UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

ENTRENAMIENTO DE JUGADORES VIRTUALES DE PÁDEL: COMBINANDO APRENDIZAJE POR REFUERZO CON DATOS DE JUGADORES PROFESIONALES

WENQI ZHOU

Director/a
CARLOS ANDUJAR GRAN (Departamento de Ciencias de la Computación)

Titulación
Grado en Ingeniería Informática (Computación)

Memoria del trabajo de fin de grado

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

01/07/2024

Agradecimientos

Quiero agradecer sinceramente a todas las personas que han contribuido a la realización de este Trabajo de Fin de Grado. En primer lugar, quiero agradecer a mi tutor, Carlos Andújar, y a mi colaborador, Jia Long Ji Qiu, por su dedicación y orientación constante a lo largo de todo el proceso. También quiero expresar mi gratitud a Mohammadreza Javadiha por proporcionar los datos sobre posicionamiento y golpeos de jugadores de pádel profesionales, que fueron utilizados para la parte de entrenamiento por imitación.

Agradezco a todas las personas que, de una manera u otra, han contribuido al desarrollo de este TFG. Sin vuestro apoyo y colaboración, este proyecto no hubiera sido posible.

Resumen

En este trabajo de fin de grado se investigan diversas metodologías para entrenar a un agente a jugar al pádel, emulando el comportamiento de un jugador humano. El enfoque se centrará en la aplicación de algoritmos de aprendizaje por refuerzo (RL) y aprendizaje por imitación (IL) dentro de un entorno virtual básico de pádel, con el objetivo de permitir que el agente tome decisiones óptimas basadas en la información observada del entorno.

El proyecto profundizará en los principios de los procesos de decisión de Markov (MDP), que son la base del aprendizaje por refuerzo, y explorará el Aprendizaje por Imitación Generativa Adversaria (GAIL). Además, se examinarán enfoques de aprendizaje por refuerzo basados en modelos y sin modelos para comprender mejor sus mecanismos de entrenamiento. Para implementar estos conceptos, se utilizará el entorno virtual de Unity, junto con ML-Agents Toolkit. A través del SDK de Unity, se llevarán a cabo varios experimentos para determinar el algoritmo más efectivo para entrenar a un agente a jugar al pádel.

Resum

En aquest treball de fi de grau s'investiguen diverses metodologies per entrenar un agent a jugar a pàdel, emulant el comportament d'un jugador humà. L'enfocament se centrarà en l'aplicació d'algoritmes d'aprenentatge per reforç (RL) i aprenentatge per imitació (IL) dins d'un entorn virtual bàsic de pàdel, amb l'objectiu de permetre que l'agent prengui decisions òptimes basades en la informació observada de l'entorn.

El projecte aprofundirà en els principis dels processos de decisió de Markov (MDP), que són la base de l'aprenentatge per reforç, i explorarà l'Aprenentatge per Imitació Generativa Adversària (GAIL). A més, s'examinaran enfocaments d'aprenentatge per reforç basats en models i sense models per comprendre millor els seus mecanismes d'entrenament. Per implementar aquests conceptes, s'utilitzarà l'entorn virtual de Unity, juntament amb ML-Agents Toolkit. A través del SDK de Unity, es duran a terme diversos experiments per determinar l'algoritme més efectiu per entrenar un agent a jugar a pàdel.

Abstract

In this final degree project, various methodologies are investigated to train an agent to play padel, emulating the behavior of a human player. The focus will be on the application of reinforcement learning (RL) and imitation learning (IL) algorithms within a basic virtual padel environment, with the goal of enabling the agent to make optimal decisions based on the observed information from the environment.

The project will delve into the principles of Markov Decision Processes (MDP), which form the basis of reinforcement learning, and will explore Generative Adversarial Imitation Learning (GAIL). Additionally, model-based and model-free reinforcement learning approaches will be examined to better understand their training mechanisms. To implement these concepts, the Unity virtual environment will be used, along with the ML-Agents Toolkit. Through the Unity SDK, various experiments will be conducted to determine the most effective algorithm for training an agent to play padel.

Índice

Índice de figuras

Índice de tablas

1. Introducción y contextualización

En los últimos 5 años, la demanda de la IA ha crecido drásticamente. Especialmente después de la aparición de ChatGPT, un modelo grande de lenguaje (LLM) que reconoce el lenguaje natural. Gracias al crecimiento de la IA, los ámbitos de uso son cada vez más numerosos. Algunos ejemplos son los videojuegos y el entrenamiento de profesionales en diferentes sectores como la medicina, la programación o el deporte. Justamente este TFG se centra en la aplicación de técnicas de IA en el ámbito deportivo, concretamente en un relativamente joven deporte de raqueta: el pádel.

Últimamente, el pádel ha experimentado un notable crecimiento de popularidad. Este deporte ofrece una experiencia única que aúna velocidad, estrategia y colaboración, atrayendo a personas de todas las edades. Como todos los deportes competitivos, los jugadores buscan perfeccionar sus técnicas y habilidades. Con este propósito de mejora surgió la necesidad de emplear tecnologías innovadoras para analizar y optimizar el rendimiento de los jugadores. Desde el análisis de los datos recopilados durante los partidos, hasta la personalización de estrategias según las necesidades específicas de cada jugador. La integración de IA en el ámbito deportivo nos lleva a la capacidad de identificar y analizar los puntos fuertes y débiles de cada jugador y cada equipo. Al recopilar y procesar datos sobre el comportamiento del atleta, podemos generar simulaciones que revelan los factores clave que pueden ser aprovechados durante una competición, y nos permiten desarrollar estrategias personalizadas y adaptativas para enfrentar al oponente. La IA posee un potencial revolucionario que puede beneficiar todo el mundo, incluidos los jugadores, entrenadores y aficionados.

1.1. Contexto

Este proyecto *Entrenamiento de jugadores virtuales de pádel: combinando aprendizaje por refuerzo con datos de jugadores profesionales* representa el trabajo final de grado (TFG) dentro del plan de estudios del Grado en Ingeniería Informática de la Facultad de Informática de Barcelona (FIB), y se realiza en la modalidad A y en la especialidad de Computación. Este proyecto está supervisado por el profesor Carlos Andújar Gran del Departamento de Ciencias de la Computación (CS). El TFG se enmarca también dentro del ViRVIG (Grupo de Investigación en Visualización, Realidad Virtual e Interacción Gráfica) que incluye como una de sus líneas de investigación el análisis deportivo, así como del proyecto PID-2021 *Entornos 3D de alta fidelidad para Realidad Virtual y Computación Visual: geometría, movimiento, interacción y visualización para salud, arquitectura y ciudades*[1]. También cuenta con el asesoramiento de Jia Long Ji Qiu, ya que este trabajo se basa parcialmente, como punto de partida, su TFG, tal y como se discute más adelante.

1.2. Conceptos

1.2.1. Aprendizaje por refuerzo

El aprendizaje por refuerzo (RL) es una rama dentro del campo de la inteligencia artificial que se inspira en cómo los seres humanos y otros organismos aprenden a través de la interacción con su entorno. En comparación con otros tipos de aprendizaje automático, basados en grandes cantidades de datos estáticos, el aprendizaje por refuerzo se centra en las decisiones del agente

que aprende a través de la experimentación, de forma similar a cómo los seres vivos aprenden de sus experiencias.

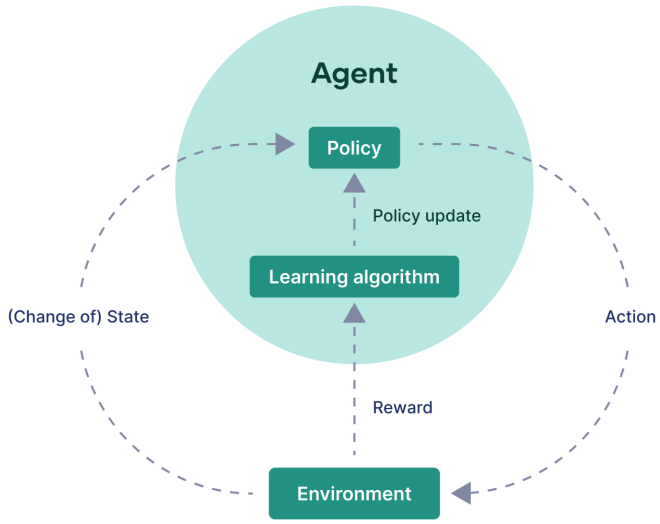


Figura 1: Flujo genérico del aprendizaje por refuerzo.(Fuente: [2])

En RL, un agente aprende mediante la interacción con su entorno, tomando ciertas acciones en el entorno y observando las recompensas o penalizaciones asociadas con la acción realizada. El objetivo del agente es maximizar la acumulación de recompensas durante el proceso de entrenamiento, identificando qué acciones generan mejores resultados en función del estado.

El proceso de aprendizaje por refuerzo se puede describir generalmente como sigue: el agente selecciona una acción $a \in A(s_t)$ basada en su política actual (que puede ser una estrategia aleatoria al principio) partiendo de su estado actual $s_t \in S$, ejecuta esa acción en el entorno actual, observa el nuevo estado s_{t+1} y la recompensa asociada r, y luego actualiza su conocimiento o política en base de la experiencia para mejorar su desempeño futuro [3].

1.2.2. Aprendizaje por imitación

El aprendizaje por imitación (IL) es otro enfoque importante dentro del aprendizaje automático. El método consiste en el que un agente adquiere conocimientos al intentar replicar el comportamiento de un modelo previamente definido.

Esto se logra buscando una política de comportamiento π para el agente que se comporta de manera lo más similar posible a la política del experto π^*. Sin embargo, a diferencia de un problema de Aprendizaje por Refuerzo (RL), donde el agente aprende de la interacción previa con el entorno, en IL no se puede observar directamente la política del experto. En su lugar, se centra más en simular el comportamiento del experto con una cierta condición, y el objetivo es aproximar esta política a través de la acción [3].

1.3. Identificación del problema

Este trabajo va a partir de un TFG previo realizado por Jia Long Ji Qiu [3]. El objetivo principal del proyecto es desarrollar una aplicación que simule una partida de pádel, tal que el agente

pueda imitar un determinado comportamiento de un jugador o experto. Cabe distinguir los dos ámbitos principales del trabajo, que son:

- Tomando como punto de partida una parte del entorno virtual creado por Jia Long, este trabajo se centrará más en el movimiento de los jugadores, así como en la física detrás del movimiento de la pelota. Se introducirán nuevos cambios al entorno para que se ajuste más al mundo real. La implementación tendrá en cuenta diferentes técnicas, así como el reglamento del pádel.
- Otra área de trabajo será el entrenamiento del agente, donde queremos que los jugadores virtuales se comporten de manera realista. Esto implica que necesitan decidir cómo moverse, qué golpes hacer y hacia dónde enviar la pelota durante un partido. Exploraremos diferentes técnicas de aprendizaje automático, específicamente el aprendizaje por refuerzo y el aprendizaje por imitación. En lugar de codificar las reglas directamente, entrenaremos a los agentes virtuales para que aprendan a comportarse, observando y practicando para aprender una política óptima.

1.4. Agentes implicados

El resultado del proyecto va dirigido a una gran variedad de campos, no solamente la industria de los videojuegos:

- Las personas relacionadas con este ámbito del deporte (jugadores, entrenadores, analistas), ya que mediante la simulación del modelo podría observar qué efectos podría tener una jugada especial o la potencia que tiene un jugador agresivo o defensivo en el juego de pádel. Por ejemplo, el modelo entrenado con aprendizaje por imitación, puede servir para analizar el comportamiento del juego y poder mejorar sus puntos débiles. También permitiría simular un partido en el que los jugadores forman equipos que no existen en la realidad.
- Las personas fuera del ámbito deportivo, como desarrolladores de videojuegos, podrán utilizar el proyecto para realizar una investigación sobre el comportamiento del aprendizaje por refuerzo y el aprendizaje por imitación. Para poder desarrollar su propio bot o NPC capaz de jugar una partida contra un jugador humano.
- Para investigadores y perfiles como estudiantes interesados en el aprendizaje automático, el producto puede servir como base para estudios o investigaciones futuras con objetivos similares a los de este proyecto, como otros deportes de raqueta similares.

2. Justificación

Si bien el aprendizaje automático abarca diversos campos, sin duda el aprendizaje profundo es uno de los más destacados en términos de popularidad. Sin embargo, áreas como el aprendizaje por refuerzo (RL) y el aprendizaje por imitación (IL) no gozan de la misma atención en comparación con el aprendizaje profundo. De manera similar, el pádel, aunque se trata de un deporte muy valorado, no goza de la misma popularidad que otros deportes más conocidos como baloncesto y tenis. Debido a esto es muy difícil encontrar una aplicación de código abierto que simule una partida de pádel; lo que generalmente encontramos son proyectos académicos realizados por estudiantes, como *Entrenamiento de agentes virtuales para el aprendizaje de políticas óptimas de los jugadores durante un partido de pádel* realizado por el estudiante Jia Long Ji Qiu [3], el cual supone el punto de partida del presente proyecto. También cabe destacar otro proyecto previo *Agentes Inteligentes para imitar el comportamiento de los jugadores de Pádel* realizado por el estudiante Iván López Rodríguez [4].

2.1. Estudio de soluciones existentes

En el mercado actual, los simuladores de pádel son escasos. En Barcelona, existe una empresa llamada PadelVR [5] que está desarrollando un simulador de pádel utilizando la realidad virtual. Además, hay una iniciativa similar a la nuestra en la que Google, en colaboración con el equipo de fútbol Manchester City F.C., va a realizar un concurso [6]. En esta iniciativa, los participantes entrenarán a sus agentes para jugar al fútbol y competirán contra los agentes de otros participantes. Aunque estos ejemplos existen, no podemos aprovechar estos proyectos, ya que no son de código abierto.

2.1.1. Motor gráfico

Un motor gráfico es un conjunto de herramientas que acelera el proceso del desarrollo de videojuegos y aplicaciones, evitando trabajos repetitivos. De esta forma se permite a los desarrolladores centrarse más en la creación y no preocuparse por las lógicas que hay detrás, evitándose las tareas repetitivas. Los más populares son *Unity* y *Unreal Engine*, que son los dos motores gráficos más importantes en la industria del desarrollo de videojuegos. Unity es reconocido por su facilidad de uso, una herramienta ideal para principiantes. Por otro lado, Unreal Engine ofrece gráficos potentes con alta fidelidad, pero la curva de aprendizaje es mucho más elevada que Unity.

Tras comparar estos dos motores gráficos, al final optamos por utilizar *Unity*, ya que es un motor mucho más simple de usar que Unreal. Además, los trabajos previos están realizados usando Unity; por conveniencia y para poder aprovechar recursos ya existentes, resulta lógico escoger Unity.

2.1.2. Librerías de entrenamiento

A continuación se describen muy brevemente las principales librerías de entrenamiento de agentes virtuales:

- **OpenAI's Gym**: Gym es una librería para Python desarrollada por OpenAI que se centra principalmente en RL. Se basa en el Modelo de Proceso de Decisión de Márkov (MDP),

un modelo dinámico de toma de decisiones. En Gym, se implementa el ciclo entre agente y entorno, donde el agente toma ciertas acciones dentro del entorno y luego observa cómo estas acciones afectan y modifican el estado del entorno [7].

- **ReAgent by Meta**: Una de las ventajas principales de ReAgent es su capacidad para evaluar el rendimiento de manera offline antes de su implementación en un entorno real. Esto facilitará a los desarrolladores ajustar sus modelos y mejorar su precisión antes de utilizarlos en producción. Otra ventaja importante de ReAgent es una librería C++ que se puede integrar en cualquier aplicación[7].
- **TensorFlow's TF-Agents**: Es una librería desarrollada por Google que ofrece varios algoritmos de RL basados en TensorFlow. Ofrece una interfaz simple y modular para desarrollar y entrenar agentes en un entorno variado. La ventaja más grande de TF-Agents es que opera en dos entornos paralelos y realiza múltiples cálculos al mismo tiempo durante el entrenamiento de una red neuronal. Esto elimina la necesidad de sincronización manual y permite que el motor optimice el cálculo de manera eficiente.[7].
- **ML-Agents**: Es un proyecto de código abierto desarrollado por *Unity Technologies*. Como es un proyecto realizado por el equipo de Unity, es muy fácil incorporar RL en los proyectos de Unity. Al estar diseñado específicamente para Unity, permite a los desarrolladores entrenar agentes de IA para interactuar con entornos creados dentro de Unity. ML-Agents ofrece una variedad de algoritmos de RL que los usuarios pueden adaptar a sus proyectos de manera oportuna, incluyendo Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), etc.

Comparando estas librerías, sin ninguna duda ML-Agents es la librería que mejor se adapta a la propuesta que plantea el trabajo. Ya que ML-Agents está integrado directamente con Unity, permite a los desarrolladores crear y entrenar agentes de IA directamente dentro del entorno de desarrollo de Unity. Esto simplifica el proceso de iteración y desarrollo.

3. Alcance

3.1. Objetivos

Este proyecto se centra principalmente en crear y analizar una aplicación capaz de reconstruir partidos de pádel en un entorno virtual que se parezca a la realidad tanto como sea posible. Un aspecto clave es el uso de aprendizaje automático en la creación de un agente inteligente que simula a un jugador real. Para alcanzar este objetivo, lo subdividiremos en los siguientes:

1. Diseñar y mejorar el entorno virtual existente para que se asemeje más a una pista de pádel real, donde los movimientos del agente y la interacción con el entorno sean más fiables.
2. Explorar y emplear distintas estrategias de aprendizaje basadas en la imitación para capacitar agentes virtuales. Estos agentes aprenderán a jugar al pádel mediante el análisis de datos de jugadores profesionales.
3. Estudiar y aplicar diversas estrategias destinadas a agilizar el entrenamiento del modelo del agente, dado que el proceso de obtener un modelo funcional mediante aprendizaje automático puede requerir largas horas de entrenamiento.
4. Estudiar y comparar distintas técnicas de aprendizaje para entrenar agentes virtuales, que principalmente se centra en la RL y IL. Queremos comparar qué técnica es más eficaz teniendo en cuenta cuál de ellas se aproxima mejor a un jugador humano y con un tiempo de entrenamiento menor.

3.2. Requerimientos

3.2.1. Requerimientos funcionales

Algunos de estos requisitos ya está mencionado en los apartados anteriores, pero existen otros, como por ejemplo durante una simulación el usuario tendrá la capacidad de alternar entre las diferentes políticas aprendizaje. Ya que con esto permitirá comparar la eficiencia de diferentes políticas. Y durante la simulación el usuario tiene capacidad de pausar y continuar una simulación en cualquier momento. Ya que el entrenamiento de agente puede ser excesivo, el usuario puede parar en el momento de que la mejora del agente ya no es significativo comparando con el tiempo.

3.2.2. Requerimientos no funcionales

Hablando de los requerimientos no funcionales debe destacar:

- Usabilidad: El usuario puede usar la aplicación sin ninguna dificultad. Proporcionará una documentación que detalla cómo entrenar un nuevo agente. Que una persona sin ningún conocimiento siguiendo la documentación del proyecto podría realizarlo sin problema. Para lograr este objetivo, escogerá un grupo de personas con un conocimiento básico de programación y dejar a ellos entrenar su propio agente. Después obtenemos sus respuestas para mejorar.
- Reusabilidad o flexibilidad: La aplicación debe ser diseñada de manera que sea altamente integrable con otros sistemas. Que el agente puede ser integrada en otra deportes

mediante modificaciones de los comportamientos para adaptarse el nuevo entorno. En este caso se podría escoger un equipo de programadores que integra el agente a otro tipo de deporte, por ejemplo el baloncesto y descubrir la inconveniencia y dificultad.

3.3. Obstáculos y riesgos

Es esencial prever los posibles contratiempos en un proyecto para reducir su impacto. Los principales desafíos y riesgos que podrían surgir en este proyecto incluyen:

- **Complejidad del aprendizaje**: La implementación de técnicas de aprendizaje por imitación y aprendizaje por refuerzo puede resultar compleja y requerir un profundo entendimiento teórico de los algoritmos implicado.
- **Disponibilidad y calidad de datos**: Obtener conjuntos de datos de buena calidad para entrenar los agentes virtuales puede ser difícil. Además, la disponibilidad de datos específicos de jugadores profesionales de pádel puede ser limitada.
- **Requerimientos de computación**: Los algoritmos de aprendizaje utilizados pueden ser intensivos en términos de recursos computacionales, lo que podría requerir hardware potente para su ejecución eficiente o un tiempo no excesivamente largo.
- **Evaluación y ajuste de las políticas**: Evaluar y ajustar las políticas aprendidas por los agentes puede ser un proceso iterativo y complejo, que requiere tiempo y recursos para obtener resultados satisfactorios.
- **Validación de resultados**: Verificar que los agentes virtuales aprendan y jueguen de manera realista y eficaz requerirá métodos de validación adecuados, así requiere la participación de expertos en el deporte del pádel para evaluar la fidelidad de los movimientos y estrategias generadas.

4. Metodología

Dado el desafío de enfrentar proyectos con tiempo limitado, se busca subdividir una tarea extensa en subtareas. Teniendo en cuenta que muchas de las subtareas dependen de otras, es conveniente utilizar una metodología flexible que pueda adaptarse constantemente a las nuevas necesidades, por lo tanto, la metodología ágil se considera la más adecuada.

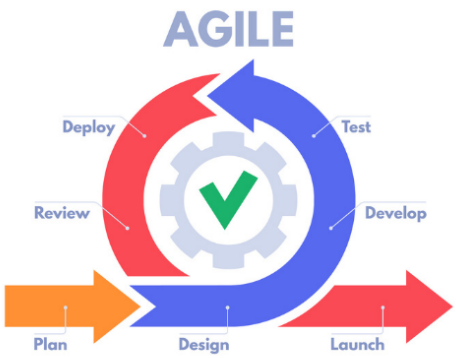


Figura 2: Ciclo de desarrollo ágil (Fuente: [8])

La metodología ágil se basa en iteraciones cortas, colaboración continua y flexibilidad a los cambios. Durante el desarrollo del Trabajo de Fin de Grado (TFG), se establecerán ciclos de trabajo cortos, en los cuales se implementarán y probarán las funcionalidades específicas.

En este contexto, durante la realización del Trabajo de Fin de Grado (TFG), cada ciclo tendrá una duración aproximada de una semana. Durante este período, el estudiante se reunirá regularmente con su director de proyecto para revisar el progreso, establecer objetivos para el próximo ciclo y realizar ajustes según sea necesario. Esta metodología ágil permite una gestión continua y adaptativa del TFG, lo que facilita la obtención de respuestas constantes y la prevención de desviaciones en el desarrollo del proyecto.

