Resumen

Este proyecto tiene como objetivo crear una IA capaz de ganar un partido 1vs1 en una cancha interactiva. La IA se entrenará mediante aprendizaje por refuerzo [2], específicamente con el algoritmo Q-Learning [1], que permitirá al agente aprender decisiones óptimas basadas en recompensas obtenidas a través de la interacción con el entorno. Inicialmente, se exploró el uso de Deep Q-Learning (DQN) [3] para manejar espacios de estados más complejos, pero tras simplificar el entorno, se optó por una versión más eficiente basada en Q-Learning clásico. Además, se documentará el progreso del entrenamiento y se analizarán los resultados obtenidos.

Palabras clave: Inteligencia Artificial, Videojuegos, Aprendizaje por Refuerzo, Q-Learning, Deep Q-Learning.

Abstract

This project aims to develop an AI capable of winning a 1vs1 match in an interactive field. The AI will be trained using reinforcement learning [2], specifically with the Q-Learning algorithm [1], which will enable the agent to learn optimal decisions based on rewards obtained through interaction with the environment. Initially, the use of Deep Q-Learning (DQN) [3] was explored to handle more complex state spaces, but after simplifying the environment, a more efficient version based on classical Q-Learning was chosen. Additionally, the training progress will be documented, and the results will be analyzed as the complexity of the environment increases.

Keywords: Artificial Intelligence, Video Games, Reinforcement Learning, Q-Learning, Deep Q-Learning.

Índice de contenidos

1. Introducción

En este proyecto, se desarrollará y entrenará una Inteligencia Artificial para competir en un entorno interactivo similar a una cancha de fútbol. El objetivo principal será diseñar una IA capaz de ganar partidos 1vs1 mediante el uso de técnicas de Aprendizaje por Refuerzo, con un enfoque inicial en el algoritmo Deep Q-Learning (DQN), y una posterior implementación basada en Q-Learning clásico, más adecuada tras la simplificación del espacio de estados.

Para ello, se emplearán las bibliotecas Python, PyTorch y Pygame. Pygame [5] se utilizará para construir una cancha de fútbol con paredes y representar visualmente el entorno, mientras que PyTorch [6] se empleará en la implementación de la versión con DQN, permitiendo entrenar redes neuronales profundas para aproximar la función de valor en entornos más complejos.

El proceso de desarrollo incluirá la creación del entorno, el diseño y entrenamiento del agente, y la documentación y análisis de su progreso. A medida que la IA avance en su aprendizaje, se implementarán nuevas recompensas, fomentando así un aprendizaje más robusto. Por último, los resultados obtenidos en cada fase serán evaluados para reflejar la evolución y efectividad del comportamiento del agente y serán comparados con los resultados de su TFG análogo.

1.1 Motivación

La inteligencia artificial siempre ha sido un área de la informática que me ha fascinado y desde que empecé la rama de computación, donde tuve la oportunidad de aprender más gracias a asignaturas como “Percepción” y “Técnicas, entornos y aplicaciones de Inteligencia Artificial”, mi interés por este campo ha aumentado con creces. Estos cursos me han proporcionado una base sólida en los principios y técnicas de la inteligencia artificial, motivándome a profundizar en este ámbito y a elegir un proyecto relacionado para mi Trabajo de Fin de Grado.

Además, siempre me han interesado los videos en internet sobre el entrenamiento de IAs para jugar a videojuegos. Ver cómo las IAs aprenden y mejoran en diversos entornos de juego ha despertado en mí una curiosidad y un deseo de entender y replicar esos procesos. La combinación de mis estudios y esta curiosidad personal me ha llevado a escoger este proyecto, con el objetivo de crear y entrenar una IA en un entorno interactivo similar a un videojuego.

Este proyecto no solo representa una oportunidad para aplicar mis conocimientos y habilidades adquiridas durante la carrera, sino también para explorar y contribuir al campo de la inteligencia artificial en videojuegos, un área que siempre me ha apasionado.

1.2 Objetivos

El propósito de este TFG será entrenar un par de IAs capaces de simular un partido de fútbol en un terreno de pruebas y analizar el proceso y resultados obtenidos. Para ello, se han establecido una serie de objetivos que servirán como puntos de inflexión para pasar al siguiente nivel del entrenamiento:

- **Aprender a moverse hacia la pelota**

  Primero, se realizará un entrenamiento en un entorno plano con solo la pelota y un agente, forzando así a que se aprendan los controles básicos del juego, es decir aprender a moverse hacia la pelota independientemente de la posición en la que se encuentre

- **Marcar gol en un entorno simple:**

  Una vez el agente aprenda a moverse hacia la pelota, se añadirán al entorno de juego las porterías, junto con una serie de nuevas recompensas encargadas de incentivar al agente a que una vez cerca de la pelota, la empuje hacía la portería enemiga y acabe anotando gol.

- **Simular un partido:**

  Por último, simularemos un partido con ambos agentes para poner a prueba todo lo que han aprendido, pudiendo así modificar el valor de alguna recompensa o añadir o quitar algunas dependiendo de los resultados.

Una vez tengamos un resultado satisfactorio, analizaremos el proceso de entrenamiento y las gráficas con los resultados que hemos ido obteniendo.

1.3 Metodología

La metodología de trabajo aplicada en este proyecto se basa en un enfoque iterativo e incremental [9], ampliamente utilizado en el desarrollo de software y en la investigación de inteligencia artificial. Este enfoque permite una progresión continua y estructurada, asegurando que cada etapa del proyecto se desarrolle de manera eficiente y efectiva.

El enfoque iterativo e incremental implica ciclos repetitivos de desarrollo y evaluación, donde cada iteración añade funcionalidad incremental al sistema. En cada ciclo, se planifican, implementan, prueban y revisan mejoras, permitiendo refinar y optimizar el producto basado en los resultados obtenidos en cada iteración. Esta metodología se adapta bastante bien al proyecto, puesto que los resultados del entrenamiento de una IA pueden ser impredecibles y requieren ajustes frecuentes para maximizar el rendimiento del agente.

Esta metodología comienza con la definición de los objetivos y la recopilación de requisitos, estableciendo una base clara y detallada para el proyecto. A continuación, se procede a la fase de diseño, donde se desarrollan las especificaciones técnicas y arquitectónicas del sistema. La implementación se lleva a cabo de manera incremental, añadiendo nuevas funcionalidades y mejoras en cada iteración. Este enfoque asegura que el sistema evolucione de forma controlada y eficiente, permitiendo realizar ajustes basados en el feedback recibido durante las pruebas.

La primera etapa se centró en establecer las bases técnicas del proyecto. Durante esta fase inicial, se desarrolló el entorno gráfico en 2D utilizando Pygame, lo que permitió crear un espacio interactivo donde el agente pudiera ejecutar y probar sus estrategias.

Paralelamente, se implementó la gestión de la Q-Table [7], sentando las bases para el entrenamiento del agente. Esta etapa fue una de las más intensivas en cuanto a creación de código, ya que implicó diseñar los componentes fundamentales que soportarían las iteraciones posteriores del proyecto.

A su vez, en las primeras etapas del entrenamiento, se probaron a implementar mediante el uso de Pytorch, otros algoritmos avanzados como Deep Q-Learning que, si bien se lograron implementar correctamente, terminaron siendo descartados en etapas posteriores debido a mejoras realizadas en la percepción del estado del agente, lo que nos permitió simplificar la complejidad del proyecto y no tener que recurrir a algoritmos más complejos como los anteriormente mencionados.

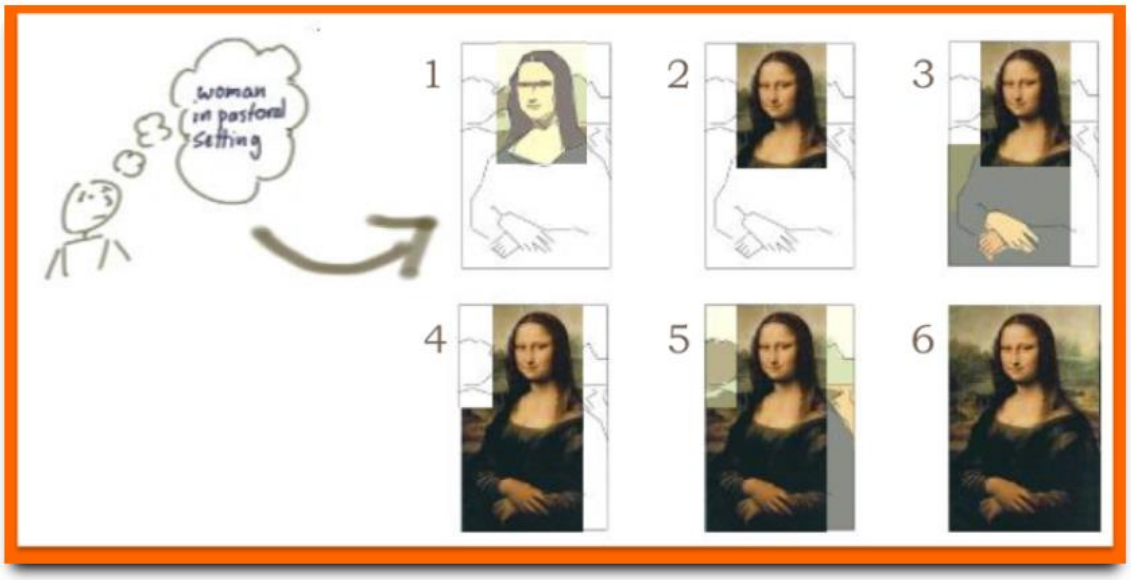

Figura 1.1: Enfoque iterativo e incremental

1.4 Colaboraciones

El entorno Pygame original fue realizado junto a Álvaro Órdoño Saiz, el cual publicó su respectivo TFG análogo centrado en el aprendizaje mediante algoritmos genéticos.

1.5 Estructura de la memoria

La memoria se dividirá en 12 capítulos, en cada uno de estos capítulos nos centraremos y desarrollaremos un apartado distinto del proyecto:

- **El primer capítulo** será la **introducción** del proyecto, donde se expondrán tanto los motivos por los cuales se ha realizado, como los objetivos y metodología de trabajo empleada.

- **El segundo capítulo**, se centrará en los **fundamentos teóricos**, en este capítulo, introduciremos y explicaremos distintos conceptos relacionados con el proyecto y que ayudarán a la hora de comprender su funcionamiento a lo largo de la memoria.

- **El tercer capítulo**, tratará sobre la **historia del arte**, de la Inteligencia artificial, las redes neuronales y los distintos tipos de entrenamiento actuales, mostrando así su desarrollo en los últimos años y los logros obtenidos.

- **El cuarto capítulo**, explicará el **análisis del problema**, mostrando así las dificultades a las que nos hemos enfrentado, como las hemos solucionado, los requisitos que se han tenido en cuenta a lo largo del trabajo y la metodología empleada.

- **El quinto capítulo**, mostrará el **diseño de la solución**, mediante el diseño arquitectónico del sistema y de sus distintos componentes. A su vez, explicaremos las herramientas utilizadas para llevar a cabo el proyecto

- **El sexto capítulo**, implicará el **desarrollo de la solución**, recorriendo así los distintos pasos que se han realizado a lo largo del proyecto para implementar los distintos componentes y superar las distintas dificultades a las que nos hemos enfrentado.

- **El séptimo capítulo**, sintetizará las distintas **pruebas** que hemos realizado para valorar el rendimiento de nuestras IAs y si han logrado los objetivos propuestos en el primer capítulo.

- **El octavo capítulo,** se centrará en las conclusiones a las que hemos llegado una vez realizado el proyecto

- Por último, los **capítulos noveno y décimo** tratarán de la **relación del proyecto con los estudios** y los **posibles trabajos a futuro** de este, explicando así que posibles añadidos se podrían agregar en el futuro.

2. Fundamentos teóricos

En este capítulo, presentaremos los fundamentos teóricos esenciales para comprender el desarrollo y funcionamiento del proyecto. La inteligencia artificial (IA) y el aprendizaje por refuerzo (RL) son áreas amplias y complejas, cuya comprensión es crucial para seguir el proceso de diseño, implementación y evaluación del agente entrenado.

La IA ha transformado numerosos campos, permitiendo que las máquinas realicen tareas que requieren inteligencia humana. Este proyecto se sitúa en la intersección de la IA y los videojuegos, un área que ha ganado atención en los últimos años por su capacidad para simular entornos complejos y dinámicos.

Para lograr que una IA aprenda a jugar y ganar en un entorno interactivo, es necesario aplicar técnicas avanzadas de aprendizaje automático, en este caso, aprendizaje por refuerzo. Este enfoque se centra en cómo un agente debe tomar decisiones en un entorno para maximizar una recompensa acumulada.

En las primeras etapas, se optó por utilizar el algoritmo Deep Q-Learning (DQN) [3], una extensión moderna y potente del Q-Learning tradicional [1], que emplea redes neuronales profundas para manejar entornos con grandes espacios de estados. Este enfoque fue implementado con la ayuda de bibliotecas como PyTorch [6], que facilita la construcción y entrenamiento de redes neuronales, y Pygame [5], que proporciona herramientas para el desarrollo de videojuegos y entornos simulados.

Sin embargo, a medida que avanzaba el desarrollo, se logró simplificar considerablemente el espacio de estados mediante una discretización más agresiva de las posiciones del agente y la pelota. Esta simplificación permitió prescindir del uso de redes neuronales, ya que el espacio resultante era lo suficientemente reducido como para ser manejado eficazmente con Q-Learning clásico. De este modo, el enfoque final del proyecto se basa en Q-Learning, debido a su menor complejidad y mayor eficiencia en este contexto.

A lo largo de este capítulo, introduciremos y explicaremos los conceptos clave necesarios para entender este proyecto. Esta base teórica permitirá comprender los principios en el diseño y desarrollo de los agentes, a la par que proporcionar una comprensión clara y completa del marco conceptual en el que se basa este proyecto, preparando así al lector para los capítulos posteriores que detallarán la implementación y los resultados del entrenamiento.

2.1 Inteligencia Artificial (IA)

La inteligencia artificial (IA) es la capacidad de las máquinas para realizar tareas que normalmente requerirían inteligencia humana [11]. Estas tareas pueden incluir el reconocimiento de patrones, la toma de decisiones, la resolución de problemas, el procesamiento del lenguaje natural y el aprendizaje de la experiencia. En esencia, la IA busca crear sistemas que puedan percibir su entorno, razonar sobre la información recibida y actuar de manera autónoma para alcanzar objetivos específicos.

La IA se puede clasificar en 3 categorías, dependiendo de su capacidad y complejidad:

- **IA débil (Narrow AI):**

  Sistemas sin conciencia ni inteligencia general que están diseñados para realizar tareas específicas. Tanto nuestro proyecto como otros agentes como Siri y Alexa se basan en este modelo de IA.

- **IA Fuerte (General AI):**

  Sistemas con capacidades cognitivas similares al ser humano. Hoy en día, esta IA solo es una meta teórica y no se ha logrado desarrollar completamente, pero sería capaz de realizar cualquier tarea intelectual que un ser humano pudiera realizar.

- **Superinteligencia Artificial (Superintelligent AI):**

  Sistemas que superan con creces a la inteligencia humana en todos los aspectos, desde la creatividad hasta la resolución de problemas y la toma de decisiones. Al igual que la IA fuerte, esta forma de IA es solo teórica actualmente y plantea importantes implicaciones éticas y de seguridad.

2.2 Aprendizaje Automático (Machine Learning)

El aprendizaje automático, o machine learning, es una subdisciplina de la inteligencia artificial que se centra en el desarrollo de algoritmos y técnicas que permiten a las máquinas aprender de los datos. En lugar de ser programadas para realizar una tarea específica, las máquinas utilizan datos para identificar patrones, hacer predicciones y tomar decisiones basadas en la información aprendida.

El aprendizaje automático se puede clasificar en tres categorías principales [10]:

- **Aprendizaje Supervisado (Supervised Learning):**

  En el aprendizaje supervisado, el modelo se entrena con un conjunto de datos etiquetados, donde cada ejemplo incluye una entrada y una salida deseada. El objetivo es aprender una función que mapee las entradas a las salidas correctas, de manera que, una vez entrenado, el modelo pueda predecir las salidas para nuevas entradas.

  Ejemplos: Clasificación (identificar correos spam), regresión (predecir precios de casas).

- **Aprendizaje No Supervisado (Unsupervised Learning):**

En el aprendizaje no supervisado, el modelo se entrena con datos que no tienen etiquetas, con el objetivo de encontrar patrones o estructuras ocultas en los datos.

Ejemplos: Agrupación de clientes en segmentos de mercado, detección de anomalías.

- **Aprendizaje por Refuerzo (Reinforcement Learning):**

En el aprendizaje por refuerzo [8], un agente interactúa con un entorno y aprende a tomar decisiones que maximicen una recompensa acumulada a lo largo del tiempo que irá aumentado o decrementando debido a las recompensas o castigos que reciba el agente en base a sus acciones. Nuestro proyecto se basa en este modelo de aprendizaje.

Ejemplos: Robots que aprenden a caminar, agentes que juegan videojuegos.

2.3 Redes neuronales

Las redes neuronales [12] son un componente fundamental de la inteligencia artificial y el aprendizaje automático, inspiradas en la estructura y funcionamiento del cerebro humano. Estas redes están diseñadas para reconocer patrones y procesar datos de manera similar a como lo hacen las neuronas en el cerebro.

Una red neuronal está compuesta por varias capas de unidades de procesamiento llamadas neuronas o nodos. Las principales capas en una red neuronal son:

- **Capa de Entrada (Input Layer):**

La primera capa de la red, donde se reciben los datos de entrada. Cada nodo en esta capa representa una característica o atributo del conjunto de datos.

- **Capas Ocultas (Hidden Layers):**

Una o más capas situadas entre la capa de entrada y la capa de salida. Estas capas realizan la mayor parte del procesamiento a través de pesos y funciones de activación. Cada nodo en una capa oculta recibe entradas de los nodos de la capa anterior y pasa su salida a los nodos de la siguiente capa.

- **Capa de Salida (Output Layer):**

La última capa de la red, donde se generan las predicciones o resultados. Cada nodo en esta capa representa una posible salida o categoría.

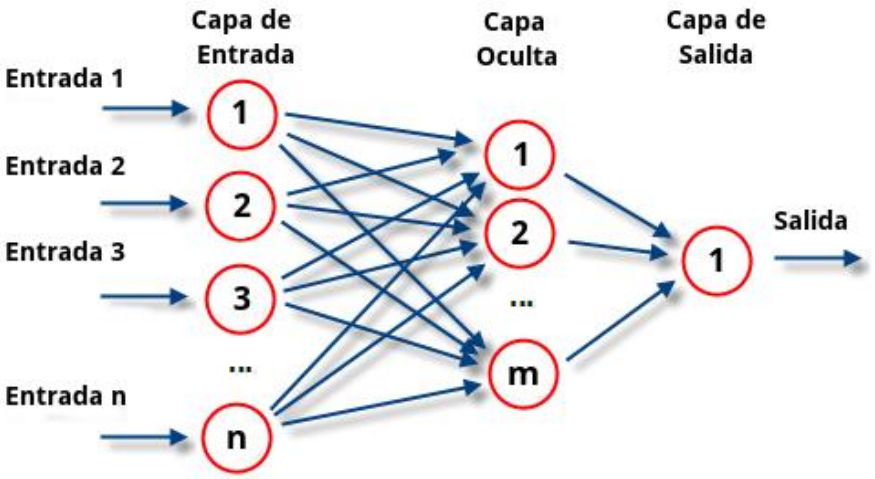

Figura 2.1: Capas de una red neuronal

2.4 Funcionamiento de una Red Neuronal

El funcionamiento de las redes neuronales se puede resumir principalmente en 4 etapas:

- **Propagación Hacia Adelante (Forward Propagation):**

  El proceso comienza con la capa de entrada, donde los datos se introducen a la red. Los datos pasan a través de las capas ocultas, donde se aplican pesos y funciones de activación para transformar las entradas en salidas. Finalmente, las salidas de la última capa oculta se pasan a la capa de salida, generando la predicción final.

- **Función de Activación:**

  Cada nodo aplica una fórmula matemática a su valor de salida lineal, transformando estos valores de manera no lineal. Esta transformación permite que la red pueda resolver problemas más complejos, donde una simple separación lineal de los datos no sería suficiente, como en el caso del reconocimiento de imágenes.

- **Retropropagación (Backpropagation):**

  Después de obtener la predicción en la fase de propagación hacia adelante, se calcula el error comparando la predicción con el valor real utilizando una función de pérdida, este error se propaga hacia atrás a través de la red, desde la capa de salida hasta la capa de entrada, ajustando los pesos en cada nodo.

