# Diseño e implementación de un entorno de aprendizaje por refuerzo para un juego de estrategia por turnos

Eric Rodríguez Merichal

1 de julio de 2025

**Resumen–** Este proyecto presenta el diseño e implementación de un entorno de simulación para el aprendizaje por refuerzo (reinforcement learning, RL) aplicado a un juego de estrategia por turnos. El objetivo es desarrollar un entorno que facilite el entrenamiento de agentes de RL, permitiéndoles aprender y adaptarse a diferentes situaciones tácticas. Para ello, se emplea la API de la librería Gymnasium y diversos algoritmos de aprendizaje por refuerzo, con el objetivo de comparar su efectividad en el entorno diseñado.

**Palabras clave–** Aprendizaje por refuerzo, Gymnasium, videojuegos de estrategia, IA en juegos, simulación de combate por turnos.

**Abstract–** This project presents the design and implementation of a simulation environment for reinforcement learning applied to a turn-based strategy game. The goal is to develop agents capable of adapting to different tactical situations using Gymnasium and RL algorithms. The work compares various techniques and analyzes their effectiveness in the created environment.

**Keywords–** Reinforcement learning, Gymnasium, strategy video games, AI in games, turn-based combat simulation.

## 1 Introducción - Contexto del trabajo

El aprendizaje por refuerzo (RL) es una técnica de aprendizaje automático en la que un agente aprende a tomar decisiones mediante la interacción con un entorno. A través de recompensas o penalizaciones recibidas por sus acciones, el agente ajusta su comportamiento con el objetivo de maximizar la recompensa acumulada a lo largo del tiempo, desarrollando así una estrategia óptima basada en la experiencia.

En este proyecto, se aplica RL al contexto de los **juegos de estrategia por turnos**, un tipo de entorno secuencial en el que dos equipos alternan sus decisiones, moviendo unidades sobre un tablero con el objetivo de eliminar al equipo rival mediante combate táctico. A diferencia de otros juegos donde existen múltiples condiciones de victoria (como controlar zonas o capturar edificios), en este trabajo se define un único objetivo claro: **eliminar todas las unidades del equipo contrario**.

El objetivo principal del trabajo es diseñar e implementar un **entorno de simulación compatible con Gymnasium** que reproduzca las mecánicas esenciales de un juego de estrategia por turnos. El entorno, titulado *Reward Wars*, permitirá entrenar agentes con distintos enfoques de aprendizaje por refuerzo y analizar su desempeño ante variaciones en el mapa, las unidades y las condiciones de combate.

El diseño del entorno se inspira en juegos como *Advance Wars* [15] o *Fire Emblem* [16], adaptando sus mecánicas a un marco simplificado orientado al aprendizaje automático.

Este documento se estructura de la siguiente forma: en primer lugar, se presentan los objetivos y la planificación del proyecto. A continuación, se expone el estado del arte relacionado con el aprendizaje por refuerzo y su aplicación a videojuegos, así como las herramientas utilizadas. Posteriormente, se describen el diseño del juego y la implementación del entorno de simulación. Finalmente, se analizan los resultados obtenidos tras el entrenamiento de los agentes, los problemas identificados y se discuten posibles mejoras futuras.

## 2 Objetivos

Los principales objetivos de este proyecto son:

1. Diseñar e implementar un entorno de juego de estrate-

gia por turnos compatible con Gymnasium.

2. Integrar y entrenar agentes de aprendizaje por refuerzo utilizando algoritmos ya implementados en la biblioteca Stable-Baselines3.
3. Comparar el desempeño de diferentes algoritmos de RL en la toma de decisiones estratégicas dentro del entorno.
4. Analizar el impacto de las condiciones del entorno en el comportamiento y rendimiento de los agentes entrenados.
5. Documentar y discutir los resultados obtenidos, identificando los problemas encontrados y proponiendo mejoras futuras.

# 3 Planificación del Proyecto

El desarrollo del proyecto se divide en cinco fases principales. Esta planificación permitirá hacer un seguimiento del progreso y asegurar el cumplimiento de los objetivos dentro del plazo establecido (ver Apéndice 3 para más detalles sobre la planificación).

## 3.1. Fases del Proyecto

- **Fase 1: Investigación y diseño** (4 semanas) En esta fase se estudian los fundamentos del aprendizaje por refuerzo y su aplicación en videojuegos, además de definir el diseño del juego.
  - Aprender sobre RL (3 semanas)
  - Revisar proyectos anteriores (1 semana)
  - Explorar cómo aplicar RL en videojuegos (1 semana)
  - Definir diseño del juego (1 semana)
  - Analizar mecánicas para facilitar el entrenamiento de la IA (2 semanas)
- **Fase 2: Diseño e implementación del entorno** (5 semanas) Se desarrolla la estructura del juego y su integración con `Gymnasium`, además de implementar el modo de juego para un jugador humano.
  - Definir reglas del juego (1 semana)
  - Diseñar tablero y unidades básicas (2 semanas)
  - Implementar mecánicas básicas (2 semanas)
  - Implementar jugador humano (1 semana)
- **Fase 3: Integración y entrenamiento de agentes RL** (6 semanas) Se integran los agentes de aprendizaje por refuerzo y se inician las primeras pruebas de entrenamiento con agentes preentrenados y agentes con políticas aleatorias.
  - Conectar entorno con `Gymnasium` (1 semana)
  - Cargar un agente de `Gymnasium` y hacer acciones aleatorias (1 semana)
  - Ajustar mecánicas (1 semana)
  - Ajustar recompensas y penalizaciones (2 semanas)
  - Entrenar agentes PPO y analizar resultados (2 semanas)
  - Entrenar agentes A2C y DQN y comparar (2 semanas)
- **Fase 4: Optimización y análisis** (3 semanas) Se analiza el rendimiento de los agentes y se realizan ajustes en el diseño del entorno si es necesario.
  - Identificar problemas en la IA (1 semana)
  - Ajustar mecánicas (1 semana)
  - Entrenar en escenarios más complejos (1 semana)
  - Comparar agentes nuevos vs. viejos (1 semana)
- **Fase 5: Documentación** (2 semanas) Se recopilan los resultados y se elabora la memoria final del TFG.
  - Redacción del informe final (1 semana)
  - Revisar y corregir memoria (1 semana)
  - Preparar presentación y defensa (1 semana)

# 4 Estado del Arte

El aprendizaje por refuerzo ha cobrado gran relevancia en el ámbito de la inteligencia artificial, especialmente en el desarrollo de agentes autónomos capaces de tomar decisiones en entornos dinámicos y complejos. En el contexto de los videojuegos, el RL ha demostrado ser una herramienta poderosa para entrenar agentes que aprenden estrategias óptimas sin necesidad de programación explícita. Esto ha permitido avances en la creación de enemigos más inteligentes, sistemas de dificultad adaptativa y agentes capaces de competir con jugadores humanos en juegos tanto de estrategia como de acción.

## 4.1. Fundamentos Teóricos

El aprendizaje por refuerzo es un paradigma del aprendizaje automático en el que un agente aprende a interactuar con un entorno mediante un sistema de recompensas y penalizaciones. Este proceso se basa en la interacción constante entre los siguientes elementos:

- **Agente:** Es quien toma decisiones con el objetivo de maximizar la recompensa acumulada. Sus elementos internos incluyen:
  - **Política:** Estrategia que define qué acción tomar ante cada observación.
  - **Función de valor:** Estimación de la recompensa total esperada desde un estado o tras realizar una acción, siguiendo una política determinada.
- **Entorno:** Es el sistema con el que el agente interactúa y que responde a sus acciones. Se compone de:
  - **Estado:** Representación completa de la situación actual del entorno (puede ser parcial u oculta para el agente).

- **Observación:** Información que el agente recibe del entorno (puede ser el estado completo o parcial).
- **Acción:** Comando enviado por el agente al entorno para influir en él.
- **Recompensa:** Valor numérico que el entorno devuelve al agente como consecuencia de su acción, indicando lo beneficiosa o perjudicial que ha sido.

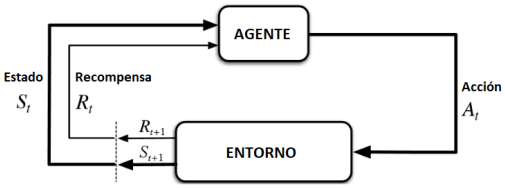


Fig. 1: Esquema general del ciclo de aprendizaje por refuerzo

A lo largo de los años se han desarrollado múltiples algoritmos de RL, cada uno con distintas estrategias de aprendizaje y optimización. En este proyecto se emplean tres de los más representativos:

- **Deep Q-Network (DQN)**: Algoritmo basado en aprendizaje por diferencia temporal (*Temporal Difference Learning*) que utiliza una red neuronal profunda para aproximar la función de valor de acción $Q(s, a)$, que estima la recompensa esperada a largo plazo tras ejecutar una acción $a$ en un estado $s$.