Al adoptar este enfoque, se fomenta la colaboración activa entre el estudiante y el director del TFG, promoviendo una comunicación fluida. Además, al dividir el trabajo en ciclos cortos, se reduce el riesgo de desviaciones y se facilita la identificación temprana de posibles problemas.

4.1. Herramientas

Para poder realizar el seguimiento y verificar las tareas, se decidió utilizar *Trello*, una herramienta de gestión de proyectos basada en bloques visibles. La herramienta permite organizar tareas y proyectos utilizando listas y tableros.

Se empleará *Google Meet* para llevar a cabo reuniones semanales con el director, con el propósito de recibir comentarios sobre el avance del trabajo y discutir su estado actual.

Para garantizar un control de versiones efectivo, respaldar el trabajo y facilitar la colaboración con el director del proyecto, se optó por utilizar GitHub como plataforma para almacenar el proyecto en un repositorio y poder tener control de las versiones.

5. Planificación temporal

El TFG consta de un total de 18 créditos, de los cuales 3 créditos corresponden a la gestión de proyecto (GEP) y 15 créditos al propio trabajo. Según el sistema de evaluación de la UPC [9], cada crédito equivale a 30 horas de trabajo. Entonces, en términos de previsión, el proyecto requiere 540 horas de dedicación. Este trabajo empezó el día 22 de enero de 2024 y la entrega es el día 21 de junio de 2024, Tenemos 155 días para poder acabar el proyecto y requiere una dedicación de 3.5 horas al día.

Para garantizar el cumplimiento del plazo previsto y mantener una buena estructura durante todo el desarrollo del proyecto, es importante tener una planificación temporal clara que nos guíe hacia nuestros objetivos. Por lo tanto, utilizamos la planificación final de Jia Long [3] como la base para tener una mejor planificación que se adapte mejor a nuestro caso.

5.1. Descripción de las tareas

En esta sección se detallan las tareas planteadas, las cuales se agrupan en bloques con el fin de facilitar una gestión más eficiente y efectiva del proyecto. Cada tarea está acompañada de una breve explicación y el tiempo de dedicación. Se puede encontrar un resumen en la Tabla 1 que lista las tareas junto con su duración y recursos.

5.1.1. Gestión del proyecto [T1]

En este apartado se agrupan todas las tareas relacionadas con la gestión del proyecto. También incluye el desarrollo de documentación, presentación y reuniones. El tiempo total previsto para la realización de la gestión del proyecto es de 150 horas.

- **Contextualización y Alcance** [T1.1]: consiste en introducir y contextualizar el proyecto que se va a realizar. También define el objetivo y alcance del proyecto. El tiempo estimado para realizar esta tarea es 20h.
- **Planificación temporal** [T1.2]: Se elabora una planificación proyectiva detallada, se identifican los recursos requeridos y se implementa un proceso de gestión de riesgos. El tiempo estimado para realizar esta tarea es 15h.
- **Presupuesto** [T1.3]: Se elabora un cálculo del presupuesto para desarrollar el proyecto. El tiempo estimado para realizar esta tarea es 7.5h.
- **Informe de sostenibilidad** [T1.4]: Realizar un análisis exhaustivo del impacto medioambiental, económico y social que conlleva el proyecto. El tiempo estimado para realizar esta tarea es 7.5h.
- **Reuniones** [T1.5]: Son reuniones semanales con el director para hacer seguimiento del trabajo. Cada sesión está entre 30min y 1h. El tiempo estimado para realizar esta tarea es 25 horas.
- **Documentación** [T1.6]: Es una parte significativa de un Trabajo de Fin de Grado (TFG), demandando una dedicación continua. Se trata de una tarea paralela que se llevará a cabo a lo largo de todo el desarrollo del TFG. El tiempo estimado para realizar esta tarea es 60h.

- **Presentación** [T1.7]: Es la preparación de la presentación que dará lugar en el día de defensa de proyecto. El tiempo estimado para realizar esta tarea es 15h.

5.1.2. Trabajo previo [T2]

Es el proceso de preparación necesaria para empezar el proyecto, que incluye estudio de los conceptos y familiarización con las herramientas. El tiempo total previsto para la realización del trabajo previo es de 115 horas.

- **Estudio del estado del arte** [T2.1]: Hacer un estudio sobre el proyecto anterior de Jia Long. [3]. Incluye estudio teórico sobre aprendizaje automático y la implementación de diferentes algoritmos utilizados, que incluye RL y IL. El tiempo estimado para realizar esta tarea es 25h.
- **Familiarización de la Implementación** [T2.2]: Se hará un estudio del código desarrollado por Jia Long en el entorno de desarrollo Unity, realizando evaluación y experimentación con el modelo existente. El tiempo estimado para realizar esta tarea es 20h.
- **Familiarización con ML-Agents** [T2.3]: Se llevará a cabo una introducción a ML-Agents, comprenderá la funcionalidad y configuración del ML-Agents integrado dentro de Unity y la experimentación con algunos ejemplos preexistentes. El tiempo estimado para realizar esta tarea es 20h.
- **Estudio de la documentación** [T2.4]: Se realizará un estudio de toda la documentación relacionada con RL y IL. Incluye documentación de estudio científico o académico, vídeos y libros. El tiempo estimado para realizar esta tarea es 50h.

5.1.3. Mejora del entorno virtual en Unity [T3]

Durante la fase de desarrollo del entorno virtual en Unity, se procederá a la elaboración del espacio destinado a las simulaciones de partidos de pádel. Este proceso implica la consideración de los requisitos y reglas establecidos dentro del pádel, garantizando así la autenticidad y fidelidad de las representaciones virtuales de los encuentros deportivos. El tiempo total previsto para el desarrollo del entorno virtual es de 90 horas.

- **Simulación de la física en torno de entrenamiento** [T3.1]: Se realizará una mejora del entorno ya existente, substituyendo el sistema de física interno de Unity por uno propio adaptable a nuestras necesidades. El tiempo estimado para realizar esta tarea es 20h.
- **Implementación** [T3.2]: Se procederá a la implementación del entorno virtual siguiendo las directrices del diseño y los requisitos establecidos. El tiempo estimado para realizar esta tarea es 50h.
- **Testing** [T3.3]: Se realizarán pruebas para confirmar que la implementación funciona de acuerdo a lo esperado y cumple con las necesidades del proyecto. El tiempo estimado para realizar esta tarea es 20h.

5.1.4. Desarrollo de técnicas de aprendizaje por imitación [T4]

En este bloque se realizará todo el trabajo relacionado con el entrenamiento de agentes mediante IL, existiendo una fuerte dependencia con el bloque T3. El tiempo total previsto para el desarrollo del aprendizaje por imitación es de 150 horas.

- **Procesamiento de datos** [T4.1]: Se procederá a la revisión y transformación de los datos existentes para ajustarlos a los requisitos específicos del entrenamiento de IL en ML-Agents. El tiempo estimado para realizar esta tarea es 25h.
- **Diseño del modelo** [T4.2]: Identificar el algoritmo de IL más adecuado para adaptar nuestro modelo al entorno de entrenamiento. El tiempo estimado para realizar esta tarea es 25h.
- **Preparación del entorno de entrenamiento IL** [T4.3]: En este apartado se estudiarán los parámetros y requisitos necesarios para sincronizar el entrenador de IL con el entorno Unity, necesario para llevar a cabo un entrenamiento mediante IL. El tiempo estimado para realizar esta tarea es 30 horas.
- **Implementación** [T4.4]: se desarrollará el modelo con los requisitos diseñados previamente para integrar al entorno virtual ya implementado. El tiempo estimado para realizar esta tarea es 50h.
- **Testing** [T4.5]:Se realizarán pruebas para confirmar que la implementación funciona de acuerdo a lo esperado y cumple con las necesidades del proyecto. El tiempo estimado para realizar esta tarea es 20h.

5.1.5. Mejorar la interfaz gráfica de usuario [T5]

Tras la finalización del desarrollo del anterior, se integrará y mejorará la interfaz gráfica de usuario de la aplicación. El tiempo total previsto para la realización de la interfaz gráfica de usuario es de 35 horas.

- **Diseño** [T5.1]: Partiendo de la interfaz existente previamente, se diseñarán nuevas características para la integración. El tiempo estimado para realizar esta tarea es 10h.
- **Implementación** [T5.2]: Se procederá a la implementación de los iconos y las funcionalidades en la aplicación, lo que permitirá una experiencia de usuario completa e intuitiva. El tiempo estimado para realizar esta tarea es 15h.
- **Testing** [T5.3]: Se realizarán pruebas para confirmar que la implementación funciona de acuerdo a lo esperado y cumple con las expectativas. El tiempo estimado para realizar esta tarea es 10h.

5.2. Recursos

5.2.1. Recursos humanos

El desarrollo del proyecto está a cargo de diferentes roles de un equipo multidisciplinario que incluye un director de proyecto, un analista de datos, un programador y un ingeniero Q&A.

- **Jefe del proyecto**: Encargado de supervisar y planificar la realización del proyecto, coordinar a los miembros del equipo y mantener al cliente informado de su avance. También se encarga de escribir la documentación.
- **Analista de datos**: se encarga de recoger e interpretar los datos recopilados de un jugador, con el objetivo de facilitar el proceso de desarrollo.
- **Programador**: Es la persona que se ocupa de toda la parte de la implantación de código.

- **Q&A Engineer**: El control de calidad es un proceso fundamental en cualquier empresa que busca garantizar que sus productos o servicios cumplan con los estándares establecidos.

5.2.2. Recursos materiales

Para llevar a cabo este proyecto, se requieren los siguientes recursos materiales:

- **Computador de alto rendimiento**: Requiere una unidad de procesamiento gráfico (GPU) potente para poder realizar el proceso de entrenamiento de agente.
- **Unity**: es un motor gráfico que facilita el desarrollo del entorno virtual. La versión utilizada es Unity 2021.3.22f1, con licencia personal.
- **TeamGantt**: Herramienta para la elaboración de diagramas de Gantt. Con la versión gratuita.
- **Overleaf**: Herramienta para editar documentación. La versión utilizada es la versión gratuita.
- **GitHub**: Repositorio utilizado para el control de versiones. La versión utilizada es la versión gratuita.
- **Google Meet**: aplicación para realizar reuniones con el miembro de equipo.

ID	Tarea	Tiempo	Dependencias	Recursos	Rol
T1	**Gestión del proyecto**	**150h**	**-**	**-**	**-**
T1.1	Alcance	20h	[]	PC, Overleaf	JP
T1.2	Planificación	15h	[T1.1]	PC, Overleaf, TeamGantt	JP
T1.3	Presupuesto	7.5h	[T1.2]	PC, Overleaf	JP
T1.4	Informe de sostenibilidad	7.5h	[T1.2]	PC, Overleaf	JP
T1.5	Reuniones	25h	[]	PC, Google Meet	JP
T1.6	Documentación	60h	[]	PC, Overleaf	JP
T1.7	Presentación	15h	[T1.6]	PC, Overleaf	JP
T2	**Trabajo previo**	**115h**	**-**	**-**	**-**
T2.1	Estudio del estado del arte	25h	[]	PC	JP
T2.2	Familiarización de la Implementación	20h	[T2.1]	PC	JP
T2.3	Familiarización con ML-Agents	20h	[]	PC	JP
T2.4	Estudio de la documentación	50h	[]	PC	JP
T3	**Mejora del entorno virtual en Unity**	**90h**	**-**	**-**	**-**
T3.1	Simulación de la física en torno de entrenamiento	20h	[T2.1,T2.2]	PC, Unity	P
T3.2	Implementación	50h	[T3.1]	PC, Unity	P
T3.3	Testing	20h	[T3.3]	PC, Unity	QA
T4	**Desarrollo de técnicas de IL**	**150h**	**-**	**-**	**-**
T4.1	Procesamiento de datos	25h	[T3]	PC	AD
T4.2	Diseño del modelo	25h	[T4.1]	PC, Unity	JP
T4.3	Preparación del entorno de entrenamiento IL	30h	[T4.2]	PC, Unity	P
T4.4	Implementación	50h	[T4.3]	PC, Unity	P,QA
T4.5	Testing	20h	[T4.4]	PC, Unity	QA
T5	**Mejorar la interfaz gráfica de usuario**	**35h**	**-**	**-**	**-**
T5.1	Diseño	10h	[T3, T4]	PC, Unity	P
T5.2	Implementación	15h	[T5.1]	PC, Unity	P
T5.3	Testing	10h	[T5.2]	PC, Unity	QA
-	**Total**	**540h**	**-**	**-**	**-**

Tabla 1: Resumen de las tareas con la duración. Roles: JP - Jefe del proyecto, P - Programador, QA - *Q&A Engineer*, AD - Analista de datos. (Elaboración propia)

5.3. Gestión del riesgo

Antes de poder realizar el proyecto también hay que tener cuenta los posibles riesgos y obstáculos pueden ocurrir durante el desarrollo, con objetivos de prevenir los impactos. Los posibles riesgos identificados son los siguientes:

- **Insuficiencia de datos**: La obtención de datos de jugadores profesionales de pádel puede resultar compleja. Para abordar esta problemática, se propone una implementación que permita la extracción automatizada de la información de un modelo de aprendizaje por refuerzo (RL). El desarrollo de este modelo implicaría una inversión inicial de aproximadamente 10 horas, incluyendo la obtención de los datos y el procesamiento posterior de los datos para su correcta utilización.
- **Estimación del tiempo**: Existe el riesgo de que algunas tareas del proyecto excedan el tiempo estimado, lo que podría afectar el cronograma general. Para prevenir este escenario, se sugiere establecer puntos de control durante el desarrollo del proyecto, que es la reunión semanal con el tutor. En caso de que se observe un escaso de tiempo, se dará prioridad a las tareas centrales del proyecto, y eliminará tareas menos importantes como la interfaz de usuario.
- **Entrenamiento de agentes**: Durante el proceso de entrenamiento del agente, existe la posibilidad de que el modelo seleccionado presente una velocidad de entrenamiento inferior a la deseada. Esto se debe a que, en cada frame, el agente debe realizar un elevado número de comprobaciones, como la ubicación de la pelota y del oponente. Este procesamiento puede resultar costoso en términos computacionales. Para evitar este problema, es necesario realizar una evaluación del modelo en un punto determinado del entrenamiento con el objetivo de determinar si su rendimiento justifica la continuación del proceso. Para acelerar el proceso de entrenamiento, se pueden ejecutar dos modelos con diferentes hiperparámetros en paralelo. Esto es posible gracias a la disponibilidad de un equipo de alto rendimiento proporcionado por el director Carlos Andújar del grupo de investigación ViRVIG.

6. Gestión económica

En este apartado, se realizará una evaluación económica de los costes relacionados con el proceso de desarrollo, con el fin de obtener una perspectiva más clara sobre la viabilidad del proyecto. El presupuesto estimado incluye tanto el personal como los recursos materiales.

6.1. Presupuesto

6.1.1. Costes de personal

El costo del personal representa una parte significativa del presupuesto, aproximadamente el 80 %. Este gasto abarca los salarios correspondientes a cada rol previamente definido. Los datos sobre el costo por hora de cada rol se obtuvieron de *talent* [10]. Además de los salarios, se debe tener en cuenta el costo de la seguridad social, que suele ser alrededor de 30 % [11].

Rol	**Salario bruto por hora**	**SS**	**Salario + SS**
Jefe de proyecto	20.67 €	6.2 €	26.87 €
Analista de datos	11.95 €	3.585 €	15.535 €
Programador	14.62 €	4.386 €	19.01 €
Q&A Engineer	17.69 €	5.307 €	22.997 €

Tabla 2: Coste personal. (Elaboración propia)

ID	Tarea	Tiempo	Rol	Coste
T1	**Gestión del proyecto**	**150h**	**-**	**4018,71 €**
T1.1	Alcance	20h	JP	537,4 €
T1.2	Planificación	15h	JP	402,31 €
T1.3	Presupuesto	7.5h	JP	201,28 €
T1.4	Informe de sostenibilidad	7.5h	JP	201,28 €
T1.5	Reuniones	25h	JP	663,25 €
T1.7	Documentación	60h	JP	1612,2
T1.8	Presentación	15h	JP	402,31 €
T2	**Trabajo previo**	**115h**	**-**	**3081,05 €**
T2.1	Estudio del estado del arte	25h	JP	663,25 €
T2.2	Familiarización de la Implementación	20h	JP	537,4 €
T2.3	Familiarización con ML-Agents	20h	JP	537,4 €
T2.4	Estudio de la documentación	50h	JP	1343,5 €
T3	**Mejora del entorno virtual en Unity**	**90h**	**-**	**1790,64 €**
T3.1	Simulación de la física en torno de entrenamiento	20h	P	380,2 €
T3.2	Implementación	50h	P	950,5 €
T3.3	Testing	20h	QA	459,94 €
T4	**Desarrollo de técnicas de IL**	**150h**	**-**	**3130,945 €**
T4.1	Procesamiento de datos	25h	AD	388,375 €
T4.2	Diseño del modelo	25h	JP	663,25 €
T4.3	Preparación del entorno de entrenamiento IL	30h	P	570,32 €
T4.4	Implementación	50h	P,QA	1050,68 €
T4.4	Testing	20h	QA	459,94 €
T5	**Mejorar la interfaz gráfica de usuario**	**35h**	**-**	**705,22 €**
T5.1	Diseño	10h	P	190,1 €
T5.2	Implementación	15h	P	285,15 €
T5.3	Testing	10h	QA	229,97 €
-	**Total**	**540h**	**-**	**10935,215 €**

Tabla 3: Roles: JP - Jefe del proyecto, P - Programador, QA - *Q&A Engineer*, AD - Analista de datos. (Elaboración propia)

6.1.2. Costes genéricos

Espacio de trabajo

Durante el proceso de desarrollo del proyecto, el espacio de trabajo se centra principalmente en la habitación del ponente. Considerando que el precio de alquiler de una habitación en el alrededor de Barcelona varía entre 450€ y 650€ según la zona geográfica, hemos optado por un promedio de 550€. Dado que la duración del proyecto es de 4 meses, el costo total del espacio de trabajo asciende a 2200€.

Hardware

Para llevar al cabo el proyecto se requiere un ordenador sobremesa o un portátil, pero para poder facilitar el desarrollo de proyecto se requiere un ordenador con una GPU potente. En este caso utilizamos Nvidia 4070, tal que el coste del ordenador es 1500€ incluyendo los

accesorios del ordenador. Suponiendo que la vida útil los dispositivos es de 60 meses (5 años), la amortización sería de $5/60 \times 1500 = 125$€.

Software

Como se ha mencionado en el apartado de planificación, todos los *software* que utilizamos son versiones gratuitas. Por lo tanto, el costo total es de 0 €.

6.1.3. Contingencia

También hay que tener, hay que contemplar la posibilidad, algún obstáculo que aparece durante el desarrollo, lo cual pude generar un sobrecoste. Por lo general, los sobrecostos se encuentra cerca de 15 %. En siguiente tabla podemos observar más detalladamente la contingencia aplicada en cada ámbito.

Tipo	**Coste**	**Contingencia**
Personal	10935,215 €	1640,28 €
Espacio	2200 €	330 €
Software	0 €	0 €
Hardware	125 €	18,75 €
Total	**13260.215 €**	**1988,75 €**

Tabla 4: Tabla de contingencia 15 % (Elaboración propia)

6.1.4. Imprevistos

Para prevenir a los posibles obstáculos e imprevistos que puedan surgir durante el proyecto, se ha estimado un coste adicional. La Tabla 5 detalla el riesgo asociado a cada obstáculo.

- Incremento del tiempo: Este imprevisto tiene una alta probabilidad de ocurrir. El proyecto podría prolongarse en aproximadamente 30 horas durante el proceso de implementación, lo que equivale a un costo adicional de 570 €. La probabilidad de que esto suceda es del 25 %.
- Fallo del dispositivo hardware: En el caso de que se produzca un fallo en el ordenador sobremesa, será necesario reemplazar uno o más componentes del dispositivo. El coste de la reparación puede variar significativamente, dependiendo del componente que haya fallado. Por ejemplo, la reparación de una memoria RAM suele ser económica, mientras que la reparación de una tarjeta gráfica puede ser muy costosa. Para estimar el coste promedio de la reparación, se ha utilizado un valor del 200€. La probabilidad que esto pasa sería 10 % porque se trata de un dispositivo nuevo.

Imprevisto	**Coste**	**Riesgo**	**Coste total**
Aumento del tiempo de desarrollo	570 €	25 %	142,5 €
Fallo de ordenador de sobremesa	200 €	10 %	20 €
Total	**-**	**-**	**162,5 €**

Tabla 5: Costes imprevistos (Elaboración propia)

6.1.5. Coste total

En la Tabla 6 se detalla el coste total de cada apartado sumándolos globalmente.

Tipo	**Coste**
Personal	13260,215 €
Espacio	2200 € €
Software	0 €
Hardware	125 €
Contingencia	1988,75 €
Imprevistos	162,5 €
Coste total	**4608,465 €**

Tabla 6: Coste total del proyecto. (Elaboración propia)

6.2. Control de gestión

Resulta fundamental establecer mecanismos de control para monitorizar las predicciones realizadas y detectar cualquier desviación inesperada. Estos mecanismos nos permiten verificar si las predicciones se están cumpliendo e Identificar desviaciones inesperadas. Para controlar las desviaciones en un proyecto, se utilizarán las siguientes métricas:

$$\text{Desviación del coste} = (c_r - c_e) * hr \tag{1}$$

$$\text{Desviación del consumo} = (he - hr) * c_e \tag{2}$$

Donde c_r y c_e representa, coste estimado y coste real en horas. he y hr representa, horas estimadas y horas reales.

7. Sostenibilidad

7.1. Autoavaluación

Al completar la encuesta de sostenibilidad, he reconocido que mis conocimientos en la materia son aún limitados. Me ha sorprendido los indicadores para valorar sostenibilidad que hay algunos yo nunca había pensado. Alguna vez he reflexionado sobre el impacto social que podrá tener un proyecto de informática, pero no he profundizado en las dimensiones económica y ambiental.

Evaluar la sostenibilidad en el TFG es muy importante la hora de planificar y desarrollar, teniendo en cuenta tres dimensiones diferentes: ambiental, social y económica. Esta evaluación nos permite detectar los posibles impactos del proyecto y determinar su viabilidad.

En resumen, el trabajo de fin de grado nos permite abordar estas dimensiones de forma práctica. Esta oportunidad nos permite descubrir cómo desarrollar una solución sostenible que considere los aspectos económicos, ambientales y sociales.

7.2. Dimensión económica

El coste estimado para el proyecto se considera justificado por sus múltiples beneficios. No solo abarca los ámbitos de entretenimiento e investigación, sino que también representa una forma innovadora en el ámbito de videojuegos que reducirá los tiempos de desarrollo.