  Finalmente, los pesos se actualizan utilizando un algoritmo de optimización, como el descenso de gradiente, para minimizar el error. Este proceso se repite iterativamente, afinando los pesos hasta que el error se reduzca a un nivel aceptable.

2.5 Q-Learning

Q-Learning es uno de los algoritmos más populares en el aprendizaje por refuerzo. En este tipo de aprendizaje, un agente aprende a tomar decisiones óptimas para obtener la mayor recompensa posible en un entorno dado. La esencia de Q-Learning es que el agente explora diferentes acciones y, a través de la experiencia, determina cuáles son las más beneficiosas.

Este algoritmo consta de 4 componentes básicos para su debido funcionamiento:

- **Estados:**

  Diferentes situaciones o posiciones en las que el agente se puede encontrar. Ej Disposición de las piezas en un tablero de ajedrez

- **Acciones:**

  Todas las acciones que el agente puede realizar. Ej Mover las piezas de ajedrez en el tablero

- **Recompensa:**

  Puntos que recibe el agente al realizar una acción. Estas recompensan indicarán si una acción fue beneficiosa o perjudicial. Ej Una recompensa positiva se daría al realizar un movimiento y eliminar una pieza enemiga, mientras que una negativa se daría al ser eliminada una pieza del agente.

- **Q-Table:**

  Tabla en la que el agente va rellenando con el paso del tiempo. Esta tabla indicará que tan buena es una acción en un estado específico, permitiendo al agente escoger la mejor acción en cada estado.

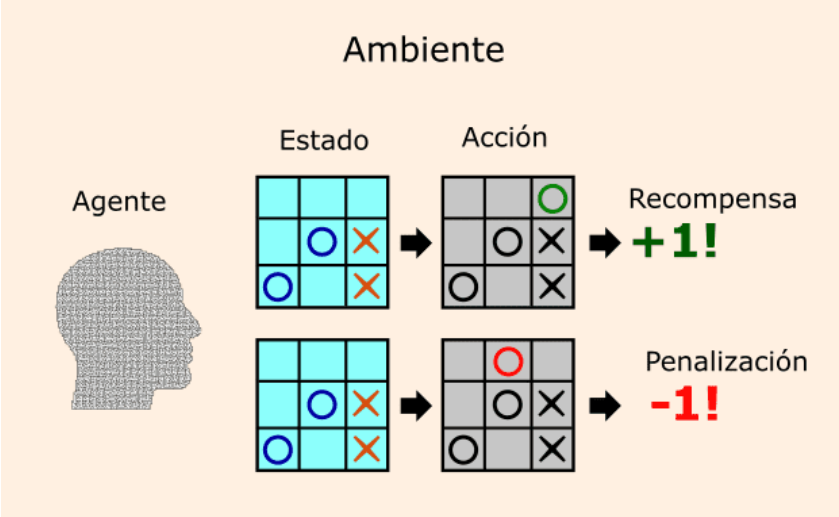

Figura 2.2: Ejemplo aprendizaje por refuerzo

En cuanto al funcionamiento del algoritmo, lo podemos dividir en 4:

- **Inicialización:**

  El proceso comienza con una q-table completamente vacía.

- **Exploración del entorno:**

  El agente observa el entorno y elige la mejor acción posible, al principio como la q-table está vacía, estas acciones serán elegidas al azar.

- **Ejecución de una acción:**

  Una vez realizada la acción escogida, el agente observará tanto la recompensa obtenida como el nuevo estado.

- **Actualización de la Q-Table:**

  Rellena una entrada de la q-table con los valores obtenidos al realizar la acción y determina la calificación de esa entrada.

- **Repetición:**

  Una vez actualizada la tabla, se repite el proceso desde el punto de exploración hasta que la q-table refleje de manera precisa las mejores acciones para cada estado.

2.6 Deep Q-Learning (DQN)

Deep Q-Learning es una extensión del algoritmo Q-Learning que utiliza redes neuronales profundas para manejar entornos con estados de alta dimensión. Esto es especialmente útil cuando los estados no pueden ser representados de manera eficiente con una tabla Q tradicional, como en el caso de imágenes, video o datos complejos.

La principal diferencia con Q-Learning es que, en lugar de almacenar los valores de cada estado en una tabla, se utiliza una red neuronal para determinar el valor de cada acción. Esto es especialmente útil en problemas complejos, donde el tamaño de la tabla Q puede escalar a proporciones inmanejables.

El proceso, similar al de Q-learning, se puede dividir en 4 etapas:

- **Inicialización**:

  Se inicializa la red con pesos aleatorios y se crea una memoria de repetición donde almacenará las futuras experiencias del agente.

- **Interacción con el Entorno**:

  El agente observa el estado actual, selecciona y ejecuta una acción, luego observa la recompensa y el nuevo estado obtenidos, y finalmente almacena la experiencia en la memoria.

- **Entrenamiento de la Red**:

  Mediante un pequeño lote de las experiencias, calculará los valores objetivos de cada una y ajustará los pesos de la red neuronal para minimizar la diferencia entre los valores calculados y los objetivos/finales.

- **Actualización de la Red de Objetivo**:

  Periódicamente, copia los pesos de la red principal a la red objetivo, que proporcionará unos valores más precisos que la red principal.

3. Estado del arte

En este capítulo vamos a tratar los orígenes de la inteligencia artificial, los avances en los últimos años y por último los mayores logros obtenidos en esta área.

Además, documentaremos otras aplicaciones existentes en el mercado con funcionalidades similares a las propuestas en el TFG y justificaremos las decisiones metodológicas adoptadas.

3.1 Creación e inicios de la IA.

La inteligencia artificial ha sido un concepto teórico desde mucho antes de que se convirtiera en una realidad tecnológica. A lo largo de la historia, la idea de crear máquinas inteligentes ha capturado la imaginación de escritores, filósofos y científicos. Este concepto ha aparecido en numerosas novelas y medios de ciencia ficción, como en las obras de Isaac Asimov con sus famosas leyes de la robótica, o en películas icónicas como "2001: Una odisea del espacio" y "Blade Runner". Estas representaciones culturales han contribuido a formar la visión y expectativas de lo que podría ser la IA.

El surgimiento real de la IA como campo de estudio se remonta a mediados del siglo XX. El término "inteligencia artificial" fue acuñado por primera vez en 1956 por John McCarthy, durante una conferencia en el campus de Dartmouth [13], que es considerada el punto de partida oficial del campo de la IA. En esta conferencia, un grupo de investigadores se reunió para discutir la posibilidad de crear máquinas que pudieran "pensar" de manera similar a los humanos.

Los primeros pasos a la hora de desarrollar los algoritmos y programas de IA se dieron durante las décadas de 1950 y 1960. Estos primeros esfuerzos se centraron en la resolución de problemas y la lógica simbólica. Por ejemplo, en 1956, Allen Newell y Herbert A. Simon desarrollaron el programa "Logic Theorist" [14], considerado uno de los primeros programas de IA, capaz incluso de demostrar teoremas de forma más precisa que varios matemáticos.

Otro hito importante en los inicios de la IA fue el desarrollo del programa "General Problem Solver" (GPS) por Newell y Simon en 1957 [14]. GPS fue diseñado para simular el proceso de pensamiento humano y resolver una amplia gama de problemas utilizando una estrategia de búsqueda heurística.

Figura 3.1: Allen Newell

Figura 3.2: Herbert Simon

Durante los años 60 y 70, la investigación en IA se diversificó en varias subdisciplinas, incluyendo la robótica, el procesamiento del lenguaje natural y la visión por computadora. A pesar de estos avances, el progreso fue más lento de lo esperado debido a las limitaciones tecnológicas de la época y a la complejidad de crear sistemas verdaderamente inteligentes, llegando así a un periodo de desilusión conocido como "inviernos de la IA" [15], donde la financiación y el interés en la investigación de IA disminuyeron significativamente.

A pesar de estos desafíos, los investigadores continuaron trabajando en la IA, y los avances en hardware, algoritmos y teorías matemáticas eventualmente llevaron a resurgimientos en el campo, especialmente a partir de los años 90 con la llegada del aprendizaje automático y, más recientemente, el aprendizaje profundo.

3.2 Avances en los últimos años

En los últimos años, la inteligencia artificial ha experimentado grandes avances que han transformado tanto la parte teórica como la práctica de este campo. Estos avances se deben en gran medida a mejoras en el hardware, el desarrollo de algoritmos más sofisticados, y la disponibilidad de grandes volúmenes de datos.

Uno de los hitos más importantes ha sido el desarrollo y la implementación de redes neuronales profundas, también conocidas como deep learning. Estas redes, que consisten en múltiples capas de nodos interconectados, han demostrado ser increíblemente eficaces para tareas de reconocimiento de imágenes, procesamiento de lenguaje natural y juegos de estrategia complejos. Modelos como los CNN (Convolutional Neural Networks) y los RNN (Recurrent Neural Networks) han superado las barreras de rendimiento anteriores [12], permitiendo así aplicaciones en áreas como la visión por computadora y la traducción automática.

Otro avance notable ha sido el crecimiento del aprendizaje por refuerzo, especialmente con la introducción de algoritmos como el Deep Q-Learning [3]. Estos métodos han permitido a los agentes aprender comportamientos complejos en entornos dinámicos a través de la interacción y la retroalimentación basada en recompensas. Este enfoque ha sido fundamental en el desarrollo de IA que puede jugar y ganar videojuegos, tales como Dota 2 y StarCraft II, superando incluso a jugadores profesionales.

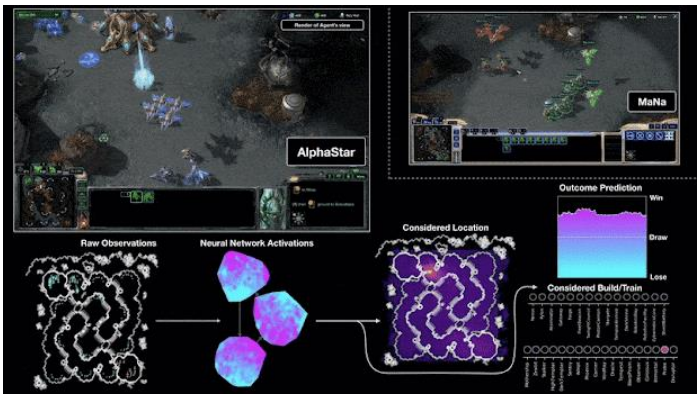

Figura 3.3 IA de Google (AlphaStar) jugando a Dota 2

El desarrollo de IA explicable y ética también ha crecido, con investigadores y empresas invirtiendo en la creación de modelos que no solo son potentes sino también transparentes y justos. La capacidad de entender y justificar las decisiones de una IA es fundamental para su aceptación y uso en áreas críticas como la medicina y el sistema judicial.

A su vez, en el ámbito del hardware, el desarrollo de unidades de procesamiento gráfico (GPUs) y unidades de procesamiento tensorial (TPUs) ha permitido la aceleración de cálculos necesarios para el entrenamiento de modelos de IA. Esto ha reducido significativamente el tiempo y el costo asociados con el desarrollo de modelos avanzados.

Además, la disponibilidad de grandes conjuntos de datos etiquetados, tales como ImageNet [16], ha facilitado el entrenamiento de modelos de IA con una precisión sin precedentes, proporcionando así a los investigadores los medios necesarios para crear modelos capaces de realizar tareas más complejas.

3.3 Mayores logros

En los últimos años, la inteligencia artificial ha alcanzado logros notables que han capturado la atención tanto de la comunidad científica como del público en general. Entre estos logros destacan el desarrollo de sistemas que han superado a los humanos en juegos complejos, avances en la medicina y el procesamiento del lenguaje natural, y la creación de modelos generativos avanzados.

A continuación, detallaremos algunos de los hitos más destacados:

- **AlphaGo de DeepMind**

  Uno de los logros más importantes en el campo de la IA fue el desarrollo de AlphaGo por DeepMind, una subsidiaria de Google [17]. En 2016, AlphaGo derrotó al campeón mundial de Go, Lee Sedol, en un partido. El juego de Go, conocido por su dificultad y gran cantidad de posibles movimientos, había sido considerado durante mucho tiempo un desafío inalcanzable para las máquinas. AlphaGo utilizó una combinación de redes neuronales profundas y aprendizaje por refuerzo para evaluar posiciones y seleccionar movimientos, demostrando la capacidad de la IA para abordar problemas extremadamente complejos.

- **IA en Diagnóstico Médico**

  Otro avance significativo ha sido el uso de IA en el diagnóstico médico. Modelos de aprendizaje profundo han sido entrenados para identificar enfermedades en imágenes médicas con una precisión comparable a la de médicos capacitados. Por ejemplo, la IA desarrollada por Google Health ha mostrado resultados impresionantes en la detección de retinopatía diabética a partir de imágenes de retina y en la identificación de cáncer de mama en mamografías. Estos avances están revolucionando la medicina, ofreciendo diagnósticos más rápidos y precisos y ayudando a los profesionales de la salud a tomar decisiones informadas.

- **GPT-3 y el Procesamiento del Lenguaje Natural**

El lanzamiento de GPT-3 (Generative Pre-trained Transformer 3) por OpenAI en 2020 representó un salto significativo en el procesamiento del lenguaje natural (NLP) [18]. GPT-3, con sus 175 mil millones de parámetros, ha demostrado una gran capacidad para generar texto coherente y realizar tareas de lenguaje natural con poca o ninguna supervisión específica. Este modelo ha sido utilizado en una gran variedad de aplicaciones, desde la generación automática de texto hasta la traducción de idiomas y la creación de chatbots avanzados.

- **Conducción Autónoma**

El desarrollo de vehículos autónomos ha sido otro logro destacado de la IA. Empresas como Tesla, Waymo y Uber han invertido significativamente en la creación de sistemas de conducción autónoma que utilizan una combinación de aprendizaje profundo, sensores avanzados y algoritmos de toma de decisiones. Estos sistemas son capaces de navegar de manera segura y eficiente en entornos urbanos complejos, y aunque todavía enfrentan desafíos regulatorios y tecnológicos, representan un paso importante hacia el futuro del transporte.

3.4 Aplicaciones existentes y alternativas

Actualmente, existen diversas aplicaciones y herramientas en el campo de la inteligencia artificial que se utilizan para entrenar agentes capaces de realizar tareas específicas o resolver problemas complejos, como los que se plantean en este proyecto. Estas aplicaciones suelen centrarse en áreas específicas como los videojuegos, la robótica, y la simulación en entornos virtuales. A continuación, revisaremos algunas de las plataformas más relevantes, sus características, y la manera en la que abordan el entrenamiento de agentes inteligentes.

- **OpenAI Gym**

OpenAI Gym es una de las plataformas más conocidas en el ámbito del aprendizaje por refuerzo [19]. Proporciona una serie de entornos simulados diseñados para que los agentes aprendan a través de la experimentación y la interacción con el entorno.

Esta plataforma permite el desarrollo de agentes que pueden aprender a maximizar sus recompensas mediante algoritmos de aprendizaje por refuerzo. Aunque ofrece múltiples entornos, su objetivo principal es funcionar como un conjunto de herramientas experimentales que faciliten la investigación en IA, buscando explorar y mejorar el rendimiento de agentes autónomos en diferentes contextos.

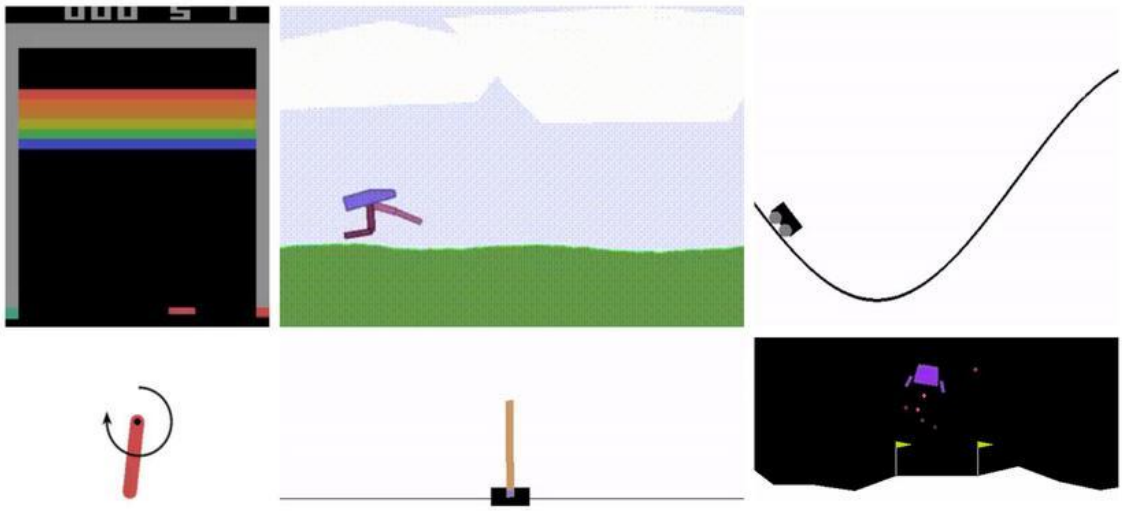

Figura 3.4: Entornos de entrenamientos de OpenAI Gym

- **Simulaciones en Robótica**

En el campo de la robótica, muchas aplicaciones integran técnicas de IA y simulación para entrenar agentes virtualmente antes de su interacción en el mundo real. Herramientas como Gazebo y ROS [20] facilitan la simulación de robots en entornos virtuales, aplicando métodos como el aprendizaje por refuerzo, lo cual permite evitar los riesgos y costos de errores en el entorno físico.

Estas simulaciones emplean recompensas y penalizaciones para guiar el aprendizaje del agente en escenarios seguros y controlados, permitiendo así un entrenamiento progresivo y sin consecuencias reales, un enfoque clave en robótica que también es ampliamente aplicable en otros entornos simulados.

- **PyTorch y Pygame**

PyTorch [6] es una biblioteca ampliamente utilizada para la construcción de redes neuronales y el desarrollo de modelos de aprendizaje profundo que permite crear redes flexibles y optimizadas para cálculos eficientes. Esta herramienta facilita el diseño, optimización y entrenamiento de redes neuronales, apoyando tanto tareas supervisadas como no supervisadas y de aprendizaje por refuerzo.

Pygame [5], por su parte, es una biblioteca de Python que está enfocada en la creación de entornos 2D para videojuegos. Su simplicidad y versatilidad la convierten en una opción perfecta para simular entornos interactivos, permitiendo desarrollar escenarios visuales y dinámicos adecuados para diversas pruebas y simulaciones.

3.5 Justificación de la opción elegida

Tras evaluar diversas plataformas y proyectos, se puede afirmar que cada alternativa ofrece ventajas específicas según el tipo de objetivo y los requerimientos técnicos. Herramientas como OpenAI Gym y Unity ML-Agents [21], por ejemplo, son perfectas para el entrenamiento de agentes en entornos simulados y videojuegos de alta complejidad, mientras que otros simuladores, como Gazebo, están diseñados para la interacción de agentes en entornos físicos simulados.

Para este proyecto, se ha optado por un entorno específico en el que el agente aprende en una cancha interactiva, lo cual facilita su implementación y se ajusta de manera óptima a los objetivos planteados. La elección de **Pygame** ha sido fundamental para construir un entorno visualmente representativo, controlado y fácilmente modificable, que permite observar en tiempo real el comportamiento del agente y evaluar su rendimiento durante el entrenamiento.

Durante el desarrollo del proyecto, se implementaron dos enfoques de aprendizaje distintos. En una primera fase, se utilizó **Q-Learning clásico**, pero al probarlo, la cantidad de estados resultó ser tan grande que la **tabla Q** se volvió inmanejable, y el rendimiento fue insuficiente debido al tamaño del espacio de estados. Ante esta dificultad, se implementó **Deep Q-Learning (DQN)**, resolviendo así el problema de rendimiento, puesto que las redes neuronales permitieron manejar el gran número de estados de forma más eficiente. Sin embargo, persistía un problema: el espacio de estados seguía siendo extremadamente grande, con cientos de millones de combinaciones posibles.

Ante esta limitación, se decidió aplicar una discretización del espacio de estados, lo que redujo significativamente el número de posibles estados a solo unos pocos cientos. Con esta simplificación, el espacio de estados pasó a ser mucho más manejable, permitiendo así mejorar considerablemente el tiempo de entrenamiento y la estabilidad del proceso.

Con el espacio de estados discretizado, se optó por volver a **Q-Learning clásico** en lugar de seguir utilizando DQN. Aunque **Deep Q-Learning** tiene la capacidad de manejar entornos con espacios de estados más grandes y continuos, uno de los desafíos que presenta es que puede no converger a una solución óptima debido a la naturaleza de las redes neuronales, que pueden quedar atrapadas en mínimos locales. Por el contrario, **Q-Learning clásico** garantiza que se encontrará una solución óptima en un espacio de estados discretizado y finito, lo que hace que sea una opción más adecuada para este entorno particular.

