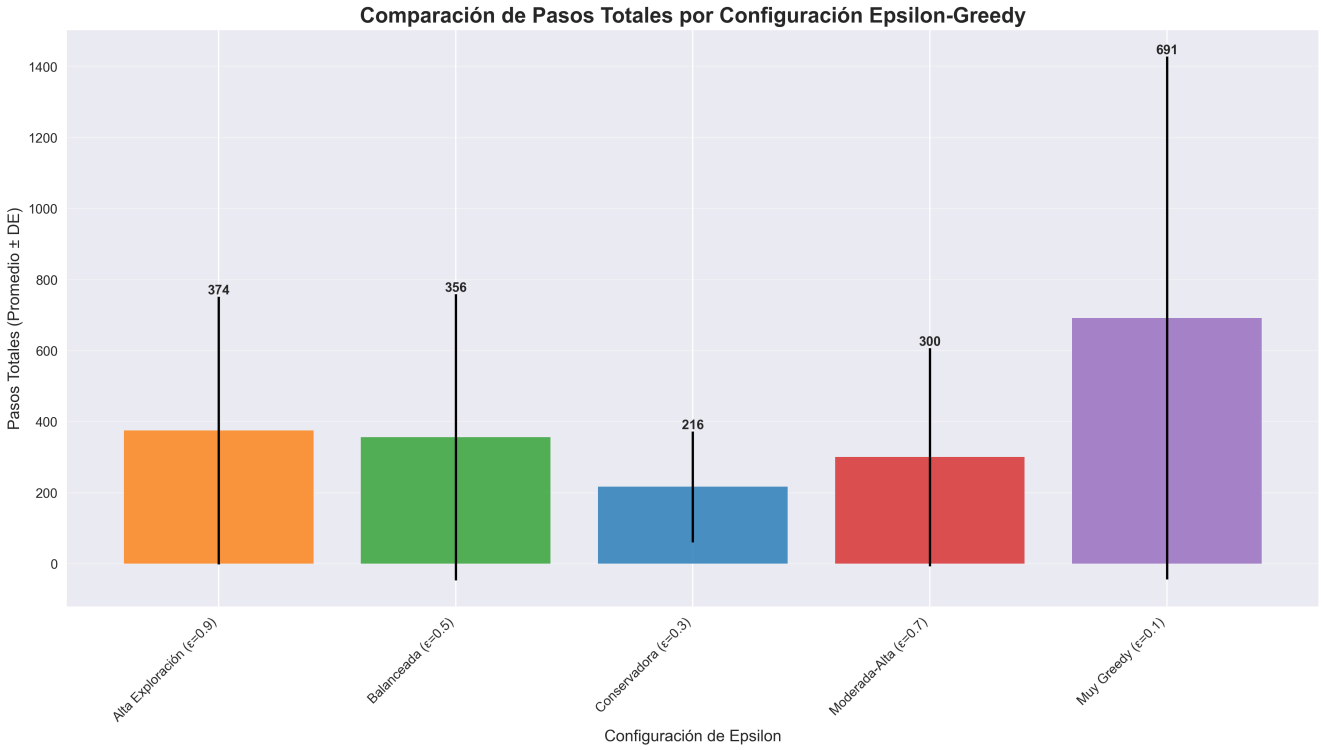


Figura 2: Comparación de Pasos Totales.