Actualmente, utilizamos aprendizaje por la imitación para entrega el modelo comparando con otras proyecto que utiliza aprendizaje por refuerzo. Pero utilizando IL se podría reducir el coste de entrenamiento consideradamente, ya que no hacer falta comenzar el entrenamiento de agente desde inicio.

El entrenamiento del modelo implica un costo inicial. Pero, el mantenimiento de la aplicación no tiene un costo adicional. Esto se debe a la poca probabilidad de actualización de la normativa del pádel.

7.3. Dimensión ambiental

En principio, el proyecto tiene un impacto ambiental relativamente bajo. Es importante considerar los recursos utilizados durante la fabricación, así como los dispositivos y la energía consumida durante el desarrollo.

Para mejorar estos aspectos, hemos optado por utilizar un ordenador de sobremesa en lugar de múltiples dispositivos simultáneamente para entrenar al agente, lo que minimiza significativamente el consumo energético. Además, hemos implementado reuniones remotas con el director para reducir la huella de carbono asociada a los desplazamientos.

Como mencionamos previamente en el apartado anterior, este proyecto es una continuación de otro. Podemos aprovechar el modelo ya previamente entrenado para desarrollar nuestro propio modelo, por lo tanto, ahorramos tiempo y energía al evitar iniciar desde cero.

7.4. Dimensión social

A nivel personal, el impacto más significativo ha sido el descubrimiento del apasionante mundo del aprendizaje automático. Esta técnica, con su amplia aplicabilidad en diversos campos, incluyendo el desarrollo de videojuegos, me ha sorprendido por su enorme utilidad y su potencial para revolucionar la industria.

Hablando de la vida útil del producto, la actual escasez de simuladores y videojuegos de pádel con IA para los personajes no jugadores (NPCs) permite este proyecto tenga oportunidad de ser muy útil. El proyecto utiliza técnicas de aprendizaje por imitación para crear un modelo con comportamientos más complejos, reducirá el tiempo de desarrollo. Esta nueva perspectiva aporta una alternativa al mercado, permitiendo una experiencia de juego más interactiva y rica.

En el mundo de videojuegos existe mucho simulador de deporte como tenis y baloncesto. Pero hay muy poco simulador o videojuegos de pádel y además utiliza una IA que imita al profesional. Entonces sí que existe una necesidad de un simulador que simule los límites de un jugador profesional, incluso para utilizarlo para encontrar el punto débil de los oponentes.

8. Estudio previo

8.1. Aprendizaje por refuerzo

8.1.1. RL basado en modelos y sin modelos

Dentro del Aprendizaje por Refuerzo (RL), podemos distinguir dos enfoques principales, clasificándolos entre algoritmos que necesitan un modelo del entorno y aquellos que pueden operar sin él. Esta distinción es importante en la hora de diseñar y implementar los algoritmos de RL.

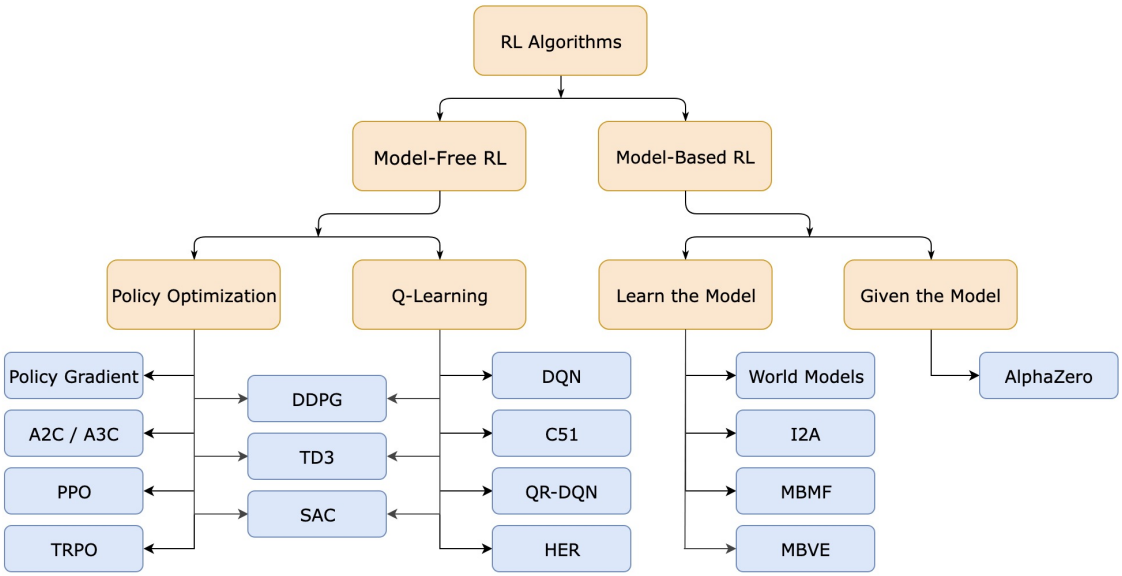


Figura 3: Taxonomía de algoritmos de RL. (Fuente: [12])

En el RL con modelo, el agente opera con una representación interna del entorno, denominada modelo del entorno. Esta herramienta le permite al agente anticipar las transiciones de estados y las recompensas asociadas a cada posible acción y estado. El modelo del entorno puede ser creado a partir de datos históricos, simulaciones o conocimiento experto. Por ejemplo, algoritmos como la Búsqueda en Profundidad (DFS) o la Búsqueda en Anchura (BFS) utilizan un modelo del entorno para determinar la secuencia de acciones que conducen al estado objetivo.

La existencia de un modelo del entorno permite al agente planificar sus acciones de manera más eficaz y explorar el espacio de estados de forma más dirigida, centrándose en áreas con mayor potencial de recompensa. Sin embargo, el rendimiento del RL con modelo depende en gran medida de la precisión de este modelo. Si el modelo es impreciso, las decisiones del agente también lo serán.

En el RL sin modelo, el agente no cuenta con una representación detallada del entorno y debe aprender a través de la interacción directa con él. A diferencia del RL con modelo, donde el agente utiliza un modelo para predecir las transiciones de estado y las recompensas asociadas, en el RL sin modelo el agente se basa en la experiencia de prueba y error. El agente estima una política para tomar decisiones.

El RL sin modelo ofrece una mayor robustez frente a cambios en el entorno, ya que el agente no depende de la precisión de un modelo predefinido. No obstante, el agente debe explorar todo el entorno para comprenderlo, lo cual puede ser un proceso lento y costoso, especialmente en entornos extensos o complejos.

8.1.2. Proceso de decisión de Márkov

El Proceso de Decisión de Márkov (MDP) es un marco matemático utilizado para modelar decisiones en entornos donde los resultados son en parte aleatorios. Este proceso se produce en cada iteración del agente en el Aprendizaje por Refuerzo (RL). Un proceso de decisión de Márkov es una 5-tupla (S, A, R, P, γ) donde:

- S es un conjunto de estados denominado espacio de estados.
- A es un conjunto de acciones denominado espacio de acciones.
- $R : S \times A \times S \to \mathbb{R}$, donde $r_t = R(s_t, a_t, s_{t+1})$ es la recompensa inmediata recibida de pasar de s_t al estado s_{t+1} mediante la acción a_t.
- $P : S \times A \to P(S)$, donde $P(s_{t+1}|s_t, a_t)$ es la probabilidad de transitar el estado s' al estado s en el momento $t + 1$ mediante la acción a.
- γ es el factor de descuento que representa la ponderación de las recompensas futuras en comparación con las presentes.

Propiedad de Markov

Esta propiedad establece que el futuro es independiente del pasado, dado el presente, se expresa mediante la siguiente fórmula:

$$P(s_{t+1}|s) = P(s_{t+1}|s_1...s_t) \tag{3}$$

Esta fórmula implica que el estado actual s_t contiene toda la información relevante de los estados pasados $(s_1, \ldots, s_{t-1})$. En otras palabras, conocer los estados pasados no tiene ningún impacto adicional en la predicción del futuro.

Espacio de observaciones y estados

El **espacio de observaciones (O)** representa todos los conjuntos de información que un agente puede observar en un momento dado. Estos conjuntos de información pueden ser discretos o continuos, e incluyen datos como la posición del agente, el estado del entorno y las acciones de otros agentes.

El **espacio de estados (S)** representa todos los posibles estados que el agente puede encontrarse. A diferencia del espacio de observaciones, el agente no siempre puede acceder directamente al espacio de estados. Sin embargo, el agente puede inferir su estado actual a partir de las observaciones que realiza.

Un ejemplo sería un laberinto. El espacio de estados del explorador del laberinto podría incluir su posición (x, y), su orientación y su estado actual. El espacio de observaciones del explorador podría incluir la distancia a las paredes y la presencia de obstáculos.

Espacio de acciones

El **espacio de acciones (A)** es el conjunto de todas las acciones posibles que el agente puede tomar. Este espacio puede ser discreto, con un número finito de acciones, o continuo, lo que implica una cantidad infinita de posibles acciones.

Retorno, episodios y el factor de descuento

El objetivo principal de un agente es tomar decisiones que maximicen la recompensa acumulada. Esto implica que el agente debe tomar decisiones que le permitan obtener la mayor cantidad de recompensas en el futuro, en lugar de centrarse solo en las recompensas inmediatas.

La recompensa total esperada, también conocida como retorno esperado, es la suma de todas las recompensas futuras que el agente espera recibir desde un momento dado. Se representa con la letra G y se calcula usando la siguiente fórmula:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T = \sum_{k=t+1}^{T} R_k \tag{4}$$

- G_t es el retorno esperado en el instante t.
- $R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T$ son las recompensas recibidas en los instantes t+1, t+2, t+3, ..., t+n.

Maximizar la recompensa acumulada esperada es aplicable en entornos con un rango temporal definido, que tiene un inicio y un fin. En estas situaciones, la interacción entre el agente y el entorno se divide en secuencias llamadas episodios. Cada episodio comienza con un estado inicial conocido, situando al agente en una situación bien definida. El episodio termina cuando el agente alcanza un estado final o cumple un objetivo específico.

En particular, existen situaciones en las que la interacción no se puede dividir en episodios discretos y nunca se detiene. Estos entornos se llaman entornos continuos. En estos casos, la suma de recompensas puede extenderse indefinidamente, lo que podría llevar a un valor infinito para la recompensa esperada.

Para resolver este problema, se utiliza el concepto de factor de descuento (γ). Este es un valor entre 0 y 1 que determina la importancia relativa de las recompensas futuras en comparación con las recompensas inmediatas. Se define de la siguiente manera:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{5}$$

La elección del valor adecuado para el factor de descuento depende de las características del entorno y los objetivos del agente. En general, se recomienda utilizar un valor de γ alto cuando las recompensas futuras son importantes y un valor de γ bajo cuando las recompensas inmediatas son cruciales.

8.1.3. Objetivo de optimización

En un Proceso de Decisión de Márkov (MDP), el objetivo es encontrar una política óptima que dirija al agente hacia las mejores decisiones en cada uno de los posibles estados. Esta política óptima se representa como π^*. Cuando el agente se encuentra en un estado s_t perteneciente al conjunto de estados S, basándose en la política actual π, elige una acción a del espacio de acciones. Esta acción lleva al agente a un nuevo estado s_{t+1} y obtiene una recompensa.

Existen dos enfoques principales para aprender la política óptima en RL:

- **Métodos basados en la política:** el agente aprende directamente la política, que define la acción a tomar en cada estado posible.
- **Métodos basados en el valor:** el agente aprende primero una función de valor, que asigna un valor numérico a cada estado. La política óptima se obtiene eligiendo la acción que maximiza el valor futuro esperado en cada estado.

8.1.4. Métodos basados en el valor

A diferencia de los métodos basados en la política, donde el objetivo es aprender directamente la política óptima, en los métodos basados en el valor, la política se define previamente y, por lo general, se trata de una estrategia sencilla, como la política *greedy*, que selecciona la acción que maximiza el valor inmediato o esperado a corto plazo.

La función de valor orienta al agente a explorar el entorno de manera efectiva y a dirigirse hacia estados con un mayor valor. A medida que interactúa con el entorno y recibe recompensas, el agente actualiza constantemente la función de valor, mejorando su estimación. Al finalizar el entrenamiento, utiliza estos valores como guía para tomar decisiones.

Función de valor

La función de valor en el RL se puede dividir en dos conceptos relacionados:

- **La función de estado-valor $V_\pi(s)$:** Estima la recompensa total acumulada que el agente puede esperar recibir si comienza en el estado s y actúa acuerdo con la política.

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s \right] \tag{6}$$

- **La función acción-valor $Q_\pi(s, a)$:** Representa el valor esperado de tomar una acción específica a en un estado s y seguir una política específica hasta el final del episodio.

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a \right] \tag{7}$$

Consideremos dos políticas, π y π', decimos que π es mejor o igual que π' denotado como $\pi \geq \pi'$. Si y solo si se cumple la siguiente condición $V_\pi(s) \geq V_{\pi'}(s)$ para todos los estados $s \in S$.

Una consecuencia importante de la introducción de este orden parcial es la existencia de al menos una política óptima. Esta política cumple la siguiente condición para todos los estados

$s \in S$ y todas las demás políticas π':

$$V_{\pi^*}(s) \geq V_{\pi'}(s) \tag{8}$$

Este razonamiento sería lo mismo para la función acción-valor:

$$Q_{\pi^*}(s) \geq Q_{\pi'}(s) \tag{9}$$

Ecuación de Bellman

Calcular directamente las funciones de valor $V(s)$ y $Q(s,a)$ usando las ecuaciones 6 y 7 puede ser un proceso lento y poco práctico, sobre todo en entornos complejos.

Para simplificar este proceso, se utilizan las Ecuaciones de Bellman, que descomponen el problema en subproblemas más pequeños de manera recursiva, aprovechando la estructura subóptima del problema.

En primer lugar, simplificamos la función de retorno G_t:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + ...) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{10}$$

Sustituye esta función en la función de estado-valor $V_\pi(s)$:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi \left[G_t \,|\, S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s \right] \\ &= \sum_a \pi\left(a \,|\, s\right) \sum_{s'} \sum_r p\left(s', r \,|s, a\right) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \,\middle|\, S_{t+1} = s' \right] \right] \\ &= \sum_a \pi\left(a \,|\, s\right) \sum_{s',r} p\left(s', r \,|s, a\right) \left[r + \gamma v_\pi(s') \right] \end{aligned} \tag{11}$$

En la función acción-valor $Q_\pi(s,a)$ sería lo mismo:

$$Q_\pi(s,a) = \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma \sum_{a'} \pi(a' \,|\, s') Q_\pi(s', a') \right] \tag{12}$$

Para obtener el valor de un estado, únicamente se requiere información sobre el estado actual y el siguiente en vez de todos los estados. La ecuación de Bellman nos indica que el valor del estado actual es la suma de la recompensa inmediata esperada, $\sum_{s',r} p(s', r \,|\, s, a) r$, sobre todos los posibles estados siguientes s' y recompensas r, y el valor descontado de las recompensas futuras, multiplicado por el factor de descuento γ y ponderado por la política π.

8.1.5. Métodos basados en la política

A diferencia de los métodos basados en valores, que se enfocan en evaluar cada estado individualmente, que los agentes aprenden directamente la estrategia óptima para tomar la mejor decisión en cada situación.

Los métodos de gradiente de política (PG) utilizan un enfoque de aprendizaje iterativo para ajustar la política del agente basándose en su interacción con el entorno, en lugar de utilizar funciones de valor para tomar decisiones.

La política está representada por un conjunto de parámetros $\theta \in \mathbb{R}^{d'}$ que definen cómo el agente debe actuar en su entorno. Generalmente, son un vector en un espacio euclidiano de dimensión d' y se define como:

$$\pi(a|s,\theta) = \mathrm{P}\{A_t = a|S_t = s, \theta_t = \theta\} \tag{13}$$

donde P es la probabilidad de elegir la acción a dado el estado s, parametrizada por θ.

Mediante la política parametrizada $\pi(a|s,\theta)$, el objetivo es ajustar estos parámetros θ de manera que maximicen una medida de rendimiento $J(\theta)$, que está relacionada con la cantidad total de recompensa esperada.

Para optimizar esta medida de rendimiento, utilizamos un algoritmo de gradiente ascendente. En cada paso de tiempo t, actualizamos los parámetros θ_t en la dirección que maximiza $J(\theta_t)$, con la siguiente fórmula:

$$\theta_{t+1} \leftarrow \theta_t + \alpha\widehat{\nabla J(\theta_t)} \tag{14}$$

- α es la tasa de aprendizaje, la cual determina el tamaño del paso que se da en la dirección del gradiente durante la actualización de los parámetros.
- $\widehat{\nabla J(\theta_t)}$ es una valoración aleatoria del gradiente de la función de rendimiento con respecto a los parámetros θ_t. Esta estimación indica la dirección en la que se deben ajustar los parámetros para mejorar el rendimiento.

El proceso de estimación del gradiente y ajuste de parámetros se repite iterativamente, actualizando los parámetros θ en cada paso de tiempo. El objetivo es que la política converja a una solución óptima.

8.2. Aprendizaje por la imitación

A diferencia del aprendizaje por refuerzo, en el aprendizaje por imitación (IL), los agentes intentan aprender una política óptima que imita una demostración generalmente proporcionada por un experto. Esto permite a los agentes aprender comportamientos complejos, observando y replicando las acciones del experto.

8.2.1. Clonación de Comportamiento

La clonación de comportamiento (BC) [13] es una técnica sencilla que se basa en aprender la política del experto mediante aprendizaje supervisado. El agente observa y memoriza las acciones del experto con el objetivo de imitarlas.

A partir de las demostraciones del experto, los datos se dividen en pares estado-acción, tratándolos como ejemplos independientes e idénticamente distribuidos (i.i.d.). Luego, se aplica aprendizaje supervisado para predecir futuras acciones basándose en los datos actuales.

Aunque BC puede ser efectivo en ciertos casos, presenta diversas limitaciones. Una de ellas es la suposición de independencia, ya que se considera que los pares estado-acción son independientes entre sí. Sin embargo, en la mayoría de los entornos reales, la acción realizada en un estado influye directamente en el siguiente estado. Otra limitación es la acumulación de errores; si el agente comete un error en un estado, puede llegar a un nuevo estado que no estaba contemplado en las demostraciones del experto.

8.2.2. Aprendizaje por refuerzo inverso

El aprendizaje por refuerzo inverso (IRL) [14] es otro enfoque del aprendizaje por imitación (IL). Como su nombre indica, invierte el problema del aprendizaje por refuerzo (RL), deduciendo la función de recompensa a partir de las acciones realizadas por un experto.

El objetivo de IRL es encontrar una función de recompensa que haga que la política del experto sea óptima, y mediante esta función derivar una política óptima a través de aprendizaje por refuerzo. El proceso puede describirse en los siguientes pasos:

1. Recopilar los datos del comportamiento del experto en forma de trayectorias, que son secuencias de estados y acciones observadas.
2. Utilizando métodos basados en máxima verosimilitud, se ajusta una función de recompensa para maximizar la probabilidad de obtener las demostraciones del experto mediante la función ajustada. Este algoritmo inicializa una función de recompensa paramétrica $R_\theta(s, a)$, donde θ son los parámetros que queremos aprender.
3. Una vez obtenida la función, se utiliza RL para obtener una política π^* que maximiza la recompensa esperada.
4. Se realiza la comparación entre la política π^* obtenida y la política del experto π_E. Esta comparación se efectúa comparando las trayectorias generadas por π^* con las trayectorias observadas del experto. Si el resultado no es similar al del experto, entonces se repite el proceso desde el paso 2. De lo contrario, el proceso se da por terminado.

8.2.3. Aprendizaje generativo por imitación adversarial

Comparado con los algoritmos anteriores, el BC enfrenta problemas de robustez ante la acumulación de errores. Por otro lado, el aprendizaje por refuerzo inverso (IRL) es costoso debido a la necesidad de ajustar una función de recompensa para alcanzar una política óptima. Para superar estos desafíos, el aprendizaje generativo por imitación adversarial (GAIL) [15] combina las ideas de las redes adversariales generativas (GAN) con el IRL, permitiendo imitar de manera eficiente y precisa el comportamiento de un experto.

El GAN consiste en entrenar dos modelos al mismo tiempo. El modelo generativo G aprende a generar muestras que se asemejan a las muestras reales, mientras que el modelo discriminador D aprende a distinguir si una muestra es real o ha sido generada por G. El objetivo de G es crear muestras que el discriminador no pueda diferenciar de las reales, y el objetivo de D es maximizar su precisión en distinguir las muestras. Tras múltiples iteraciones, llegará un punto en que el generador conseguirá producir muestras que el discriminador no

podrá diferenciar de las muestras expertas, indicando que el agente ha aprendido una política similar a la del experto.

Mediante la combinación de GAN e IRL, GAIL introduce la recompensa intrínseca. Cuando el generador consigue engañar al discriminador, se le asigna una recompensa. Esto hace que el discriminador se vuelva más estricto y preciso, forzando al generador a mejorar aún más su política.

El objetivo del algoritmo es llegar un punto donde D ya no puede distinguir los datos generados por G.

$$\mathbb{E}_{\pi}[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi)$$

- $\mathbb{E}_{\pi}$: Representa la esperanza matemática bajo la política π.
- $\log(D(s,a))$: Es el logaritmo de la salida del discriminador D cuando recibe un par estado-acción (s,a). Donde $D_w : S \times A \to (0,1)$ con el peso w.
- $\mathbb{E}_{\pi_E}$: Representa la esperanza matemática bajo la política experta π_E.
- $H(\pi)$: La entropía de una política π mide la incertidumbre o aleatoriedad de dicha política. Con un valor pequeño de λ, se minimiza la entropía, haciendo que la política se vuelva más determinista.

$\mathbb{E}_{\pi}[\log(D(s,a))]$ incentiva al D a asignar un valor alto a las acciones (s,a) generadas por la política del generador G. Por otro lado, $\mathbb{E}_{\pi_E}[\log(1 - D(s,a))]$ incentiva a asignar un valor bajo a las acciones (s,a) generadas por la política experta.

Algorithm 1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}\left[\nabla_w \log(D_w(s,a))\right] + \hat{\mathbb{E}}_{\tau_E}\left[\nabla_w \log(1 - D_w(s,a))\right]$$

5: Take a policy step from θ_i to θ_{i+1} using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i}\left[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\right] - \lambda \nabla_\theta H(\pi_\theta),$$

where $Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}\left[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}\right]$
6: **end for**

8.3. Unity ML-Agents

ML-Agents de Unity proporciona un SDK para desarrollar entornos de aprendizaje automático dentro de Unity. Permite a los desarrolladores crear entornos virtuales donde los agentes pueden aprender y mejorar utilizando técnicas de aprendizaje automático. Además de ofrecer un conjunto de herramientas para el desarrollo de estos entornos, el toolkit incluye implementaciones de algoritmos de aprendizaje automático basados en aprendizaje automático.