Si bien es cierto que el uso de redes neuronales en DQN ofrece una mayor capacidad de generalización, especialmente en entornos más amplios o menos estructurados, esta ventaja no resulta determinante en el proyecto. El entorno diseñado es completamente observable, finito y controlado, lo que minimiza la necesidad de generalización. Además, el coste computacional, la complejidad de implementación y el mayor tiempo de entrenamiento asociado a DQN no compensan las posibles ventajas en adaptabilidad, dado que el objetivo del proyecto es lograr un aprendizaje efectivo en un entorno cerrado y específico. Por tanto, el uso de Q-Learning clásico representa una solución más eficiente sin una pérdida significativa de rendimiento o versatilidad.

De esta manera, se evitó la complejidad innecesaria de seguir con un enfoque basado en redes neuronales, lo cual podía complicar aún más el proceso de entrenamiento sin ofrecer una ventaja significativa en cuanto a la calidad de las soluciones. La simplicidad de **Q-Learning clásico**, combinada con la discretización del espacio de estados, permitió una solución eficiente y estable para el problema planteado.

A pesar de que la versión final del agente utiliza **Q-Learning**, se considera relevante incluir en este trabajo la experiencia obtenida con **DQN**, tanto por el valor formativo que aportó como por tratarse de una implementación completamente funcional que permitió entender mejor los retos y ventajas del aprendizaje profundo en entornos interactivos.

4. Análisis del problema

4.1 Definición y contexto del problema

La inteligencia artificial ha avanzado notablemente en las últimas décadas, con un amplio abanico de aplicaciones desde la optimización de sistemas de recomendación hasta la conducción autónoma. Sin embargo, la creación de agentes inteligentes capaces de aprender, adaptarse y tomar decisiones efectivas en entornos dinámicos y desafiantes sigue siendo un gran reto. Esto es especialmente complejo en el campo del aprendizaje por refuerzo, donde el agente no recibe instrucciones explícitas, sino que aprende mediante la interacción directa con el entorno.

Uno de los problemas clave en este ámbito es desarrollar un agente capaz de no solo explorar y aprender estrategias óptimas en entornos simulados, sino también adaptarse a condiciones que puedan evolucionar o cambiar, como ocurre en muchos escenarios reales. Por lo tanto, la dificultad radica en dotar al agente de una estructura de aprendizaje que le permita optimizar su rendimiento de manera autónoma a medida que acumula experiencia, enfrentando situaciones diversas y desconocidas.

A su vez, el proyecto plantea la implementación de un entorno en el cual el agente pueda explorar y perfeccionar sus habilidades sin las limitaciones de hardware o los riesgos asociados a la interacción física en entornos reales. Este tipo de entorno virtual no solo facilita el entrenamiento, sino que ofrece un terreno controlado para probar y ajustar distintos algoritmos.

Por último, este problema no solo es relevante desde una perspectiva académica o de investigación, representa también una oportunidad de innovación en sectores que requieren que agentes autónomos toman decisiones en tiempo real y en escenarios complejos. Algunos de estos sectores son la conducción autónoma, donde los vehículos deben reaccionar de manera segura y eficiente a posibles imprevistos, y las operaciones de rescate, en las que agentes pueden desempeñar un papel crucial al explorar áreas peligrosas o inaccesibles para los humanos.

4.2 Análisis de necesidades

Para desarrollar un sistema de inteligencia artificial eficiente y autónomo, es necesario identificar las necesidades específicas que nos servirán para establecer el diseño y funcionalidad del agente.

En concreto, para este proyecto se requiere la creación de un agente capaz de aprender y adaptarse en un entorno simulado. Esto implica que el sistema debe estar diseñado para:

- **Aprender de la experiencia:**

  El agente necesita un entorno donde pueda interactuar y recibir retroalimentación sobre sus acciones, en forma de recompensas o penalizaciones.

- **Adaptarse a la complejidad:**

El entorno debe permitir una evolución gradual de la dificultad, para que el agente pueda desarrollar habilidades progresivamente.

- **Optimizar el rendimiento:**

El agente necesita una estructura de aprendizaje que lo impulse a optimizar sus decisiones a medida que acumula experiencia en el entorno, maximizando las recompensas y mejorando la eficiencia.

4.3 Especificación de requisitos

Para satisfacer las necesidades previamente establecidas, son necesarios una serie de requisitos que el proyecto debe cumplir:

- **Requisitos funcionales del entorno:**

RF1

Movimiento del jugador:

El jugador deberá permitir a la IA o a la persona que lo controle ser movido con total libertad hacía cualquier dirección cardinal con una cierta aceleración y deceleración. El jugador no deberá poder salirse del terreno de juego ni atravesar obstáculos.

RF2

Interacción con la pelota:

El jugador será capaz de interactuar con la pelota, pudiendo aplicarle su velocidad y dirección al colisionar con esta.

RF3

Rebote de la pelota:

La pelota rebotará con cualquier agente o límite del mapa que se encuentre, alterando su dirección en base al rebote.

RF4

Anotar gol:

Al colisionar la pelota con una portería se registrará la portería en la que se ha anotado, mostrando las puntuaciones de cada jugador.

RF5

Reinicio del nivel:

Al marcar un gol o al pasar un periodo x de tiempo, se reiniciarán las posiciones y velocidades de tanto jugadores como pelota, permitiendo realizar otra iteración.

RF6

Registro de gráficas:

Al realizar un reinicio, se guardará en una gráfica para cada agente la recompensa obtenida en esa iteración y la media de la suma de todas las recompensas.

- **Requisitos funcionales de la Inteligencia Artificial:**

RF7

Percepción del entorno:

La IA deberá de ser capaz de comprender donde se encuentran los elementos situados en el mapa y como interactuar con estos.

RF8

Cercanía a la pelota:

La IA deberá de poder mover a los jugadores hacia la dirección donde se encuentre la pelota.

RF9

Marcar gol:

La IA deberá entender que debe acercar la pelota a la portería rival, para anotar un punto.

RF10

No marcar gol en propia:

La IA deberá de ser capaz de diferenciar entre portería amiga y enemiga para no marcar en propia.

RF11

Interacción con el rival:

La IA deberá de competir con el rival por la pelota y evitar que este marque gol.

RF12

Detectar cuando le han marcado gol:

La IA deberá de percibir cuando le han marcado gol para intentar evitarlo en siguientes iteraciones.

- **Requisitos no funcionales**

RNF1

Rendimiento:

El entorno de entrenamiento deberá estar optimizado para evitar un uso excesivo de recursos en este durante el entrenamiento.

RNF2

Reusabilidad:

El entorno y los jugadores podrán ser utilizados en otros proyectos/niveles realizando las mínimas modificaciones.

RNF3

Escalabilidad:
El programa deberá soportar realizar entrenamientos simultáneos para agilizar el proceso de entrenamiento

RNF4

Robustez:

El programa deberá de evitar en la medida de lo posible los errores que puedan entorpecer el entrenamiento de los agentes.

4.4 Solución propuesta

La solución propuesta en este proyecto busca crear un agente inteligente dentro de un entorno simulado en 2D, que aprenda a maximizar recompensas a través de la toma de decisiones autónomas. Para ello, se ha implementado un modelo de aprendizaje por refuerzo, mediante el cual el agente mejora sus estrategias en función de sus experiencias dentro del entorno.

En las primeras etapas del proyecto, se planteó una solución basada en Deep Q-Learning (DQN), utilizando la biblioteca PyTorch para la construcción y entrenamiento de redes neuronales profundas. Esta aproximación permitió explorar el aprendizaje en entornos con espacios de estados complejos.

No obstante, tras simplificar la representación del entorno y reducir significativamente el espacio de estados, se optó por una implementación final basada en Q-Learning clásico, que resultó más adecuada y eficiente en este contexto. Esta versión prescinde de redes neuronales, pero mantiene el enfoque de aprendizaje basado en la interacción del agente con el entorno y la optimización de recompensas acumuladas.

Para el diseño y visualización del entorno en 2D se utiliza Pygame, que permite representar gráficamente el campo de juego y simular los elementos clave de la interacción. Esta herramienta facilita que el agente observe, actúe y reciba retroalimentación en tiempo real, lo que contribuye a un entrenamiento visual y comprensible.

Así, el agente sigue un ciclo continuo de observación, decisión, acción y retroalimentación, ajustando progresivamente sus estrategias en función de las recompensas recibidas. Este enfoque facilita la mejora progresiva de su rendimiento, asegurando que pueda adaptarse a distintas tareas dentro del entorno y perfeccionar sus decisiones de acuerdo con las recompensas obtenidas.

La solución final, basada en Q-Learning clásico, proporciona una plataforma accesible y eficiente para experimentar con aprendizaje por refuerzo en entornos controlados. Además, la arquitectura modular del proyecto permite futuras extensiones hacia simulaciones más complejas o el uso de técnicas más avanzadas como DQN, si el entorno lo requiere.

4.5 Plan de trabajo

Como comentamos en apartados anteriores, hemos optado por seguir una metodología de trabajo basada en el desarrollo iterativo e incremental, permitiendo así dividir el proyecto en objetivos específicos y secuenciales, organizando el trabajo de manera clara y eficiente.

El proyecto puede dividirse en las siguientes fases:

- **Fase 1**

  Planificación y Definición de Requisitos

  En esta fase inicial del proyecto, se llevó a cabo una identificación y documentación exhaustiva de los objetivos y requisitos del sistema de inteligencia artificial (IA) que se desarrollaría. Se establecieron los objetivos generales del proyecto, como el desarrollo de un agente capaz de aprender y mejorar sus habilidades en un entorno simulado. Se definieron los requisitos específicos del entorno de juego, los algoritmos de IA a utilizar y las herramientas necesarias, como Pygame. Esta planificación detallada creó una base sólida y un plan claro que guiaría las posteriores fases del desarrollo.

- **Fase 2**

  Diseño del Sistema

  Aquí se definieron la arquitectura y los componentes funcionales del sistema. Se diseñó el entorno interactivo con Pygame, incluyendo la cancha, los jugadores y la pelota. Paralelamente, se elaboró un primer diseño para un agente basado en Deep Q-Learning, detallando la estructura de la red neuronal, funciones de activación y parámetros de entrenamiento. Este diseño detallado permitió que todas las partes del sistema estuvieran bien definidas y que el plan de desarrollo fuera coherente y factible. Posteriormente, y tras simplificar el espacio de estados, este enfoque fue reemplazado por un modelo basado en Q-Learning clásico, más adecuado para la nueva configuración del entorno.

- **Fase 3**

  Desarrollo IterativoP

  El desarrollo se llevó a cabo mediante ciclos iterativos. En cada iteración se implementaron y probaron componentes específicos, como la integración del entorno con el agente o los algoritmos de aprendizaje. Este enfoque permitió introducir mejoras gradualmente, facilitando la incorporación controlada de nuevas funcionalidades y el ajuste progresivo del sistema.

- **Fase 4**

  Pruebas y Validación

  Durante cada iteración, se llevó a cabo una fase de pruebas para validar que los componentes desarrollados funcionaran según lo esperado. Se probaron las capacidades del agente en el entorno de juego, evaluando su desempeño y ajustando los parámetros de entrenamiento en base a los resultados obtenidos. Esta fase de pruebas tempranas permitió detectar y corregir errores, garantizando la efectividad del sistema.

- **Fase 5**

  Incremento de la Complejidad

  A medida que se avanzaba en las iteraciones, se incrementó gradualmente la complejidad de las tareas asignadas al agente. Inicialmente, se desarrollaron habilidades básicas, como moverse y chutar la pelota, y progresivamente se introdujeron desafíos más complejos, como el posicionamiento respecto de la portería. Este enfoque gradual facilitó un aprendizaje progresivo, permitiendo que la IA desarrollara habilidades fundamentales antes de enfrentar problemas más complicados.

- **Fase 6**

  Documentación y Análisis

  A lo largo del proceso de desarrollo, se documentó el progreso del agente y se analizaron los resultados obtenidos en cada iteración. Se registraron los cambios en los parámetros de entrenamiento, las mejoras en el desempeño del agente y las dificultades encontradas. Esta documentación detallada fue crucial para mantener un registro del desarrollo y el rendimiento de la IA, proporcionando una base sólida para realizar ajustes y mejoras continuas.

- **Fase 7**

  Iteración y Mejora Continua

  Basado en el análisis de los resultados obtenidos en cada iteración, se realizaron iteraciones adicionales para mejorar el diseño del agente y del entorno de entrenamiento. Este ciclo de retroalimentación continua permitió refinar y optimizar el rendimiento del agente, asegurando así un proceso constante de mejora y adaptación a los desafíos del entorno de juego.

- **Fase 8**

  Evaluación Final

  En la fase final del proyecto, se llevó a cabo una evaluación del desempeño del agente. Se analizaron su capacidad para cumplir con los objetivos establecidos y su eficacia en el entorno de entrenamiento. Esta evaluación final permitió identificar áreas de mejora y realizar ajustes finales antes de la conclusión del proyecto, asegurando que el agente desarrollado cumpliera con las expectativas y objetivos planteados desde el inicio.

A continuación, se presenta un diagrama de Gantt que resume gráficamente la planificación temporal del proyecto, incluyendo las distintas fases de desarrollo y los periodos de pausa debido a la realización de prácticas externas. Este cronograma refleja de forma visual cómo se ha distribuido el trabajo a lo largo del tiempo y permite observar la secuencia, duración y solapamiento de las actividades realizadas.

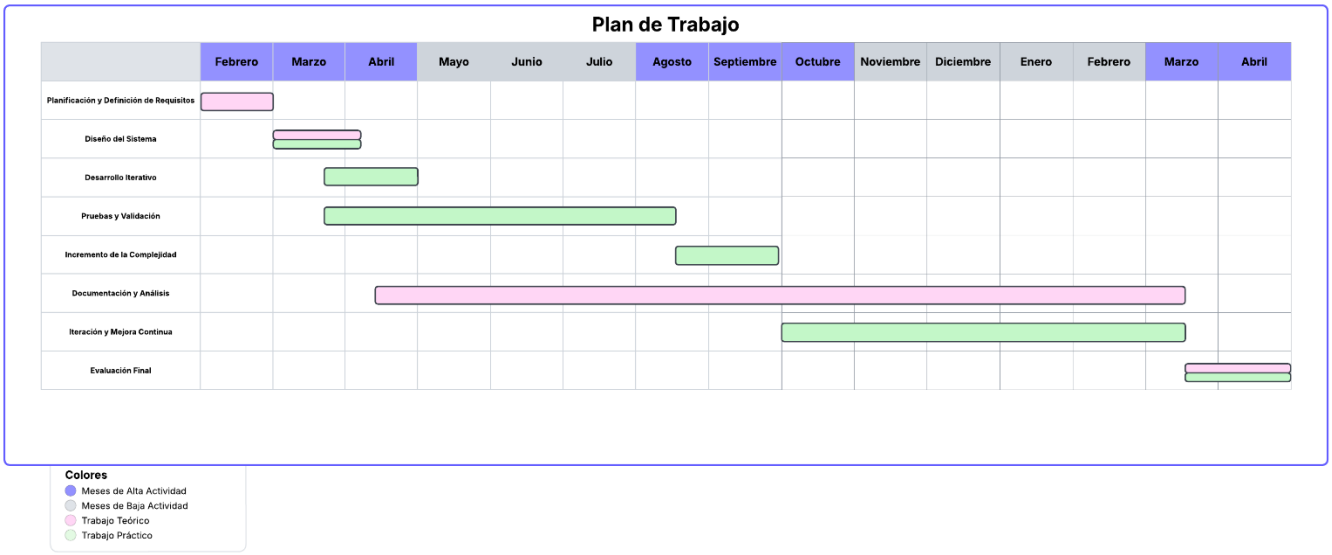

Figura 4.1: Diagrama de Gantt del proyecto

5. Diseño de la solución

5.1 Análisis de las herramientas

En este capítulo mostraremos las herramientas que hemos utilizado para desarrollar este proyecto y describiremos la arquitectura general de la solución propuesta, identificando los principales bloques o subsistemas que la componen. A continuación, se detallarán estos subsistemas y se especificarán, a nivel de diseño, los aspectos técnicos que definen la solución, logrando así, una comprensión clara y estructurada del sistema que a su vez facilitará su implementación y mantenimiento.

- **Python**

Nos hemos decantado por Python porque durante nuestra carrera, nos hemos familiarizado con este lenguaje, especialmente en el ámbito del entrenamiento de inteligencia artificial. Esta elección se debe en gran medida a la gran variedad de bibliotecas especializadas que Python ofrece, facilitando en gran medida el desarrollo de IA. Python destaca por su simplicidad, versatilidad y facilidad de uso, lo que lo convierte en una base ideal para nuestro proyecto.

A su vez, una de las mayores fortalezas de Python es su vasta y activa comunidad. Esta comunidad no solo contribuye con un flujo constante de recursos, documentación y tutoriales, sino que también desarrolla y mantiene una gran variedad de bibliotecas que amplían las capacidades del lenguaje. Entre estas bibliotecas, se encuentran herramientas específicas para el desarrollo de inteligencia artificial y simulaciones, como PyTorch y Pygame, que son componentes cruciales en nuestro proyecto.

Python también es conocido por su integración y compatibilidad con otras tecnologías y lenguajes, lo que lo hace extremadamente flexible y adaptable a diferentes necesidades de desarrollo.

Teniendo en cuenta todas estas ventajas, hemos decidido elegir Python como el lenguaje de programación principal para nuestro proyecto, asegurando así un desarrollo eficiente y eficaz en el ámbito de la inteligencia artificial.

- **PyTorch**

Hemos elegido PyTorch [6] como la biblioteca principal para el desarrollo de redes neuronales y modelos de aprendizaje profundo durante la fase experimental del proyecto, concretamente en la implementación del agente basado en Deep Q-Learning (DQN). Aunque finalmente se optó por una versión basada en Q-Learning clásico para la solución definitiva, PyTorch fue clave en el desarrollo y prueba de la alternativa con redes neuronales profundas.

PyTorch es conocido por su facilidad de uso y flexibilidad, lo que permite a los desarrolladores construir, modificar y entrenar modelos de manera eficiente. Aunque no lo hemos utilizado mucho a lo largo de nuestra carrera, hemos investigado y encontrado que PyTorch parecía ser una de las opciones más cómodas y sencillas de implementar en nuestro proyecto.

Una de las mayores ventajas de PyTorch es su fuerte integración con Python, lo que no solo facilita la escritura de código, sino que también permite una fácil depuración y ajuste de los modelos. Esto es crucial en proyectos de inteligencia artificial, donde el proceso de ajuste fino y la iteración rápida son esenciales para alcanzar resultados óptimos. A su vez, PyTorch ofrece una amplia gama de herramientas y módulos predefinidos, como torch.nn, que simplifican la creación y el entrenamiento de redes neuronales complejas.

Además, PyTorch es ampliamente adoptado en la industria, lo que asegura que los conocimientos adquiridos sean relevantes para el futuro profesional. Su compatibilidad con otras bibliotecas y herramientas de Python también amplía sus capacidades y permite la integración con otros componentes esenciales para nuestro proyecto, como Pygame.

Pese a no formar parte de la solución final, su inclusión en las fases de diseño y pruebas contribuyó significativamente a comprender y explorar enfoques alternativos, y su uso resultó valioso como parte del proceso iterativo de desarrollo.

- **Pygame**

Hemos optado por utilizar Pygame [5] para la creación del entorno interactivo de nuestro proyecto después de considerar varias alternativas. Inicialmente, nos debatíamos entre usar Unity para desarrollar un entorno 3D o utilizar Pygame para un entorno 2D. Tras evaluarlo cuidadosamente, decidimos optar por Pygame tanto por consejo de nuestro tutor como por su simplicidad a la hora de programar, dado que utiliza Python, el lenguaje con el que estamos más familiarizados.

Pygame es una biblioteca de Python diseñada para la programación de videojuegos en 2D, proporcionando una plataforma robusta y accesible para la creación de entornos interactivos. Su sintaxis sencilla y su integración directa con Python permiten un desarrollo rápido y eficiente, lo cual es crucial para iterar y ajustar el entorno de simulación según las necesidades del proyecto.

Una de las principales ventajas de Pygame es su facilidad de uso. La simplicidad de su API facilita la creación de gráficos, la gestión de eventos y la implementación de lógica de juego, permitiéndonos concentrarnos en el desarrollo del agente. Además, Pygame está bien documentado y cuenta con numerosos recursos y ejemplos, lo que reduce significativamente la curva de aprendizaje.

Aunque Unity ofrece capacidades avanzadas para la creación de entornos 3D y una gran cantidad de herramientas, su complejidad y la necesidad de utilizar C# representaban un desafío adicional que podría haber desviado nuestro enfoque del objetivo principal del proyecto. Pygame, al estar basado en Python, nos permitió aprovechar nuestras habilidades existentes y concentrarnos en la implementación del aprendizaje por refuerzo y el desarrollo del agente.