  Esta función se basa en la **ecuación de Bellman**, que define el valor de una acción como la recompensa inmediata más el valor estimado de la mejor acción futura:

$$Q(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q(s', a')] \quad (1)$$

  DQN es eficiente en espacios de acción discretos, pero presenta limitaciones en entornos donde la planificación a largo plazo es clave, debido a su exploración limitada y a cierta inestabilidad en la política [1].

- **Advantage Actor-Critic (A2C)**: Algoritmo híbrido que combina los enfoques basados en valor y política. Emplea dos redes: una (*Actor*) que aprende la política óptima $\pi(a|s)$, y otra (*Crítico*) que evalúa las acciones usando la función de ventaja $A(s, a)$, la cual mide cuán beneficiosa es una acción en comparación con la media esperada. Esta combinación reduce la varianza y mejora la estabilidad del entrenamiento [3].

- **Proximal Policy Optimization (PPO)**: Algoritmo basado en políticas, donde una política $\pi(a|s)$ define la probabilidad de ejecutar una acción $a$ en un estado $s$. PPO ajusta esta política de forma gradual aplicando restricciones en la función de pérdida, lo que evita cambios bruscos en las actualizaciones. Destaca por su estabilidad y eficiencia, siendo especialmente útil en entornos complejos o con espacios de acción más amplios [2].

- **Maskable PPO**: Variante del algoritmo PPO que incorpora máscaras dinámicas para restringir acciones inválidas, lo que resulta especialmente útil en entornos con múltiples restricciones contextuales [14].

Estos algoritmos han demostrado su eficacia en múltiples contextos, desde juegos clásicos de Atari hasta entornos más sofisticados como *StarCraft II* [17] o *Dota 2* [18].

## 4.2. Revisión de Trabajos Previos

La aplicación de RL a videojuegos ha sido objeto de numerosos estudios. Algunos de los más relevantes son:

- **AlphaStar (DeepMind, 2019)**: Agente basado en RL que superó a jugadores humanos en *StarCraft II*, demostrando la capacidad del RL para abordar tareas de planificación estratégica compleja [4].
- **OpenAI Five (OpenAI, 2018)**: Sistema basado en aprendizaje profundo y RL que logró coordinación táctica entre múltiples agentes en partidas de *Dota 2*, un entorno multijugador altamente dinámico [5].
- **Juegos de mesa**: Proyectos como AlphaGo y variantes aplicadas al ajedrez han impulsado avances en búsqueda de árboles, redes de política y funciones de valor [6].

Si bien estos logros se han producido en entornos complejos y en su mayoría en tiempo real, hay una clara escasez de investigaciones aplicadas a videojuegos de estrategia por turnos con control de unidades individuales, como *Fire Emblem* o *Advance Wars*. En estos títulos, las decisiones tácticas tienen un alto grado de secuencialidad y estructura espacial. Este trabajo explora cómo el aprendizaje por refuerzo puede ser aplicado a este tipo de entornos, investigando si los agentes pueden adaptarse a situaciones tácticas sin depender de reglas programadas explícitamente.

## 4.3. Tecnologías y Herramientas Utilizadas

Para el desarrollo del entorno y entrenamiento de los agentes, se han utilizado las siguientes herramientas:

- **Gymnasium**: Biblioteca estándar para la implementación y evaluación de entornos de RL. Se emplea para diseñar el entorno personalizado que simula el juego de estrategia por turnos [7].
- **Stable-Baselines3 (SB3)**: Biblioteca que proporciona implementaciones robustas de algoritmos de RL como PPO, A2C y DQN. En este proyecto, se utilizan estos algoritmos como base para entrenar agentes dentro del entorno desarrollado [9].
- **Pygame**: Utilizada para crear la visualización del entorno, permitiendo representar gráficamente los turnos, unidades y elementos del mapa [8].

## 4.4. Limitaciones y Desafíos del Estado del Arte

A pesar de los avances recientes, el aprendizaje por refuerzo aún presenta desafíos importantes en su aplicación a juegos de estrategia por turnos:

- **Generalización del aprendizaje**: Los agentes pueden tener dificultades para transferir lo aprendido a configuraciones nuevas, especialmente si difieren mucho de los escenarios de entrenamiento [10].
- **Equilibrio entre exploración y explotación**: Encontrar estrategias eficaces sin depender excesivamente de recompensas inmediatas sigue siendo un reto, particularmente en entornos con retroalimentación escasa o diferida [11].
- **Coste computacional**: El entrenamiento de agentes en entornos con múltiples unidades y reglas complejas puede requerir largos tiempos de cómputo y recursos significativos [12].

Este trabajo busca contribuir a la mejora de estas limitaciones mediante el diseño de un entorno simplificado pero tácticamente significativo, donde se puedan evaluar distintas estrategias de RL en condiciones progresivamente más complejas.

# 5 Diseño del Juego

*Reward Wars*, se ha desarrollado siguiendo un modelo basado en casillas, donde dos equipos se enfrentan por turnos. Cada unidad pertenece a una de tres clases (Soldado, Arquero y Caballero) y se desplaza y ataca una casilla en direcciones ortogonales.

El terreno influye en la estrategia: los bosques ofrecen defensa adicional, los campamentos curan a las unidades que terminan su turno sobre ellos, las colinas otorgan bonificaciones ofensivas a los arqueros, y los obstáculos bloquean completamente el paso. El objetivo del juego es **eliminar a todas las unidades del equipo contrario**. Esto fomenta el enfrentamiento directo y la gestión táctica del posicionamiento y del combate.

Una vista general del entorno del juego puede consultarse en la Figura 4, incluida en el Apéndice.

## 5.1. Mecánicas de Juego

El juego sigue una estructura de **turnos alternos**, donde cada jugador activa a sus unidades una por una en orden, completando primero la fase de movimiento y después la fase de ataque. Es decir, cada unidad actúa en dos fases consecutivas:

- **Fase de movimiento**: La unidad puede moverse una casilla en dirección ortogonal o quedarse en su lugar.
- **Fase de ataque**: La unidad puede atacar en una dirección válida según su tipo o decidir no atacar.

El agente debe aprender a coordinar estas fases para optimizar su posicionamiento y daño.

## 5.2. Control de Acciones Inválidas

El entorno permite la ejecución de acciones inválidas (como moverse fuera del mapa, chocar con aliados u obstáculos, o atacar sin enemigos en rango), pero aplica penalizaciones ligeras. Este enfoque favorece el aprendizaje autónomo del agente mediante retroalimentación negativa, permitiendo que descubra qué decisiones evitar sin necesidad de restricciones explícitas.

## 5.3. Tipos de unidades y triángulo de debilidades

Cada jugador controla un ejército compuesto por tres tipos de unidades, cada una con fortalezas, debilidades y mecánicas específicas. El sistema sigue un **triángulo de ventajas y desventajas**, similar al de otros juegos de estrategia:

- **Soldado**: Unidad equilibrada de combate cuerpo a cuerpo. Se mueve una casilla en dirección ortogonal y ataca a enemigos adyacentes. Es fuerte contra arqueros, pero débil frente a caballería.
- **Arquero**: Unidad de ataque a distancia. Se mueve una casilla ortogonal y ataca en línea recta a una distancia de 1 a 3 casillas, impactando automáticamente al **primer enemigo que encuentre en esa dirección**. Es fuerte contra caballería, pero vulnerable contra soldados. Además, si ataca desde una colina, inflige más daño gracias a la bonificación de altura.
- **Caballero**: Unidad especializada en embestidas. Se mueve una casilla ortogonal y, al atacar, empuja al enemigo una casilla en la misma dirección. Si el enemigo no puede ser empujado (por un obstáculo o unidad), recibe daño adicional. Es fuerte contra soldados, pero débil frente a arqueros.

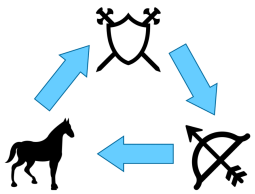

Fig. 2: Triángulo de debilidades

## 5.4. Condiciones de Victoria

Un jugador gana la partida si logra **eliminar a todas las unidades del equipo rival**. Esto simplifica el diseño del entorno y enfoca el aprendizaje del agente en tácticas de combate, posicionamiento eficiente y aprovechamiento del terreno.