8.3.1. Componentes de SDK

En la documentación de ML-Agents [16] detalla que el SDK de ML-Agents está compuesto por los siguientes componentes:

- **Entorno de aprendizaje:** Es el núcleo de la aplicación, donde se desarrolla el aprendizaje. Se trata de un entorno virtual, construido dentro de una escena de Unity.
- **API de bajo nivel de Python:** permite una interacción y manipulación precisas del entorno de aprendizaje desde un entorno Python externo. Utiliza un componente llamado Comunicador, que actúa como puente entre la API y el entorno de aprendizaje. Esta API se utiliza principalmente para interactuar con la *Academy*. Este componente controla los comportamientos de los agentes, asegurando que las acciones de cada uno estén sincronizadas en el entorno.
- **Comunicador externo**: Sirve como puente entre el entorno de aprendizaje en Unity y la API de Python. Facilita el intercambio de información y la ejecución de comandos entre ambos sistemas.
- **Entrenador basado en Python:** Este componente emplea algoritmos de IA en Python para entrenar al agente y se comunica a través de una API de bajo nivel de Python.
- **Wrappers de Gym y PettingZoo**: Son adaptadores que permiten que Unity ML-Agents se comunique con las bibliotecas de Gym y PettingZoo, permitiendo que el entorno de aprendizaje utilice los algoritmos de aprendizaje automático disponibles en estas bibliotecas.

Dentro del entorno virtual existen dos componentes importantes que ayudan a organizar la escena de Unity:

- La **clase Agent**: se puede integrar como *GameObject* dentro de la escena de Unity. Los agentes generan observaciones del entorno, que son las informaciones que el agente utiliza para comprender su situación actual. Basándose en estas observaciones, el agente realiza una acción y recibe una recompensa asociada con esa acción.
- El **Comportamiento (Behavior)** de un agente: Representa el conjunto de acciones que un agente puede ejecutar en un entorno. Los agentes responderán con una determinada acción basada en la información que reciben del entorno. Hay tres tipos de comportamientos:
    - **Aprendizaje**: El comportamiento aún no está definido y está destinado a ser entrenado a través de algoritmos de aprendizaje automático.
    - **Heurística**: El comportamiento de agente está controlado por un conjunto de reglas predefinido en el código.
    - **Inferencia**: Se incorpora una red neuronal entrenada que el agente utiliza para tomar decisiones. De hecho, un comportamiento de aprendizaje se convierte en un comportamiento de inferencia una vez finalizado su entrenamiento.

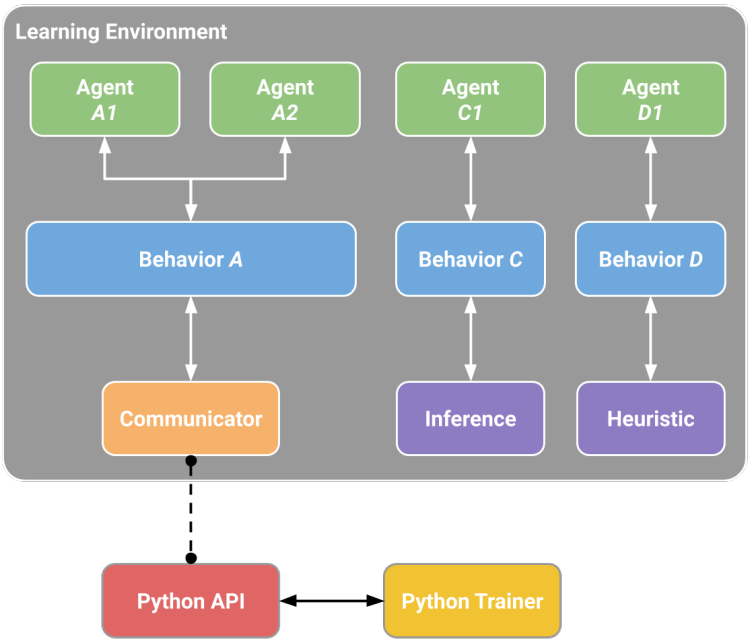


Figura 4: Diagrama de componentes de ML-Agents (Fuente: [16])

8.3.2. Métodos de entrenamiento implementados en ML-Agents

En ML-Agents se ofrecen muchos algoritmos de aprendizaje, pero dentro de RL, los más importantes son Proximal Policy Optimization (PPO) [17] y Soft Actor-Critic (SAC). Debido a la necesidad de entrenar varios agentes en el mismo entorno, se optó por utilizar PPO, ya que permite combinarse con *Self-play* [16].

ML-Agents dispone de varios algoritmos para entrenar múltiples agentes en un mismo entorno, como por ejemplo: Self-Play [18] y MultiAgent Posthumous Credit Assignment (MA-POCA). En el Self-Play permite que varios agentes compitan entre sí en el mismo entorno. Esto permite que los agentes aprendan unos de otros y desarrollen estrategias más complejas. Por otra parte, MA-POCA es un algoritmo específico para entornos multi-agente cooperativos. Asigna recompensas a los agentes basándose en el éxito del equipo en su conjunto.

En el mundo del pádel, un deporte de equipo, se podría considerar un entorno multi-agente. Sin embargo, el enfoque se centra principalmente en el entrenamiento de agentes individuales. Considerando las características y limitaciones de cada método, el Self-Play se destaca como la opción más adecuada para entrenar agentes en el entorno de pádel, especialmente por su compatibilidad con PPO.

A medida que los agentes interactúan con su entorno y reciben retroalimentación sobre sus acciones, aprenden a tomar decisiones que no solo maximizan su propia ganancia, sino que también consideran el éxito general del equipo. Esto puede conducir a comportamientos cooperativos, donde los agentes trabajan juntos hacia un objetivo común, incluso si inicialmente estaban centrados en maximizar su propia recompensa individual.

Self-play

A diferencia del RL tradicional, donde el agente en entrenamiento se enfrenta a un oponente estático, el método Self-Play introduce un oponente dinámico que se adapta durante el proceso

de aprendizaje. En este método, un agente en entrenamiento compite contra otro agente, manteniendo su estrategia constante, mientras que el oponente ajusta la suya utilizando versiones actualizadas de la estrategia del agente en entrenamiento. A medida que el agente a entrenar se enfrenta a un oponente cada vez más fuerte, su propia política también evoluciona y se vuelve más efectiva, creando una mejora mutua entre ambos agentes.

El Self-Play no se limita a entornos de uno contra uno. Se puede extender a escenarios con múltiples agentes utilizando equipos. Por ejemplo, en pádel de dos contra dos, cada equipo podría utilizar Self-Play para mejorar sus estrategias de manera conjunta.

Proximal Policy Optimization

La Optimización de la Política Proximal (PPO) [19] utiliza técnicas de aprendizaje por refuerzo basadas en gradientes de política. Un desafío que enfrenta el aprendizaje por refuerzo es la inestabilidad de aprendizaje que el agente se comporte de manera errática o que no converja a una política óptima. La PPO mejora este problema mediante la introducción de un *"recorte" (clipping)* en las actualizaciones de la política. El recorte limita la magnitud en la que la política puede cambiar en cada iteración.

La PPO usa la función objetivo sustituta con límite (*clipped surrogate objective function*) que equilibra la recompensa esperada con la restricción de cambios drásticos en la política. Utiliza el término substituta porque intenta aproximar la función objetivo real y la función PPO tiene siguiente forma:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}\Big[\min(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \tag{15}$$

- $r_t(\theta)$: Es la relación entre la política actual $\pi_\theta(a_t|s_t)$ y la anterior $\pi_{\theta_{\text{old}}}(a_t|s_t)$, representa de siguiente manera:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{16}$$

- $\hat{A}_t$: La ventaja estimada en el paso de tiempo t.
- ϵ: El hiperparámetro que varía entre 0 y 1 que controla el rango del recorte, suele ser pequeño 0.1 o 0.2.

El primer término, $r_t(\theta)\hat{A}_t$, representa el objetivo original sin restricciones. Por otro lado, $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ introduce una función de *clip* que limita la magnitud de la actualización de la política, restringiendo la relación de probabilidad $r_t(\theta)$ a un rango de $[1-\epsilon, 1+\epsilon]$. Se seleccionará el mínimo entre estos dos términos para proporcionar una cota inferior en cada actualización, garantizando que la política no se desvíe excesivamente del objetivo original.

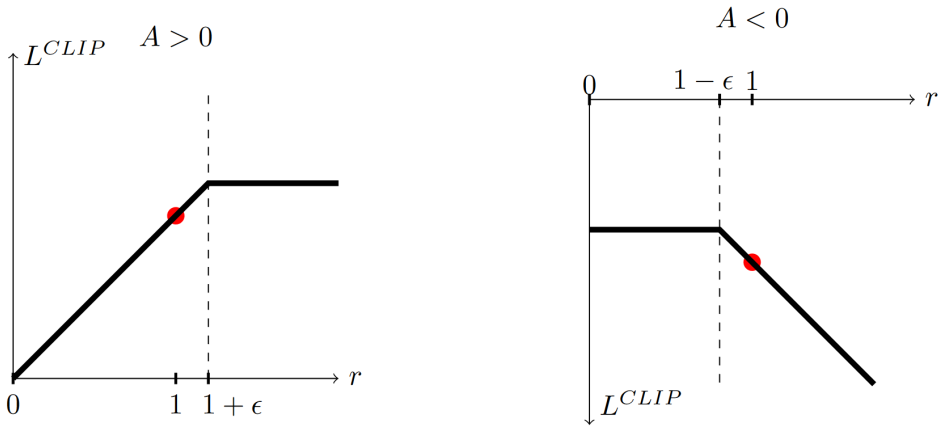


Figura 5: El gradiente del sustituto L^{CLIP} con respecto a la relación $r_t(\theta)$ varía dependiendo de la ventaja $\hat{A}_t$. (Fuente: [19])

8.3.3. Configuración del entrenamiento

En esta sección se proporcionan detalles sobre las configuraciones y los hiperparámetros que serán utilizados en el entrenamiento de agentes en ML-Agents. Se recomienda consultar la documentación de ML-Agents [20] para obtener una explicación más completa de estos parámetros.

Ajuste	Descripción
`trainer_type`	Define el tipo de entrenamiento a utilizar, que puede ser `ppo`, `sac` o `poca`.
`summary_freq`	La cantidad de veces que se debe realizar una operación o ejecutar un conjunto de experiencias antes de que se genere una visualización o un modelo.
`max_steps`	Número total de pasos de entrenamiento
`keep_checkpoints`	La cantidad de copias del estado del modelo que se deben mantener guardadas durante el entrenamiento del modelo.
`even_checkpoints`	se refiere a si los puntos de guardado del modelo deben distribuirse de manera uniforme a lo largo del proceso de entrenamiento.
`hyperparameters/ learning_rate`	La tasa de aprendizaje inicial que se utiliza en el algoritmo de descenso de gradiente durante el entrenamiento de un modelo de aprendizaje automático.
`hyperparameters/ learning_rate_ schedule`	La estrategia que se utiliza para ajustar la tasa de aprendizaje a lo largo del tiempo durante el entrenamiento de un modelo de aprendizaje automático.
`hyperparameters/ batch_size`	Define cuántas experiencias se utilizan en cada paso del descenso de gradiente, influyendo en la frecuencia de actualización de los parámetros del modelo y en el uso de memoria durante el entrenamiento.
`hyperparameters/ buffer_size`	La cantidad de experiencias que se deben acumular en la memoria temporal antes de que el algoritmo PPO actualice la política del agente,
`network_settings/ num_layers`	Especifica cuántas capas ocultas (*hidden layers*) tendrá cada una de las redes neuronales (del actor y del crítico) dentro del modelo, determinando la profundidad y complejidad de estas redes.
`network_settings/ hidden_units`	El número de neuronas en cada capa oculta de las redes neuronales.

Tabla 7: Configuración común (Fuente: [20])

Configuración específica de PPO

Ajuste	Descripción
`hyperparameters/ beta`	Coeficiente de ajuste para el término de entropía.
`hyperparameters/ epsilon`	Define el margen de cuánto puede diferir la nueva política de la anterior en cada actualización durante el entrenamiento.
`hyperparameters/ beta_schedule`	Cómo debe variar el valor de `beta` durante el proceso de entrenamiento.
`hyperparameters/ epsilon_schedule`	Cómo debe variar el valor de `epsilon` durante el proceso de entrenamiento.
`hyperparameters/ num_epoch`	Es el número de veces que se recorre el conjunto de datos de experiencias almacenadas en el buffer durante el proceso de optimización mediante el algoritmo de gradiente descendente.

Tabla 8: Configuración específica de PPO. (Fuente: [20])

Configuración de Self-Play

Ajuste	Descripción
`save_steps`	El número de pasos de entrenamiento que deben transcurrir antes de realizar una copia de seguridad de la política que se está entrenando
`team_change`	Especifica la frecuencia con la que el equipo que aprende cambia los oponentes con los que se entrena en el entorno de aprendizaje por refuerzo. Un valor más grande puede mejorar la capacidad del equipo, pero también aumenta el riesgo de *overfitting* a las estrategias de los oponentes actuales.
`swap_steps`	Se refiere al número de pasos "fantasma" que deben transcurrir antes de cambiar la política del oponente. Un *paso fantasma* es un paso dado por un agente cuya política está fija y no está siendo actualizada o aprendiendo durante estos pasos.
`play_against_ latest_model_ratio`	Determina la probabilidad de que el agente en entrenamiento se enfrente a su versión más reciente contra versiones anteriores de su política. Por ejemplo, si *play_against_latest_model_ratio* se establece en 0.7, hay un 70 % de probabilidad de que el agente se enfrente a la versión más reciente de su propia política en cada episodio de entrenamiento. Con una probabilidad de 1 - 0.7 = 0.3 (30 %), el agente se enfrentará a una de las versiones anteriores de su política, seleccionada aleatoriamente.
`window`	El número de copias pasadas de la política del agente que se mantienen en una ventana deslizante, afectando la diversidad de comportamientos a los que el agente se expone durante el entrenamiento. Por ejemplo, si `window` se establece en 10, se guardarán las 10 versiones más recientes de la política del agente. Cada vez que se crea una nueva copia de la política, la copia más antigua se elimina para mantener el tamaño de la ventana constante.

Tabla 9: Configuración de self-play. (Fuente: [20])

Recompensas intrínsecas de GAIL

Ajuste	Descripción
`gail/strength`	Ajusta la magnitud de las recompensas proporcionadas durante el proceso de entrenamiento.
`gail/gamma`	El factor de descuento determina la importancia de las recompensas futuras en comparación con las recompensas inmediatas.
`gail/demo_path`	Ruta del directorio donde guarda `.demo` para entrenar el agente.
`gail/ network_settings`	Especificaciones de la red neuronal.
`gail/learning_rate`	Factor de aprendizaje determina el tamaño de los pasos que da el algoritmo al ajustar los pesos del modelo en respuesta al error observado en cada iteración de entrenamiento.
`gail/use_actions`	Determina si el discriminador del modelo debe usar tanto las observaciones (estados) como las acciones, o solo las observaciones para hacer sus predicciones.
`gail/use_vail`	Habilita el *variational bottleneck* del discriminador de GAIL, que regulariza el discriminador para promover la generalización y evitar el overfitting

Tabla 10: Configuración de las recompensas intrínsecas procedentes de GAIL. (Fuente: [20])

9. Desarrollo

En este capítulo se expondrán las implementaciones del proyecto, y se explicarán las diferentes implementaciones del proyecto original, detallando las distintas modificaciones realizadas con respecto al proyecto original de Jia Long. [3]

9.1. Desarrollo del entorno virtual

9.1.1. Reglamento de juego

Para entrenar a los agentes de manera efectiva, es necesario establecer reglas en el entorno virtual. Estas reglas tienen como objetivo facilitar el entrenamiento mientras se mantienen fieles a la realidad del deporte. Las reglas son:

- Dado que RL e IL utilizan episodios para el entrenamiento, se simplifica un partido de pádel a un único set que se repite indefinidamente, en lugar de los tres sets de un partido real.
- Para visualizar los resultados, cada set en el pádel se puntúa de la siguiente manera: 0, 15, 30 y 40 puntos. Para ganar un juego, un equipo debe alcanzar los cuatro puntos. En caso de empate a 40 puntos, el juego continúa hasta que un equipo logre una ventaja de dos puntos.
- Al inicio de la ejecución, el servicio siempre lo realiza el equipo 2 desde el lado derecho de la pista. Al finalizar un episodio, se cambia el lado de servicio.
- Durante el servicio, para realizar un saque válido, la pelota debe botar una vez en el suelo detrás de la línea de saque. Al enviar la pelota al campo contrario, esta debe botar primero dentro del cuadro de saque del rival en diagonal. Sin embargo, la pelota no puede chocar contra la pared después de haber rebotado en el suelo.
- Cuando se devuelve la pelota, se permite rebotar una vez contra una pared de tu campo, pero no está permitido golpearla dos veces seguidas contra la misma pared. Tras el golpeo, la pelota no puede botar en el propio campo y pared.
- Tras recibir la pelota, el jugador solo tiene permitido un bote en el suelo antes de devolverla. Rebotar la pelota en la pared está permitido de forma ilimitada. Sin embargo, si la pelota toca el suelo dos veces consecutivas, el punto se le otorga al equipo contrario.
- Si, después de devolver la pelota, ésta toca la red o a un compañero de equipo, el punto se otorga al equipo contrario. En cambio, si la pelota golpeada toca a un oponente, se obtiene un punto.

9.1.2. Descripción del Entorno Virtual

Ahora describiremos la pista que forma parte del entorno virtual. La pista originalmente desarrollada se basa en una pista oficial del World Padel Tour. Dado que esta pista refleja fielmente una pista real, no se han realizado muchas modificaciones en su estructura.

La pista tiene una medida de 10x20m (Figura 6), el rango de la pista se extiende desde $x \in [-5, 5]$, y $z \in [-10, 10]$. En medio de la pista existe una red de 0.92m. Las paredes que rodean la pista tienen una altura de 3m y con una coeficiente de restitución $e = 0.5$. Y el coeficiente de restitución del suelo es $e = 0.76$.

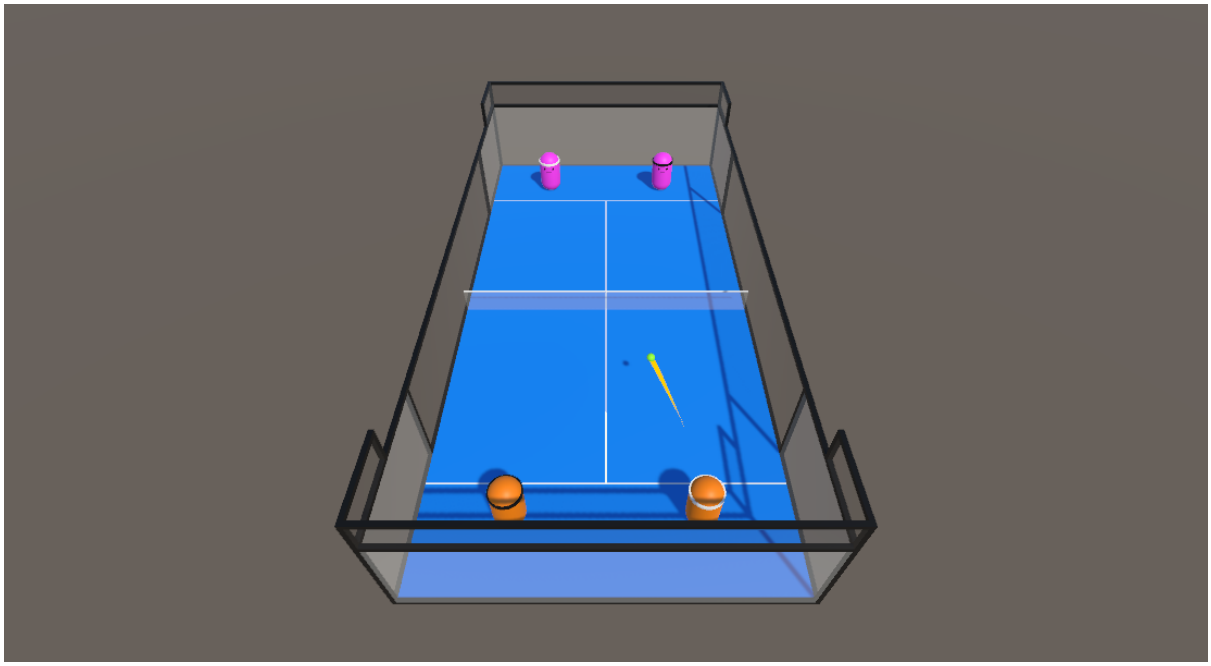


Figura 6: Entorno virtual. (Fuente: [3])

Los agentes (jugadores de la partida), se identifican por el color de su equipo: naranja para el Equipo 1 y morado para el Equipo 2. Al iniciar cada partida, los agentes se ubican en posiciones predefinidas fijas, como se muestra en la Figura 7. Estas posiciones se intercambian en función del equipo que inicia el saque.

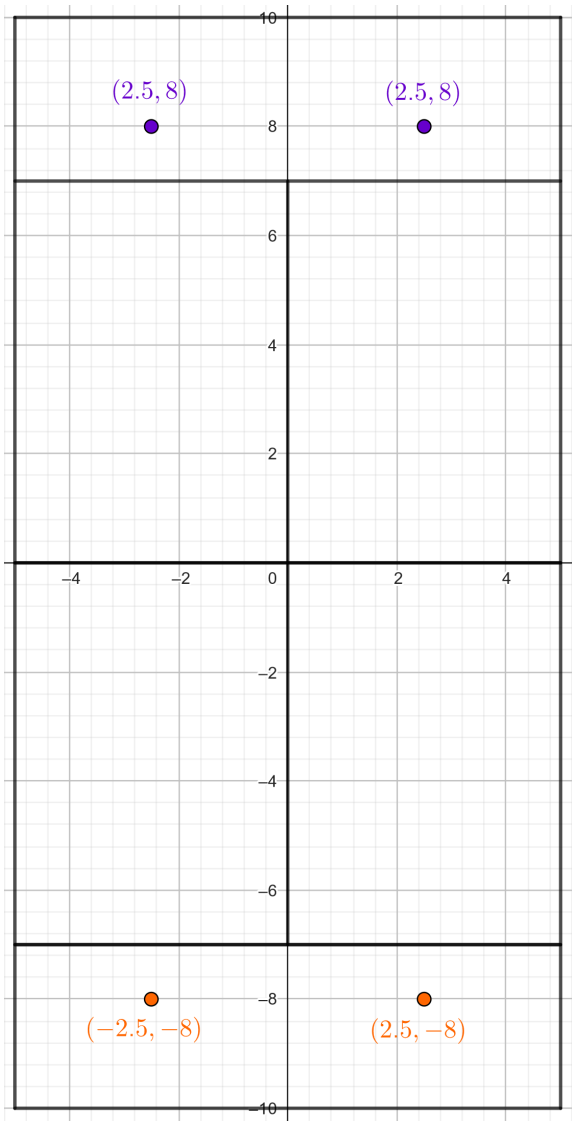


Figura 7: Posiciones de servicio del agente. (Fuente [3])

9.2. Controladores de la lógica

9.2.1. Controlador del entorno

El controlador del entorno actúa como intermediario entre los demás controladores; es la capa superior que tiene la visibilidad de los demás. Cuando un controlador requiere información de otro, envía una solicitud al controlador del entorno, quien se encarga de hacer la consulta al controlador responsable mediante una función de consulta.