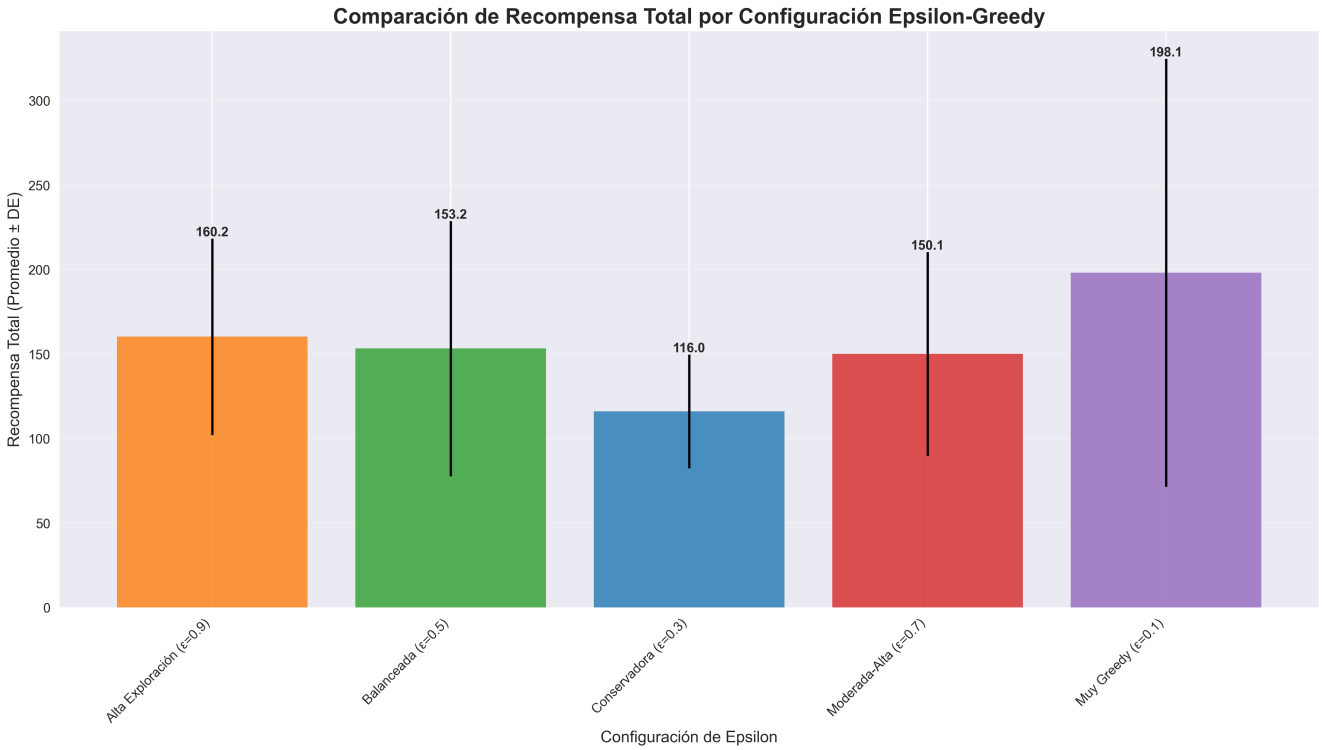


Figura 3: Comparación de Recompensa Total.

Como se observa en la Figura 2 y 3, la configuración con $\epsilon = 0{,}3$ no solo minimizó los pasos, sino que también mantuvo una recompensa competitiva, indicando una trayectoria de alta calidad. Todas las configuraciones lograron una tasa de éxito del 100 %, lo que demuestra que, aunque ineficiente en algunos casos, Epsilon-Greedy es un método robusto para este problema.

0.6.2. Comparativa General: PPO vs. Epsilon-Greedy

Al comparar las tres familias de algoritmos, emerge una clara jerarquía de rendimiento para esta tarea específica:

1. **PPO (Aprendizaje por Refuerzo Profundo):** El agente PPO, aunque mucho más lento que los de búsqueda, demostró una política de navegación coherente, completando la tarea en un promedio de 350 pasos y 23 segundos. Su fortaleza radica en que no necesita un modelo del mundo; aprende directamente de la experiencia (píxeles), lo que lo hace más generalizable a tareas donde el mapa no se conoce de antemano.

2. **Epsilon-Greedy:** Fue la estrategia menos eficiente en términos de tiempo y pasos. Aunque robusta, su método de aprendizaje simple y su exploración (a menudo aleatoria) la hacen inadecuada para problemas que requieren una secuencia larga y precisa de acciones, como lo

demuestran sus tiempos de ejecución, que fueron órdenes de magnitud mayores que los de PPO.