# 6 Implementación y Arquitectura del Entorno

El entorno ha sido desarrollado en `Python` utilizando la biblioteca `Gymnasium`, siguiendo su interfaz estándar de entornos de aprendizaje por refuerzo. La lógica del juego por turnos, previamente descrita, ha sido traducida a una arquitectura modular que gestiona el estado del tablero, la secuencia de turnos, las acciones válidas y el sistema de recompensas. Las observaciones se representan como tensores multicanal y el espacio de acciones permite controlar

a las unidades en fases secuenciales. El entorno es compatible con algoritmos de RL como PPO, A2C, DQN y MaskablePPO, y ha sido diseñado para entrenamientos progresivos mediante *transfer learning*.

El código fuente del entorno y los modelos entrenados está disponible en el repositorio público https://github.com/Ciireex/TFG_1496793.git

## 6.1. Estructura general

La clase principal del entorno es `StrategyEnv`, que hereda de `gymnasium.Env`, la clase base de la biblioteca Gymnasium. Esta define la interfaz estándar que deben seguir todos los entornos compatibles con algoritmos de aprendizaje por refuerzo.

La API de Gymnasium exige la implementación de dos métodos fundamentales:

- `reset()`: reinicia el entorno y devuelve la primera observación del nuevo episodio.
- `step(action)`: aplica una acción, devuelve la nueva observación, la recompensa, un indicador de si el episodio ha terminado (`done`) y un diccionario con información adicional (`info`).

Esta estructura permite que los algoritmos de RL interactúen con cualquier entorno de forma estandarizada, sin necesidad de conocer su lógica interna.

`StrategyEnv` implementa esta interfaz y gestiona el flujo completo del episodio: inicializa el tablero, controla las fases de cada unidad (`move` y `attack`), calcula recompensas, aplica las reglas del juego y detecta el final del episodio.

Durante la función `reset()`, se colocan las unidades en posiciones aleatorias dentro de las dos filas más cercanas a su lado del tablero, evitando casillas con obstáculos. El terreno incluye casillas especiales como bosques, colinas, campamentos y obstáculos, generados aleatoriamente pero garantizando caminos transitables.

## 6.2. Observaciones y acciones

Cada observación es una matriz tridimensional de dimensiones $(21, H, W)$, donde $H \times W$ es el tamaño del tablero (por ejemplo, $6 \times 4$ o $7 \times 5$), y cada uno de los 21 canales representa una capa de información distinta:

- **Canal 0**: Obstáculos (casillas bloqueadas).
- **Canales 1–3**: Presencia de unidades aliadas (Soldado, Arquero, Caballero).
- **Canales 4–6**: Salud de unidades aliadas por clase.
- **Canales 7–9**: Presencia de unidades enemigas (Soldado, Arquero, Caballero).
- **Canales 10–12**: Salud de unidades enemigas por clase.
- **Canal 13**: Unidad activa actual.
- **Canal 14**: Casillas válidas para moverse.
- **Canal 15**: Casillas en las que se puede atacar.
- **Canal 16**: Fase actual (1 si es `attack`, 0 si es `move`).
- **Canal 17**: Porcentaje del número de turnos completado.
- **Canal 18**: Jugador activo (0 para azul, 1 para rojo).
- **Canales 19–21**: Tipos de terreno especiales (bosque, colina y campamento)

El espacio de acciones está definido como `Discrete(5)`, lo que indica que el agente debe seleccionar una acción entre cinco posibles opciones codificadas como enteros del 0 al 4:

- 0: No hacer nada (pasar turno).
- 1–4: Ejecutar una acción en dirección ortogonal: izquierda, derecha, arriba, abajo.

El significado exacto de la acción seleccionada depende de la fase actual (`move` o `attack`), que el agente debe interpretar a partir de la observación.

## 6.3. Interacción y flujo del episodio

Cada unidad activa debe completar una fase de movimiento y una fase de ataque. Tras realizar ambas fases, se pasa a la siguiente unidad del mismo equipo. Cuando todas las unidades han actuado, se alterna el turno al equipo contrario.

- En la fase `move`, la unidad puede desplazarse una casilla ortogonal libre. Si no lo hace (por acción 0 o por falta de movimientos válidos), permanece en su lugar.
- En la fase `attack`, puede atacar en una dirección válida según su tipo o decidir no atacar.

Si la acción es inválida (por ejemplo, colisionar con otra unidad u obstáculo, o atacar sin objetivo), se aplica una penalización ligera. Además, si la unidad activa no tiene movimientos válidos disponibles, se penaliza con un pequeño castigo adicional.

El método `step()` devuelve la tupla estándar de Gymnasium:

```
(observation, reward, terminated,
        truncated, info)
```

## 6.4. Recompensas y finalización

El sistema de recompensas utiliza *reward shaping*, es decir, señales intermedias diseñadas para guiar el aprendizaje hacia un comportamiento táctico deseado. Se otorgan bonificaciones por moverse a casillas ventajosas, acercarse a enemigos, atacar desde buena posición o explotar el triángulo de debilidades. El caballero gana puntos extra al empujar contra obstáculos, y el arquero por atacar desde su distancia óptima. Acciones inválidas o pasivas se penalizan levemente. Eliminar enemigos y ganar la partida otorgan recompensas altas. Si se alcanza el límite de turnos, se penaliza según el número relativo de unidades vivas. El episodio termina al eliminar a todo el equipo rival o alcanzar el máximo de turnos.

### 6.5. Componentes principales

Los principales módulos del entorno son:

- **`Unit` y subclases**: Representan a las unidades del juego, incluyendo `Soldier`, `Archer` y `Knight`. Cada clase define su comportamiento específico, como el alcance de ataque, las interacciones especiales o los efectos del terreno sobre su rendimiento.
- **`StrategyEnv`**: Clase principal del entorno, encargada de gestionar el estado del juego, los turnos, la ejecución de acciones, la validación de movimientos y ataques, el sistema de recompensas y la detección de condiciones de finalización.
- **`Renderer`**: Módulo visual implementado con `pygame` que permite representar gráficamente el estado del tablero en tiempo real. Muestra las unidades de ambos equipos, la unidad activa, su rango de movimiento y ataque, las fases del turno y animaciones como ataques, desplazamientos y muertes.

### 6.6. Uso de una red convolucional (EnhancedTacticalFeatureExtractor)

Para mejorar la capacidad del agente de interpretar el entorno a partir de observaciones espaciales, se ha implementado un extractor de características personalizado basado en redes convolucionales con atención, denominado `EnhancedTacticalFeatureExtractor`. Este módulo reemplaza al extractor por defecto de `Stable-Baselines3` y está diseñado para procesar observaciones multicanal que representan el estado completo del tablero.

Su arquitectura se basa en un bloque convolucional híbrido (`HybridCNN`) compuesto por tres capas convolucionales con activación ReLU. Además, incorpora mecanismos de atención de canal (`ChannelAttention`) y atención espacial dinámica (`DynamicSpatialAttention`) para mejorar la sensibilidad del modelo a regiones relevantes del mapa. La red incluye normalización por lotes, `Dropout2d` y un módulo de `AdaptiveAvgPool2d` que condensa las representaciones espaciales a un tamaño fijo $2 \times 2$, permitiendo su uso en tableros de distintos tamaños.

Tras aplanar la salida, se aplica una capa totalmente conectada con `LayerNorm`. De forma adicional, se extrae y procesa por separado el valor medio del canal correspondiente al jugador activo, combinando ambas salidas en la capa final. Esta arquitectura modular permite capturar patrones espaciales complejos y facilita el aprendizaje de estrategias eficaces en el entorno.

## 7 Entrenamiento de los Agentes de Aprendizaje por Refuerzo

En esta sección se detallan los algoritmos de aprendizaje por refuerzo entrenados en el entorno desarrollado. El objetivo es comparar su rendimiento y analizar sus comportamientos en escenarios de complejidad creciente. Se han entrenado cuatro algoritmos: **PPO**, **A2C**, **DQN** y **Maskable PPO**, aplicados de forma progresiva mediante *transfer learning*.

### 7.1. Algoritmos entrenados

En la fase inicial (Fase 1), los agentes se enfrentan a una IA heurística simple para aprender las mecánicas básicas del entorno. A partir de la Fase 2, cada agente entrena enfrentándose a una versión congelada entrenada en la fase anterior, lo que permite evaluar su evolución táctica de forma incremental.