El controlador del entorno contiene diferentes variables globales; las más importantes son:

- **Reward*, son las variables que acaban en `Reward`. Cuando el agente realiza una determinada acción, se le asigna la recompensa correspondiente. Por ejemplo, `WinningReward` es la recompensa por ganar un punto, y `HittingBallReward` es la recompensa por devolver la pelota.
- *DebugMode*, la variable que utiliza para activar el modo depuración. Este modo nos permite visualizar la trayectoria de la pelota y las informaciones relacionadas con el agente y la pelota.
- *RecordingDemonstrations*, La variable utilizada para activar el modo de grabación de una demostración, que posteriormente se emplea para entrenar con el algoritmo de IL.

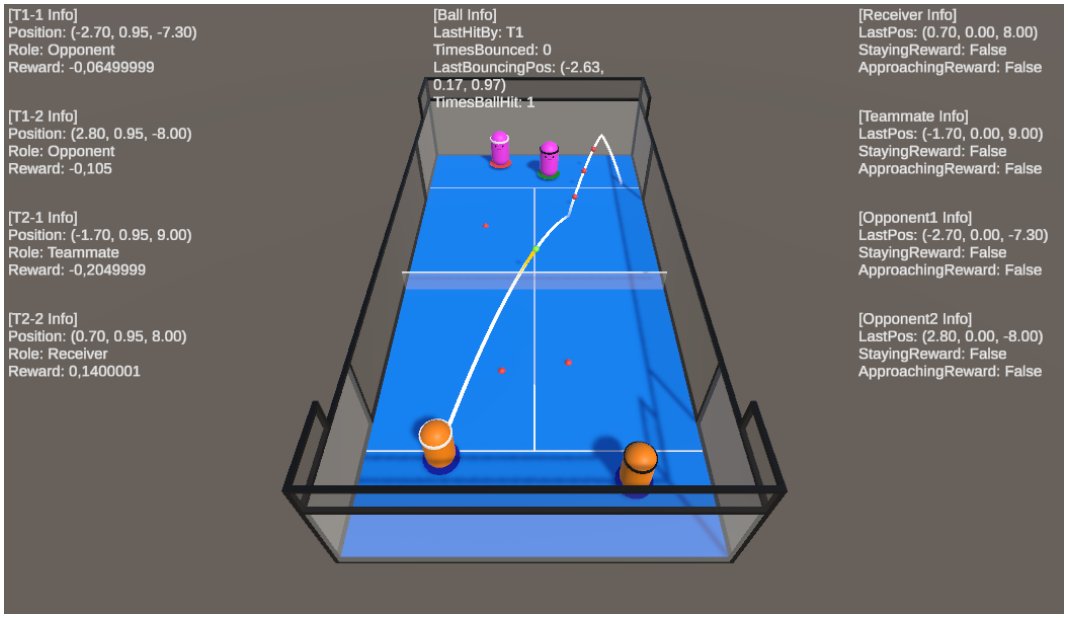


Figura 8: El modo de depuración. (Fuente: [3])

En este controlador no se han realizado cambios en la funcionalidad existente, sino que se han añadido nuevas características. Por ejemplo, se ha introducido la variable *logEnabled*, que activa el modo de registro de información del entorno. Para ello, se ha creado una nueva clase llamada *Logger*, que se instancia al inicio de la ejecución. Cuando *logEnabled* está activado, el *Logger* recopila información relevante del entorno en cada actualización. Esta información incluye la posición de la pelota y los jugadores, los tipos de golpes realizados durante la partida, entre otros datos. Una vez obtenidos, los datos se procesan y se guardan en el formato correcto. Cuando se ha recopilado suficiente información, todo se guarda automáticamente en un archivo Excel que se utilizará posteriormente para generar gráficos.

9.2.2. Generador de estadísticas de juego

Este script de Python tiene como objetivo principal procesar datos del entorno virtual almacenados en formato .xlsx. El script utiliza las librerías *pandas* y *numpy* para manipular y procesar los datos, y la librería *seaborn* para generar las gráficas correspondientes. Las gráficas resultantes pueden observarse en la sección de experimentación 10.

9.2.3. Controlador del servicio

Este controlador asegura que cada servicio comience de manera determinista, colocando la pelota en la misma posición inicial para realizar el servicio, sin importar el equipo o lado que saque.

Al inicio de cada punto, los agentes se colocan en posiciones fijas y la pelota se sitúa a una distancia predeterminada del servidor. Una vez realizado el saque, los agentes pueden moverse libremente y el control de la pelota pasa al script correspondiente. Al finalizar un punto, se rota la posición del servicio y el proceso se repite.

9.2.4. Controlador del marcador

Controlador del marcador se encarga de los puntos obtenidos por cada equipo. Cada punto ganado se suma al marcador actual, siguiendo la secuencia estándar: 0, 15, 30 y 40. Si un equipo alcanza los 40 puntos, gana el juego en condición de que el equipo oponente no alcanza este punto. En caso de que ambos equipos lleguen a 40 puntos sin que ninguno haya obtenido una ventaja previa, se regresa al estado de igualdad. Un punto de ventaja se obtiene cuando el equipo rival no tiene ninguno, y el juego se gana después de conseguir dos puntos de ventaja consecutivos. Tras cada punto, el controlador indica al controlador del entorno que cambie el lado del servicio y reinicie el entorno.

9.2.5. Controlador de los agentes

El controlador de agentes es responsable de gestionar las posiciones de los jugadores dentro del entorno virtual. Este controlador se ocupa de establecer la posición inicial de cada agente al comienzo de cada punto y asigna la recompensa al equipo ganador al finalizar el juego. Además, administra el estado de cada jugador, determinando si puede realizar un movimiento.

9.2.6. Controlador de la pelota

El controlador de la pelota es el encargado de controlar todo el comportamiento de la pelota, como su movimiento, rebote contra el suelo, etc.

Uno de los problemas de la implementación original del script es la falta de sincronización entre la lógica y la física de la pelota. Esto se debe a que el comportamiento de la pelota estaba controlado por el sistema de física propio de Unity, lo que podía generar discrepancias cuando un jugador golpea la pelota mientras el controlador de trayectoria aún estaba calculando su movimiento. Esta discordancia podía provocar que la fuerza aplicada a la pelota fuera incorrecta.

Para solucionar este problema, se decidió controlar la física de la pelota mediante un *script* propio. En cada *FixedUpdate* de Unity, que representa un intervalo de tiempo de 0.02 segundos,

se actualiza la posición de la pelota y se comprueban las colisiones con otros objetos. Para lograr este objetivo, se re-implementó el script, creando diversas variables que representan el estado de la pelota, como la velocidad, la posición y el tiempo. Utilizando estos valores, el script calcula la posición de la pelota en cada instante de tiempo utilizando fórmulas de movimiento parabólico y movimiento rectilíneo uniforme (MRU). Una vez calculada la nueva posición, se actualizan las variables con los nuevos valores. Esta solución permitió acelerar el proceso de entrenamiento asignando un *time-scale* de 20 al agente de aprendizaje automático (ML-Agents). El *time-scale* controla la velocidad a la que se entrena el entorno; al aumentar este valor, se incrementa la velocidad de entrenamiento. Al establecer el *time-scale* en 20, el proceso se hizo 20 veces más rápido.

Otra funcionalidad del script es calcular la velocidad necesaria para enviar la pelota a una posición específica. Para simplificar el problema se considera el campo como una cuadrícula de 5×5, como se muestra en la Figura 9, donde el jugador elegirá un punto de la cuadrícula como zona de impacto. Entre los puntos de la cuadrícula existe una separación de $\frac{10}{6}$. Respecto cada equipo, el punto $(0, 0)$ de la cuadrícula representa la esquina superior izquierda del campo contrario y la posición $(4, 4)$ corresponde a la esquina inferior derecha.

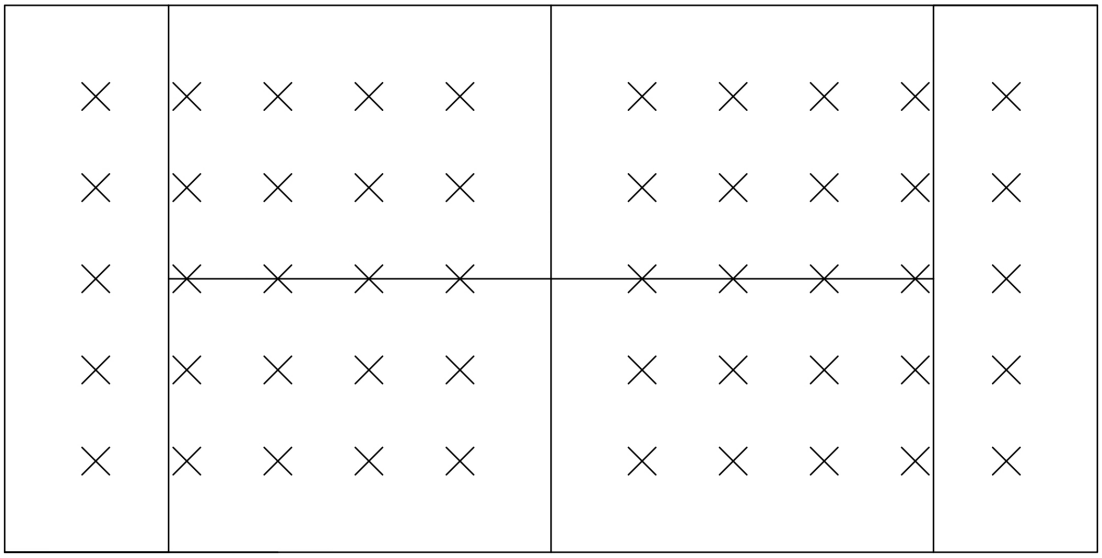


Figura 9: División de la pista.(Fuente: [3])

Para el cálculo de la fuerza que necesita la pelota para llegar al punto elegido, se utilizan las ecuaciones de movimiento parabólico en el eje Y, y la ecuación de movimiento rectilíneo uniforme (MRU) en los ejes X y Z:

$$x = x_0 + v_x + t \tag{17}$$

$$y = y_0 + v_{y_0} t + \frac{1}{2} a t^2 \tag{18}$$

$$v_x = v_{x_0} \tag{19}$$

$$v_y = v_{y_0} + at \tag{20}$$

La información disponible para calcular la velocidad de golpeo incluye: la posición inicial de la pelota (x_0, y_0, z_0), la posición final deseada $(x_f, 0, z_f)$, la aceleración de la gravedad

$a = -9{,}8\,\mathrm{m/s^2}$ y la altura máxima de la pelota y_{max}. Mediante esta condición podemos deducir las velocidades iniciales $(v_{x_0}, v_{y_0}, v_{z_0})$ partiendo de las fórmulas anteriores. [3]

$$v_{y_0} = \sqrt{(y_0 - y_{max})2a} \tag{21}$$

$$v_{x_0} = \frac{x_f - x_0}{t} \tag{22}$$

$$v_{z_0} = \frac{z_f - z_0}{t} \tag{23}$$

Además de controlar la posición de la pelota, el controlador también maneja la lógica de impacto con otros objetos y recalcula la velocidad de la pelota después de colisionar con la pared o el suelo. Para lograr esto, se predefinen los límites del campo y las dimensiones de cada objeto (en el proyecto original, se utilizaban objetos específicos de Unity llamados *colliders* para que éste detectara las colisiones). En cada instante de tiempo, el script comprueba si la pelota ha colisionado con un objeto, ya sea una pared, un jugador o la red. Si la pelota colisiona con la pared o el suelo, su velocidad se recalcula utilizando la siguiente fórmula:

$$\text{Velocidad Nueva} = \text{Coeficiente de Restitución} * \text{Velocidad} * \text{Dirección de Reflexión}$$

Si la pelota colisiona contra la red o un jugador, activará el proceso de gestionar la puntuación. El script enviará la información de la puntuación al controlador del entorno, que a su vez la comunicará a otros controladores. Al finalizar cada punto, el controlador de la pelota restablece su estado y notifica al controlador del entorno que debe asignar un punto al equipo ganador.

Para hacer que la simulación se asemeje más a la realidad, también se ha introducido un cierto nivel de ruido al golpear la pelota. Cuando el agente golpea la pelota con más fuerza, el ruido es más grande, lo que desvía la trayectoria de la pelota con mayor intensidad. En el mundo real, la precisión de los jugadores, es decir, la desviación entre la trayectoria deseada y la trayectoria que realmente sigue la pelota, depende de múltiples factores, pero la velocidad previa de la pelota antes del impacto tiene un papel muy relevante. Las pelotas muy rápidas impiden controlar con precisión la velocidad de salida tras el impacto, y con frecuencia modifican el instante de impacto y en consecuencia la orientación de la pala puede diferir de la requerida para conseguir la trayectoria deseada. Añadir este ruido en el entorno virtual introduce una pequeña diferencia entre la acción que elige el agente de acuerdo con su política, y el efecto que causa en el entorno, algo que también ocurre en el mundo real y que lleva a los jugadores a considerar un cierto margen de error en sus acciones técnicas, evitando enviar la pelota a zonas con riesgo cuando la pelota a devolver viaja a gran velocidad.

9.2.7. Controlador de la trayectoria de la pelota

Al igual que con el controlador de la pelota, este script se ha re-implementado debido a los cambios en la física de la misma. El controlador de trayectoria simula el movimiento de la pelota tras cada golpe, recreando una simulación física para cada impacto y replicando los obstáculos del campo correspondiente. La principal diferencia es que no calcula únicamente la posición de la pelota en el siguiente frame, sino toda la trayectoria hasta que, en caso de no ser devuelta por ningún jugador, se decida el punto (segundo bote en el suelo, bola sale de la pista, etc). Este permite calcular toda una serie de posiciones en las que, teniendo en cuenta la

posición de los jugadores, una devolución de la pelota sería posible. Estas posiciones se llaman posiciones clave, y excluyen por ejemplo los tramos de la trayectoria en los que la pelota está a mucha altura (por ejemplo, tras un globo) o bien a una distancia que el jugador no puede cubrir ni desplazándose a la velocidad máxima. En el mundo real, los jugadores de pádel con cierta experiencia pueden estimar mentalmente la trayectoria de la pelota, incluyendo rebotes con las paredes, y eso les permite identificar las posiciones en las que la devolución de la pelota es viable. A diferencia del tenis, para el cual normalmente solamente existe una única región con devoluciones viables, la existencia de paredes en pádel hace que a menudo la devolución pueda ser viable antes o después del rebote en las diferentes paredes. Para este cálculo, se ha creado una clase llamada `GhostBall`, que actúa como réplica de la pelota real en la posición donde recibe la fuerza del golpe. Esta `GhostBall` simula la trayectoria futura de la pelota en frames (*Time.fixedDeltaTim*), utilizando una velocidad inicial.

El cálculo de la posición futura de la pelota sigue el mismo procedimiento que el controlador de la pelota. La simulación se detiene una vez que se alcanza un número máximo de frames o cuando la pelota rebota en el suelo dos veces.

Durante la simulación, las posiciones de la `GhostBall` se envían al controlador de posición clave para calcular los puntos de interés en la trayectoria de la pelota, los cuales representan los puntos clave para devolver la pelota. Además, en modo debug, se utiliza `LineRenderer` para definir estas posiciones y visualizar la trayectoria calculada.

9.2.8. Controlador de las posiciones clave

Este controlador se encarga de calcular los puntos clave (puntos en los que la devolución de la pelota sería viable, si el jugador se desplaza a ese punto) que son relevantes para un agente durante la partida. Cuando un agente se encuentra cerca de uno de estos puntos o intenta acercarse a él, se le asigna una recompensa. El propósito de esta asignación de recompensas es que el agente aprenda a posicionarse en los puntos más importantes del campo durante el entrenamiento del modelo.

Dentro del controlador se consideran diferentes roles de los jugadores: `Receiver, Opponent, Teammate`. El cálculo de los puntos para cada uno de estos roles es indiferente. Cuando se calcula la trayectoria de la pelota, también se calculan los puntos claves de la pelota. Utilizando la cuadrícula de la Figura 9, se divide el eje Z del propio campo en 5 tramos equidistantes, que se representan de siguiente manera $z \in [(i+1)\frac{10}{6}, (i+2)\frac{10}{6})]$ donde i = 0...4. Según la trayectoria de la pelota, en cada tramo existirá a lo sumo un punto clave donde la pelota es alcanzable en ese instante de tiempo. Según el agente que tenga más puntos claves de devolución cercanos a él, se le asigna el rol de `Receiver`, siendo el responsable de devolver la pelota. Las posiciones se actualizan continuamente para reflejar los cambios en el campo, ya que algunas pueden volverse inalcanzables. Por lo tanto, los roles pueden alternarse dinámicamente entre los jugadores según las circunstancias.

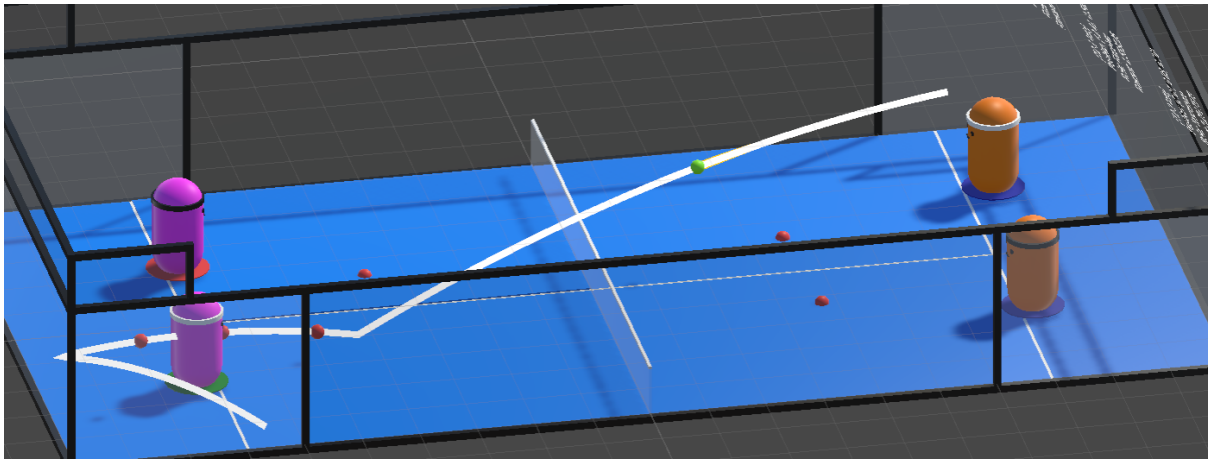


Figura 10: Punto de clave de la trayectoria. Bajo esta condición, el jugador 1 del equipo T2 recibe el rol de `Receiver`, representado por el color verde en la base de su avatar, debido a su proximidad a más posiciones clave en el campo. (Elaboración propia)

Además de los puntos claves descritos anteriormente, relevantes para el jugador que vaya a realizar la devolución de la pelota, existe otro tipo de puntos clave: los puntos de cobertura, que consideran el punto de campo más descubierto por el jugador en aquel instante de tiempo. El punto se describe de siguiente forma:

$$(x,z)_{KP} = (x,z)_{Player} + 0.5 \arg\max(\|\overrightarrow{x_1}\|, \|\overrightarrow{x_2}\|) + 0.5 \arg\max(\|\overrightarrow{z_1}\|, \|\overrightarrow{z_2}\|) \tag{24}$$

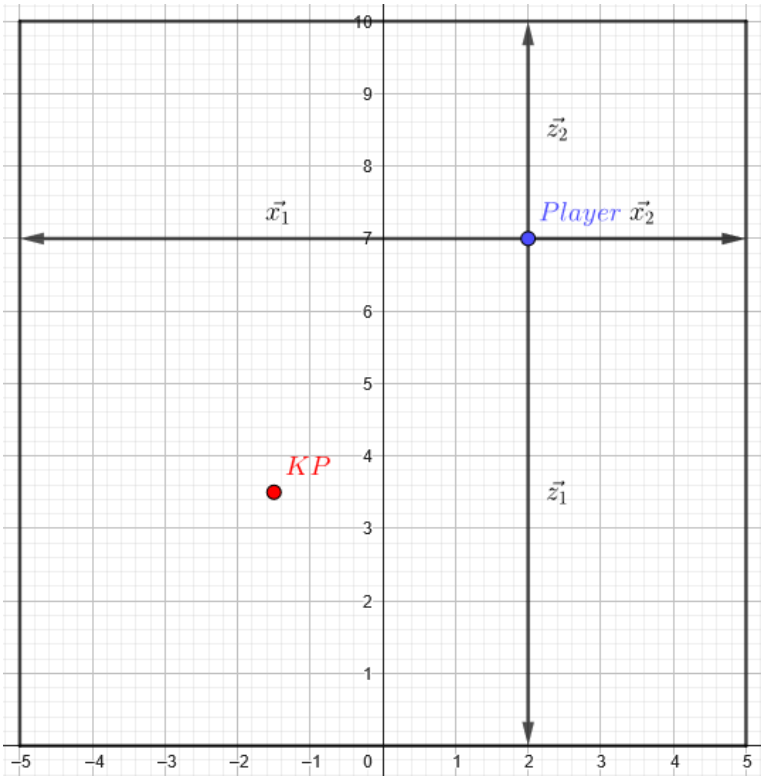


Figura 11: Ilustración de un punto de cobertura *KP*, generada por el jugador *Player*. [3]

La implementación de posiciones de cobertura tiene como objetivo incentivar a los agentes a distribuirse estratégicamente en el campo, cuando todavía no es posible predecir la trayectoria

de la pelota, porque los rivales todavía no han golpeado la pelota. La asignación de puntos de cobertura es independiente del rol de cada agente. Los agentes no tienen visibilidad de los puntos asignados a otros roles. El rol `Teammate` se asigna al compañero del mismo equipo de `Receiver`, y el rol `Opponent` se asigna a los agentes del equipo contrario.