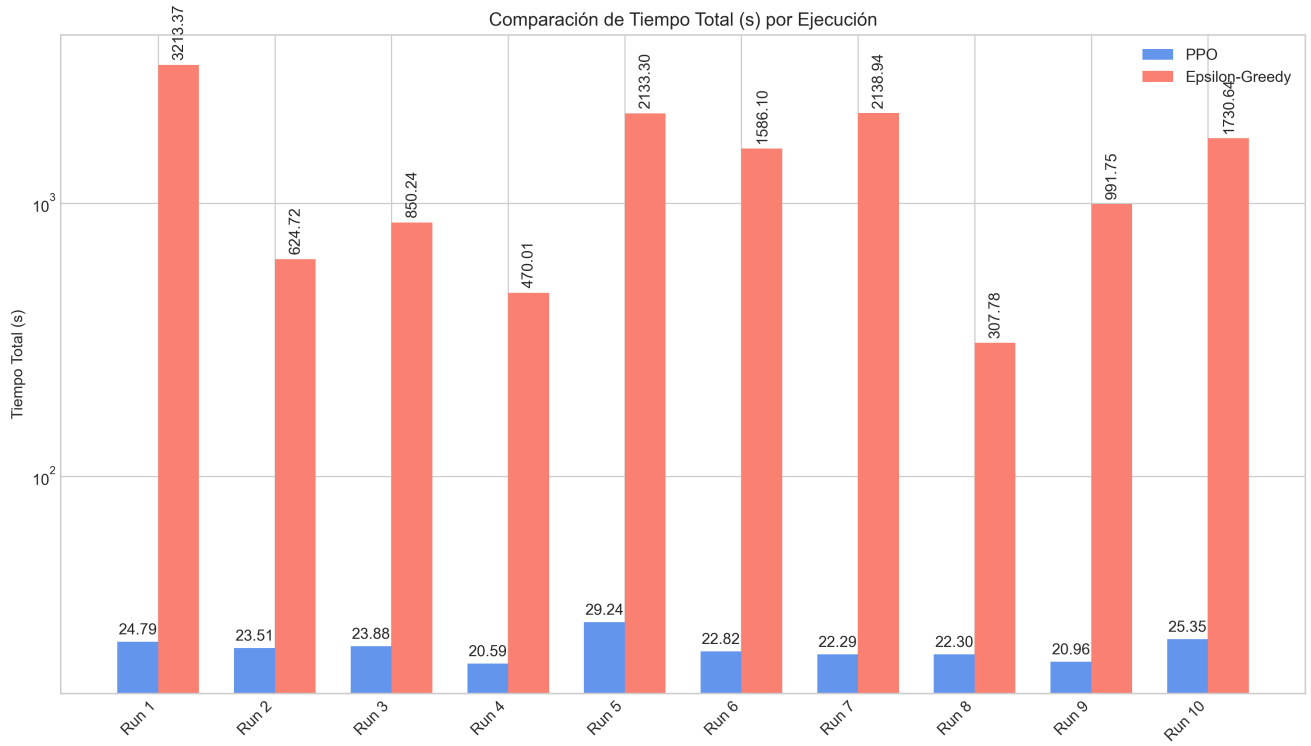


Figura 4: Comparación de Tiempos de Ejecución Totales por tipo de Agente. Nótese la escala logarítmica en el eje Y, que evidencia la enorme diferencia de rendimiento.

La Figura 4 ilustra dramáticamente esta diferencia. Mientras que los algoritmos de búsqueda resuelven la tarea casi instantáneamente, PPO toma segundos y Epsilon-Greedy puede tardar minutos.