Además, la elección de Pygame se alinea con los objetivos del proyecto, proporcionando un entorno suficientemente complejo para demostrar los principios del aprendizaje profundo y la inteligencia artificial, pero sin la sobrecarga adicional de un entorno 3D complejo. Esta decisión ha facilitado un proceso de desarrollo más ágil y ha permitido iteraciones rápidas para mejorar y ajustar el comportamiento del agente.

- **Q-Learning**

Tras investigar y evaluar varios algoritmos de aprendizaje por refuerzo, hemos decidido utilizar Q-Learning para el desarrollo final del agente. Nuestra elección de Q-Learning se vio influenciada por su amplia documentación técnica y la claridad con la que se explica el funcionamiento y las aplicaciones prácticas de este algoritmo.

Q-Learning es un algoritmo de aprendizaje por refuerzo que permite a un agente aprender cómo actuar en un entorno, con el objetivo de maximizar una recompensa acumulada a lo largo del tiempo. La simplicidad y eficacia de Q-Learning lo convierten en una opción ideal para proyectos que buscan implementar técnicas de inteligencia artificial en entornos simulados.

Uno de los motivos principales para elegir Q-Learning fue la accesibilidad y la claridad de la información disponible en un gran número de artículos académicos [1][7]. Además, al comparar Q-Learning con otros algoritmos, encontramos que era el más adecuado para nuestro entorno específico, ya que es relativamente fácil de implementar y no requiere un modelo del entorno.

Q-Learning se basa en la idea de aprender una función de valor que indica la calidad de una acción en un estado particular. A medida que el agente explora el entorno, actualiza esta función de valor basándose en las recompensas recibidas, lo que le permite mejorar gradualmente su política de decisiones. Este proceso iterativo de aprendizaje y mejora se adapta bien a nuestro entorno de simulación, donde el agente debe aprender a tomar decisiones efectivas para maximizar su desempeño.

Además, Q-Learning es conocido por su capacidad para manejar problemas de toma de decisiones en entornos discretos y su facilidad de ajuste a diferentes escenarios y requisitos de proyecto. Su estructura basada en tablas permite un seguimiento claro del progreso del agente y facilita la identificación y corrección de errores durante el entrenamiento.

- **Deep Q-Learning**

Durante las primeras fases del proyecto, exploramos el uso de Deep Q-Learning (DQN) [3] [4] como una posible solución para entrenar al agente. A diferencia de Q-Learning clásico, que utiliza una tabla para representar el valor de cada par estado-acción, DQN emplea una red neuronal para aproximar la función Q, lo que permite manejar espacios de estados más grandes y complejos.

El uso de Deep Q-Learning se consideró debido a que el entorno inicial tenía una alta dimensionalidad, y una tabla de Q-Learning se volvía rápidamente inviable por cuestiones de memoria y rendimiento. La red neuronal permitió solucionar este problema puesto que a diferencia de Q-Learning no necesitaba guardar en memoria cada posible estado.

Sin embargo, tras varias pruebas, logramos reducir el tamaño del estado en gran medida, lo que hizo que Q-Learning clásico resultara más eficiente y fácil de controlar. Aunque Deep Q-Learning no se empleó en la versión final del proyecto, su implementación fue útil para comparar enfoques y comprender mejor las limitaciones de cada técnica.

Esta fase experimental con DQN también nos permitió familiarizarnos con el uso de redes neuronales en el contexto del aprendizaje por refuerzo, así como con herramientas como PyTorch, que resultaron clave para esa etapa del desarrollo.

5.2 Arquitectura

La arquitectura de nuestro proyecto se puede dividir en 6 grandes módulos que se encargarán de las distintas tareas durante el entrenamiento:

1. Interfaz de Usuario y Control de Juego

Este componente es el responsable de la interacción con el usuario y el manejo de la lógica del juego. Utilizando Pygame, se diseñará una interfaz que permita visualizar el entorno de juego, incluyendo el campo, los jugadores y la pelota. Además, se implementarán controles básicos para que el usuario pueda interactuar con el sistema, ya sea directamente o a través de parámetros de configuración.

2. Entorno de Simulación

El entorno de simulación recrea el campo de fútbol donde el agente interactúa. Pygame es utilizado para construir este entorno 2D, que incluye la física básica de los movimientos de los jugadores y la pelota, las colisiones y los rebotes. Este subsistema asegura que el entorno sea lo más realista posible dentro del ámbito 2D, proporcionando un espacio controlado para el entrenamiento del agente.

3. Agentes

Este bloque se centra en el agente que tomará decisiones dentro del entorno simulado. Aquí es donde se aplican los algoritmos de Q-Learning. El agente recibe el estado actual del entorno, decide una acción basada en su política entrenada, y actualiza sus conocimientos basados en la recompensa obtenida.

4. Algoritmos de Aprendizaje

Este subsistema implementa los algoritmos de aprendizaje por refuerzo empleados a lo largo del proyecto. Inicialmente, se exploró el uso de **Deep Q-Learning (DQN)** para manejar entornos con un espacio de estados continuo y complejo. Sin embargo, al lograr discretizar el entorno final y la necesidad de una mayor simplicidad a la hora de entrenar, se optó por utilizar **Q-Learning clásico** como solución definitiva. Este algoritmo permitió un entrenamiento más rápido y controlado, facilitando el análisis del comportamiento del agente en cada fase del aprendizaje.

5. Evaluación y Análisis de Rendimiento

Este subsistema se encarga de evaluar el rendimiento del agente a lo largo del proceso de entrenamiento. Incluye herramientas y métricas para analizar la eficacia de las decisiones del agente, la eficiencia del entrenamiento y el progreso hacia los objetivos del proyecto. Los resultados de esta evaluación permiten realizar ajustes y mejoras continuas.

6. Componente Principal (Main)

El componente principal se encarga de coordinar y desplegar todos los componentes del programa. Este módulo inicializa el entorno, ejecuta el bucle principal del juego, gestiona la interacción del usuario y orquesta la comunicación entre los diferentes subsistemas. Su rol es crucial para asegurar que todos los componentes funcionen en armonía y que el entrenamiento del agente se realice de manera efectiva.

5.3 Diseño detallado

- **Interfaz de Usuario y Control de Juego:**

Para la interfaz de usuario, adoptamos un enfoque simple y directo, evitando complejidades innecesarias manteniendo así el foco en el entrenamiento del agente. La interfaz principal muestra el marcador de goles en pantalla, proporcionando información simple y clara sobre el desempeño del agente. Durante las fases iniciales del entrenamiento, también mostramos los toques que los jugadores realizaban a la pelota, lo cual ayudaba a monitorear las interacciones en tiempo real y realizar ajustes según fuera necesario.

En las primeras versiones del entorno, implementamos controles básicos utilizando las teclas WASD y las flechas direccionales, permitiendo a los usuarios mover los jugadores manualmente. Esto facilitó las pruebas iniciales y el ajuste de la simulación antes de ceder el control completamente a las IAs. Estos controles manuales permitieron verificar la funcionalidad del entorno y asegurar que los jugadores respondían correctamente a las entradas del usuario.

Aunque en las versiones posteriores los controles manuales fueron desactivados para permitir que las IAs asumieran el control completo de los jugadores, los controles básicos permanecen implementados en el código. Esto permite su reactivación si es necesario para futuras pruebas o ajustes.

- **Entorno de simulación**

El entorno de simulación se ha diseñado para replicar un pequeño campo de fútbol 5, proporcionando un espacio adecuado para el entrenamiento de los agentes. Este campo incluye todos los elementos esenciales para simular un juego de fútbol en 2D, lo que permite a los agentes interactuar y aprender en un entorno controlado y realista.

Se han incorporado porterías en cada extremo del campo, capaces de detectar automáticamente cuándo se marca un gol. En las etapas iniciales del proyecto, las porterías eran de tamaño reducido, pero debido a la baja precisión de los agentes en sus primeros entrenamientos, se optó por ampliarlas para facilitar el aprendizaje, emulando así el comportamiento del **Pong**. Cada vez que se anota un gol, el entorno se reinicia de forma automática, permitiendo el entrenamiento continuo y una acumulación eficiente de experiencias. Este reinicio del entorno mantiene un flujo constante de eventos y proporciona un feedback claro y consistente al agente.

Además, se han establecido límites en el mapa que actúan como las paredes del campo, impidiendo que los jugadores y la pelota salgan del área de juego. Estas paredes no solo delimitan el campo, sino que también forman parte de la física de la simulación, permitiendo que la pelota rebote contra ellas de manera realista.

Por último, la pelota ha sido diseñada con un conjunto de físicas que incluye aceleración, frenado y rebotes. La interacción de la pelota con los jugadores y las paredes sigue principios físicos básicos, asegurando que los movimientos y las colisiones se sientan naturales. Esto incluye la transferencia de impulso y ángulo cuando un jugador chuta la pelota y el comportamiento de la pelota al impactar contra diferentes obstáculos. Además, para evitar que los agentes se sobreentrenen en una única posición y desarrollen estrategias limitadas, hemos implementado que la pelota aparezca de forma aleatoria en seis puntos diferentes del campo, obligando a los agentes a adaptarse a diferentes situaciones y mejorando la robustez de las estrategias aprendidas.

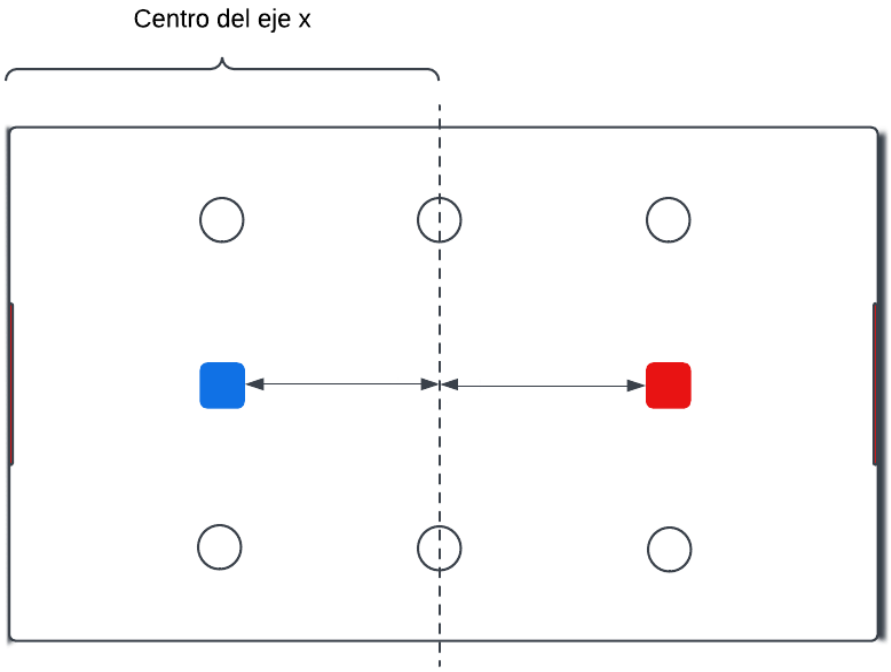

Figura 5.1: Primera versión del entorno de simulación

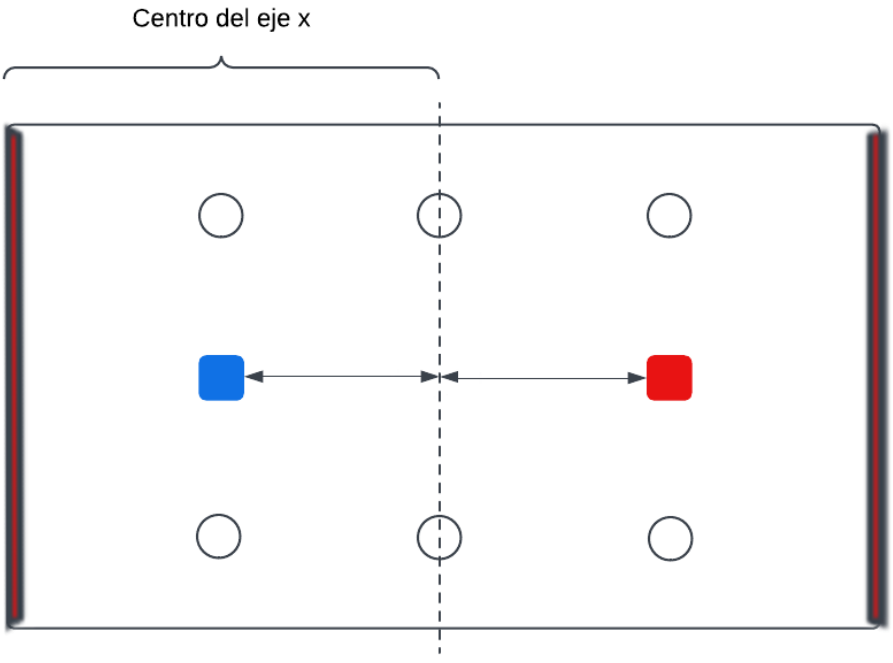

Figura 5.2: Versión final del entorno de simulación

- **Agentes**

Los agentes son los componentes que controlan a los jugadores dentro del entorno de simulación. Su principal objetivo es entrenarse para marcar goles en las porterías enemigas y esquivar los obstáculos presentes en el campo. Para ello, hemos diseñado un sistema de recompensas que guía el aprendizaje de los agentes, incentivando comportamientos deseados y penalizando acciones ineficientes o contraproducentes.

Para cada acción realizada por el agente, se asigna una recompensa basada en su eficacia y resultado. La recompensa más significativa es, por supuesto, la de marcar un gol, cada vez que un agente logra anotar un gol en la portería enemiga, se le otorgan +40 puntos, por el contrario, si marca en propia, se le penaliza con -40 puntos. Esta mecánica refuerza de forma clara la intención del agente de buscar el gol mientras protege su propia portería.

Además de las recompensas por goles, se han implementado otras recompensas menores que oscilan entre -1 y +4 puntos. Estas recompensas están centradas en el posicionamiento del agente, fomentando que se acerque a la pelota, que la empuje hacia la portería y que cuando está mal posicionado con la pelota respecto del ángulo de tiro, la rodee.

- **Algoritmos de Aprendizaje**

En las primeras etapas del proyecto, el entorno generaba un espacio de estados extremadamente amplio, ya que cada estado consideraba múltiples factores como la posición exacta de la pelota, del jugador, sus velocidades, orientación, etc. Esto provocaba que el número total de estados aumentase exponencialmente, llegando fácilmente a decenas o incluso cientos de millones de combinaciones. Debido a esta magnitud, Q-Learning no era una opción viable en ese momento, ya que habría requerido almacenar y actualizar una tabla con millones de entradas, algo totalmente inviable a nivel de rendimiento y recursos.

Por este motivo, optamos por utilizar Deep Q-Learning (DQN), que nos permitía aproximar los Q-Values [7] mediante una red neuronal en lugar de una tabla explícita. En este enfoque, la red recibe como entrada el estado actual y devuelve los Q-values para cada acción posible. Esto reduce drásticamente el uso de memoria y permite abordar espacios de estados complejos. Para mejorar el aprendizaje, implementamos técnicas como Prioritized Experience Replay (PER) [22], que da prioridad a aquellas experiencias con mayor error temporal (TD error), es decir, aquellas donde el modelo se equivoca más y por tanto puede aprender más. PER se apoyaba en estructuras tipo árbol para poder seleccionar estas experiencias de forma eficiente, acelerando y estabilizando el entrenamiento.

A pesar de estas mejoras, nos encontramos con una nueva barrera: el entrenamiento con Deep Q-Learning seguía siendo demasiado lento. El tamaño del espacio de estados seguía siendo tan grande que, incluso con una red neuronal, se necesitaban millones de episodios para empezar a ver comportamientos útiles. En nuestras pruebas, esto se traducía en días o incluso semanas de entrenamiento sin señales claras de aprendizaje, algo que no podíamos permitirnos.

Por esta razón, decidimos cambiar de estrategia y rediseñar por completo la representación del estado. En lugar de utilizar coordenadas absolutas y valores continuos, empezamos a codificar la información de forma relativa y discreta. Por ejemplo, tal y como explicamos en apartados anteriores, la posición de la pelota respecto al jugador ya no se representaba como una coordenada (x, y), sino como una dirección en cada eje: -1, 0 o 1. Esta misma lógica se aplicó a otros elementos del entorno, como la posición de la portería, permitiendo reducir drásticamente la complejidad del estado.

Gracias a esta discretización, conseguimos reducir el número total de combinaciones posibles a tan solo 324 estados únicos, lo cual volvió viable utilizar Q-Learning clásico con una tabla Q manejable. Esto nos permitió no solo acelerar el proceso de entrenamiento de forma significativa, sino también ganar en simplicidad y control sobre el aprendizaje. El agente ya no necesitaba millones de episodios para aprender algo útil: en unas pocas decenas de miles, ya empezaba a mostrar comportamientos coherentes.

Este cambio supuso un punto de inflexión en el proyecto. Pasar de Deep Q-Learning a Q-Learning no fue una vuelta atrás, sino una adaptación lógica que nos permitió sacar el máximo partido al entorno y a los recursos disponibles, sin perder capacidad de aprendizaje. Gracias a la reducción del espacio de estados y al uso de recompensas bien diseñadas, el agente pudo entrenarse de forma eficiente, robusta y predecible.

- **Evaluación y análisis de rendimiento**

La Evaluación y Análisis de Rendimiento es una parte fundamental de nuestro proyecto, ya que nos permite monitorizar y entender cómo evolucionan los agentes a lo largo del tiempo. Esta evaluación se realiza de manera iterativa, analizando el desempeño de los agentes en cada ciclo de entrenamiento.

En cada iteración, que se define como el período durante el cual un agente intenta marcar un gol o hasta que transcurren 1000 acciones del agente, registramos las recompensas totales obtenidas por el agente. Estas recompensas reflejan las acciones del agente y su capacidad para alcanzar objetivos específicos, como mover la pelota hacia la portería contraria o evitar obstáculos. Al finalizar cada iteración, las recompensas acumuladas se registran y se representan en una gráfica, que guardamos en un fichero.

Esta gráfica de recompensas totales por iteración nos permite visualizar cómo fluctúan las recompensas a lo largo del tiempo para cada agente. Podemos observar si hay patrones de mejora o si el agente enfrenta dificultades específicas en ciertas fases del juego. Estas fluctuaciones nos ofrecen una visión detallada del comportamiento y la adaptación del agente al entorno de entrenamiento.

Además, llevamos un registro acumulativo de las recompensas para cada agente. Creamos una segunda gráfica que muestra la suma de las puntuaciones en relación con el número total de iteraciones. Esta gráfica, que representa la media de las recompensas, es crucial para entender la tendencia general del rendimiento del agente. A través de esta gráfica, podemos evaluar si hay una mejora continua en el desempeño del agente y si las estrategias aprendidas están siendo efectivas a lo largo del tiempo.

Mediante el análisis de ambas gráficas, podemos sacar conclusiones significativas sobre el desarrollo de los agentes. Si observamos un aumento en las puntuaciones medias, podemos afirmar que el agente está aprendiendo y mejorando sus habilidades. Por otro lado, si las gráficas muestran una estancación o disminución en las recompensas, esto puede indicar la necesidad de ajustar los parámetros de entrenamiento, modificar las estrategias de recompensa, o reevaluar el diseño del entorno de simulación.

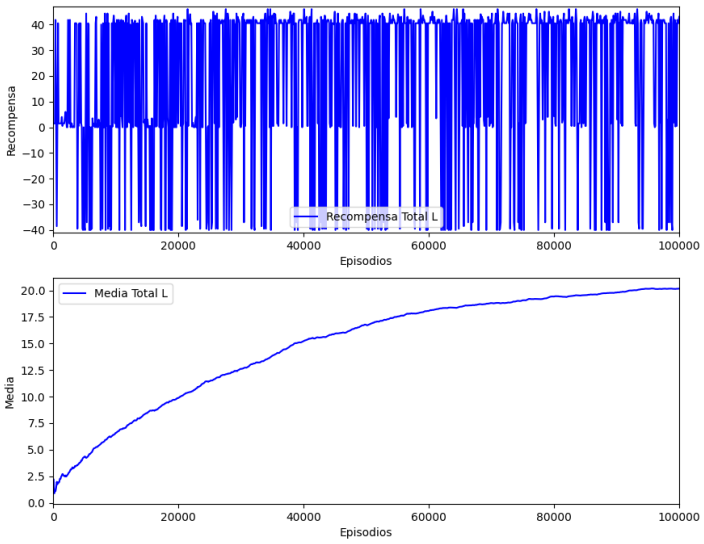

Figura 5.4: Ejemplo de gráfica del entrenamiento de los agentes

- **Componente principal**

El componente principal actúa como el centro de mandos del programa, coordinando la interacción entre los diferentes componentes del sistema. En primer lugar, se crea una instancia de cada agente, del entorno de pruebas y del monitor que se encargará de llevar las gráficas. Estas instancias son fundamentales para asegurar que cada parte del sistema esté correctamente inicializada y lista para interactuar entre sí.