El agente asignado como `Teammate` debe cubrir el espacio vacío dejado por el agente receptor. En contraste, ambos agentes `Opponent` deben encargarse de cubrir el espacio desocupado. En esta situación, el punto de cobertura es dinámico, y los agentes buscan un equilibrio que les permita cubrir la máxima área de la pista en cada momento.

Cuando un agente se encuentra a una distancia de al menos 1.5 metros de un punto clave, se le asignará una recompensa previamente definida. Además, se otorgará una recompensa a los agentes que intenten acercarse a ese punto. Esto se determinará comparando su posición anterior con la actual para verificar si se han acercado o no. De nuevo, esta recompensa trata de imitar el comportamiento real de los jugadores de pádel, que incluso en niveles de iniciación comprenden que deben cubrir la pista lo mejor posible, evitando dejar huecos que puedan aprovechar los rivales.

Agente de pádel

Es el script donde integra el componente de ML-Agents que controla los movimientos de los agentes durante el entrenamiento.

El script controla internamente el episodio, restableciendo los parámetros de control al finalizar cada uno. El inicio del episodio está marcado por el servicio y finaliza con la obtención del punto por un equipo.

El sistema cuenta con un total de 17 observaciones, que incluyen las posiciones y estados de todos los agentes y la pelota. Estos valores constituyen las entradas que reciben los agentes, y el modelo entrenado actuará en consecuencia según estos datos. Cuando *logEnabled* está activado, el script también enviará las observaciones al **Logger**, que las procesará internamente para almacenarlas, con el objetivo de generar diferentes gráficas.

El espacio de acción de los agentes consiste en conjuntos de acciones que pueden realizar. En la implementación inicial, solo existían acciones discretas. Sin embargo, para lograr un comportamiento más realista, se han introducido nuevas acciones tanto discretas como continúas, permitiendo un control más preciso. En total, disponemos de 7 ramas, cada una de las cuales es un subconjunto de valores:

Acción discreta

- La rama 0, de tamaño 5, utilizada para definir hacia qué columna de la propia cuadrícula debe moverse el agente (reflejado en el eje X).
- La rama 1, de tamaño 5, utilizada para definir hacia qué fila del propio la cuadrícula debe moverse el agente (reflejado en el eje Z).
- La rama 2, de tamaño 5, utilizada para definir hacia qué columna de la cuadrícula debe enviar la pelota (reflejado en el eje X).
- La rama 3, de tamaño 5, utilizada para definir hacia qué fila de la cuadrícula debe enviar la pelota (reflejado en el eje Z).
- La rama 4, de tamaño 6, especifica qué tipo de acción realizar al devolver la pelota:

0 no golpear.

1 realizar golpe normal (derecha/revés) plano, es decir, sin aplicar ningún efecto a la pelota.

2 realizar un golpe normal pero con efecto cortado. El coeficiente de restitución se reduce al 0.65 y la gravedad a $-6{,}5$ m/s, lo cual hace que la pelota caiga más lentamente y rebote menos. Esta es una forma sencilla de simular el efecto de las fuerzas de sustentación que ejerce el aire sobre la pelota cuando ésta gira sobre sí misma con un efecto cortado; así como su efecto en el rebote.

3 realizar un golpe normal pero con efecto top-spin. Es contrario de un cortado, el coeficiente de restitución se aumenta al 0.85,

4 realizar un globo, el cual se ha definido con una altura $y_{max} = 4$ metros.

5 realizar un remate, el cual se ha definido con $y_{max} = -1$ metros. Al calcular la velocidad inicial, si $y_{\max} < y_0$, se ajusta $y_{max} = y_0$, Lo que provoca que la pelota siga una trayectoria parabólica con una curvatura tan pequeña que se asemeja a una línea recta (lo cual requiere una gran velocidad de golpeo).

Acción continua

- La rama 0 varía entre [-1,1] y determina la velocidad a la que debe moverse el agente. Este valor se multiplicará por la velocidad máxima predefinida (por defecto 5 m/s) y `Time.deltaTime` para obtener la velocidad final. Con una menor velocidad, el agente obtendrá una mayor recompensa, incentivando así que se mueva con el mínimo esfuerzo, similar a cómo lo hacen los humanos para minimizar el gasto energético.
- La rama 1 varía entre [-1,1] y determina la altura a la que se debe devolver la pelota. El rango de esta altura está entre [1 y 4] metros, y corresponde a la altura máxima de la pelota durante el vuelo. El código se encarga de traducir esta altura a un vector velocidad adecuado en el momento del golpeo.

9.2.9. Implementación de la grabación de demostraciones (aprendizaje por imitación)

Aparte de las configuraciones para el RL, también se han implementado nuevas funcionalidades para generar grabaciones de demostraciones, con el fin de entrenar mediante GAIL.

Normalmente, para generar una demostración se utiliza la funcionalidad de heurística que proporciona ML-Agents, donde el agente puede ser controlado directamente por el desarrollador mediante el teclado. Sin embargo, dado que se trata de un entorno multi-agente, se decidió utilizar un conjunto de datos reales obtenidos de una partida final femenina proporcionada por el director. Los agentes actuarán basándose en estos datos en cada instante.

La idea principal es que, en cada momento, el agente envía los parámetros que observa (posición de jugador y la pelota) y el *Control de entrenador*, donde están cargados los datos, le responderá cómo debe actuar en función de los datos proporcionados.

Heurística del agente

La función heurística dentro del script del *controlador del agente* controla los movimientos del agente cuando se activa la configuración *Heuristic only* de ML-Agents.

En cada instante de tiempo, el agente envía una solicitud de movimiento o golpeo al *controlador del entrenador*. Esta solicitud incluye las posiciones locales de todos los agentes, su compañero, los oponentes y la pelota. Además, para adaptar la estructura de los datos procesados, se incluye el último equipo que ha golpeado la pelota. La función heurística del agente no solo envía la solicitud, sino que también recibe el resultado de la petición, sobre la cual basa sus movimientos.

La función heurística se ejecuta de manera rápida y continua. Para evitar el envío de solicitudes duplicadas, se ha implementado un mecanismo de bloqueo. Este mecanismo garantiza que el agente solo pueda enviar una solicitud cuando no está esperando una respuesta. Una vez que se envía la petición, el agente entra en un estado de espera, que se desbloquea al recibir la respuesta.

Entrenador

En el proyecto de partida, era un script de Python que funcionaba como un servidor que recibía solicitudes de movimiento y golpeo desde Unity. Dentro del script, se procesaban los datos reales de una partida y se almacenaban en una estructura KD-tree, la cual permite acelerar el cálculo de búsquedas de los k-vecinos más próximos. Cuando el script recibía una petición, realizaba una búsqueda en el KD-tree y devolvía el vecino más cercano. Luego, postprocesaba los datos obtenidos y los enviaba a Unity. Por ejemplo, para un determinado estado de los jugadores y la pelota, se devolvía la acción observada en los datos del partido con jugadores reales.

Al principio, se planeó utilizar TCP-sockets para conectar Unity con el script. Sin embargo, surgió un inconveniente: el tiempo de transmisión resultó ser demasiado alto, llegando a 20 ms. Esto causaba movimientos inesperados en los agentes porque las solicitudes se enviaban a intervalos muy cortos pero las respuestas tardaban demasiado tiempo en llegar.

Tras analizar el proceso, se detectó que el cuello de botella se encontraba en la transmisión de información, en lugar del procesamiento de datos. Para resolver este problema, se decidió reemplazar el TCP-socket por ZMQ-socket con el fin de acelerar la velocidad de transmisión. Sin embargo, aunque el tiempo de transmisión de los mensajes se redujo, seguía siendo demasiado alto.

Finalmente, se decidió eliminar este script y re-implementar el entrenador directamente dentro de Unity, eliminando así la necesidad de establecer un socket entre ambas partes. De esta manera, se elimina el tiempo de transmisión. Todas estas funcionalidades se explicarán con más detalle en el controlador del entrenador.

Controlador del entrenador

Este script carga y procesa los datos recibidos de las solicitudes necesarias durante el entrenamiento por imitación, eliminando la información innecesaria. Cada solicitud incluye información sobre el estado del entorno virtual. Utilizando esta información, el script busca en el conjunto de datos la situación más similar a la actual y devuelve la acción correspondiente al solicitante.

El conjunto de datos es una partida final femenina proporcionada por el tutor, almacenada en un archivo Excel (.xlsx). Este archivo contiene las posiciones y velocidades de cada jugadora de pádel y de la pelota en cada instante de tiempo, para más de 21.000 fotogramas del video

del partido. Los datos utilizan el origen de coordenadas $(0,0)$ en la esquina inferior izquierda de la pista de pádel con una medida de 10x20. El campo se divide en 2 zonas, la mitad superior de la pista ($y > 10$) pertenece al equipo `T` (top) que representa equipo T2 dentro Unity y ($y < 10$) pertenece al equipo `B` que representa equipo T1. Dentro de cada equipo, los jugadores se distinguen por su posición inicial: `TL` (izquierda) y `TR` (derecha) en el equipo `T`, y `BL` (izquierda) y `BR` (derecha). La Figura 12 muestra una representación del campo.

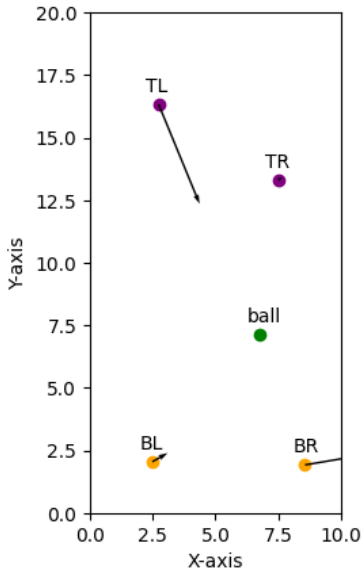


Figura 12: Representación del campo de los datos de pádel procesado (Fuente: [3])

Antes de operar con el conjunto de datos, se lleva a cabo un preprocesamiento que elimina la información innecesaria, como por ejemplo el rol de cada jugador, el tiempo de la partida, etc. El resultado final se encuentra de la siguiente manera:

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	y1	x2	y2	x3	y3	x4	y4	ballx	bally	speed1x	speed1y	speed2x	speed2y	speed3x	speed3y	speed4x	speed4y	shot	targetx	targety	lastHit
2	18.28	7.49	13.35	2.50	1.60	7.87	1.66	1.89	18.28	-0.82	-0.39	-0.01	-0.04	-0.14	-0.31	0.15	-1.20	normal	8.98	2.11	T
3	18.27	7.49	13.35	2.50	1.59	7.88	1.62	2.14	17.72	0.70	-2.61	-0.05	-0.20	-0.22	1.27	0.39	-0.66	undef	0.00	0.00	T
4	18.18	7.49	13.35	2.49	1.63	7.89	1.60	2.38	17.17	1.36	-2.46	-0.02	-0.05	-0.76	2.13	0.34	-0.84	undef	0.00	0.00	T
5	18.10	7.49	13.34	2.47	1.70	7.90	1.57	2.63	16.61	0.86	-4.29	-0.01	-0.03	-0.25	0.92	0.97	0.10	undef	0.00	0.00	T
6	17.96	7.49	13.34	2.46	1.74	7.93	1.58	2.87	16.05	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	undef	0.00	0.00	T
7	17.96	7.49	13.34	2.46	1.74	7.93	1.58	3.11	15.49	1.41	-1.83	0.03	0.06	-0.33	1.69	1.10	0.10	undef	0.00	0.00	T
8	17.90	7.49	13.34	2.45	1.79	7.97	1.58	3.36	14.94	1.87	-10.08	0.41	-0.15	-0.30	1.26	1.01	-0.43	undef	0.00	0.00	T
9	17.56	7.50	13.34	2.44	1.83	8.00	1.57	3.60	14.38	1.04	0.79	0.07	-0.03	-0.46	1.06	2.01	-0.20	undef	0.00	0.00	T
10	17.59	7.51	13.34	2.42	1.87	8.07	1.56	3.85	13.82	1.02	-1.56	0.00	0.07	0.08	1.17	1.48	0.64	undef	0.00	0.00	T
11	17.53	7.51	13.34	2.42	1.91	8.12	1.58	4.09	13.26	2.08	-5.82	-0.01	0.10	-0.17	-0.01	1.55	-0.04	undef	0.00	0.00	T
12	17.34	7.50	13.34	2.42	1.91	8.17	1.58	4.34	12.71	0.00	0.00	0.00	-0.01	-0.02	0.01	0.03	0.12	undef	0.00	0.00	T
13	17.34	7.50	13.34	2.42	1.91	8.17	1.58	4.58	12.15	1.81	-4.52	-0.04	0.30	-0.25	1.31	1.87	1.45	undef	0.00	0.00	T
14	17.19	7.50	13.35	2.41	1.95	8.23	1.63	4.83	11.59	2.49	-3.90	-0.01	-0.17	0.15	-0.16	1.54	2.24	undef	0.00	0.00	T
15	17.06	7.50	13.35	2.41	1.95	8.29	1.71	5.07	11.03	-0.23	-1.41	0.12	-1.58	0.07	1.13	1.63	2.03	undef	0.00	0.00	T

Figura 13: Conjunto de datos procesado. (Elaboración propia)

A partir de estos valores se selecciona un subconjunto de parámetros más importante, formado por las posiciones de jugadores y la pelota:

$$[\texttt{x.1}, \texttt{y.1}, \texttt{x.2}, \texttt{y.2}, \texttt{x.3}, \texttt{y.3}, \texttt{x.4}, \texttt{y.4}, \texttt{ball.x}, \texttt{ball.y}]$$

Se realiza una separación en los espacios de búsqueda para las solicitudes de movimiento y golpeo, dado que la información necesaria varía según el tipo de solicitud. Esto se debe a que, al realizar la búsqueda, es necesario saber qué equipo ha golpeado la pelota por última vez.

A partir de estos datos separados, se crea un árbol de k-dimensiones para cada uno de los conjuntos de datos. El árbol de k-dimensiones almacena los valores en múltiples dimensiones, lo que permite realizar búsquedas eficientes de los vecinos más próximos.

Hemos separado los datos en siguiente 4 grupos:

1. Conjuntos de datos de movimientos que la pelota ha sido golpeada por el equipo `T`.
2. Conjuntos de datos de movimientos que la pelota ha sido golpeada por el equipo `B`.
3. Conjuntos de datos de golpeos realizados por el equipo `T` eliminando los golpes `undef` (es decir, incluyendo solo los fotogramas en los que un jugador golpea la pelota).
4. Conjuntos de datos de golpeos realizados por el equipo `B` eliminando los golpes `undef`.

En las solicitudes de movimiento enviadas por el script del agente de pádel, se especifica el último equipo que haya golpeado la pelota (`T1` o `T2`) para determinar qué árbol KD-tree debe buscar los resultados. Por otro lado, las solicitudes de golpeo no requieren esta especificación, ya que se asume que el agente que envía la solicitud es el jugador que golpeará la pelota.

A la hora de procesar las peticiones, el script bloqueará el proceso hasta que se devuelva el resultado al agente. Esto se debe a que los 4 agentes pueden enviar una solicitud al mismo tiempo. En cada instante de tiempo, el script solo procesará una solicitud, ya que a partir de la información proporcionada puede calcular los próximos pasos para los 4 agentes.

Para el postprocesamiento de los golpeos, solamente utilizará estas informaciones:

[`Shot`, `Target.x`, `Target.y`]

El tipo de golpeo `shot` tiene tres tipos (`normal`, `lob`, `smash`), que representan la acción numérica (2, 3, 4) de la rama de acción 4. Para encontrar el punto de impacto, es necesario realizar una conversión de los valores, ya que utilizamos una cuadrícula de 5x5 para calcular los puntos de impacto. Para lograr esto, se inicializa una matriz de 5x5 para cada equipo, donde se asigna a cada punto la posición real del campo. Luego, se recorre esta matriz para calcular los índices más cercanos a `Target.x` y `Target.y`. Así, logramos obtener los valores de las ramas 2 y 3 de las acciones discretas del agente.

Para el postprocesamiento de los movimientos, utilizamos el siguiente subconjunto de resultados:

[`x.1`, `y.1`, `x.2`, `y.2`, `x.3`, `y.3`, `x.4`, `y.4`, `speed1.x`, `speed1.y`, `speed2.x`, `speed2.y`, `speed3.x`, `speed3.y`, `speed4.x`, `speed4.y`]

El resultado utiliza un rango $x \in [0, 10]$ y $y \in [0, 20]$, que debe convertirse a $x \in [-5, 5]$ y $z \in [-10, 10]$, en este caso el eje y representa el eje z de Unity. El segundo paso es asociar cada posición con el agente correspondiente, que en realidad es el agente más cercano al punto. Una vez hecho esto, calculamos la inversa de la velocidad `Speed` en X e Y, sabiendo que los agentes tienen una velocidad fija de 5 m/s.

$$\hat{u} = \frac{\vec{u}}{|\vec{u}|} = \frac{|\vec{v}|}{5 \text{ m/s}} \tag{25}$$

$$\text{Target} = \text{Pos\_player} + \hat{u} \cdot |\vec{u}| \tag{26}$$

De esta manera, obtenemos la posición del campo hacia la que deben moverse los agentes. Utilizando la misma metodología de la matriz de procesamiento de golpeo, obtenemos los valores de las ramas 0 y 1 de las acciones discretas del agente.

10. Resultados

En este apartado, exploramos los efectos de las nuevas funcionalidades que influyen en el entorno de entrenamiento mediante el uso de técnicas de aprendizaje por refuerzo profundo.

En este estudio, nos enfocamos principalmente en evaluar el impacto de la nueva funcionalidad introducida en el entorno virtual sobre el comportamiento del entrenamiento. Sin embargo, es importante reconocer que existen otros parámetros que también podrían haber influido en los resultados obtenidos. Entre ellos, se encuentran los hiperparámetros de las redes neuronales utilizadas y los valores de recompensa.

La siguiente tabla (Tabla 11) resume la configuración probada según el experimento.

Experimento	Velocidad de los agentes	Altura máxima	RL/IL
`Ex.1`	Variable en [0, 5m/s]	Variable en [1m, 4m]	RL
`Ex.2`	Fija	Variable en [1m, 4m]	RL
`Ex.3`	Fija	Fija según golpe elegido	RL
`Ex.4`	Fija	Fija según golpe elegido	IL

Tabla 11: Activación de las funcionalidades de experimentos (Elaboración propia)

Sin embargo, se decidió priorizar la evaluación de los elementos directamente definidos en el entorno virtual. Esto se debe a que la inclusión de parámetros adicionales en la fase de experimentación habría extendido considerablemente este apartado, requiriendo un aumento significativo de tiempo y recursos. Hay que tener en cuenta que el entrenamiento de los agentes, para un número razonable de episodios (del orden de varios millones) requiere varias horas de cálculo.

Los valores que tomaremos para comparar diferentes modelos son:

- **La recompensa acumulada:** Representa el beneficio total obtenido durante un episodio. Al inicio del entrenamiento, la recompensa acumulada suele ser muy variable, ya que la política aún no sabe cómo obtener recompensas en el entorno. A medida que la política aprende, se espera que la recompensa acumulada aumente, ya que la política es capaz de realizar acciones que le permiten obtener más recompensas. Sin embargo, la recompensa acumulada puede ser engañosa si la política se enfoca en obtener recompensas inmediatas sin considerar el objetivo final a largo plazo.
- **Duración del episodio:** Indica la cantidad de pasos que la política ha realizado en el entorno. Al igual que la recompensa acumulada, inicialmente la duración de cada episodio es muy corta, pero aumenta con el tiempo. Sin embargo, este parámetro tiene sus limitaciones, ya que la duración del episodio puede incrementarse artificialmente si la política adopta comportamientos no deseados, como repetir acciones simples, devolviendo la pelota de una forma muy suave.
- **La puntuación Elo:** A diferencia de otras métricas como la duración del episodio o la recompensa acumulada, la puntuación Elo obtenida mediante *self-play* ofrece una evaluación más robusta y confiable del rendimiento de la política que se está entrenando. La puntuación Elo se actualiza después de cada episodio, aumentando o disminuyendo en función del resultado de la política contra su oponente. En cada enfrentamiento, el agente con mayor puntuación Elo tiene más probabilidades de ganar, pero si el agente

con menor puntuación Elo logra vencer, ganará más puntos Elo que el ganador. Por el contrario, si el agente con mayor puntuación Elo gana, ganará menos puntos. Este sistema de ajuste dinámico permite reflejar la habilidad relativa de cada agente de forma precisa.

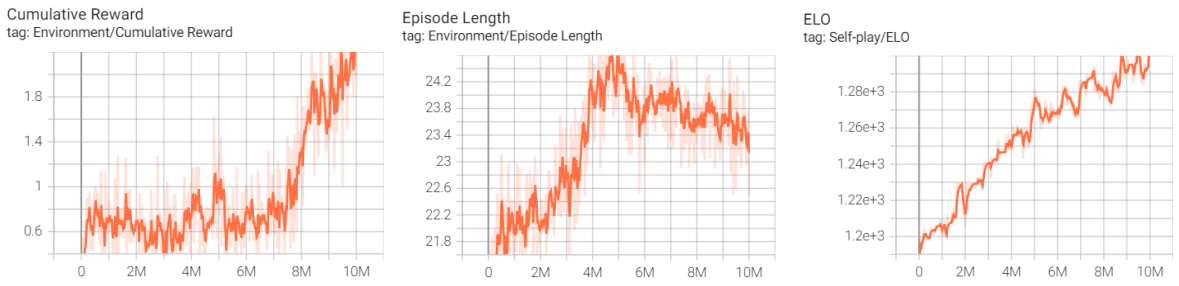
Para la experimentación del entorno con aprendizaje por refuerzo profundo, utilizamos las mismas recompensas indicadas en la Tabla 12 y los hiperparámetros especificados en el Apéndice B.1 para todos los entrenamientos de RL y el Apéndice B.2 para el entrenamiento de IL.

Recompensa	Valor
`WinningReward`	10
`LosingReward`	-10
`ApproachingKeyPositionsReward`	0,01
`StayingAroundKeyPositionsReward`	0,1
`HittingBallReward`	0,05
`SpeedReward`	0,001

Tabla 12: Función de recompensa para aprendizaje por refuerzo profundo (Elaboración propia)

10.1. Experimento 1. Entrenamiento con todas las funcionalidades activadas

Tras 10 millones de pasos de entrenamiento, observamos los resultados presentados en la Figura 14. Al inicio del proceso de entrenamiento, la recompensa acumulada muestra una gran variabilidad, reflejando la fase inicial de aprendizaje de los agentes. A medida que se acerca a los 8 millones de pasos, los agentes comienzan a adquirir de manera efectiva la dinámica y las estrategias del juego. En contraste, la duración de los episodios alcanza un pico alrededor de los 4 millones de pasos, que luego disminuye progresivamente. La puntuación Elo, por otro lado, muestra un aumento constante a lo largo del entrenamiento, indicando una mejora continua en la política de los agentes.