Cada fase consta generalmente de **500 mil pasos de entrenamiento**, salvo la fase final (en un mapa más grande), donde se emplea **1 millón de pasos** para favorecer la consolidación de estrategias. Todo el proceso ha sido monitorizado mediante **TensorBoard**, permitiendo visualizar métricas clave como la recompensa media, la pérdida de valor, la entropía y la estabilidad de la política.

A continuación, se describen las observaciones más relevantes de cada modelo según los datos registrados.

### 7.2. Interpretación de métricas

Las tablas anteriores recogen las métricas más relevantes extraídas de los logs de entrenamiento de TensorBoard. A continuación, se explican brevemente los indicadores más utilizados:

- **loss**: Representa la pérdida total del modelo, habitualmente la suma o promedio de otras pérdidas (de política y valor). Un valor bajo indica mejor ajuste y convergencia.
- **approx_kl**: Aproximación de la divergencia KL entre la política nueva y la anterior. Valores muy altos indican actualizaciones demasiado agresivas; valores estables reflejan buen control sobre la política.
- **explained_variance**: Proporción de varianza de la recompensa explicada por la red crítica. Cuanto más cercana a 1, mejor capacidad de predicción tiene el valor estimado. Valores negativos indican un crítico poco informativo.
- **entropy_loss**: Refleja la aleatoriedad de la política. Valores negativos cercanos a cero indican una política determinista, mientras que valores más bajos (más negativos) corresponden a una mayor exploración.
- **value_loss**: Error cuadrático medio al predecir el valor esperado. Cuanto más bajo, mejor es la estimación del crítico sobre el estado actual.
- **clip_fraction**: Fracción de actualizaciones donde la política fue recortada por el algoritmo PPO. Valores moderados (alrededor de 0.1) suelen indicar un aprendizaje estable.
- **policy_gradient_loss**: Pérdida asociada a la política. Su magnitud refleja el grado de actualización aplicada a la política durante el entrenamiento.
- **learning_rate**: Tasa de aprendizaje utilizada. En estos experimentos se mantuvo fija para cada modelo.

Estas métricas permiten analizar no solo la eficacia del agente, sino también la estabilidad y eficiencia del proceso de entrenamiento.

## 7.3. Procedimiento y configuración del entrenamiento

El entrenamiento se ha realizado de forma progresiva mediante **transfer learning**, reutilizando los pesos aprendidos en cada fase para escalar la complejidad del entorno. En la Fase 1 se usaron los wrappers `DummyRedWrapper` y `DummyBlueWrapper`, donde uno de los equipos permanece inactivo para facilitar el aprendizaje inicial. Desde la Fase 2, los agentes se enfrentan a versiones congeladas entrenadas en la fase anterior.

Se han utilizado los algoritmos `PPO`, `A2C`, `DQN` y `MaskablePPO`, mediante la biblioteca `Stable-Baselines3` y el vectorizador `DummyVecEnv`. Las observaciones espaciales multicanal se procesan con un extractor convolucional personalizado (`TacticalFeatureExtractor`), registrado con `policy_kwargs`.

El proceso se divide en 7 fases:

- **Fases 1–3 (Fundamentos y obstáculos):** Entrenamiento con soldados en tableros pequeños ($4 \times 4$ y $6 \times 4$), incorporando obstáculos progresivamente para fomentar exploración y planificación.
- **Fases 4–5 (Nuevas unidades):** Se introducen arqueros y caballeros, ampliando las mecánicas de ataque a distancia y empuje.
- **Fase 6 (Terreno especial):** Se activan casillas estratégicas (bosques, colinas y campamentos), afectando movimiento y defensa.
- **Fase 7 (Entorno final):** Mapa de $8 \times 6$ con todos los elementos anteriores. Es la fase usada para la comparación de modelos, al ser la más representativa del juego completo.

## 7.4. PPO

A lo largo de las fases 1 a 6, el agente PPO mostró una mejora progresiva tanto en rendimiento como en estabilidad a medida que aumentaba la complejidad del entorno. En las primeras fases, con soldados únicamente y sin obstáculos, aprendía rápido y resolvía los episodios de forma eficaz. A medida que se añadían obstáculos, unidades a distancia, caballería y terreno con efectos especiales, el rendimiento fue más variable entre agentes, pero en general mantuvo una buena capacidad de adaptación. El agente azul destacó por su estabilidad y eficiencia, mientras que otros agentes exploraban más pero con resultados menos consistentes.

En la Fase 7, el **agente azul mostró un rendimiento superior**, con una recompensa media por episodio más alta (35.87 frente a 33.27 del rojo) y resolviendo los episodios en menos pasos (370 frente a 467). También obtuvo un valor de `explained_variance` notablemente mayor (0.88 frente a 0.66), lo que indica que su red crítica aprendió a estimar con mayor precisión el valor esperado de los estados.

Por otro lado, el agente rojo exploró más intensamente, como refleja su menor `entropy_loss` (-1.09 frente a -0.68), pero sin alcanzar la misma eficacia en recompensa ni estabilidad. El azul mantuvo un `value_loss` más bajo y una política más consolidada, lo que le permitió resolver las partidas con mayor consistencia.

La Tabla 1 resume las métricas más representativas del rendimiento de ambos agentes en esta fase final.

TABLA 1: RESUMEN DE MÉTRICAS FINALES EN FASE 7 (PPO AZUL Y ROJO).

| Métrica | PPO Azul | PPO Rojo |
|---|---|---|
| Recompensa media | **32.15** | 31.19 |
| Longitud episodio | **369.50** | 467.00 |
| Policy loss | **-0.06** | -0.07 |
| Value loss | **0.26** | 0.32 |
| Entropy loss | -0.68 | **-1.01** |
| Explained variance | **0.88** | 0.67 |

Las gráficas completas con la evolución de estas métricas pueden consultarse en el Apéndice 5.

## 7.5. A2C

A lo largo de las fases 1 a 6, el agente A2C mostró una buena capacidad de adaptación en entornos simples y con pocas unidades, obteniendo recompensas altas y episodios cortos. Sin embargo, a medida que se incrementaba la complejidad del entorno (obstáculos, variedad de unidades y terrenos), su rendimiento se volvió más inestable, especialmente en las fases con terreno, donde los episodios se alargaron y las recompensas disminuyeron.

En la Fase 7, el entorno más completo del entrenamiento, el agente rojo mostró una **recompensa media por episodio** significativamente más alta (34.98 frente a 21.30 del azul) y resolvió los episodios en menos pasos (495 frente a 535), lo que indica una mejor capacidad para adaptarse a situaciones complejas. Su valor de `explained_variance` también fue más alto (0.9618 frente a 0.9293), reflejando una estimación más precisa del valor futuro.

Por otro lado, el agente azul destacó por tener un `value_loss` y `policy_loss` más bajos, lo que sugiere una política más estable y conservadora, aunque menos eficaz en términos de puntuación. Ambos agentes convergen a una política determinista, como indica el valor casi nulo de `entropy_loss`.

La Tabla 2 recoge las métricas finales más relevantes en esta fase.

TABLA 2: RESUMEN DE MÉTRICAS FINALES EN FASE 7 (A2C AZUL Y ROJO).

| Métrica | A2C Azul | A2C Rojo |
|---|---|---|
| Recompensa media | 21.30 | **34.98** |
| Longitud episodio | 535.24 | **495.96** |
| Policy loss | **-0.50** | 3.01 |
| Value loss | **0.17** | 0.19 |
| Entropy loss | 0.00 | 0.00 |
| Explained variance | 0.93 | **0.96** |

Las gráficas completas con la evolución de estas métricas pueden consultarse en el Apéndice 7.

## 7.6. DQN

DQN se basa exclusivamente en la estimación de la función de valor $Q(s, a)$, sin una política explícita ni com-

ponente crítico, por lo que ofrece menos métricas durante el entrenamiento. La principal es la **total loss**, que mide el error cuadrático medio entre $Q(s, a)$ y su objetivo $r + \gamma \max Q(s', a')$, según la ecuación de Bellman. Su reducción indica una mejor aproximación del valor esperado.

Las observaciones espaciales se procesan mediante una red convolucional, cuya salida se aplana antes de pasar por capas densas, generando una representación vectorial del tablero útil para predecir $Q(s, a)$.

A diferencia de PPO y A2C, DQN no utiliza entrenamiento por fases, ya que no se beneficia de la transferencia de pesos entre mapas: cualquier cambio en el tamaño del tablero, posición de unidades u obstáculos puede invalidar los valores aprendidos. Por eso, el entrenamiento se realizó directamente sobre el entorno final (Fase 7) desde cero.