Una vez que estas instancias están cargadas, inicia un bucle principal. Este bucle se encarga de llamar y actualizar todos los procesos del entorno de pruebas y del entrenamiento del agente. Durante cada iteración del bucle, el entorno de pruebas genera valores que reflejan el estado actual del juego, como la posición de los jugadores, la pelota y otros elementos relevantes.

Estos valores son pasados al agente, permitiéndole evaluar el estado del entorno y decidir las acciones a tomar. El agente guarda en su Q-table estos valores actualizando así sus conocimientos y mejorando su estrategia, basándose en las recompensas obtenidas y las experiencias previas.

El componente principal también monitorea el reseteo del entorno al final de cada iteración. Cuando detecta un reseteo, comunica esta información tanto al agente como al monitor, permitiéndoles ajustar sus procesos en consecuencia. El agente puede utilizar esta información para actualizar su estrategia y el monitor para actualizar las gráficas con los datos obtenidos durante la iteración. Estas gráficas proporcionan una visualización continua del progreso del agente, permitiendo evaluar su rendimiento y realizar ajustes si es necesario.

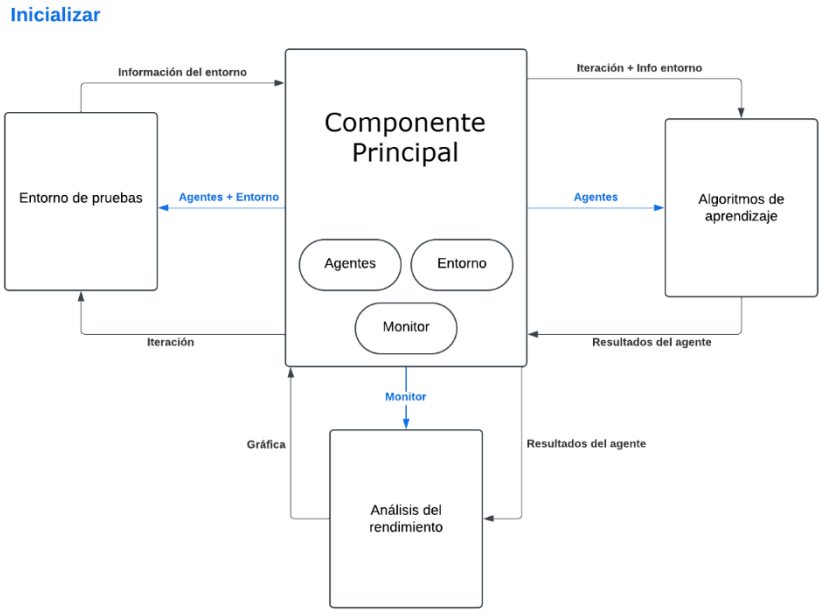

Figura 5.5: Estructura del componente principal

Cabe destacar que, si bien hemos presentado el proyecto dividido en bloques funcionales para facilitar su explicación conceptual, la estructura real del código se organiza de manera diferente. Esta organización queda reflejada en el siguiente diagrama de clases, que representa con mayor fidelidad cómo se relacionan entre sí los distintos componentes del sistema. En el próximo capítulo profundizaremos en esta estructura, explicando cómo se implementa cada módulo, cómo interactúan entre ellos y cómo se gestiona el flujo de datos y control a lo largo del programa.

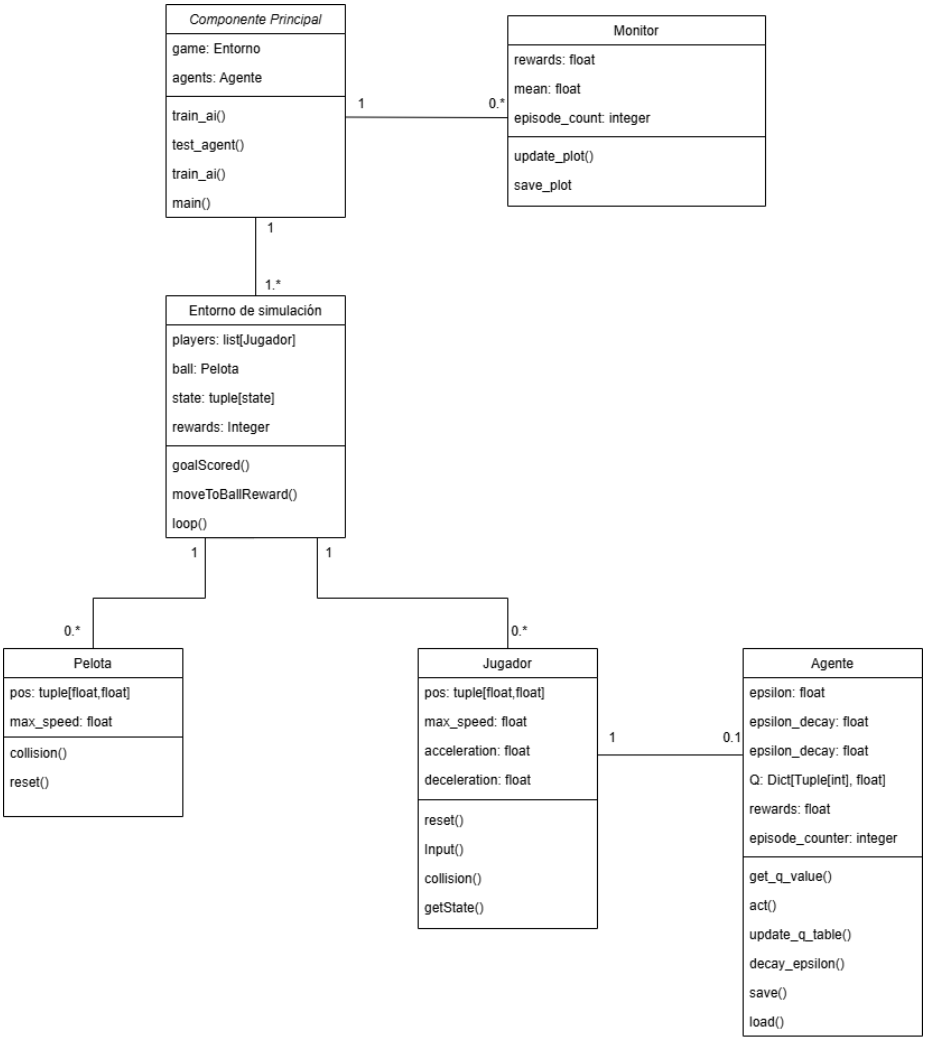

Figura 5.6: Diagrama de clases del proyecto

6. Desarrollo de la solución

En este capítulo se detalla el proceso que hemos seguido para pasar de la propuesta inicial a la solución final del proyecto. Exploraremos tanto los problemas y dificultades que encontramos en el camino, como las decisiones clave que tomamos para superarlos. Este capítulo también abordará las particularidades de la solución final, destacando aspectos relevantes, novedosos y complejos, esenciales para entender el sistema desarrollado.

Dividiremos el capítulo en varias secciones principales:

- **Estructura del Código:**

  Describiremos cómo hemos organizado y estructurado nuestro código, explicando los componentes principales y su interacción. A pesar de haber presentado una división conceptual en la sección de arquitectura, aquí profundizaremos en la verdadera organización del código para facilitar su comprensión.

- **Desarrollo del Entrenamiento:**

  Analizaremos el proceso de entrenamiento de los agentes, desde la implementación inicial hasta la optimización final. Discutiremos las estrategias utilizadas, los ajustes realizados y los resultados obtenidos, proporcionando una visión detallada del progreso y la evolución de los agentes a lo largo del tiempo.

- **Problemas y Dificultades Encontradas:**

  Identificaremos los desafíos más significativos que enfrentamos durante el desarrollo del proyecto, describiendo cómo los abordamos y las soluciones que implementamos para superarlos.

- **Decisiones Clave:**

  Detallaremos las decisiones importantes que influyeron en la dirección del proyecto, explicando el razonamiento detrás de cada elección y su impacto en la solución final.

- **Particularidades de la Solución Final:**

  Destacaremos los aspectos más distintivos de la solución final, incluyendo innovaciones y mejoras que se introdujeron para alcanzar los objetivos del proyecto.

6.1 Desarrollo del Código

Para explicar el desarrollo del código, vamos a dividir este apartado en dos partes. En primer lugar, abordaremos la **estructura general**, con el objetivo de ofrecer una visión global sobre cómo se relacionan los distintos componentes del sistema. A continuación, nos centraremos en los **componentes esenciales**, donde explicaremos de forma más detallada el funcionamiento interno de cada uno de ellos.

6.1.1 Estructura del Código

Tal y como se muestra en la imagen, todo nuestro código gira en torno al main, que es el componente central encargado de comunicar los métodos entre sí y transmitir los resultados de los demás. A continuación, describiremos los módulos principales y su interconexión:

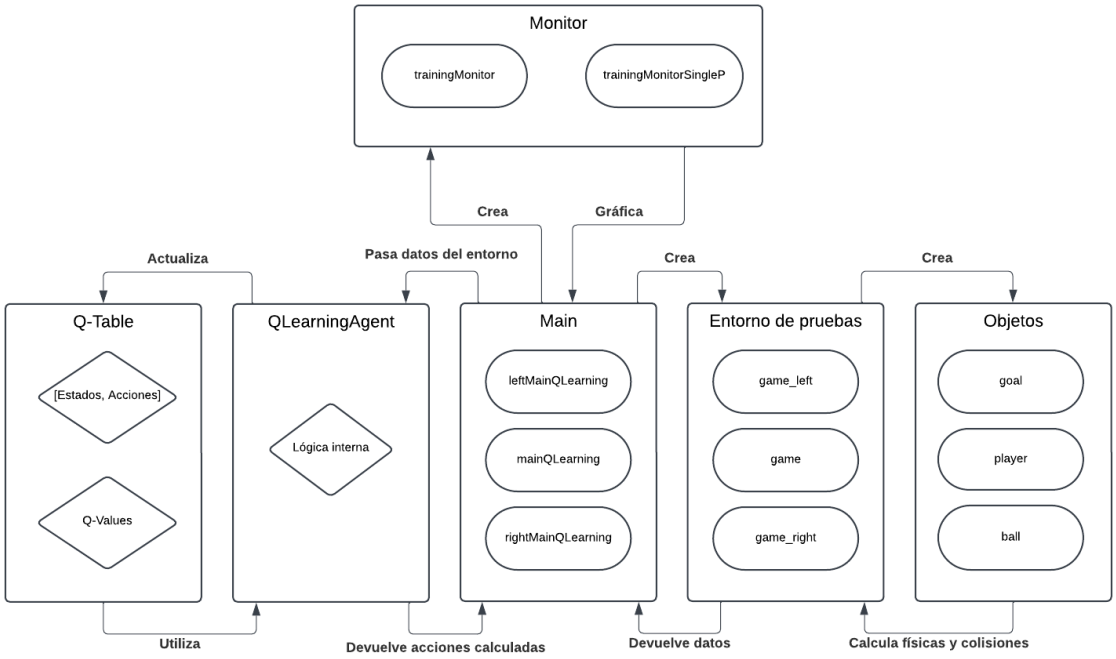

Figura 6.1: Ilustración de la estructura del código

- **Main:**

Tenemos tres archivos principales para el main, cada uno con una función específica:

  - Main para el Jugador Izquierdo: Este archivo se encarga de llamar a los métodos que crearán una instancia de todo lo necesario para el jugador izquierdo.
  - Main para el Jugador Derecho: Similar al anterior, pero centrado en el jugador derecho.
  - Main para Ambos Jugadores: Este archivo se utiliza en la fase final de entrenamiento, donde se experimenta con un partido completo, involucrando a ambos jugadores.

- **Clases Game (Entorno de pruebas):**

También contamos con tres clases game, una para cada jugador y una para ambos:

  - Game para el Jugador Izquierdo: Esta clase crea el entorno de pruebas para el jugador izquierdo, manejando todos los cálculos y reseteos del nivel. Además, se encarga de añadir los objetos al campo, tales como porterías, jugadores y la pelota.
  - Game para el Jugador Derecho: Funciona de manera similar, pero centrada en el entorno del jugador derecho.
  - Game para Ambos Jugadores: Utilizada para el entorno compartido en la fase de entrenamiento final.

- **Clases de Objetos:**

Los objetos en el campo se dividen en dos tipos:

  - Obstacle y Goal: Estas clases solo se encargan de generar un tipo de objeto que luego se dará forma y colisiones en la clase game.
  - Player y Ball: Estas clases tienen físicas y mecánicas más complejas. Se encargan de generar objetos con las características deseadas y de manejar todas las físicas, controles y colisiones de estos con el resto de elementos del entorno.

- **QLearningAgent:**

En este módulo es donde tenemos toda la lógica del entrenamiento de nuestros agentes, podemos dividir su funcionalidad en 3:

  - **Carga y guardado de la Q-Table:** Aquí se guarda y carga la Q-Table el entrenamiento del agente.
  - **Cálculo de jugadas:** Utilizando los datos obtenidos en las clases game y proporcionados por los main, se calcula la mejor jugada posible.
  - **Comunicación con Main:** Una vez calculada la jugada, se pasa al main para que este se la comunique a la clase player y se efectúe el movimiento.

- **Q-Table:**

Este módulo, aunque lo hemos separado en la explicación, forma parte del código de QLearningAgent, principalmente se encarga de:

  - **Gestión de la Q-Table:** La tabla se implementa como un diccionario, donde cada entrada corresponde a una tupla [estado, acción] y a un valor Q asociado. Esta estructura permite al agente recordar la utilidad estimada de cada acción en diferentes situaciones.
  - **Actualización del Q-Value**: A medida que el agente interactúa con el entorno, los Q-Value se actualizan en función de las recompensas obtenidas, permitiendo ajustar el comportamiento del agente en base a estos valores.

- **Clases TrainingMonitor:**

Estas clases se encargan de la visualización mediante una gráfica de los resultados del entrenamiento, contamos con dos tipos de TrainingMonitor:

  - **Entrenamiento Individual:** Una clase trainingMonitor se utiliza cuando los agentes se entrenan de forma individual.
  - **Entrenamiento Conjunto:** Otra clase trainingMonitor se utiliza para el entrenamiento conjunto de los agentes.

6.1.2 Componentes esenciales

En esta sección, desglosaremos los componentes esenciales que forman la columna vertebral de nuestro proyecto, destacando sus funciones y cómo se interconectan para formar un sistema cohesivo y robusto.

- **MainQLearning**

  El main, como hemos comentado en puntos anteriores, es el núcleo del proyecto, puesto que se encarga de inicializar, ejecutar, gestionar y comunicar todos los componentes. Se puede dividir en 3 grandes funciones:

  - **Método constructor**

    Esta sección del código es responsable de inicializar los atributos de la instancia de la clase, encargándose así de establecer el entorno inicial y las variables que se utilizarán durante el juego y el entrenamiento de los agentes, tales como los resets, las puntuaciones, entidades creadas...

  - **Bucle del entrenamiento del agente (train_ai)**

    El método train_ai ejecuta una iteración de entrenamiento para los agentes izquierdo y derecho en el entorno de fútbol. Este método se puede desglosar en las siguientes etapas:

    - **Estado y Acción de los Agentes**:

      - Se obtiene el estado actual de los jugadores izquierdo y derecho utilizando la posición de la pelota y otros parámetros del entorno.
      - Ambos agentes deciden una acción basada en su política entrenada y estas acciones se aplican a sus respectivos jugadores.

    - **Recompensas de los Agentes:**

      - Se calcula la recompensa para cada agente en función de la acción tomada y el estado resultante.
      - Estas experiencias (estado actual, acción, recompensa, siguiente estado) se utilizan para recalcular y actualizar el nuevo Q-value.

    - **Guardado Periódico:**

      - Los modelos de los agentes se guardan periódicamente si ha pasado un intervalo de tiempo predefinido desde el último guardado.

    - **Reseteo:**

      - Si los agentes han alcanzado un estado de reseteo (si se ha marcado un gol o se ha alcanzado el límite de acciones), se actualiza el valor de épsilon, fomentando cada vez más una menor exploración.
      - Se actualizan y guardan periódicamente los gráficos de rendimiento, mostrando las recompensas promedio y totales para cada agente.

Es importante mencionar que muchos de los métodos nombrados en este proceso, como get_state, act, input, remember y save, están definidos en sus respectivas clases (por ejemplo, en las clases de los agentes y en la clase Game). Aquí, el main solo realiza las llamadas a estos métodos, pero los procesos reales se llevan a cabo en las otras clases.

- **Test_agents**

  El módulo test_agents se encarga de evaluar el rendimiento de los agentes una vez finalizado su entrenamiento. Para ello, replica la estructura de train_agents, pero con algunas diferencias.

  En primer lugar, el valor de épsilon se fija en 0, lo que obliga al agente a explotar siempre la mejor acción posible según su Q-Table, eliminando completamente la exploración aleatoria. Además, durante esta fase no se actualiza la Q-Table, evitando así cualquier sobreentrenamiento o modificación de los valores ya aprendidos.

  Finalmente, se imprime por pantalla el porcentaje de goles marcados respecto al número total de episodios, proporcionando una medida clara y directa del rendimiento del agente.

- **Bloque principal**

  Por último, el bloque principal (__main__) se encarga de inicializar el flujo del programa. Primero, se configura la interfaz gráfica de la ventana del juego mediante pygame. Luego, se crean las instancias de los agentes izquierdo y derecho (agentL y agentR), y se inicializa el monitor de entrenamiento. Si existen modelos previamente entrenados, estos se cargan en los agentes.

  Finalmente, se inicia el bucle de entrenamiento del juego llamando al método loop de la clase Game, donde se gestionan las interacciones entre los agentes, el entorno y el proceso de entrenamiento. Este bloque principal organiza la ejecución de todo el sistema, asegurando que los agentes entrenen y se guarden los resultados periódicamente.

- **QLearningAgent**

La clase QLearningAgent es la encargada de implementar las funciones necesarias para el pleno funcionamiento del agente. Este agente tendrá como objetivo aprender a jugar al fútbol en un entorno simulado mediante el manejo de una Q-Table, para ello implementaremos los siguientes métodos.

- **Constructor:**

  Al igual que en el main, el constructor de la clase QLearningAgent inicializa las variables esenciales para el debido funcionamiento de la clase, algunos de estos parámetros incluyen la propia Q-Table, que se inicializa como un diccionario vacío o la cantidad de las acciones posibles (action_size).

- **get_q_value:**

Este método permite acceder al valor Q de una pareja estado-acción. Si la tupla (estado, acción) no se encuentra previamente en la Q-Table, se inicializa automáticamente con un valor de 0.0. Esto asegura que cada vez que el agente se enfrente a una nueva situación, pueda empezar a construir su conocimiento desde cero sin errores de acceso, y permite una exploración continua del entorno.

Este método, será utilizado por el método que vamos a explicar a continuación(act)

- **act**

El método act es el encargado de decidir qué acción tomará el agente en un estado determinado, utilizando una estrategia ε-greedy.

Para ello, primero se evalúa si el agente debe explorar o explotar. Esto se decide generando un número aleatorio entre 0 y 1 y comparándolo con epsilon, que representa la probabilidad de exploración. Si el valor es menor o igual que epsilon, se selecciona una acción aleatoria.

Este enfoque favorece la exploración durante las primeras etapas del entrenamiento, permitiendo al agente probar diferentes acciones y aprender del entorno.

Por el contrario, si el valor aleatorio es mayor que epsilon, el agente elige la mejor acción conocida hasta el momento: se calculan los Q-values de todas las acciones posibles en el estado actual, mediante get_q_value, y se selecciona aquella con el mayor valor.

Este mecanismo, permite al agente explotar lo aprendido en su tabla Q, eligiendo acciones que le proporcionen una mayor recompensa esperada según su experiencia previa.

❖ **Ejemplo de Flujo**

Supongamos que el agente está en un estado particular y epsilon es 0.1.

- **Exploración (10 % del tiempo):** Si np.random.rand() devuelve un valor menor o igual a 0.1, el agente tomará una acción aleatoria.

- **Explotación (90% del tiempo):** Si el valor generado es mayor a 0.1, el agente utilizará su Q-Table para obtener los valores de las acciones y seleccionar la mejor acción.

- **update_q_table:**

El método update_q_table se encarga de actualizar los valores de la Q-Table del agente tras cada interacción con el entorno, siguiendo la fórmula clásica de Q-Learning con algunas optimizaciones.