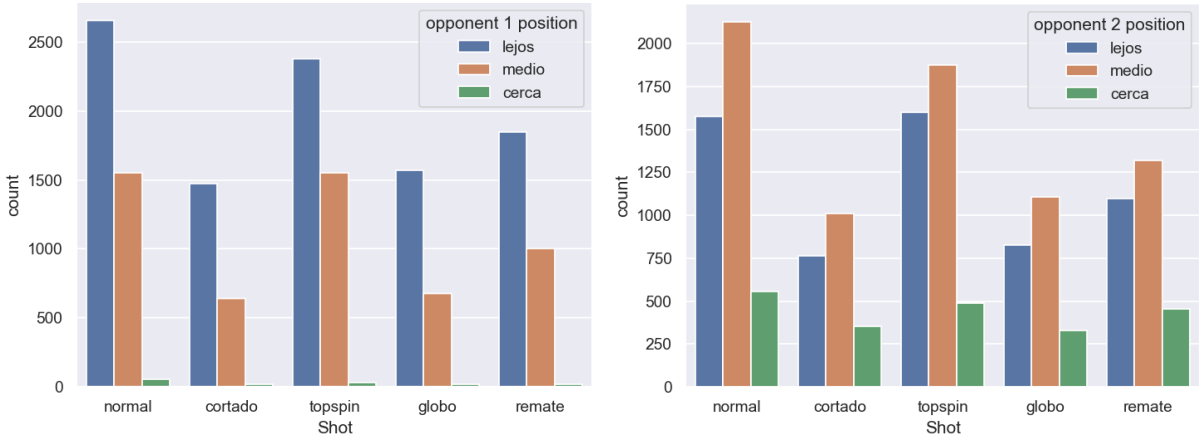


(a) Recompensa acumulada (b) Duración del episodio (c) Puntuación Elo

Figura 14: Resultados del experimento 1. (Elaboración propia)

La Figura 15 muestra la distribución de la distancia de los oponentes a la red cuando el agente devuelve la pelota. Se observa una clara división de roles: el oponente 1 defiende el fondo (zona posterior) del campo, mientras que el oponente 2 se ubica en zonas medias y

cerca de la red para mayor versatilidad. Los golpes más frecuentes son el topspin y el golpe normal, por su control y seguridad.

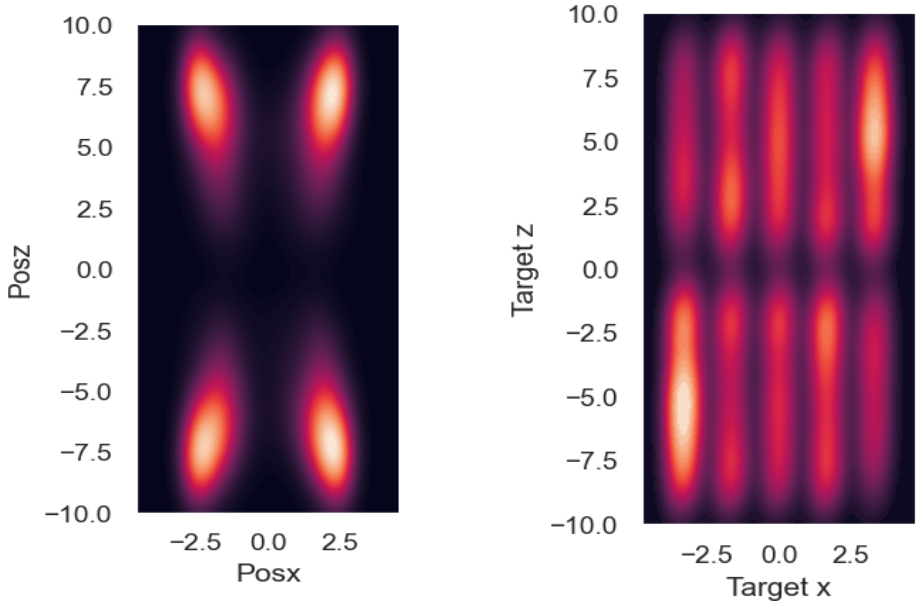


(a) Oponente 1

(b) Oponente 2

Figura 15: Experimento 1. Frecuencia absoluta de cada tipo de golpe, según la distancia de los oponentes cuando se devuelve la pelota (Elaboración propia)

La Figura 16a muestra el mapa de calor de los movimientos de los agentes, donde se observa que la mayoría del tiempo los agentes se sitúan en la parte posterior del campo y raramente se acercan a la red. En la Figura 16b, se presentan las posiciones a las que se envía la pelota, representadas por franjas, debido a que el campo está dividido en coordenadas de 5x5 a las cuales el agente puede enviar la pelota. En la gráfica, se aprecia que los bordes del campo tienen un color más intenso, lo que indica que el agente tiende a enviar la pelota hacia las esquinas del campo.



(a) Posición de los agentes

(b) Posición donde envia la pelota

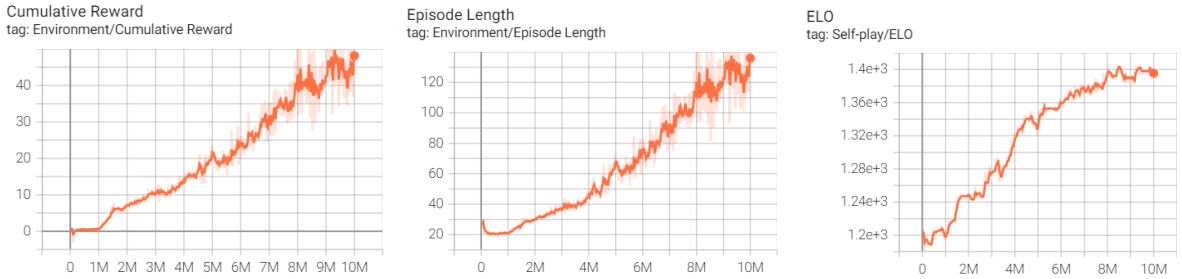
Figura 16: Experimento 1. Mapa de calor de los agentes (Elaboración propia)

Finalmente, al observar el comportamiento del modelo en el entorno de Unity, descubrimos que el posicionamiento de los agentes no es óptimo; se sitúan demasiado en el fondo del campo y no cubren adecuadamente la pelota que se acerca. Sin embargo, un aspecto positivo que encontramos es que los agentes saben separarse y no se agrupan al ir hacia la pelota. Este problema podría solucionarse con más pasos de entrenamiento, ya que la gráfica de ELO muestra una tendencia positiva.

10.2. Experimento 2. Entrenamiento desactivando el control de la velocidad del agente

En este experimento, examinamos el efecto del control de la velocidad. En esta nueva condición, los agentes ya no pueden ajustar su velocidad de movimiento; en su lugar, se mueven a una velocidad constantemente de 5 m/s.

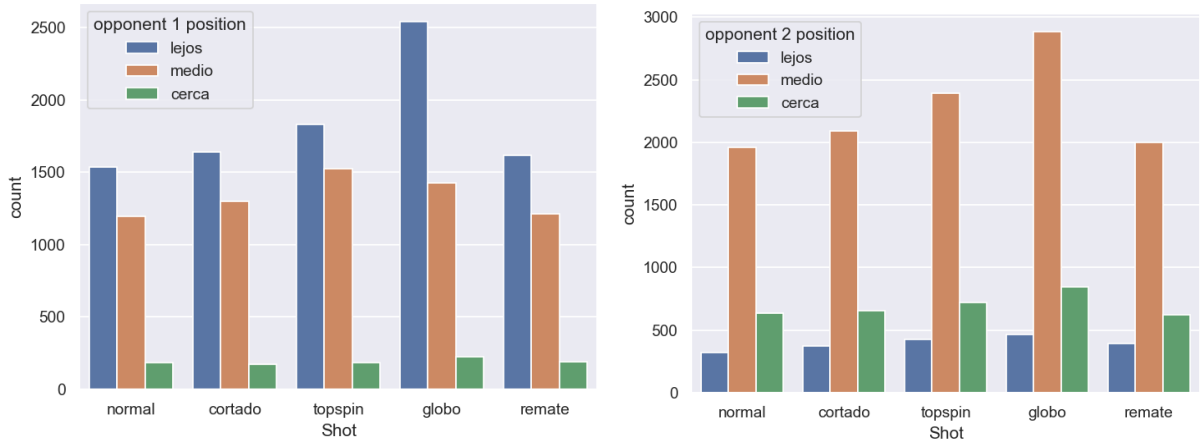
Mediante la Figura 17, podemos observar que los resultados obtenidos son significativamente mejores que en el experimento anterior. Los parámetros aumentan de manera progresiva, y el periodo de aprendizaje es mucho más corto. Podemos ver que los parámetros comienzan a crecer de manera constante a partir de los 3 millones de pasos. Este resultado concuerda con la nuestra expectativa, ya que al eliminar una acción continua, los agentes pueden aprender más rápido al tener menos parámetros que gestionar.



(a) Recompensa acumulada (b) Duración del episodio (c) Puntuación Elo

Figura 17: Resultados del experimento 2. (Elaboración propia)

El resultado mostrado en la Figura 18 es mucho más extremo que en el Experimento 1. Observamos que un agente del equipo se especializa en cubrir el centro del campo, mientras que otro se encarga de cubrir tanto el centro como la parte posterior del campo. Además, el tipo de golpe preferido es el globo. Sospechamos que los agentes utilizan este golpe para alargar la duración de cada episodio, ya que el globo es el golpe que permanece más tiempo en el aire.

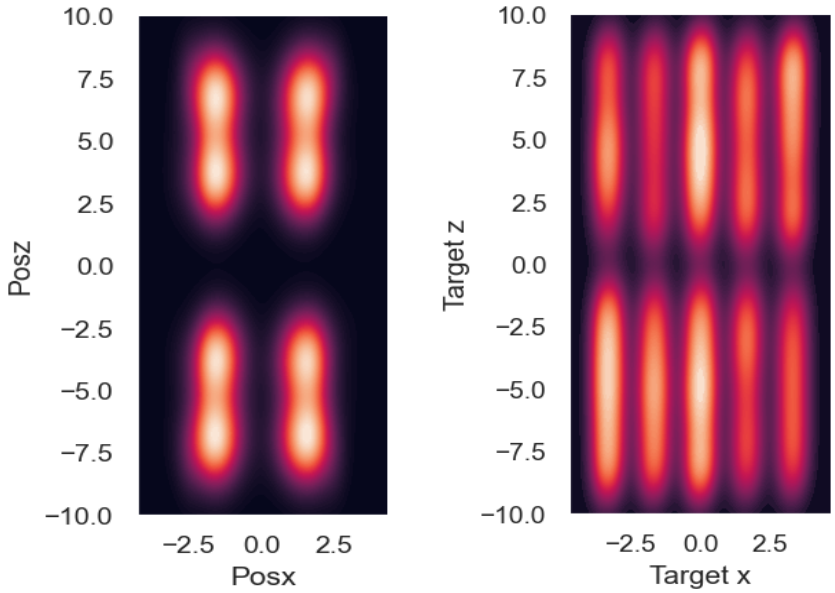


(a) Distancia de oponente 1

(b) Distancia de oponente 2

Figura 18: Experimento 2. Frecuencia absoluta de cada tipo de golpe, según la distancia de los oponentes cuando se devuelve la pelota (Elaboración propia)

La Figura 19a muestra un resultado interesante. Los agentes se concentran en el centro del campo, lo cual se debe a que siempre se mueven a una velocidad constante de 5 m/s, permitiéndoles cubrir una mayor área del campo si inician su movimiento desde el centro. En la Figura 19b, observamos que los agentes en el campo inferior (rango [-10, 0]) tienden a enviar la pelota hacia la parte derecha del campo contrario. De manera simétrica, los agentes en el campo superior presentan una tendencia similar, enviando la pelota hacia la derecha desde su perspectiva. Esto sugiere una estrategia coordinada para maximizar el control del juego y aprovechar la disposición espacial de sus posiciones.



(a) Posición de los agentes

(b) Posición donde envia la pelota

Figura 19: Experimento 2. Mapa de calor de los agentes (Elaboración propia)

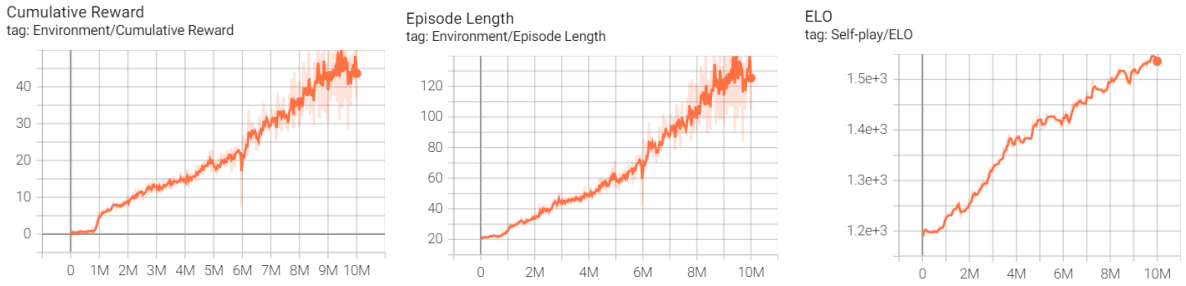
En el entorno de Unity, observamos que los agentes se mueven de forma más errática. Esto se debe a que el agente realiza una acción cada 0.2 segundos; en un instante se mueve hacia

la izquierda y en el siguiente instante puede moverse en sentido contrario. El Experimento 1 resolvía este problema al hacer que los agentes se movieran solo cuando era necesario, en lugar de moverse constantemente.

10.3. Experimento 3. Entrenamiento desactivando el control de la velocidad y altura de la pelota

En este experimento, similar al experimento 2, los agentes ya no pueden controlar la altura de la pelota al devolverla. En esta nueva condición, la altura de la devolución está predefinida según el efecto de la pelota seleccionada.

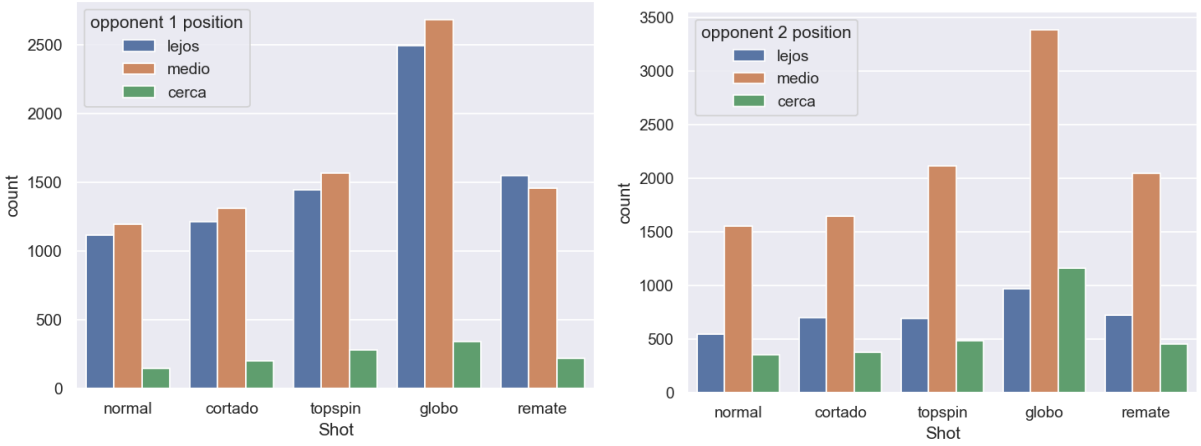
Mediante los gráficos de la Figura 20, podemos observar que los resultados obtenidos son casi idénticos a los del experimento 2. La única diferencia es que el ELO ha alcanzado un nuevo nivel. Esto sugiere que la funcionalidad de control de la altura de la pelota no tiene un impacto significativo en el entrenamiento.



(a) Recompensa acumulada (b) Duración del episodio (c) Puntuación Elo

Figura 20: Resultados del experimento 3. (Elaboración propia)

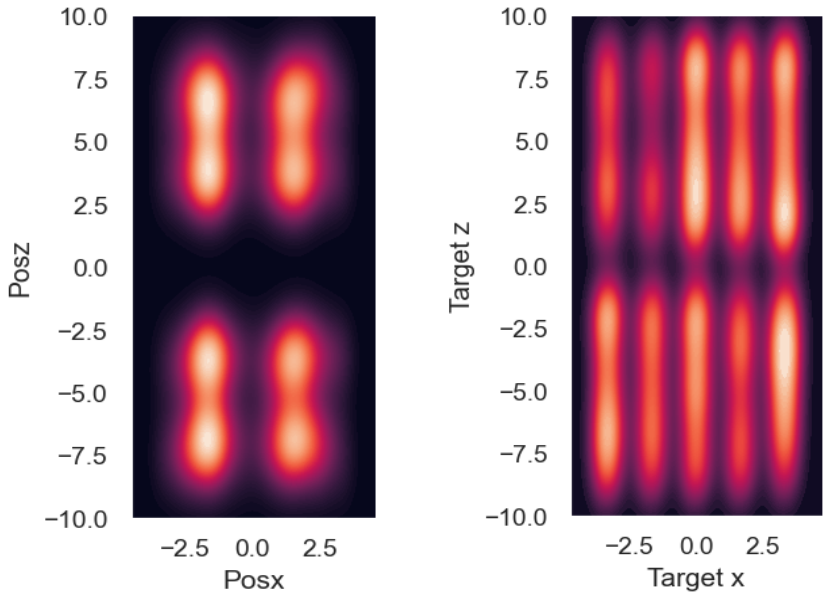
El resultado mostrado en la Figura 21 es casi idéntico al del experimento 2; el efecto más utilizado sigue siendo el globo. Además, los agentes continúan distribuyéndose en la parte media-trasera del campo.



(a) Distancia de oponente 1 (b) Distancia de oponente 2

Figura 21: Experimento 3. Frecuencia absoluta de cada tipo de golpe, según la distancia de los oponentes cuando se devuelve la pelota (Elaboración propia)

La Figura 22a muestra los mismos resultados que el experimento anterior: los agentes se concentran en el centro del campo, manteniendo una distancia adecuada entre sí. Sin embargo, en la Figura 19b se observan más diferencias, ya que las posiciones desde las que se devuelve la pelota están más distribuidas por todo el campo en lugar de concentrarse en una zona concreta.



(a) Posición de los agentes (b) Posición donde envia la pelota

Figura 22: Experimento 3. Mapa de calor de los agentes (Elaboración propia)

En el entorno de Unity, observamos el mismo comportamiento convulsivo de los agentes. Además, los agentes también han logrado devolver más pelotas, aunque sin un aumento significativo. Esto sugiere que el control de altura, en comparación con el control de velocidad, no tiene un impacto tan relevante en el rendimiento durante el entrenamiento.

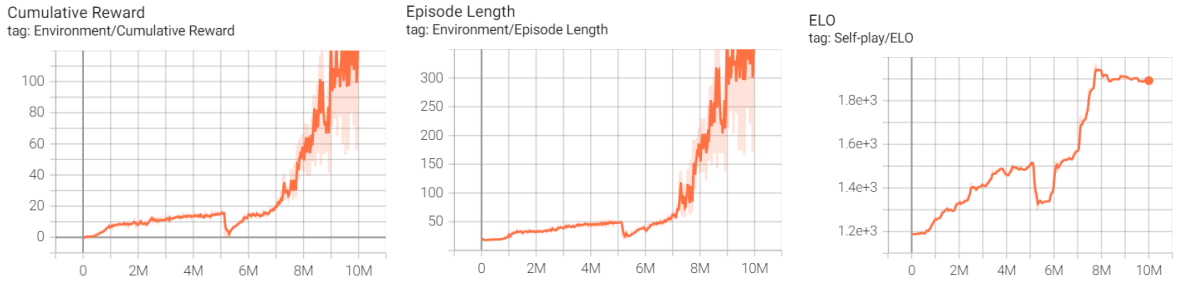
10.4. Experimento 4. Entrenamiento de aprendizaje por imitación, desactivando el control de la velocidad

En este experimento, observamos el comportamiento de los agentes utilizando las configuraciones de GAIL de Unity, desactivando el control de la velocidad de los agentes para ajustar su comportamiento. Durante los primeros 50.000 pasos de entrenamiento, utilizamos el algoritmo de clonación de comportamiento, obligando al agente a imitar las acciones del experto. Una vez completado este proceso, el entrenamiento continuará con GAIL.

Mediante las gráficas de la Figura 23, se puede observar que los resultados obtenidos son muy diferentes en comparación con otros experimentos. Notamos que, a los 5 millones de pasos, hubo una caída en los resultados, y a partir de los 7 millones, la gráfica muestra un rápido crecimiento, que al llegar cerca de 10 millones de pasos, se observa un cuello de botella donde la velocidad de crecimiento se ralentiza.

Mediante las gráficas, podemos observar los resultados obtenidos son mucho mejores que los anteriores, que por ejemplo el ELO final se alcanza 1900 y la longitud de un episodio llega

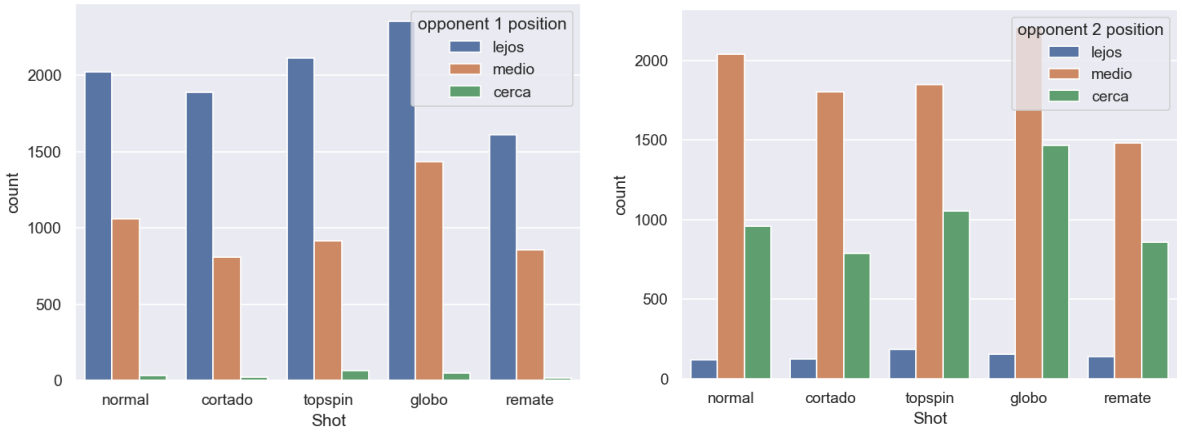
a más de 300 pasos (devoluciones de la pelota).