El agente azul fue entrenado enfrentando a un modelo de **MaskablePPO rojo**, elegido por su alto rendimiento. Luego, el agente rojo se entrenó contra el DQN azul ya entrenado. Esta configuración permite evaluar la capacidad de DQN para adaptarse a políticas fuertes.

La Tabla 3 muestra las métricas tras 1 millón de pasos. El agente rojo logró mejor **recompensa media** tanto en entrenamiento (39.97 vs. 37.29) como en evaluación (39.83 vs. 39.48), completando además los episodios en menos pasos y con una **pérdida total** ligeramente inferior.

TABLA 3: RESUMEN DE MÉTRICAS FINALES EN FASE 7 (DQN AZUL Y ROJO).

| Métrica | DQN Azul | DQN Rojo |
|---|---|---|
| Recompensa media | 39.48 | **39.83** |
| Longitud episodio | 90.2 | **150.83** |
| Total loss | 0.117 | **0.102** |

La evolución de estas métricas puede consultarse en el Apéndice 8.

## 7.7. MaskablePPO

En la fase 7, ambos agentes han consolidado su rendimiento, pero el agente azul destaca en varias métricas clave. Obtiene una mayor recompensa media por episodio (`rollout/ep_rew_mean`) y una menor longitud media de los episodios (`rollout/ep_len_mean`), lo cual indica que resuelve las partidas de forma más eficiente. Su `value_loss` también es más bajo, lo que sugiere un modelo de valor más preciso.

Por otro lado, el agente rojo presenta una ligera ventaja en `policy_gradient_loss` y en `entropy_loss`, lo cual podría reflejar una política más exploratoria. Sin embargo, su `explained_variance` es algo menor, indicando que su red crítica tiene una estimación menos precisa del valor esperado.

Ambos agentes han convergido con un `learning_rate` estable y unas pérdidas totales bajas, lo que indica un entrenamiento correctamente finalizado. La tabla 4 resume las métricas finales más relevantes para esta fase.

TABLA 4: RESUMEN DE MÉTRICAS FINALES EN FASE 7 (MASKABLEPPO (MPPO) AZUL Y ROJO).

| Métrica | MPPO Azul | MPPO Rojo |
|---|---|---|
| Recompensa media | **40.60** | 35.88 |
| Longitud episodio | **357.56** | 344.30 |
| Loss total | **-0.02** | -0.03 |
| KL Divergence | **0.12** | 0.17 |
| Entropy loss | -0.24 | **-0.22** |
| Explained variance | 0.92 | **0.97** |
| Value loss | **0.13** | 0.15 |
| Clip fraction | 0.22 | **0.19** |
| Policy gradient loss | -0.03 | **-0.02** |

Las gráficas completas con la evolución de estas métricas se encuentran en el Apéndice 6.

# 8 RESULTADOS Y DISCUSIÓN

## 8.1. Resultados entre modelos entrenados (Fase 7)

Todos los modelos entrenados en la Fase 7 se enfrentaron entre sí en duelos de 100 partidas por combinación. Los agentes entrenados con **DQN** y **Maskable PPO** fueron los más eficaces, destacando en número de victorias y puntuación global. El modelo `dqn_blue` obtuvo la mejor puntuación total (**985 puntos**), seguido muy de cerca por `maskableppo_blue` (**974 puntos**), lo que confirma su capacidad de generalización y dominio estratégico.

Aunque los modelos azules dominaron la mayoría de los cruces, `ppo_red` logró una puntuación especialmente alta entre los modelos rojos (**704 puntos**), superando incluso a varios modelos azules gracias a su gran cantidad de empates. En contraste, `a2c_red` fue el agente con peor desempeño, acumulando solo **141 puntos**.

Para facilitar la comparación cuantitativa, se ha calculado una **puntuación total** asignando **3 puntos por victoria**, **1 por empate** y **0 por derrota**. En el caso de los modelos rojos (que figuran como oponentes en la tabla), se ha contado cada derrota del rival como una victoria propia, permitiendo una comparación equitativa entre todos los agentes. La siguiente tabla muestra la clasificación final:

TABLA 5: PUNTUACIÓN TOTAL POR MODELO EN FASE 7, ORDENADA DE MAYOR A MENOR. (VICTORIA = 3 PTS, EMPATE = 1 PT, DERROTA = 0 PT)

| Modelo | Puntos totales |
|---|---|
| dqn_blue | **985** |
| maskableppo_blue | **974** |
| ppo_red | 704 |
| a2c_blue | 662 |
| maskableppo_red | 613 |
| dqn_red | 359 |
| ppo_blue | 306 |
| a2c_red | 141 |

Los resultados detallados de los duelos entre modelos pueden consultarse en la Tabla 6, incluida en el Apéndice.

## 8.2. Discusión de los resultados

El análisis de las puntuaciones revela diferencias significativas en el rendimiento de los algoritmos evaluados, explicables por sus características internas y su capacidad de adaptación al entorno.

- **DQN (azul)** fue el agente con mejor puntuación global. Este resultado puede deberse a su política determinista y su capacidad para consolidar comportamientos óptimos a través de su memoria de repetición. DQN suele ser eficaz en entornos donde las acciones válidas son claras y las consecuencias son relativamente predecibles, como es el caso de este entorno por turnos. Su estilo resolutivo también indica una agresividad táctica eficiente que evita empates prolongados.

- **Maskable PPO (azul)** mostró un rendimiento muy cercano al de DQN. Su éxito se explica en parte por el uso de *masking*, que impide elegir acciones inválidas. Esto resulta especialmente valioso en entornos con múltiples restricciones de movimiento o ataque. Además, como algoritmo basado en políticas, tiene una mejor exploración que DQN, lo que le permite adaptarse bien a escenarios más complejos y variados.

- **PPO (rojo)** sorprendió al obtener una puntuación alta a pesar de no destacar en victorias. Su capacidad para empatar sistemáticamente sugiere un comportamiento conservador, posiblemente fruto de una política que minimiza errores pero también evita riesgos. Es probable que este agente haya aprendido a sobrevivir sin optimizar el cierre de partidas, lo cual penaliza su efectividad ofensiva pero premia su solidez defensiva.

- **A2C (azul)** logró un rendimiento aceptable. Al ser un algoritmo menos sofisticado en la exploración que PPO o Maskable PPO, su estrategia parece centrarse en patrones de acción más simples. Esto le permite alcanzar cierta estabilidad pero lo limita frente a oponentes que requieren respuestas más adaptativas. El equilibrio entre victorias y empates sugiere un agente funcional, pero con margen de mejora.

- **PPO (azul)** fue el modelo azul con peor puntuación. A pesar de ser un algoritmo robusto, su alto número de empates y bajas victorias indican que no logró encontrar estrategias eficaces. Esto puede deberse a una mala convergencia en el entrenamiento o a una política demasiado cauta. En entornos donde se premia la acción resolutiva, una actitud excesivamente defensiva resulta en desventaja.

- **Maskable PPO (rojo)** obtuvo un buen rendimiento, aunque algo por debajo de su versión azul. Esto podría deberse a que, al figurar como oponente en la tabla, no aprovechó plenamente el masking en las situaciones decisivas. Aun así, su puntuación demuestra una buena comprensión del entorno.

- **A2C (rojo)** fue el modelo con peor puntuación global. Su bajo número de victorias y predominancia de empates indican una política poco eficaz, probablemente basada en mantener la posición sin ejecutar acciones ganadoras. Este comportamiento puede haber sido reforzado si el entrenamiento priorizó evitar derrotas en lugar de buscar la victoria.

En conjunto, los resultados reflejan que **DQN** y **Maskable PPO** son los algoritmos más resolutivos y adaptativos para este entorno de estrategia por turnos, mientras que **PPO** y **A2C** presentan limitaciones ligadas a una menor agresividad o a estrategias más estáticas. La precisión táctica, la capacidad de exploración efectiva y la gestión de acciones válidas son factores clave que marcan la diferencia entre un agente competitivo y uno meramente funcional.