Primero, se calcula la **tasa de aprendizaje** (learning rate). En lugar de mantener un valor fijo (alpha), se ha ajustado con un término logarítmico que depende del número de episodios. Gracias a esto, a medida que el agente acumule experiencia al explorar, el *learning rate* se incrementará ligeramente, mejorando el aprendizaje en fases más avanzadas sin afectar al entrenamiento.

Luego, se obtiene el valor Q actual correspondiente a la tupla (state, action) desde la tabla. Si no existe, se inicializa automáticamente a 0:

A continuación, se calcula el **máximo valor Q del siguiente estado** (next_state), teniendo en cuenta todas las acciones posibles. Si no hay datos para una acción en ese estado, se utiliza un valor inicial de 1.0, lo que ayuda a incentivar la exploración:

Con estos datos, se aplica la **ecuación de actualización de Q-Learning**, que corrige el valor anterior (old_q) en función de la recompensa recibida y del valor estimado del siguiente estado. El resultado se guarda en la Q-Table:

- **decay_epsilon:**

Este método se encarga de reducir gradualmente el valor de épsilon después de cada episodio, permitiendo al agente explorar menos con el tiempo y centrarse en explotar lo aprendido. El decaimiento se realiza de forma controlada para que el agente no deje de explorar demasiado pronto, utilizando epsilon_decay para ajustar la tasa de reducción.

A su vez, para evitar que el agente deje de explorar por completo, se establece un valor mínimo de épsilon, min_epsilon. Gracias a este valor, incluso cuando el agente ha reducido significativamente la tasa de exploración aún puede realizar una pequeña cantidad de exploración, lo que le permite seguir ajustando sus decisiones de forma gradual mientras perfila sus acciones. Esto asegura que el agente mantenga algo de exploración para evitar caer en un comportamiento subóptimo, incluso en las etapas finales de entrenamiento.

- **log_episode_result:**

Este método gestiona el registro de resultados al finalizar un episodio. Durante el transcurso del episodio, va guardando el valor del total de recompensas obtenidas por el agente y si dicho episodio es terminal, indica que ha de ocurrir un reseteo del entorno.

- **Métodos de guardado y carga.**

Por último, qLearningAgent cuenta con un par de métodos encargados de guardar y cargar la Q-Table, lo que facilita la persistencia del estado del agente y la continuidad del entrenamiento.

Estos métodos son:

- **save**: Guarda la Q-Table del agente junto con los parámetros de entrenamiento (como epsilon y el contador de episodios) en archivos separados. Esto permite preservar el estado del agente y reanudar el entrenamiento en cualquier momento.

- **load**: Carga la Q-Table y los parámetros guardados previamente. De este modo, el agente puede continuar aprendiendo desde donde lo dejó, manteniendo la coherencia en su proceso de entrenamiento.

Gracias a estos métodos aseguramos que el agente pueda persistir y recuperar su estado de aprendizaje, permitiendo entrenar en múltiples sesiones sin pérdida de información.

- **Player**

Lo más destacable de esta clase es cómo están diseñados los controles para que los utilice el agente, cómo se manejan las colisiones y el método get_state que proporciona al agente toda la información relevante del entorno y del propio jugador.

- **Inputs**

Para que el agente elija una acción, tenemos una lista llamada actions con las 4 posibles acciones [“UP”, “DOWN”, “LEFT”, “RIGHT”]

Cuando el agente decide qué acción tomar, selecciona un número del 1 al 4 correspondiente a cada acción. Este número se pasa al método input, que a su vez lo comunica al método move.

Este último ajustará la velocidad del jugador, permitiéndole desplazarse en la dirección deseada con una leve aceleración.

- **Colisiones**

En cuanto a las colisiones, desde la clase Game obtenemos todos los objetos con los que el jugador puede entrar en contacto, como otros jugadores, y los añadimos a una lista llamada collision_sprites_with_players. A partir de aquí, simplemente realizamos comprobaciones para cada uno de los bordes del jugador, con el objetivo de saber si están en contacto con alguno de los elementos de la lista. Si alguno lo está, evitamos que el jugador se mueva más allá de ese objeto, impidiendo así que lo atraviese.

Además, también manejamos la llamada de algunas recompensas, como penalizar al jugador por entrar en contacto con un obstáculo.

- **Estado**

Por último, el método get_state, proporciona al agente toda la información importante relacionada con el entorno. Los elementos que se le pasan al agente son:

- **Posición relativa de la pelota (X, Y)**: Indica dónde está la pelota respecto al agente, en una cuadrícula 3x3 (valores -1, 0, 1).

- **Dirección de la portería rival**: Según el equipo del agente, se representa como un valor fijo que orienta su comportamiento ofensivo.

- **Alineación al disparo**: Evalúa si el agente está bien orientado para chutar hacia portería (valores -1, 0, 1).

- **Distancia a la pelota**: Se clasifica en tres niveles (cerca, media o lejos) para determinar la urgencia de acercarse a ella.

- **Proximidad de la pelota a la portería rival**: Un valor binario que indica si la pelota está en una zona peligrosa para el rival.

Este método actúa como la visión del agente permitiendo que tenga un noción completa y general, lo que es esencial para tomar decisiones informadas y mejorar su rendimiento en el juego.

A su vez, al haber generalizado los elementos del estado, solo hay un total de 324 estados que el agente deba aprender

- **Ball**

En esencia, la clase Ball se compone principalmente del método de colisiones, que es casi idéntico al de la clase de los jugadores. La principal diferencia radica en la implementación de los rebotes y la colisión con las porterías.

- **Colisiones**

El método de colisiones sigue la misma lógica que el de los jugadores. Sin embargo, en lugar de simplemente impedir que la pelota atraviese un obstáculo, hemos implementado una lógica para que la pelota cambie de trayectoria invirtiendo su dirección, simulando así un rebote.

- **Rebotes**

  Cuando la pelota colisiona con un obstáculo o una pared, su dirección se invierte para simular un rebote. La lógica es similar a la utilizada para evitar que un jugador atraviese un obstáculo, pero en lugar de detener el movimiento, se invierte la velocidad de la pelota.

- **Colisión con Jugadores**

  Para darle mayor realismo, cuando la pelota colisiona con un jugador, no solo se invierte su dirección, sino que también adopta la velocidad del jugador multiplicada por 1.25. Esto simula el efecto de que el jugador está "chutando" la pelota.

- **Colisión con las Porterías**

  Por último, hemos implementado que, si la pelota entra en contacto con una portería, se lo comunique a game y este resetee el nivel y actualice los parámetros necesarios, tales como las puntuaciones.

- **Game**

La clase Game se encarga de gestionar el entorno de juego y se puede dividir en tres funcionalidades principales:

- **Inicialización de las entidades**

  En esta parte, se inicializan todas las entidades necesarias para el partido, es decir, jugadores, pelota, portería y obstáculos. Esto asegura que todos los elementos del entorno están configurados y listos para interactuar en el juego.

- **Gestión de las recompensas**

  En esta parte, se definen una serie de métodos que se encargan de recompensar o penalizar las acciones de los agentes, y luego transmitir estas recompensas al modelo para realizar los cálculos. Estos métodos pueden ser llamados de dos formas: periódicamente si son pasivos, como cuando la pelota está en el campo rival, o mediante colisiones, como cuando un agente marca un gol.

Las recompensas que hemos elegido dar a nuestro agente son las siguientes:

- **Goal Scored**

  Esta recompensa da +40 al agente que marque un gol, incentivando fuertemente el comportamiento de anotar y -40 al que han marcado.

- **Directional Reward**

  Esta es una de las recompensas más relevantes, y a la vez más sencillas, se basa en la posición relativa de la pelota respecto al agente. Esta estrategia permitió reducir considerablemente el espacio de estados: en lugar de proporcionar coordenadas exactas de la pelota, se pasan al agente dos direcciones (una para el eje X y otra para el eje Y), codificadas como -1, 0 o 1. Por ejemplo, si la pelota se encuentra arriba a la derecha del agente, este recibirá la señal [1, -1]. Si se mueve hacia alguna de esas direcciones, obtiene una recompensa de +1.5 por cada dirección acertada, y un adicional de +1 si acierta ambas, fomentando así el movimiento diagonal.

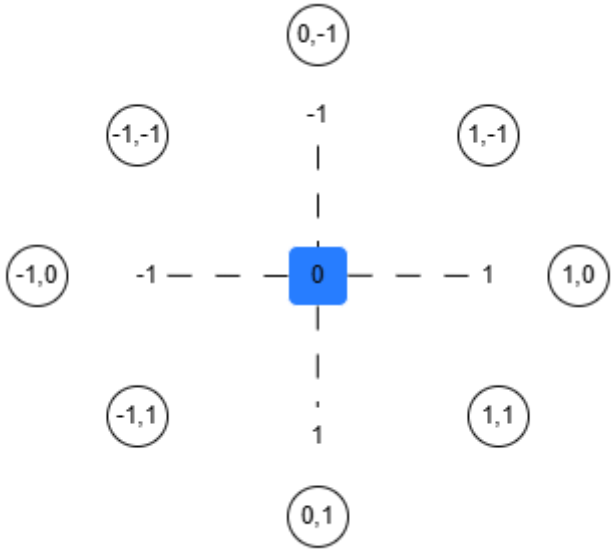

Figura 6.2: Diagrama de las posiciones de la pelota

❖ **Alignment Reward**

Una vez implementada la recompensa que permitía al agente perseguir la pelota, se añadieron recompensas adicionales orientadas al posicionamiento estratégico. En concreto Alignment Reward se encarga de premiar al agente respecto a la alineación con la pelota+2 si el agente estaba alineado con la pelota y la portería, +0.3 si estaba aproximadamente alineado y -1 si se encontraba bloqueando la trayectoria entre ambos.

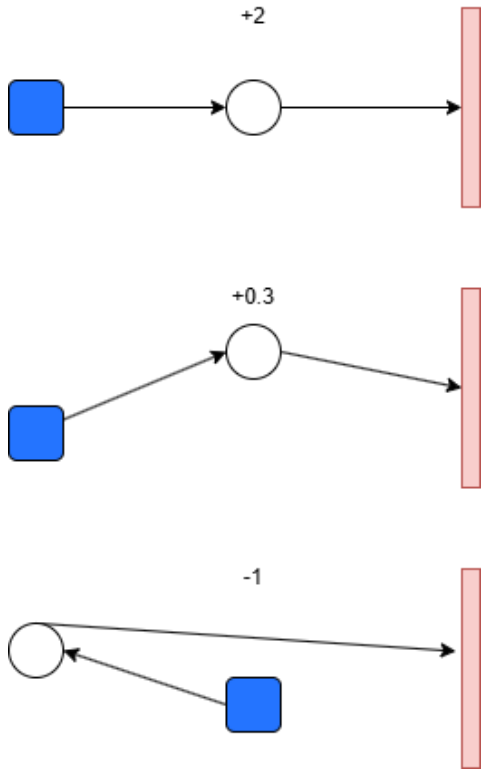

Figura 6.3: Diagrama recompensa por alineación

❖ **Encoraugment Reward**

A modo de complementar la recompensa anterior y para fomentar el comportamiento ofensivo del agente, añadimos una recompensa que se encargaba de recompensar al agente con +1.5 cuando acercaba la pelota a la portería rival.

❖ **Shot Reward**

Finalmente, se añadió una recompensa muy similar a la anterior, que se activa únicamente cuando el agente se encuentra en una posición cercana a la portería rival y alineado correctamente con la pelota. En estas situaciones, si el agente consigue empujar la pelota hacia la portería, se le premia con +1.5. Aunque su implementación es parecida a la recompensa de alineación, esta está pensada para reforzar aún más el comportamiento ofensivo en contextos con alta probabilidad de gol.

- **Bucle principal**

Por último, está el bucle principal, que se ejecuta continuamente para mantener el juego en funcionamiento.

Este método se encarga tanto de dibujar y refrescar el entorno para conseguir una animación fluida, como de actualizar los parámetros del entorno y llamar al método train_ai con la situación actual de los agentes.

Además, también se encarga de llamar en cada iteración de un partido a las recompensas que comprueban si se está cumpliendo las condiciones de las recompensas.

6.2 Entrenamiento del Agente

6.2.1 Proceso de Entrenamiento

Tal y como indicamos antes en la metodología, utilizamos un enfoque de desarrollo iterativo e incremental. Realizamos distintos entrenamientos en base a la dificultad del agente y regulamos los parámetros o hacemos cambios según sea necesario. Nuestro objetivo era obtener una gran cantidad de resultados y, si los considerábamos aceptables, pasábamos a la siguiente etapa más complicada. Si, por el contrario, los resultados no eran aptos, adaptábamos la etapa disminuyendo un poco la dificultad.

El proceso de entrenamiento se puede dividir en cuatro etapas:

- **Discretización del estado y movilidad hacia la pelota:**

En esta etapa inicial, entrenamos un solo agente en un entorno sin porterías y con solo la recompensa de guiarlo hacia la pelota. Esta primera etapa fue la que nos permitió reducir el número de estados a un número manejable y sentar las bases de como serían el resto de las fases.

- **Implementación marcar gol:**

En esta fase, añadimos las porterías al entorno e introducimos las recompensas y elementos del estado encargados de guiar al agente a marcar gol en la portería enemiga.

- **Refinamiento marcar gol:**

En esta etapa nos centramos en refinar los elementos relacionados a que el agente anotase gol.

- **Prueba con ambos agentes:**

Finalmente, implementamos a ambos agentes ya pre entrenados para observar su comportamiento y realizar ajustes adicionales

6.2.2 Desarrollo del entrenamiento

- **Etapa 1: Discretización del estado y movilidad hacia la pelota:**

En esta primera etapa, nos enfrentamos a una de las principales dificultades del proyecto: el tamaño del espacio de estados. Inicialmente, el estado del agente incluía la posición exacta de la pelota (por ejemplo, [144, 256]), lo que generaba un número de combinaciones posibles inmanejable para un entrenamiento eficiente.

- **Primeros intentos de discretización del entorno**:

Para abordar este problema, probamos a discretizar la posición de la pelota utilizando un *grid* sobre el campo completo, con el objetivo de reducir el tamaño del estado. Esta estrategia, sin embargo, trajo consigo dos problemas:

- Si las *grids* eran muy grandes, el estado se reducía, pero el agente perdía mucha precisión.
- Si las *grids* eran más pequeñas (y, por tanto, más precisas), el espacio de estados seguía siendo demasiado amplio para ser entrenado eficientemente.

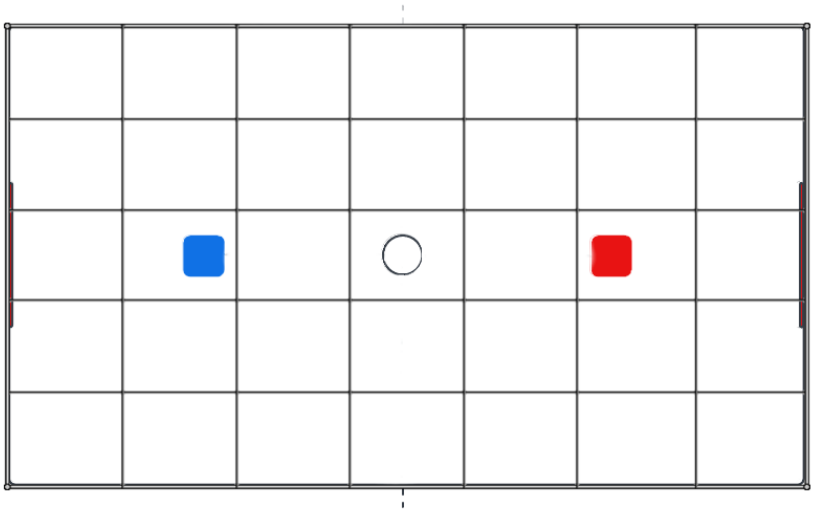

Figura 6.4: Diagrama discretización del entorno con grids

Para solucionarlos, se evaluó la posibilidad de reducir drásticamente el tamaño del entorno (hasta dimensiones de 100x80 píxeles) para facilitar el entrenamiento. Sin embargo, esto comprometía la escalabilidad y modularidad del entorno, por lo que se descartó.

- **Solución optada**:

  Tras varias pruebas, dimos con una solución más efectiva: discretizar un entorno alrededor del agente en un *grid* de 9x9. De esta forma, el agente ya no "veía" directamente la posición de la pelota, sino que percibía su dirección relativa, como si estuviera siendo guiado hacia la dirección en la que se encuentra.

  Este enfoque redujo el número de posibles combinaciones de estados de cientos de miles a solo 81, haciendo el aprendizaje mucho más eficiente sin sacrificar la calidad del comportamiento.

  Junto con este cambio, se añadió la recompensa calculate_directional_reward, que premia al agente por desplazarse en las direcciones adecuadas hacia la pelota, reforzando su orientación en el entorno local.

- **Etapa 2: Implementación marcar gol**

Una vez validado que el agente era capaz de desplazarse correctamente por el entorno y alcanzar la pelota, nos centramos en el siguiente objetivo: lograr que el agente empujara la pelota hacia la portería rival, rodeándola si era necesario, y evitando marcar en propia puerta.

Para ello, continuamos utilizando un enfoque basado en la discretización del estado, lo que nos permitía reducir el número total de combinaciones posibles y, al mismo tiempo, mantener la generalización del aprendizaje.

- **Ampliación del estado del agente:**

  En esta fase, incorporamos nueva información al estado del agente con el fin de que pudiera tomar mejores decisiones en situaciones complejas cercanas al área de gol. Las nuevas variables añadidas al estado fueron las siguientes:

  - Dirección de la portería del rival: [-1, 1]
  - Alineación al disparo: [-1, 0, 1]
  - Distancia a la pelota: [0, 1, 2]
  - Proximidad de la pelota a la portería del rival: [0, 1]

  Con estas variables adicionales, el estado del agente pasó de contar con 81 combinaciones posibles a un total de **324 estados distintos** (3x3x2x3x3x2 = 324). Aunque esto supuso un aumento relativo en la cantidad de estados, también significó una mejora considerable en la riqueza del entorno percibido por el agente.

- **Optimización del entorno de entrenamiento**:

  El incremento en la complejidad del entorno exigía un mayor número de episodios para que el agente pudiera aprender de manera efectiva. Dado que el entrenamiento inicial solo alcanzaba unos pocos miles de episodios al día, decidimos optimizar la velocidad de ejecución del entorno.

Para ello:

- ❖ Se desactivó la visualización gráfica en tiempo real, permitiendo así que el entorno funcionara sin ventana.
- ❖ Se desactivaron los mensajes por consola, que representaban un cuello de botella cuando el entorno operaba a alta velocidad.
- ❖ La gráfica de rendimiento solo se actualizaba cada 100 episodios, reduciendo así la sobrecarga.

Gracias a estas optimizaciones, pasamos de ejecutar unos **10.000 episodios por día** a lograr una velocidad de **100 episodios cada 6 segundos**, alcanzando un total de **8.640.000 episodios diarios**, lo que representó una mejora del **86.300% en el rendimiento** del entrenamiento.

- o **Recompensas enfocadas en el gol:**

Con el nuevo estado ya implementado, añadimos también nuevas recompensas que guiarían al agente hacia el objetivo de marcar gol en la portería rival. Las funciones principales que definieron este comportamiento fueron:

- ❖ **goal_scored**
- ❖ **alignment_reward**
- ❖ **shot_reward**
- ❖ **encouragement_reward**

Estas recompensas complementaban la del movimiento hacía la pelota, permitiendo al agente un aprendizaje continuo que fomentaba tanto dirigirse a la pelota, como recolocarse respecto de esta para poder empujarla bien hacía la portería.

Estos cambios sentaron las bases para el entrenamiento intensivo posterior.

- **Etapa 3: Refinamiento marcar gol**

En esta etapa, el objetivo principal era mejorar la precisión del agente a la hora de marcar goles, centrándonos en afinar su comportamiento en las proximidades de la portería. Para ello, experimentamos con diversas estrategias que buscaban dotar al agente de una mayor capacidad de alineación y control sobre la pelota.

Probamos múltiples enfoques, desde el cálculo del ángulo entre el agente, la pelota y la portería, hasta el diseño de recompensas que incentivaran al agente a alinear correctamente la pelota con la portería antes de disparar. Sin embargo, estos métodos resultaron ser poco eficaces. Debido a que los controles del agente habían sido diseñados de forma intencionadamente tosca para aportar realismo, guiar con precisión la pelota resultaba especialmente difícil.

Además, muchos de estos intentos no solo no mejoraron la tasa de éxito al marcar, sino que en algunos casos incluso la empeoraron, añadiendo a su vez una complejidad innecesaria al estado del agente, lo que aumentaba significativamente el espacio de búsqueda y ralentizaba el aprendizaje.