0.7. Conclusiones

Este proyecto ha permitido realizar un análisis exhaustivo y comparativo de diferentes estrategias de inteligencia artificial aplicadas a un problema complejo y no estándar: la navegación inicial en *Pokémon Rojo*. Los resultados obtenidos nos llevan a las siguientes conclusiones clave:

0.8. Conclusiones

Este proyecto realizó un análisis comparativo riguroso de tres paradigmas de inteligencia artificial —búsqueda clásica, aprendizaje por refuerzo simple y aprendizaje por refuerzo profundo—

aplicados a la tarea de seleccionar el Pokémon inicial en *Pokémon Rojo*. A través de 143 ejecuciones controladas, se ha establecido una jerarquía de rendimiento clara y se han extraído conclusiones fundamentales sobre la idoneidad de cada enfoque en función de las características del problema.

1. **El conocimiento del entorno es el factor dominante.** El hallazgo más contundente es que la eficiencia de un agente está directamente ligada a la cantidad de información previa que posee sobre el entorno. Los algoritmos de búsqueda clásica, que operan con un mapa explícito del juego, resolvieron la tarea en órdenes de magnitud menos tiempo y pasos que cualquier otro método. Esto confirma un principio de ingeniería fundamental: si se dispone de un modelo preciso del problema, la planificación directa sobre ese modelo es casi siempre la solución más eficiente.

2. **PPO: una solución general pero costosa.** El agente PPO logró completar la tarea de forma consistente aprendiendo directamente desde los píxeles, sin ningún conocimiento previo del mapa. Este es su mayor mérito: su capacidad de generalización. Sin embargo, esta capacidad tiene un costo evidente: necesitó significativamente más pasos y tiempo de ejecución que los algoritmos de búsqueda. El agente tuvo que .ªprender.ª navegar un mundo que los otros agentes ya çonocían", lo que resultó en trayectorias menos directas y una mayor carga computacional por la inferencia de su red neuronal.

3. **Epsilon-Greedy: la ineficiencia de la exploración no dirigida.** La estrategia Epsilon-Greedy fue, por un amplio margen, la menos eficiente. Su mecanismo de exploración, basado en acciones aleatorias, es ineficaz en un entorno con recompensas escasas", donde la retroalimentación positiva solo se obtiene al final de una larga secuencia de acciones correctas. El agente se comporta como una persona caminando a ciegas en una habitación, esperando tropezar con la salida. Aunque es robusto y eventualmente encuentra la solución, su uso no es práctico para tareas que requieren precisión y eficiencia.

4. **Recomendación para problemas similares.** Basado en los resultados, la recomendación es clara. Si el entorno es desconocido o cambia dinámicamente, un enfoque de aprendizaje por refuerzo como PPO es viable, asumiendo que se puede costear el tiempo de entrenamiento y la menor eficiencia en la ejecución. Epsilon-Greedy debe ser relegado a un rol de línea base para medir la mejora de otros algoritmos, pero no como una solución final.

Para finalizar este informe, queremos enfocarnos en que el proyecto ha demostrado exitosamente cómo diferentes arquitecturas de agentes se desempeñan en un entorno de videojuego real, Pokémon Rojo. La conclusión principal no es que una familia de algoritmos sea intrínsecamente superior a otra, sino que la elección de la herramienta correcta depende fundamentalmente del problema a

resolver y de la información disponible, un principio central en el diseño de cualquier sistema de ingeniería inteligente.

Para finalizar, este trabajo no solo ha identificado las estrategias más efectivas para un objetivo concreto, sino que también ha proporcionado una profunda visión sobre las fortalezas, debilidades y compromisos inherentes a cada familia de algoritmos de inteligencia artificial cuando se enfrentan a un desafío del mundo real, amplio y, sobre todo, complejo.

0.9. Anexo: Demostración en Video

Para complementar el análisis presentado en este informe, se ha preparado una demostración en video que muestra el funcionamiento y la comparación de los diferentes algoritmos en tiempo real. El video permite visualizar de manera directa la eficiencia, velocidad y comportamiento de cada agente al enfrentarse a la tarea de seleccionar el Pokémon inicial.

El video se encuentra disponible en la siguiente dirección:

https://youtu.be/YHPC_FzwH0o