# 9 Problemas Encontrados y Mejoras Futuras

## 9.1. Problemas Durante el Desarrollo

Durante el diseño del entorno y el entrenamiento de los agentes se identificaron varios problemas clave. Inicialmente, las recompensas mal ajustadas provocaban comportamientos subóptimos, como moverse sin combatir, por lo que se reformuló el sistema hacia uno más simple y centrado en los objetivos reales del juego. También se detectaron observaciones incompletas, al faltar información como la unidad activa o enemigos cercanos, lo que llevó a añadir canales específicos para unidades, obstáculos y máscaras de acción. Además, el uso inicial de redes MLP mostró limitaciones para captar la estructura espacial del tablero, siendo reemplazadas con éxito por redes convolucionales (CNN). Las penalizaciones excesivas fueron otro obstáculo, ya que generaban agentes pasivos; por ello, se ajustaron para fomentar una exploración más equilibrada. Finalmente, algunos agentes quedaban estancados en estrategias poco útiles, un problema mitigado gracias al uso de *transfer learning* y al diseño progresivo por fases.

## 9.2. Líneas de Mejora

A raíz de los problemas detectados, se plantean varias posibles mejoras para futuras versiones del entorno. En primer lugar, se podría añadir visión parcial para obligar a los agentes a planificar bajo incertidumbre. También sería interesante entrenar políticas específicas por tipo de unidad, con el fin de mejorar la especialización táctica. Otra línea prometedora sería explorar arquitecturas con mecanismos de atención o memoria (como LSTM), que permitan tomar decisiones basadas en el historial reciente. Finalmente, se propone desarrollar una IA heurística más compleja, inspirada en juegos como Advance Wars o Fire Emblem, para utilizarla como referencia comparativa frente a los agentes entrenados mediante aprendizaje por refuerzo.

# 10 Conclusiones

Este trabajo ha permitido diseñar e implementar un entorno de aprendizaje por refuerzo adaptado a juegos de estrategia por turnos, con mecánicas inspiradas en *Advance Wars*. A lo largo del proyecto se han entrenado y enfrentado agentes basados en diferentes algoritmos (PPO, A2C, DQN y Maskable PPO), permitiendo evaluar su rendimiento en condiciones simétricas.

Los resultados muestran que algoritmos como `Maskable PPO` y `DQN` destacan por su capacidad para tomar decisiones eficaces en escenarios complejos, gracias a la enmascaración de acciones inválidas y a estructuras de observación espaciales bien diseñadas. En cambio, algoritmos como PPO y A2C tienden a adoptar estrategias más conservadoras con mayores tasas de empate.

Entre las dificultades encontradas destacan el diseño de la función de recompensa, el impacto de observaciones incompletas y la necesidad de evitar redes MLP en entornos espaciales. También se comprobó que un mal ajuste de penalizaciones puede llevar a comportamientos pasivos.

En resumen, el entorno desarrollado ha servido como plataforma sólida para comparar estrategias de RL en juegos por turnos, sentando las bases para futuras investigaciones en IA táctica y entrenamiento competitivo entre agentes.

## AGRADECIMIENTOS

Quiero agradecer a mi tutor, Jordi Casas Roma, por su guía y apoyo durante todo el proyecto. También a mi familia, amigos y pareja, por su ánimo constante.

Agradezco al profesorado y compañeros del Grado en Ingeniería Informática por acompañarme en este camino académico.

Finalmente, gracias a la comunidad de desarrollo libre, especialmente a los creadores de `Gymnasium`, `Stable-Baselines3` y `Pygame`, por las herramientas que han hecho posible este trabajo.

Asimismo, se agradece al autor *Zerie* por el pack de sprites *Tiny RPG Character Asset Pack*, utilizado para la representación visual de las unidades en el juego [19].

## REFERENCIAS

[1] Mnih, V., Kavukcuoglu, K., Silver, D., et al., "Human-level control through deep reinforcement learning." *Nature*, vol. 518, pp. 529–533, 2015. Disponible en: nature.com. [Accedido: 4 de marzo de 2025].

[2] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., "Proximal Policy Optimization Algorithms." *arXiv preprint arXiv:1707.06347*, 2017. Disponible en: arXiv. [Accedido: 4 de marzo de 2025].

[3] Mnih, V., Badia, A. P., Mirza, M., et al., "Asynchronous methods for deep reinforcement learning." *arXiv preprint arXiv:1602.01783*, 2016. Disponible en: arXiv. [Accedido: 4 de marzo de 2025].

[4] Vinyals, O., Babuschkin, I., Czarnecki, W. M., et al., "Grandmaster level in StarCraft II using multi-agent reinforcement learning." *Nature*, vol. 575, pp. 350–354, 2019. Disponible en: nature.com. [Accedido: 4 de marzo de 2025].

[5] Berner, C., Brockman, G., Chan, B., et al., "Dota 2 with Large Scale Deep Reinforcement Learning." *arXiv preprint arXiv:1912.06680*, 2019. Disponible en: arXiv. [Accedido: 4 de marzo de 2025].

[6] Silver, D., Huang, A., Maddison, C. J., et al., "Mastering the game of Go with deep neural networks and tree search." *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. Disponible en: nature.com. [Accedido: 4 de marzo de 2025].

[7] Farama Foundation, "Gymnasium Documentation." Disponible en: gymnasium.farama.org. [Accedido: 4 de marzo de 2025].

[8] Pygame Community, "Pygame - Python Game Development." Disponible en: pygame.org. [Accedido: 12 de abril de 2025].

[9] Raffin, A., Hill, A., Ernestus, M., et al., "Stable-Baselines3: Reliable implementations of reinforcement learning algorithms in PyTorch." *GitHub repository*, 2021. Disponible en: github.com. [Accedido: 4 de marzo de 2025].

[10] Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J., "Quantifying Generalization in Reinforcement Learning." *arXiv preprint arXiv:1812.02341*, 2019. Disponible en: arXiv. [Accedido: 4 de marzo de 2025].

[11] Bellemare, M. G., Srinivasan, S., Ostrovski, G., et al., "Unifying Count-Based Exploration and Intrinsic Motivation." *NeurIPS*, vol. 29, pp. 1471–1479, 2016. Disponible en: arXiv. [Accedido: 4 de marzo de 2025].

[12] Espeholt, L., Soyer, H., Munos, R., et al., "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures." *ICML*, PMLR 80:1407–1416, 2018. Disponible en: mlr.press. [Accedido: 12 de abril de 2025].

[13] Gulli, A., Pal, S., "Convolutional Neural Networks with Reinforcement Learning." *Packt Tutorial*, 2017. Disponible en: packtpub.com. [Accedido: 4 de marzo de 2025].

[14] Stable Baselines3 Contrib, "Maskable PPO – Implementation of invalid action masking for Proximal Policy Optimization." Documentación oficial, 2025. Disponible en: sb3-contrib.readthedocs.io. [Accedido: 15 de junio de 2025].

[15] Nintendo Co., Ltd., *Advance Wars*, Intelligent Systems, Game Boy Advance, 2001. Disponible en: nintendo.com. [Accedido: 25 de junio de 2025].

[16] Nintendo Co., Ltd., *Fire Emblem: The Blazing Blade*, Intelligent Systems, Game Boy Advance, 2003. Disponible en: nintendo.com. [Accedido: 25 de junio de 2025].

[17] Blizzard Entertainment, *StarCraft II*, 2010. Disponible en: https://starcraft2.com/. [Accedido: 25 de junio de 2025].

[18] Valve Corporation, *Dota 2*, 2013. Disponible en: https://www.dota2.com/. [Accedido: 25 de junio de 2025].

[19] Zerie, *Tiny RPG Character Asset Pack*, Itch.io, 2020. Disponible en: https://zerie.itch.io/tiny-rpg-character-asset-pack. [Accedido: 30 de junio de 2025].