Para mantener la simplicidad y eficacia del proyecto, decidimos buscar inspiración en uno de los videojuegos clásicos que sirvieron de base conceptual: PONG. En este juego, toda la pared actúa como portería, lo que reduce drásticamente la necesidad de alineación precisa.

Aplicamos este mismo principio a nuestro entorno: en lugar de una portería tradicional, ahora toda la pared del fondo funcionaba como portería. Esta modificación resolvió uno de los principales problemas detectados, ya que anteriormente el agente tenía dificultades para marcar cuando la pelota se acercaba a los bordes del campo, lo que requería maniobras complejas de centrado.

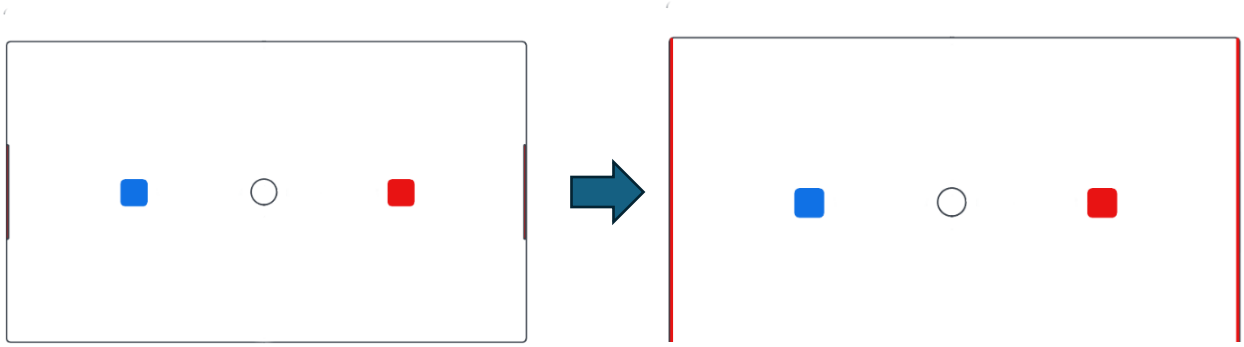

Figura 6.6: Aumento del tamaño de las porterías

Esta nueva configuración fue adoptada como estándar en el entorno a partir de esta etapa.

- **Etapa 4: Prueba con ambos agentes**

En esta última fase del proyecto, el objetivo fue **poner a prueba el sistema completo** enfrentando a **dos agentes entrenados de forma independiente**. Para ello, cargamos los modelos pre entrenados de ambos agentes y reactivamos la **visualización en tiempo real** del entorno, con el fin de observar su comportamiento de manera directa durante las partidas.

Procedimos a ejecutar unos **cientos de episodios de prueba**, analizando cuidadosamente la interacción entre ambos jugadores. Durante esta fase, realizamos varios **reentrenamientos individuales**, afinando por separado a cada agente para obtener su mejor versión.

6.3 Problemas y Dificultades Encontradas

A lo largo del desarrollo del proyecto, nos enfrentamos a diversos problemas tanto a nivel técnico como durante el entrenamiento de los agentes. En esta sección, se presentan los desafíos más significativos, clasificados por su naturaleza.

6.3.1 Desafíos Técnicos

- **Q-Learning y espacio de estados**

Al principio, entrenábamos con Q-Learning, pasándole valores continuos como las coordenadas exactas de la pelota, lo que generaba una cantidad inmanejable de combinaciones posibles. Esto provocaba dos grandes problemas:

  - **Tamaño excesivo de la tabla**: El número de estados crecía exponencialmente, haciendo que la tabla ocupara varios gigabytes con solo unas pocas horas de entrenamiento.

  - **Duración del entrenamiento**: La gran cantidad de estados hacía que el agente necesitara millones de episodios para cubrir una fracción razonable del entorno.

Para solucionar esto, aplicamos un sistema de discretización que permitió reducir de forma drástica el espacio de estados. Primero intentamos discretizar la posición global de la pelota, pero aún era insuficiente. Finalmente, adoptamos una discretización relativa alrededor del agente (un grid de 9x9), junto con una representación compacta de variables clave como alineación o dirección a portería. Gracias a esto, conseguimos reducir el número total de estados a tan solo unos pocos cientos, haciendo viable el uso de Q-Learning sin sacrificar rendimiento ni generalización.

- **Problemas con el Equipo**

A lo largo del entrenamiento de los agentes, nos enfrentamos a un problema externo al proyecto que supuso grandes dificultades: el equipo empezó a estropearse, llegando incluso a tener que formatearlo. Aunque teníamos una copia de seguridad del proyecto, descubrimos que si alguno de los fallos que provocaban que el equipo se reiniciase ocurría mientras se guardaba la red neuronal, estos archivos se podían corromper, perdiendo el entrenamiento de varios días e incluso semanas.

Para evitar esto, decidimos tener siempre una copia guardada de la red neuronal creada justo antes del entrenamiento, evitando así posibles pérdidas innecesarias.

6.3.2 Dificultades en el Entrenamiento

- **Velocidad de Simulación y episodios por día:**

  Uno de los mayores cuellos de botella al inicio del proyecto fue la baja cantidad de episodios que el entorno era capaz de procesar por día. Con la visualización activada y múltiples print() en consola, el entorno tardaba varios segundos por episodio, haciendo inviable un entrenamiento eficiente.

  Para solucionarlo:

  - Se desactivó completamente la visualización gráfica durante el entrenamiento.
  - Se eliminaron todos los mensajes de consola innecesarios.
  - Se optimizó el bucle principal del juego.

- **Porcentaje de goles marcados**

  Una vez que el agente ya era capaz de empujar la pelota, se buscó mejorar su precisión al marcar goles. Se probaron múltiples estrategias, como:

  - Cálculo de ángulos entre agente, pelota y portería.
  - Recompensas por alinear la pelota con la portería.

  Sin embargo, ninguna de estas opciones ofrecía mejoras estables. Por el contrario, muchas veces empeoraban el comportamiento del agente o aumentaban innecesariamente el número de estados.

  La solución fue inspirarse en el juego PONG y rediseñar la portería: en lugar de ser una zona limitada, toda la pared rival se consideró como portería. Esto simplificó la tarea del agente y eliminó la necesidad de precisión extrema, lo que resultó en una mejora inmediata del rendimiento sin incrementar la complejidad del estado.

A pesar de todos estos desafíos, cada dificultad nos obligó a replantear aspectos clave del diseño, permitiéndonos optimizar el entorno, aprender nuevas técnicas y construir un sistema mucho más robusto y eficiente. Estos problemas no han sido un tiempo perdido, sino experiencias que han enriquecido nuestro conocimiento y han dado lugar a un mejor proyecto.

7. Pruebas y resultados

En este capítulo presentaremos las distintas pruebas realizadas a lo largo del entrenamiento de los agentes, así como los resultados obtenidos en cada fase. También se incluye una evaluación adicional en la que los agentes se enfrentaron a jugadores humanos, seguida de una tabla resumen con los principales indicadores cuantitativos del proyecto.

7.1 Pruebas realizadas durante el entrenamiento

A lo largo de las distintas etapas del desarrollo, realizamos pruebas específicas con el objetivo de validar el comportamiento aprendido por los agentes. Estas pruebas fueron fundamentales para ajustar el diseño del entorno, la estructura del estado y las funciones de recompensa. A continuación, se describen las pruebas más relevantes junto con los resultados obtenidos.

Etapa 1 – Movimiento hacia la pelota

En esta etapa inicial, el objetivo era simplemente verificar que el agente aprendía a desplazarse hacia la pelota. Dado que se trataba de una tarea sencilla y con un estado muy reducido, optamos por evaluarla de forma cualitativa, observando directamente el comportamiento del agente en el entorno.

- **Objetivo**: Comprobar que el agente se movía de manera coherente hacia la pelota.
- **Resultado**: El comportamiento observado fue correcto en la gran mayoría de los casos, por lo que se consideró superada esta etapa.
- **Evaluación**: En esta etapa, no se generamos métricas cuantitativas ni gráficas puesto que se trataba de una tarea fácil de verificar visualmente y difícil de cuantificar con precisión.

Etapa 2 – Empujar balón a portería

Con una representación del estado más rica y recompensas centradas en marcar, el agente fue entrenado un total de 100.000 episodios.

- **Objetivo**: Lograr que el agente empujara la pelota hacia la portería rival evitando marcar en propia puerta.
- **Resultado**: Al realizar unas 100 ejecuciones del entorno, el agente alcanzó una tasa de goles cercana al 60% en una media de 500 acciones, es decir, la mitad del tiempo límite por episodio, mostrando conductas ofensivas coherentes.

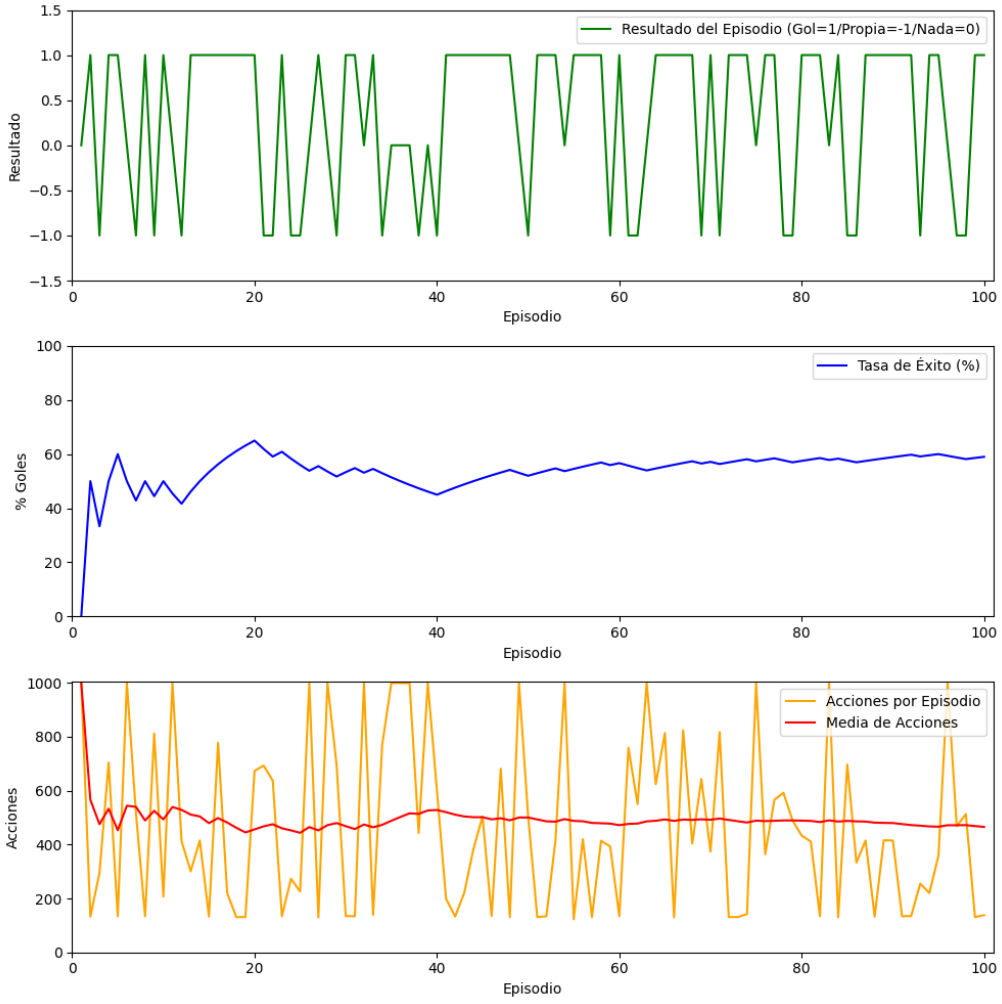

Figura 7.1: Gráfica de recompensas con porterías pequeñas

Etapa 3 – Simplificación de la portería

Al modificar la portería para abarcar toda la pared del fondo, reiniciamos el agente y volvimos a entrenar otros 100.000 episodios.

- **Objetivo**: Mejorar la tasa de goles sin añadir complejidad al estado.
- **Resultado**: La tasa de goles se incrementó hasta aproximadamente un 90%, y se redujo las acciones necesarias para anotar gol a la mitad.

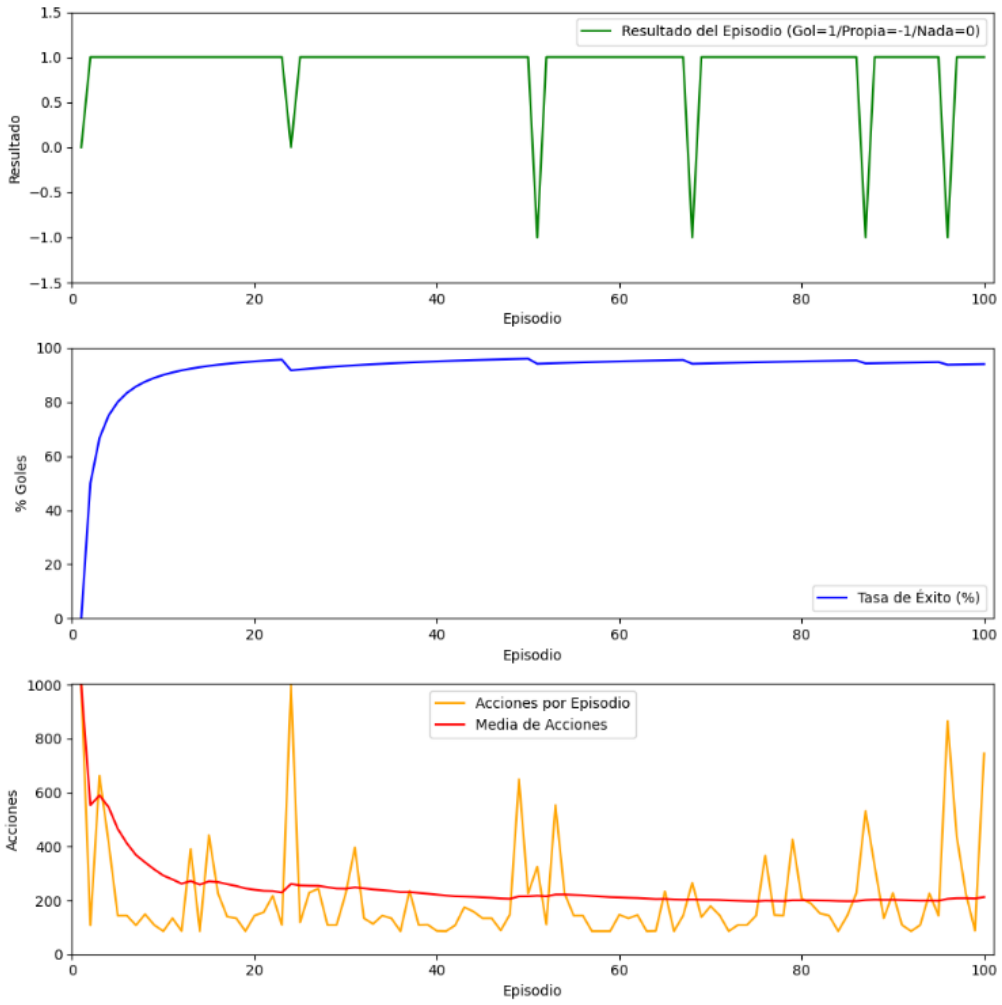

Figura 7.2: Gráfica de recompensas tras el cambio

Etapa 4 – Enfrentamiento entre agentes

Por último, juntamos agentes previamente entrenados por separados y realizamos varias ejecuciones del entorno con una duración de 100 episodios. Al final de cada ejecución, realizamos ajustes en los agentes y los volvimos a reentrenarlos hasta que obtuvimos **el mejor modelo de cada uno**, es decir, aquellos agentes que mostraban el comportamiento más coherente y competitivo frente al otro.

- **Resultado**: Ambos lograron tasas de goles entre el 45% y el 55%, indicando un equilibrio competitivo.

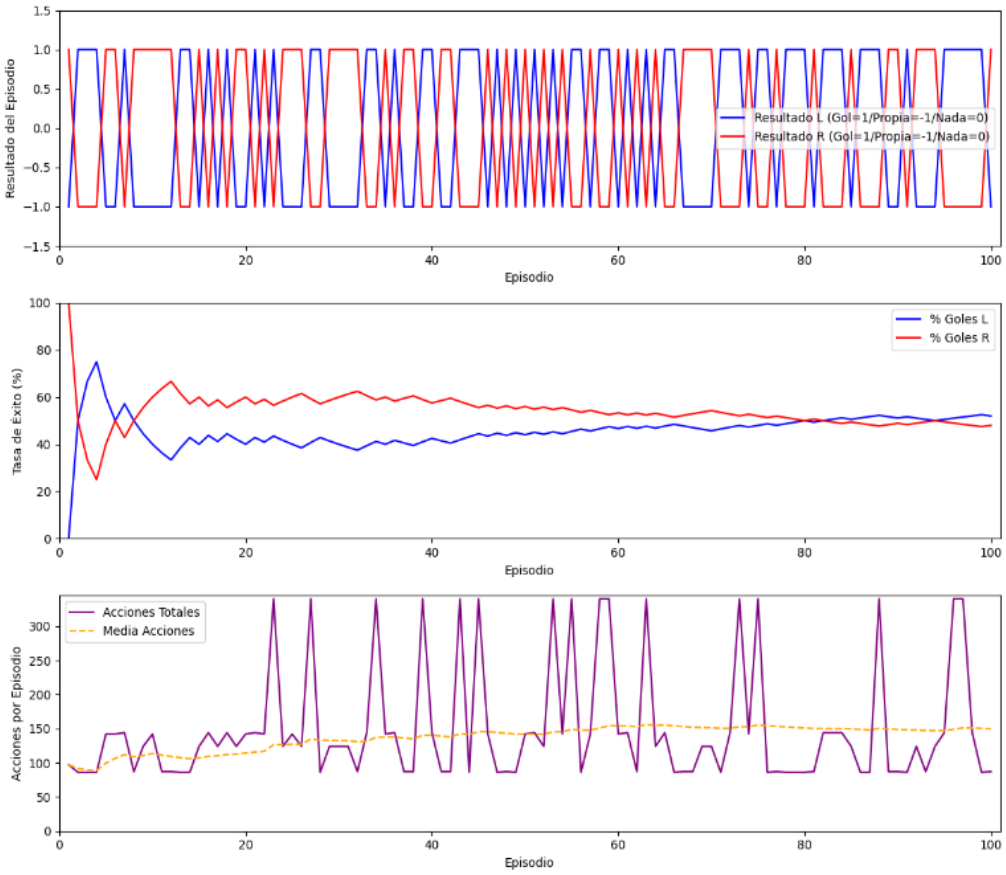

Figura 7.3: Gráfica de recompensas de ambos agentes en un partido

7.2 Pruebas realizadas con personas

Para evaluar la capacidad del agente en situaciones reales y no programadas, realizamos una serie de pruebas donde sustituimos al agente izquierdo por un jugador humano.

- **Configuración**: Se realizaron 50 partidas de duración fija, enfrentando al agente contra distintos jugadores humanos.
- **Objetivo**: Analizar cómo respondía el agente ante comportamientos impredecibles y estrategias humanas.
- **Resultado**: El agente ganó un 60% de encuentros, mostrando adaptabilidad y toma de decisiones efectiva.