# A Apéndices

## A.1. Anexo: Diagrama de Gantt

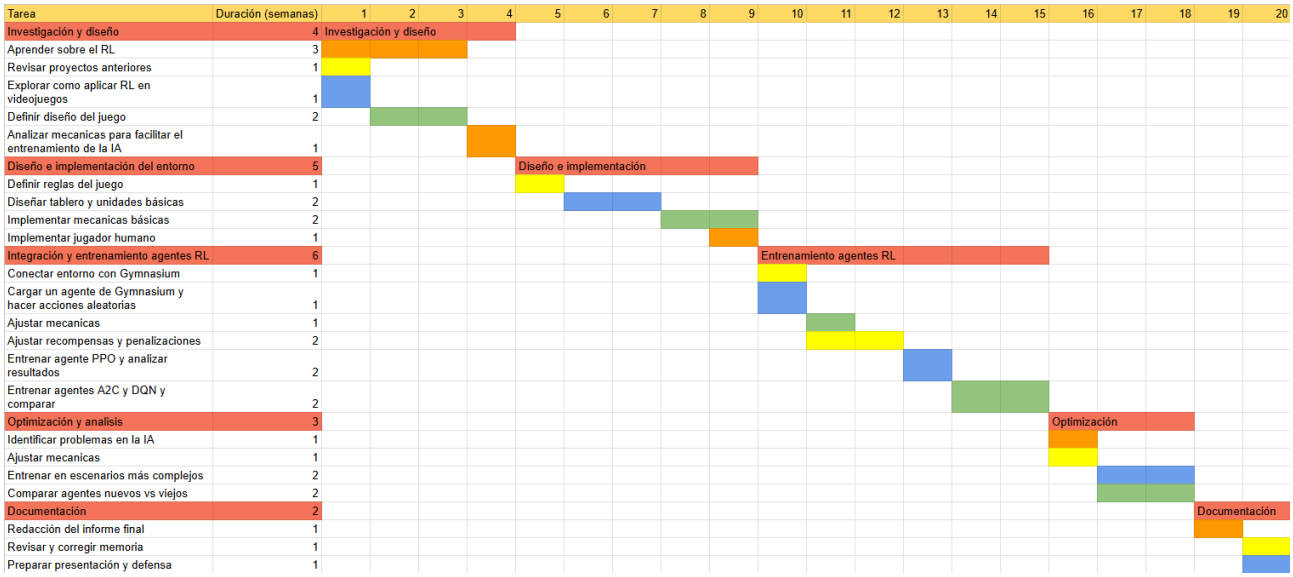


Fig. 3: Diagrama de Gantt detallado.

## A.2. Anexo: Resultados entre modelos (Fase 7)

TABLA 6: Resultados entre modelos entrenados en Fase 7 (100 partidas por cruce). Formato: V / D / E.

| Modelo azul | a2c_red | ppo_red | maskableppo_red | dqn_red |
|---|---|---|---|---|
| a2c_blue | 69V / 0D / 31E | 17V / 0D / 83E | 25V / 64D / 11E | 61V / 18D / 21E |
| ppo_blue | 0V / 2D / 98E | 1V / 8D / 91E | 1V / 66D / 39E | 4V / 36D / 60E |
| maskableppo_blue | 97V / 0D / 3E | 54V / 0D / 46E | 70V / 30D / 0E | 86V / 10D / 4E |
| dqn_blue | 97V / 0D / 3E | 84V / 0D / 16E | 71V / 27D / 2E | 66V / 24D / 10E |

## A.3. Anexo: Captura del juego

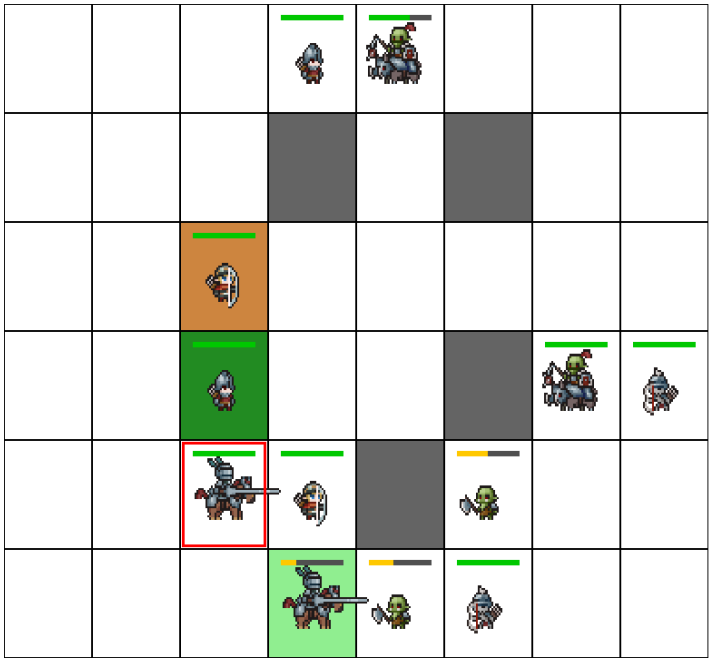

Fig. 4: Captura de una partida entre agentes entrenados.

## A.4. Curvas de entrenamiento completas

### A.4.1. PPO (azul y rojo, entrenados en la fase 7)

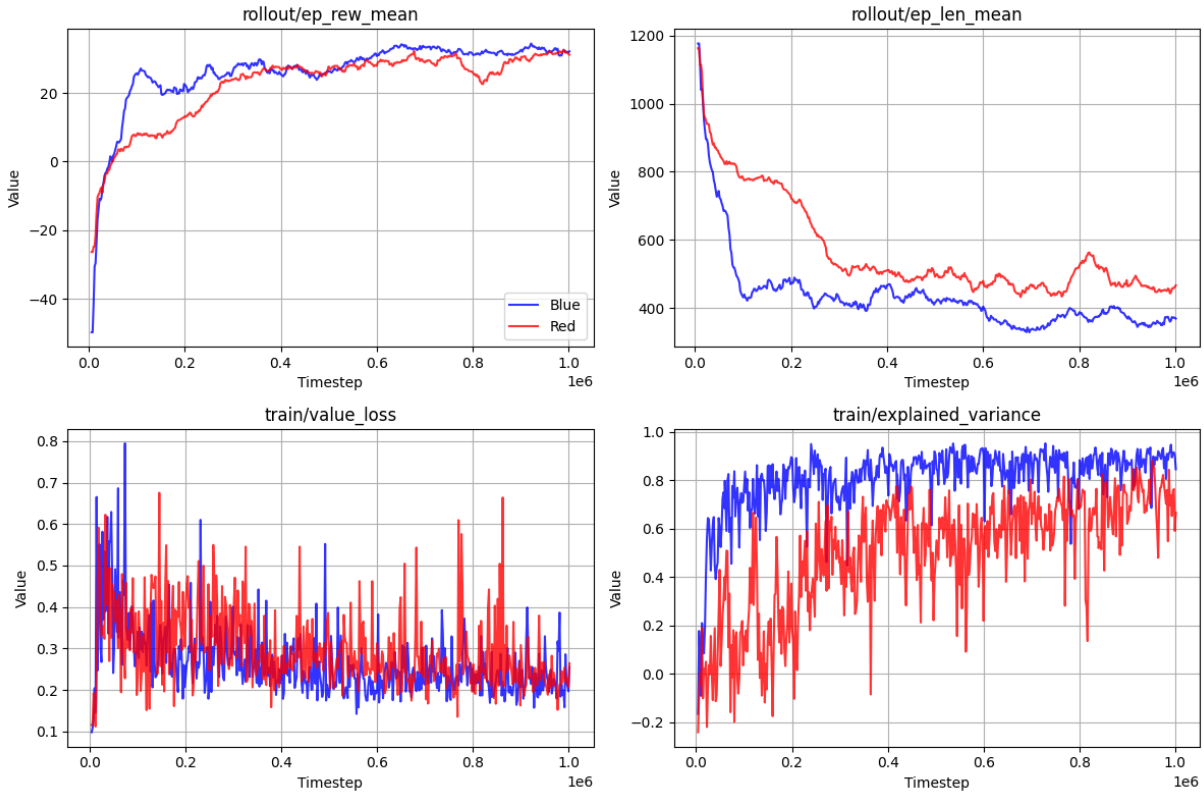


Fig. 5: Evolución de las métricas de entrenamiento para PPO azul y rojo.