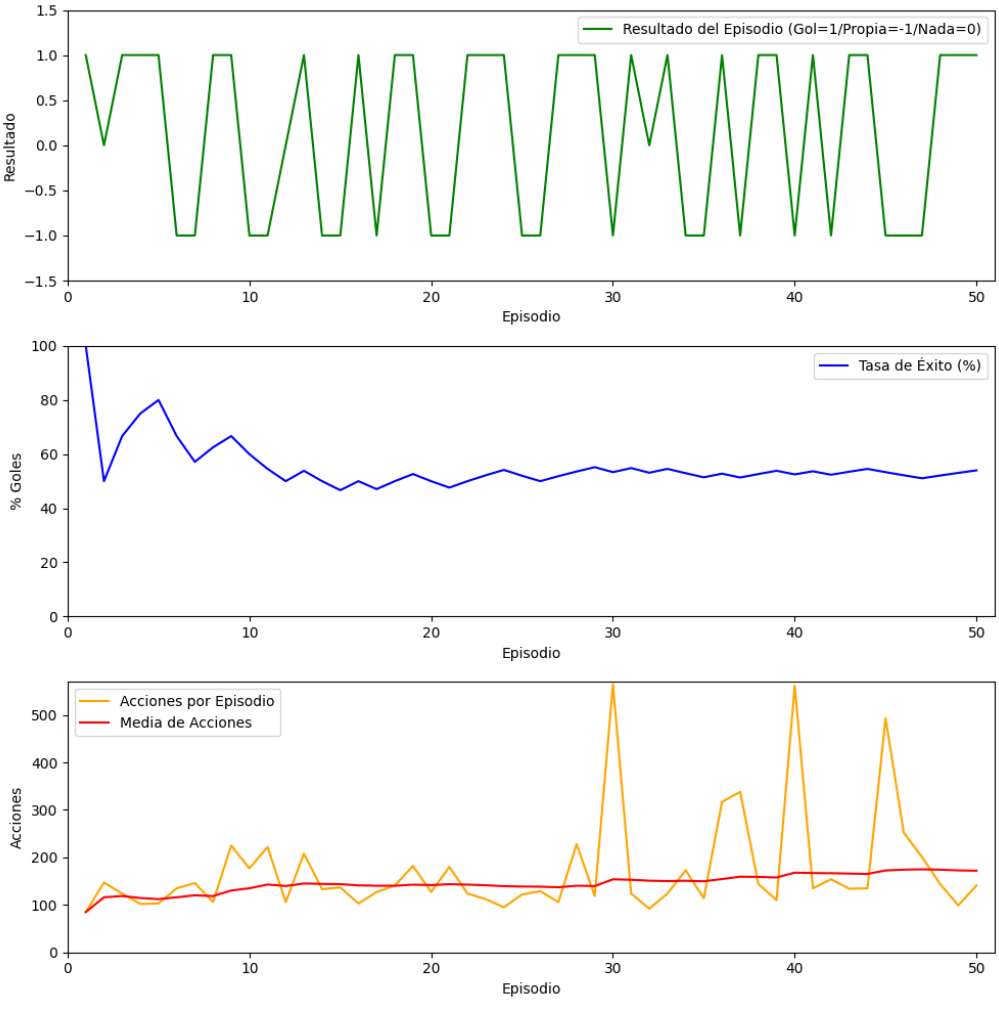

Figura 7.4: Gráfica de recompensas del agente derecho al enfrentarlo con un humano

7.3 Tabla resumen de resultados

A continuación, mostraremos una tabla resumen con los resultados más relevantes de cada fase del entrenamiento y las pruebas finales:

Fase	Episodios de prueba	Métrica Principal	Resultado
Etapa 1 Movimiento	50	Movimiento hacía la pelota	Funcional
Etapa 2 Gol portería	100	Tasa de goles	~60%
Etapa 3 Porterías grandes	100	Tasa de goles	~90%
Agente vs Agente	100	Tasa de goles por agente	45-55%
Pruebas con humanos	50	Tasa de goles + Comportamiento aceptable	~60%

Figura 7.5: Tabla resumen de resultados

Al observar los resultados recogidos en la tabla, podemos concluir que el comportamiento del agente es funcional y efectivo para el objetivo planteado. No solo aprendió a moverse correctamente hacia la pelota, sino que también desarrolló estrategias ofensivas capaces de marcar goles con una tasa de éxito elevada. Además, su rendimiento competitivo frente a otro agente entrenado y su adaptación frente a jugadores humanos refuerzan la validez del enfoque empleado y la solidez del sistema desarrollado.

A partir de estos resultados, podemos afirmar que se han cumplido con los objetivos establecidos al principio del proyecto y, por lo tanto, dar por concluido su desarrollo.

8. Conclusiones

8.1 Conclusiones de los resultados

El desarrollo de este proyecto ha sido un proceso enriquecedor y desafiante, permitiéndonos adentrarnos en el mundo de la inteligencia artificial aplicada a entornos simulados. A lo largo de las distintas etapas del proyecto, hemos logrado avances significativos y enfrentado desafíos que han ido dando forma a nuestros resultados. En este capítulo, analizamos los resultados obtenidos, las dificultades encontradas y los aprendizajes que hemos adquirido a lo largo del proceso.

- **Resultados Obtenidos**
    - **Desarrollo de Agentes Inteligentes:** Hemos conseguido implementar y entrenar agentes inteligentes capaces de jugar un partido de fútbol en un entorno simulado. Los agentes muestran habilidades básicas de control del balón y estrategia ofensiva, logrando marcar goles de manera efectiva.
    - **Optimización de Q-Learning:** Aunque se utilizó Q-Learning clásico [1], conseguimos reducir el espacio de estados de manera significativa mediante técnicas de discretización. Esto hizo viable el entrenamiento sin necesidad de redes neuronales, evitando los problemas de rendimiento y escalabilidad típicos del Q-Learning cuando se usan variables continuas.
    - **Resolución de cuellos de botella:** Se mejoró el rendimiento del sistema al eliminar elementos innecesarios como gráficos en tiempo real y mensajes en consola, lo que permitió aumentar la velocidad de simulación a millones de episodios por día, facilitando el entrenamiento masivo de los agentes.

- **Desafíos y Limitaciones**
    - **Estrategias Defensivas:** Uno de los aspectos que no logramos desarrollar plenamente fue la capacidad de los agentes para ejecutar estrategias defensivas efectivas. Los agentes tienden a enfocarse más en la ofensiva, lo que deja áreas de mejora en la recuperación del balón y la defensa contra los goles del oponente.
    - **Porcentaje de goles marcados con porterías pequeñas:** Otro de los problemas que no pudimos llegar a solucionar perfectamente fue que el agente no era capaz de centrar la pelota para marcar gol en situaciones donde la pelota estaba cerca de una pared.
    - **Problemas Técnicos:** Durante el proyecto, nos enfrentamos a problemas técnicos como la corrupción de archivos debido a fallos en el equipo y la complejidad añadida por los obstáculos en el entorno. Estas dificultades ralentizaron el progreso y requirieron soluciones creativas para mitigarlas.

- **Aprendizajes y Mejoras**
    - **Profundización en Aprendizaje por Refuerzo:** Este proyecto nos ha permitido dominar el uso práctico del Q-Learning clásico, incluyendo técnicas de discretización, diseño de recompensas, e implementación de estrategias para ambientes parcialmente estocásticos.
    - **Optimización y Gestión de Recursos:** A su vez, también aprendimos a identificar y resolver cuellos de botella críticos, desde problemas de rendimiento hasta estructuras de datos ineficientes, lo cual permitió escalar el entrenamiento sin necesidad de recursos computacionales excesivos.
    - **Adaptabilidad y resolución de problemas:** Por último, cada obstáculo, desde fallos técnicos hasta desafíos de comportamiento del agente, nos obligó a buscar soluciones creativas y a iterar sobre nuestros enfoques. Esta experiencia fortaleció nuestra capacidad de adaptación y mejoró la solidez del proyecto.

8.2 Comparativa con otro enfoque

Durante el desarrollo de este proyecto, se elaboró en paralelo un Trabajo de Fin de Grado análogo por parte de Álvaro Ordoño Saiz, quien abordó el mismo problema —el entrenamiento de agentes para competir en partidos de fútbol 1vs1 en un entorno simulado— utilizando una aproximación distinta basada en **algoritmos genéticos** en lugar de Q-Learning.

Ambos proyectos compartían tanto la base técnica, como el entorno de simulación, lo cual permitió establecer una comparación directa y objetiva entre resultados. Mientras que nuestro enfoque utilizó Q-Learning clásico con discretización del espacio de estados, priorizando la eficiencia computacional y la simplicidad del diseño, el enfoque alternativo se basó en técnicas de neuroevolución, entrenando redes neuronales mediante procesos evolutivos.

En cuanto a resultados, el agente entrenado con algoritmos genéticos logró un rendimiento superior en precisión ofensiva, llegando a ganar 14 de 20 partidos contra humanos (70%), e incluso alcanzando una tasa de éxito del 75% en las últimas fases de entrenamiento. Además, los modelos especializados desarrollados (por ejemplo, centrados en el ataque) mostraron un comportamiento más adaptativo, ganando hasta 7 de 10 partidos en sus respectivas pruebas.

Por otro lado, nuestro agente basado en Q-Learning alcanzó tasas de goles entre el 45% y el 55% frente a otros agentes, y un rendimiento de aproximadamente 60% en enfrentamientos contra humanos. Si bien estos resultados son algo inferiores a los obtenidos mediante algoritmos genéticos, nuestro enfoque presentaba una ventaja clara en cuanto a tiempo de entrenamiento y recursos computacionales necesarios. Gracias a la discretización y simplificación del entorno, el agente pudo entrenarse completamente desde cero en **menos de una hora**, alcanzando millones de episodios diarios con una infraestructura modesta y sin necesidad de redes neuronales. Por el contrario, los modelos basados en neuroevolución requerían **varios días de entrenamiento** y una mayor complejidad computacional, lo que hacía el desarrollo más costoso en tiempo y recursos.

En resumen, la comparación entre ambos enfoques muestra que:

- El enfoque basado en algoritmos genéticos obtiene mejores resultados en tareas complejas o con menor margen de error, gracias a su capacidad de generalización y adaptación.
- El enfoque basado en Q-Learning clásico es más adecuado para tareas simples o que pueden discretizarse eficazmente, ya que permite lograr un comportamiento funcional con un coste computacional y temporal significativamente menor.

Ambas estrategias han demostrado ser viables y efectivas dentro de sus respectivos contextos, y su comparación aporta una visión más completa sobre las distintas formas de abordar problemas de aprendizaje por refuerzo en entornos simulados.

9. Relación con los estudios

9.1 Asignaturas Relacionadas

Las asignaturas que más nos han servido para la realización de este Trabajo de Fin de Grado (TFG) han sido las siguientes:

- **Técnicas, Entornos y Aplicaciones de Inteligencia Artificial:** En esta asignatura adquirimos conocimientos fundamentales sobre diferentes técnicas y entornos en los que se aplica la inteligencia artificial, así como una comprensión general de sus aplicaciones prácticas.
- **Aprendizaje Automático:** Aquí aprendimos las bases del aprendizaje automático, incluyendo cómo funcionan las redes neuronales y los algoritmos de entrenamiento. Este conocimiento fue crucial para implementar y entrenar a los agentes utilizando Q-Learning.
- **Percepción:** Esta asignatura nos proporcionó una comprensión sobre cómo los agentes pueden percibir y procesar información del entorno, lo cual es esencial para desarrollar agentes inteligentes que interactúen de manera efectiva en un entorno simulado.
- **Agentes Inteligentes:** En esta materia profundizamos en el diseño y desarrollo de agentes capaces de tomar decisiones autónomas basadas en la información del entorno, algo directamente aplicable al proyecto de simular un entorno de fútbol con agentes Q-Learning.
- **Desarrollo de Videojuegos 3D:** De esta asignatura obtuvimos la metodología de trabajo que fue fundamental a la hora de gestionar el proyecto.

9.2 Competencias Transversales

En cuanto a las competencias transversales que hemos necesitado y desarrollado para la realización del TFG caben destacar:

- **Análisis y Resolución de Problemas:** La capacidad para identificar, analizar y resolver problemas ha sido esencial en todo el proceso de desarrollo del proyecto, desde la implementación de las redes neuronales hasta la solución de problemas técnicos y de rendimiento.
- **Innovación, Creatividad y Emprendimiento:** Hemos tenido que ser creativos e innovadores para superar los desafíos técnicos y de diseño que surgieron durante el desarrollo del proyecto. Esta competencia también se refleja en nuestra capacidad para probar otras soluciones como cuando usamos DQN.
- **Planificación y Gestión del Tiempo:** La organización y gestión eficiente del tiempo han sido cruciales para cumplir con los plazos y objetivos del proyecto. Desde la planificación del entrenamiento de los agentes hasta la implementación de nuevas funcionalidades, la planificación ha sido una competencia clave para el éxito del proyecto.

Estas competencias transversales no solo han sido vitales para la realización de este TFG, sino que también nos preparan para enfrentar futuros desafíos profesionales en el campo de la inteligencia artificial y el desarrollo de software.

10. Trabajos futuros

En este capítulo abordaremos las posibles mejoras y añadidos que se habrían implementado si hubiéramos contado con una mayor cantidad de tiempo. Estas mejoras están enfocadas en profundizar y ampliar las capacidades de los agentes y del entorno de juego.

- **Estrategias Defensivas**

  Uno de los aspectos que nos habría gustado desarrollar más son las estrategias defensivas de los agentes. Actualmente, los agentes se centran mucho más en la ofensiva, priorizando el ataque y la anotación de goles. Sin embargo, mejorar su capacidad para recuperar la pelota y evitar que les marquen gol es crucial para crear un juego más equilibrado y realista. Implementar técnicas defensivas avanzadas permitiría a los agentes tener un enfoque más completo y dinámico en el juego.

- **Añadir Más Jugadores con Distintos Roles**

  Otra mejora significativa habría sido la inclusión de más jugadores en cada equipo, asignándoles diferentes roles y tareas específicas. Esto permitiría desarrollar tácticas de equipo más complejas, donde cada jugador tiene un papel específico que contribuye al éxito del equipo. Por ejemplo, podríamos tener jugadores especializados en defensa, mediocampo y ataque, trabajando juntos para lograr objetivos comunes. Esta ampliación no solo haría el juego más interesante, sino que también presentaría nuevos desafíos y oportunidades de aprendizaje para los agentes.

- Disminuir el tamaño de las porterías

  Otro posible apartado de mejora sería disminuir el tamaño de las porterías poco a poco y realizar los debidos cambios en los agentes hasta lograr que sean capaces de marcar gol en porterías de cualquier tamaño. Este cambio mejoraría en gran medida la portabilidad del agente permitiéndole poder entrenar en cualquier tipo de entorno posible.

- **Creación de Entornos Diversos**

  Finalmente, también nos habría gustado crear una variedad de entornos distintos donde los agentes deban enfrentarse a diferentes problemas y desarrollar estrategias adaptativas. Estos entornos podrían incluir variaciones en la configuración del campo de juego, la presencia de obstáculos adicionales, o incluso condiciones climáticas que afecten el comportamiento del balón y los jugadores. Cada entorno presentaría un conjunto único de desafíos, obligando a los agentes a aprender y adaptar sus estrategias en función del contexto.

Bibliografía

[1] Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press.

https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf

[2] Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research, 4, 237-285.

https://doi.org/10.1613/jair.301

[3] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529–533.

https://doi.org/10.1038/nature14236

[4] OpenAI. (2018). OpenAI Baselines: High-quality implementations of reinforcement learning algorithms.

https://github.com/openai/baselines

[5] Pygame Community. (2024). Pygame: Set of Python modules designed for writing video games.

https://www.pygame.org/

[6] PyTorch Team. (2024). PyTorch Documentation.

https://pytorch.org/docs/stable/index.html

[7] Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. Machine Learning, 8(3–4), 279–292.

https://doi.org/10.1007/BF00992698

[8] Silver, D. (2015). Lecture Slides on Reinforcement Learning. University College London.

http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

[9] Larman, C., & Basili, V. R. (2003). Iterative and Incremental Development: A Brief History. IEEE Computer, 36(6), 47–56.

https://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf

[10] Mitchell, T. M. (1997). Machine Learning. McGraw-Hill.

[11] Russell, S., & Norvig, P. (2021). Artificial Intelligence: A Modern Approach (4th ed.). Pearson.

[12] Nielsen, M. (2015). Neural Networks and Deep Learning.

http://neuralnetworksanddeeplearning.com/

[13] McCarthy, J. (2007). The Dartmouth Conference Proposal. En John McCarthy: Papers.

[14] Newell, A., & Simon, H. A. (1956). The Logic Theorist.

[15] Lighthill, J. (1973). Artificial Intelligence: A Paper Symposium (Study Group on Artificial Intelligence). Science Policy Research Unit (SPRU), University of Sussex.

https://rodsmith.nz/wp-content/uploads/Lighthill_1973_Report.pdf

[16] Deng, J., et al. (2009). ImageNet: A Large-Scale Hierarchical Image Database. 2009 IEEE Conference on Computer Vision and Pattern Recognition.

[17] Silver, D., et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484–489.

https://doi.org/10.1038/nature16961

[18] Brown, T. B., et al. (2020). Language Models are Few-Shot Learners.

https://arxiv.org/abs/2005.14165

[19] Brockman, G., et al. (2016). OpenAI Gym. arXiv:1606.01540.

https://arxiv.org/abs/1606.01540

[20] Koenig, N., & Howard, A. (2004). Design and use of ROS (Robot Operating System). In Robotics and Autonomous Systems.

[21] Juliani, A., Vinyals, O., Heess, N., et al. (2018). Unity ML-Agents Toolkit. arXiv:1809.02627.

https://arxiv.org/abs/1809.02627

[22] Schaul, T., Qureshi, A. R., Antonoglou, I., et al. (2016). Prioritized Experience Replay. arXiv:1511.05952.

https://arxiv.org/abs/1511.05952

Glosario

- **Agente (Agent):** Entidad autónoma que percibe su entorno a través de sensores y actúa sobre él mediante actuadores para alcanzar sus objetivos.
- **Entorno (Environment):** Contexto en el que un agente opera, que incluye todos los elementos y condiciones que el agente puede percibir y con los que puede interactuar.
- **Estado (State):** Representación completa de la situación actual del entorno, utilizada por el agente para tomar decisiones informadas.
- **Acción (Action):** Movimiento o decisión tomada por un agente en respuesta al estado actual del entorno, con el objetivo de maximizar su recompensa futura.
- **Recompensa (Reward):** Valor numérico que un agente de aprendizaje por refuerzo recibe del entorno para evaluar y guiar sus acciones hacia el logro de un objetivo.
- **Recompensa Negativa**: Penalización otorgada al agente por realizar acciones indeseables o ineficaces, utilizada para desalentar ciertos comportamientos.
- **Función de Valor:** En el aprendizaje por refuerzo, una función que estima la recompensa esperada de un agente en un estado dado o al realizar una acción específica.
- **Política (Policy):** Estrategia que define las acciones que debe tomar un agente en diferentes estados del entorno para maximizar su recompensa futura.
- **Q-Learning:** Algoritmo de aprendizaje por refuerzo que busca aprender una política que maximiza la recompensa acumulada a lo largo del tiempo mediante la estimación de valores de acción.
- **Deep Q-Learning (DQN):** Algoritmo de aprendizaje profundo que combina Q-Learning con redes neuronales profundas para manejar espacios de estados y acciones continuos y de alta dimensión.
- **Red Neuronal (Neural Network):** Modelo computacional inspirado en la estructura del cerebro humano, compuesto por capas de nodos (neuronas) que procesan información y aprenden patrones complejos a partir de datos.
- **Estrategia Defensiva:** Conjunto de acciones y tácticas empleadas por el agente para evitar que el oponente marque goles y recuperar la posesión del balón.
- **Táctica:** Plan específico de acción utilizado por el agente para alcanzar sus objetivos de manera efectiva dentro del juego.
- **Metodología de Desarrollo Iterativo e Incremental:** Enfoque de desarrollo de software que se basa en realizar mejoras y adiciones graduales a través de ciclos repetidos, permitiendo ajustes continuos y adaptación a nuevas necesidades y descubrimientos.
- **Q-value (Valor Q):** Valor numérico que representa la utilidad esperada de realizar una acción específica en un estado determinado, siguiendo una determinada política. En Q-Learning, los Q-values se almacenan en una tabla y se actualizan iterativamente a medida que el agente interactúa con el entorno, permitiéndole aprender qué acciones conducen a mayores recompensas acumuladas.

Relación con los ODS

ANEXO II. Relación del trabajo con los Objetivos de Desarrollo Sostenible de la agenda 2020

Anexo al TFG/M: Relación del trabajo con los Objetivos deDesarrollo Sostenible de la agenda 2020

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**			X	
ODS 4. **Educación de calidad.**		X		
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**			X	
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades**.			X	
ODS 11. **Ciudades y comunidades sostenibles**.			X	
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**			X	

Descripción de la alineación del TFG/M con los ODS con un grado de relación más alto.

Nuestro Trabajo de Fin de Grado se alinea especialmente con el **Objetivo de Desarrollo Sostenible 9: Industria, innovación e infraestructuras**, cuyo propósito es promover la industrialización inclusiva y sostenible, y fomentar la innovación.

Si bien es cierto que el proyecto no está directamente aplicado al sector industrial, **sí resulta extrapolable a este contexto**, ya que el desarrollo de agentes inteligentes mediante técnicas de aprendizaje por refuerzo representa un avance tecnológico con gran potencial para optimizar recursos y automatizar tareas complejas. El uso de la inteligencia artificial en estos ámbitos puede suponer una **nueva revolución en la industria**, comparable a la introducción de la automatización o la robótica.

Además, este trabajo impulsa la innovación tecnológica al investigar y aplicar algoritmos de **Q-Learning**, y promueve el uso de tecnologías accesibles mediante herramientas de código abierto como **PyTorch** y **Pygame**. Esto facilita la extrapolarización del sistema en otros entornos, tanto académicos como industriales, favoreciendo así el desarrollo de infraestructuras adaptables y eficientes.