### A.4.2. Maskable PPO (azul y rojo, entrenados en la fase 7)

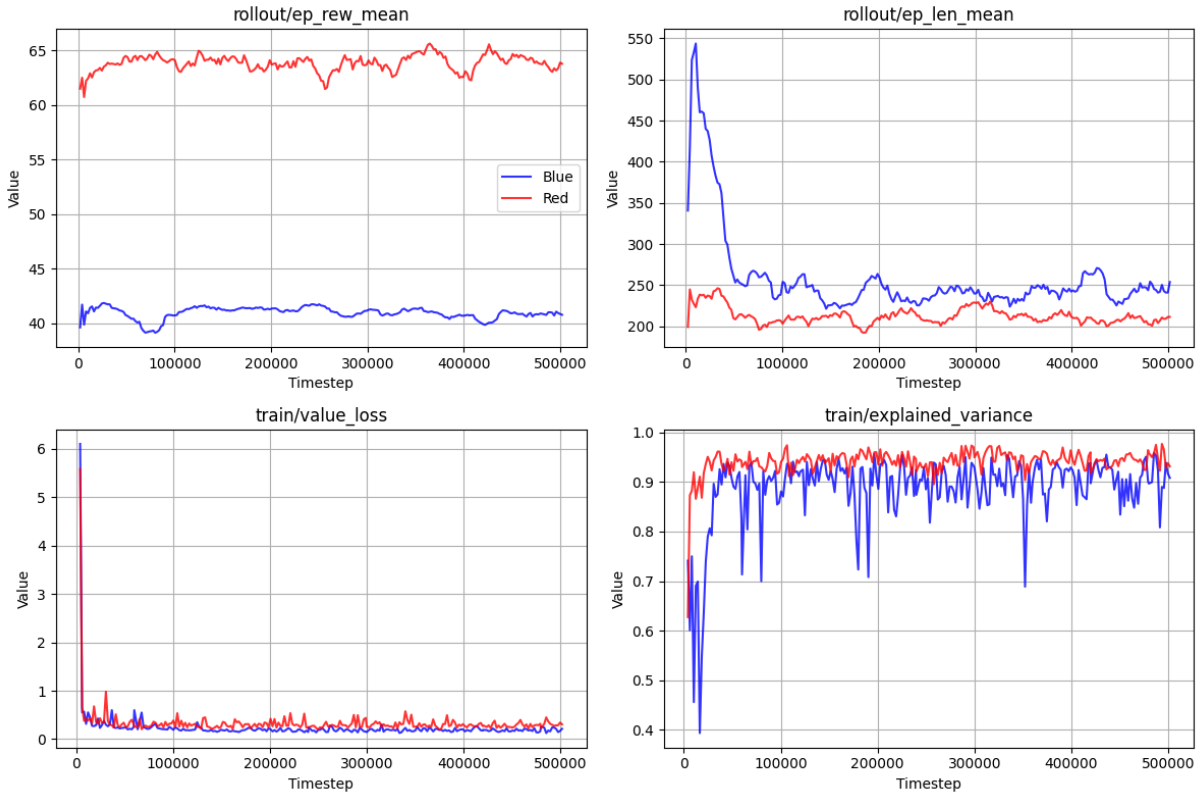


Fig. 6: Evolución de las métricas de entrenamiento para MaskbalePPO azul y rojo.

### A.4.3. A2C (azul y rojo, entrenados en la fase 7)

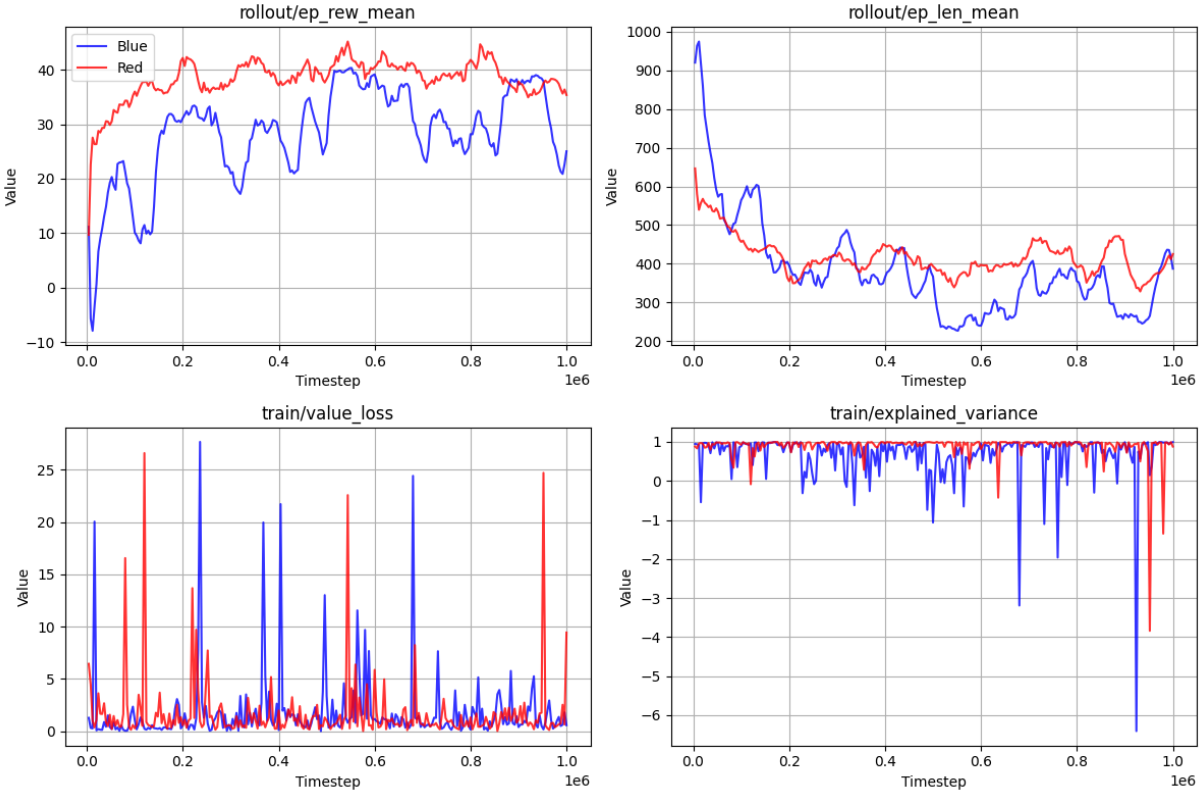


Fig. 7: Evolución de las métricas de entrenamiento para A2C azul y rojo.

### A.4.4. DQN (azul y rojo, entrenados en la fase 7)

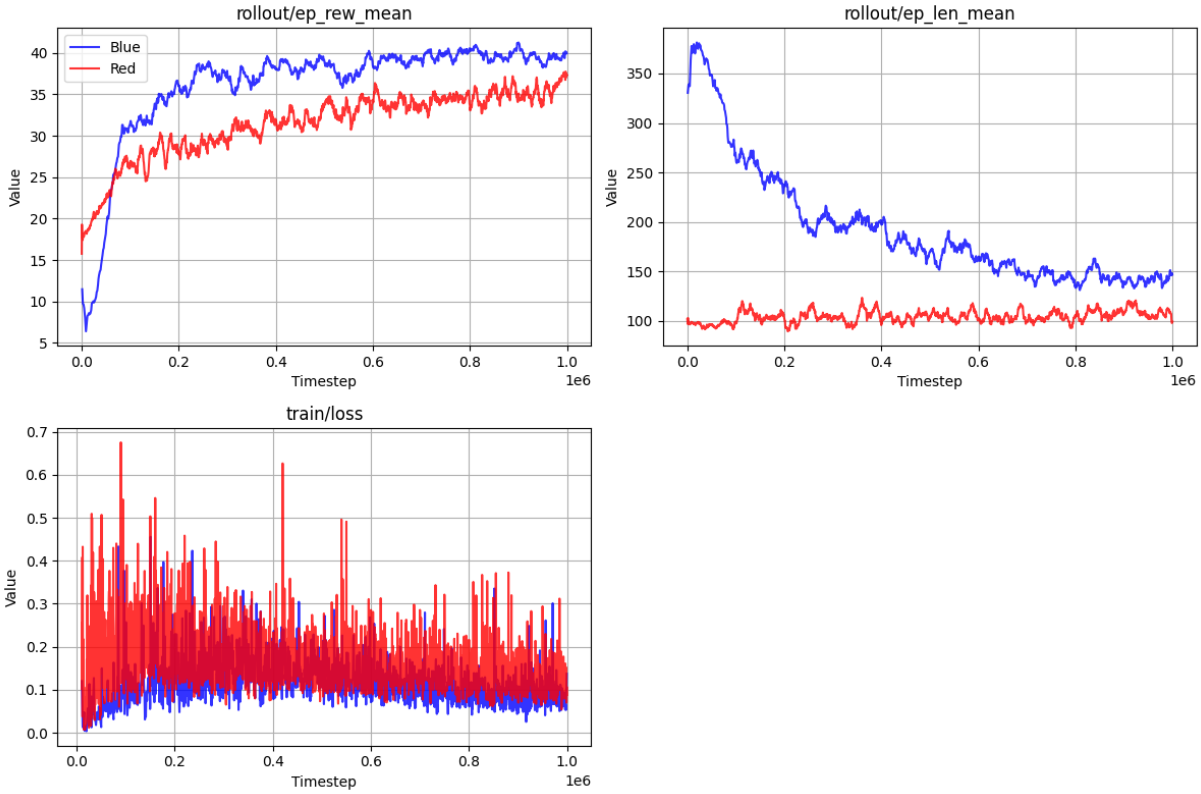


Fig. 8: Evolución de las métricas de entrenamiento para DQN azul y rojo.