(a) Recompensa acumulada (b) Duración del episodio (c) Puntuación Elo

Figura 23: Resultados del experimento 4. (Elaboración propia)

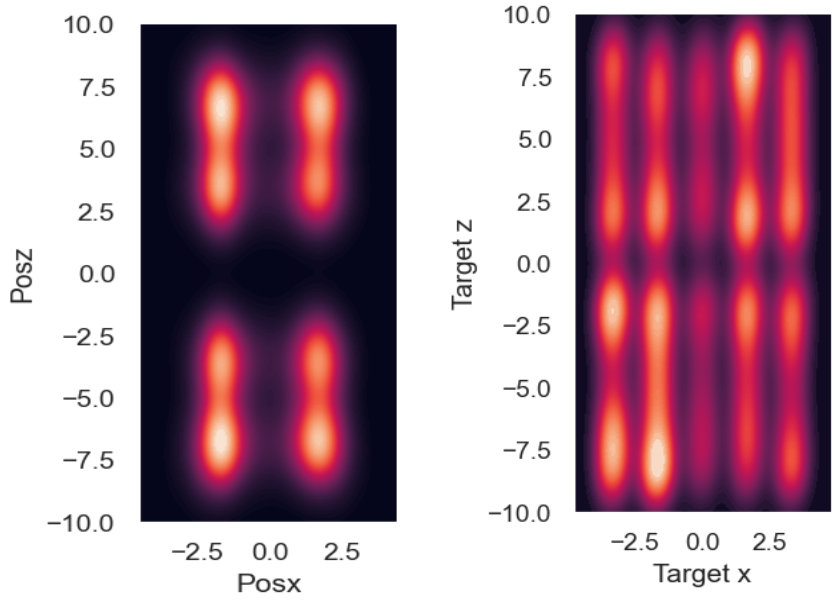
Los resultados de la Figura 24 son prácticamente equivalentes a los del experimento 2, el globo sigue siendo el más utilizado por los agentes. Al igual que en el experimento 2, los agentes continúan aplicando la estrategia de cubrir la zona media-trasera del campo.



(a) Distancia de oponente 1 (b) Distancia de oponente 2

Figura 24: Experimento 4. Frecuencia absoluta de cada tipo de golpe, según la distancia de los oponentes cuando se devuelve la pelota (Elaboración propia)

Las figuras 25a y 25b son prácticamente iguales a las de los experimentos anteriores: los agentes se ubican principalmente en el centro del campo, manteniendo una distancia adecuada entre ellos. No obstante, este experimento presenta una diferencia notable: la distribución de las posiciones desde las que se devuelve la pelota es más dispersa por todo el campo.



(a) Posición de los agentes (b) Posición donde envia la pelota

Figura 25: Experimento 4. Mapa de calor de los agentes (Elaboración propia)

En el entorno de Unity, se obtuvieron resultados significativamente mejores que en los experimentos anteriores. Los agentes ya no presentan movimientos tan convulsivos, logrando devolver más pelotas, aunque con facilidad para que el equipo contrario también pueda alcanzarlas. Esta estrategia les permite obtener una recompensa mayor que vencer al rival, lo cual nos lleva a pensar que ajustando los pesos de las diferentes recompensas, podríamos obtener comportamientos más próximos al juego real.

El aprendizaje por imitación, basado en un modelo predefinido para el comportamiento de los agentes, supone una notable aceleración del proceso de entrenamiento. Cabe suponer que la utilización de un conjunto de datos más extenso para entrenar el modelo potenciaría aún más esta aceleración y permitiría alcanzar resultados aún mejores. Si bien los datos utilizados, de una final femenina en un circuito profesional, contenía más de 21.000 fotogramas, en realidad la variedad de situaciones de juego era muy limitada, si se tiene en cuenta que, debido a la coherencia temporal, la variación de estado entre fotogramas consecutivos es muy pequeña.

10.5. Comparación de los experimentos

En esta sección comparamos las experimentaciones anteriores para ver los impactos de cada parámetro en el entrenamiento.

En la siguiente tabla 13 se puede encontrar las gráficas de la recompensa, el episodio y el ELO de 4 experimentos. A partir de esta tabla, se observa que el Experimento 1 presenta una tendencia muy inestable en comparación con los otros tres. Esto se debe a que, en el Experimento 1, el agente controla su velocidad de movimiento. Esta acción continua dificulta el proceso de aprendizaje del agente, lo que resulta en un rendimiento inferior en comparación con los otros experimentos.

Ex.1	Ex.2	Ex.3	Ex.4

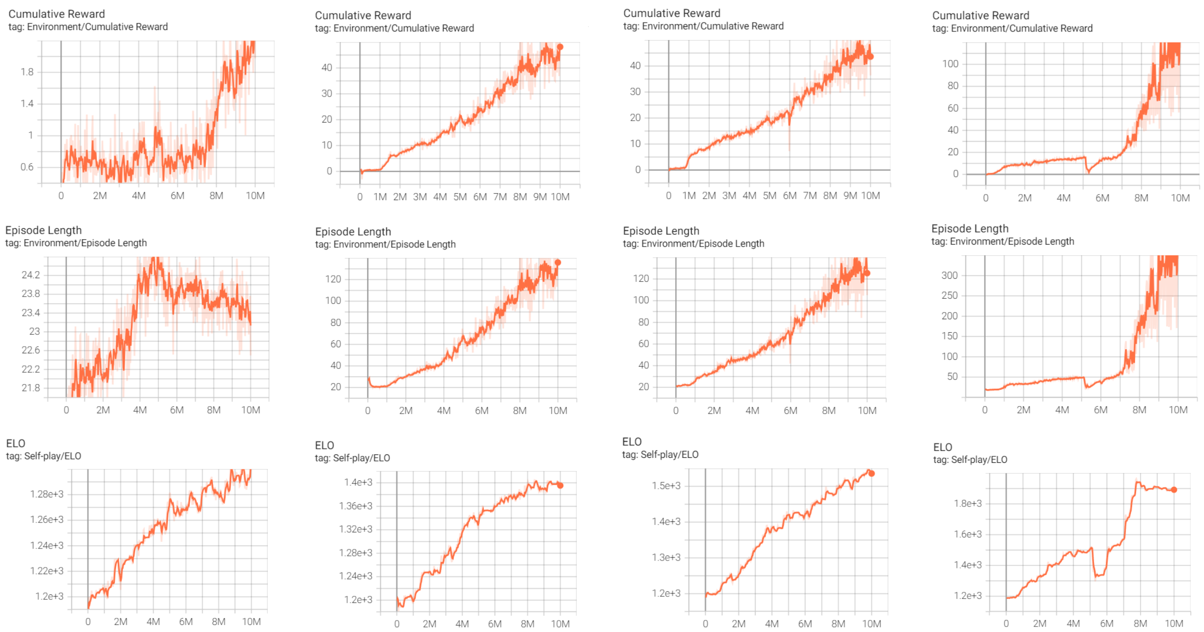


Tabla 13: Tabla de recompensa, ELO y episodio de 4 experimentos (Elaboración propia)

En la siguiente tabla 14 se muestran las gráficas de la posición de los agentes durante un periodo de tiempo y la dirección en la que envían la pelota. Comparando las gráficas, podemos observar que en el Experimento 1, los agentes se mantienen en la parte posterior del campo, mientras que en los otros experimentos se encuentran principalmente en el centro del campo. Esta distribución se produce porque en el Experimento 1, los agentes aprovechan la recompensa que reciben por moverse lentamente y permanecen en la posición inicial. En cambio, en los otros experimentos, los agentes se mueven a una velocidad constante de 5 m/s, lo que les permite llegar mejor a cualquier punto desde el centro de la pista.

También observamos que en los Experimentos 1 y 4, los agentes tienen una tendencia a devolver la pelota hacia los bordes del campo. En comparación con los otros experimentos, la distribución es más equilibrada.

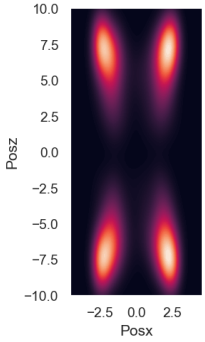
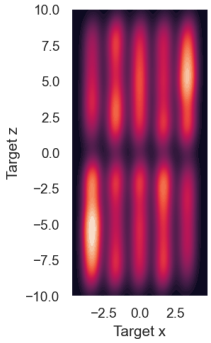
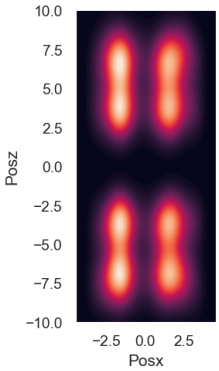
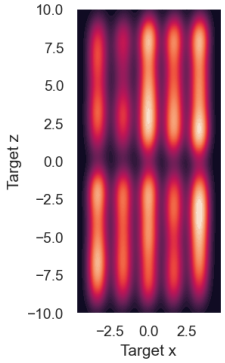
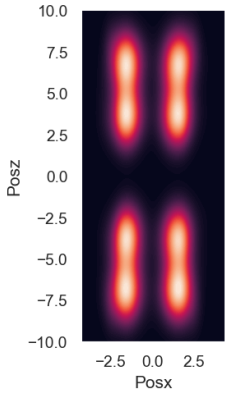
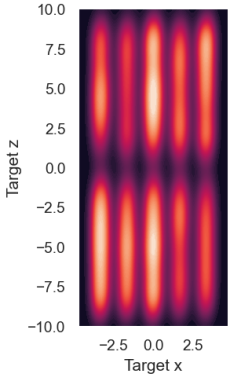
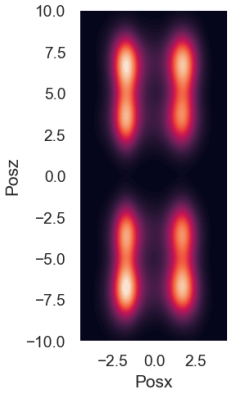
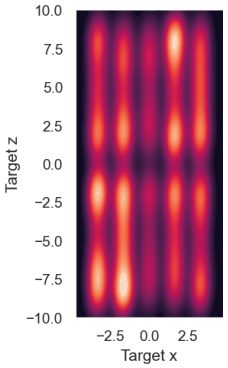
-	Posición de agente	Dirección de golpeo	Posición de agente	Dirección de golpeo
Ex.1			Ex.3 	
Ex.2			Ex.4 	

Tabla 14: Tabla de la posición de agentes y la dirección de golpeo de 4 experimentos (Elaboración propia)

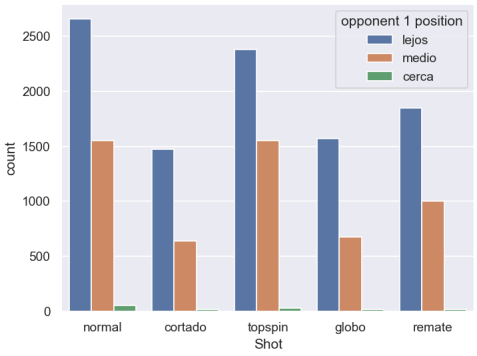
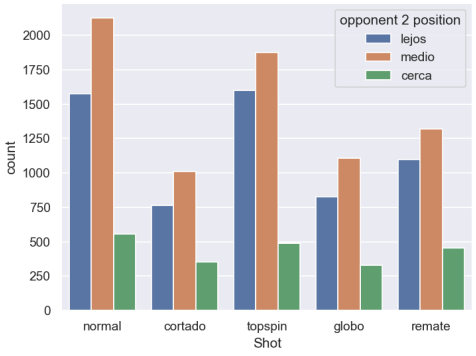
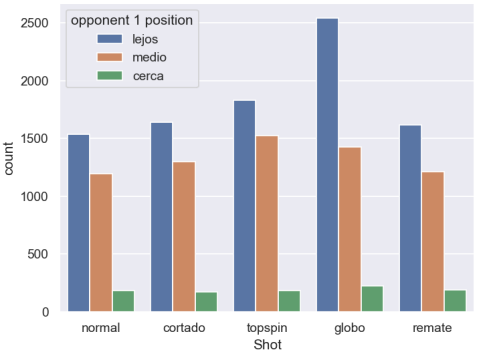
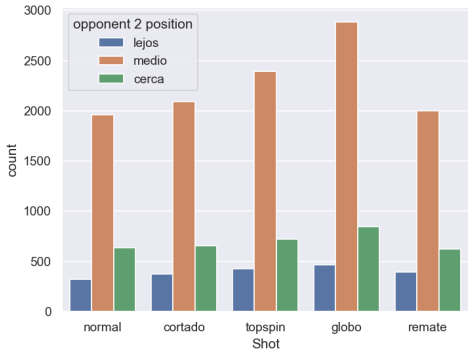
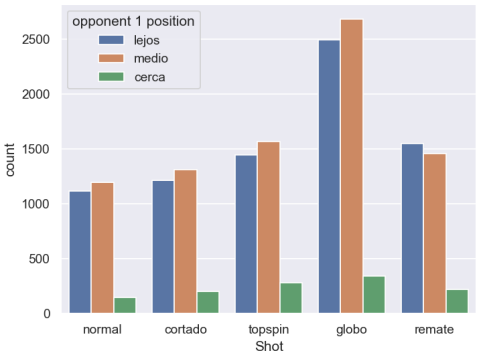
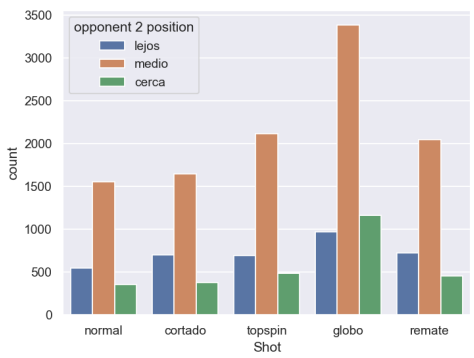
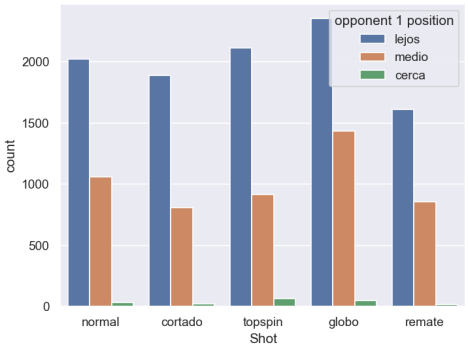
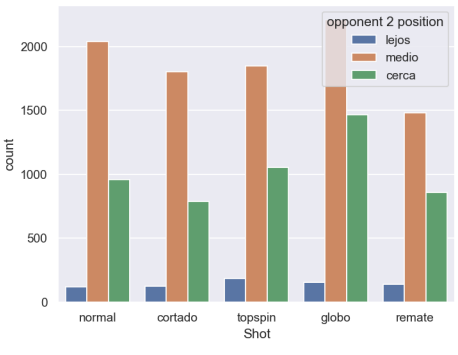
-	Oponente 1	Oponente 2
Ex.1		
Ex.2		
Ex.3		
Ex.4		

Tabla 15: Tabla de la frecuencia de golpe, respeto la distancia de oponentes de 4 experimentos (Elaboración propia)

En la tabla 15 se muestran las gráficas de la frecuencia de golpeo en relación con la distancia de los oponentes. En el Experimento 1, los agentes prefieren utilizar el golpe normal y el topspin. Sin embargo, en los otros experimentos, el golpe más común es el globo. Una posible explicación para este fenómeno es que los agentes tienen una velocidad de movimiento muy

alta. En lugar de enviar la pelota cerca de la red, donde los oponentes podrían devolverla fácilmente con un rebote rápido, los agentes prefieren enviar la pelota al fondo del campo. Esto reduce el tiempo de respuesta de los oponentes y les dificulta devolver la pelota.

Para investigar más a fondo, se podría reducir la velocidad máxima de movimiento de los agentes y observar cómo cambia su comportamiento cuando no pueden cubrir todo el campo tan rápidamente. Estos experimentos adicionales forman parte del trabajo futuro de este proyecto.

11. Planificación final

11.1. Cambios en la planificación

A continuación, se explicarán detalladamente los cambios en la planificación tras la finalización del desarrollo del proyecto.

ID	Tarea	Tiempo	Dependencias
T1	**Gestión del proyecto**	**180h**	**-**
T1.1	**Alcance**	**20h**	**[]**
T1.2	**Planificación**	**15h**	**[T1.1]**
T1.3	**Presupuesto**	**7.5h**	**[T1.2]**
T1.4	**Informe de sostenibilidad**	**7.5h**	**[T1.2]**
T1.5	**Reuniones**	**25h**	**[]**
T1.6	**Documentación**	**90h**	**[]**
T1.7	**Presentación**	**15h**	**[T1.6]**
T2	**Trabajo previo**	**115h**	**-**
T2.1	**Estudio del estado del arte**	**25h**	**[]**
T2.2	**Familiarización de la Implementación**	**20h**	**[T2.1]**
T2.3	**Familiarización con ML-Agents**	**20h**	**[]**
T2.4	**Estudio de la documentación**	**50h**	**[]**
T3	**Mejora del entorno virtual en Unity**	**140h**	**-**
T3.1	**Simulación de la física en torno de entrenamiento**	**50h**	**[T2.1,T2.2]**
T3.2	**Implementación**	**60h**	**[T3.1]**
T3.3	**Testing**	**10h**	**[T3.2]**
T3.4	**Entrenamiento de modelos RL**	**20h**	**[T3.3]**
T4	**Desarrollo de técnicas de IL**	**90h**	**-**
T4.1	**Procesamiento de datos**	**25h**	**[T3]**
T4.2	**Diseño del modelo**	**10h**	**[T4.1]**
T4.3	**Implementación**	**35h**	**[T4.2]**
T4.4	**Testing**	**10h**	**[T4.3]**
T4.5	**Entrenamiento de modelos IL**	**15h**	**[T4.4]**
T5	**Mejorar la interfaz gráfica de usuario**	**10h**	**-**
T5.1	**Diseño**	**3h**	**[T3]**
T5.2	**Implementación**	**5h**	**[T5.1]**
T5.3	**Testing**	**2h**	**[T5.2]**
-	**Total**	**540h**	**-**

Tabla 16: La planificación final. (Elaboración propia)

En los bloques T1 y T2, solo hubo cambios en la duración de las tareas, principalmente en el estudio y desarrollo de las documentaciones. La profundización en el desarrollo también requirió más tiempo para elaborar los informes de los diferentes componentes del proyecto.

En los bloques T3 y T4, se observó un aumento en el tiempo de desarrollo de T3 debido a la aparición de nuevas necesidades. Por otro lado, el tiempo de desarrollo de T4 se redujo, ya que resultó ser menos complicado de lo previsto. Además, se requirió tiempo adicional para

entrenar diferentes modelos y verificar la efectividad de los cambios realizados en el entorno, lo cual no se había contemplado en la planificación inicial. Por otra parte, no fue necesario adaptar el entorno para IL, dado que el entorno virtual ya es compatible con ambos tipos de entrenamiento.

En el bloque T5 se ha reducido el tiempo de desarrollo, ya que no es un enfoque principal del proyecto. Se decidió centrarse principalmente en entrenamiento. En este bloque desarrollaron pequeñas mejoras en la interfaz de usuario para adaptar nuevas funcionalidades del entorno.

Después de los cambios producidos, se ha actualizado la planificación 16 y el diagrama de Gantt en el apéndice A.2.

12. Conclusiones

En este proyecto se han explorado diferentes algoritmos de Aprendizaje por Refuerzo (RL) y Aprendizaje por Imitación (IL) para comprender su funcionamiento al entrenar a un agente para jugar a pádel en un entorno complejo. Tomando como punto de partida la implementación del TFG de Jia Long Ji Qiu, este proyecto ha evitado la necesidad de comenzar desde cero, permitiendo reutilizar el entorno virtual existente. Esta base ha facilitado la implementación de nuevas funcionalidades, como la reimplementación de la física de la pelota y la ampliación del espacio de acciones y observaciones para los agentes. Estas mejoras han permitido que los jugadores virtuales se comporten de manera más realista y desarrollen estrategias más eficientes.

Utilizando los algoritmos disponibles dentro Unity ML-Agents, hemos obtenido resultados muy satisfactorios que cumplen con nuestros objetivos iniciales. Con la nueva configuración del entorno y el entrenamiento mediante Proximal Policy Optimization y self-play, hemos acelerado significativamente la velocidad de entrenamiento. Ahora, el modelo puede completar hasta 10 millones de pasos en un tiempo mucho más corto, evitando esperas de varios días. Esta mejora nos permite entrenar múltiples modelos con diferentes configuraciones para comparar y determinar cuál es el mejor.

Al comparar los diferentes resultados de los modelos, observamos que el control de la velocidad permite simular la resistencia humana, que los agentes intentan realizar acciones con un mínimo esfuerzo. Por otro lado, el modelo entrenado con IL muestra los mejores resultados, alcanzando un ELO significativamente superior a otros modelos tras 10 millones de pasos. Sin embargo, debido a la falta de datos de un jugador real, el modelo IL no se entrena en las mismas condiciones que el modelo RL. Por lo tanto, solo podemos concluir que, en términos de velocidad de entrenamiento, IL es más rápido que RL para entrenar un agente de pádel.

12.1. Trabajo futuro

Aunque logramos cumplir nuestro objetivo y obtener resultados satisfactorios, todavía queda mucho margen de mejora. Durante el desarrollo del proyecto, surgieron varias ideas para posibles mejoras. Algunas propuestas son las siguientes:

- Una situación a la que nos enfrentamos durante el entrenamiento con IL es la insuficiencia de datos de un jugador real. El conjunto de datos que utilizamos actualmente es insuficiente para grabar una demostración adecuada, lo que provoca que los agentes repitan ciertas acciones debido a la falta de muestras en esas situaciones. Por lo tanto, en trabajos futuros, sería beneficioso recopilar datos adicionales para obtener una muestra más representativa y variada. También podrían utilizarse técnicas de *data augmentation*, considerando por ejemplo ciertas simetrías en el comportamiento de los jugadores (por ejemplo, jugador de derecha y jugador de revés).
- Uno de los problemas que se va a encontrar es el comportamiento de la física de la pelota. Debido a que la simulación precisa de la física de la pelota podría ser una tarea costosa y no es el foco principal del proyecto, no se profundizó en este aspecto. Actualmente, se utiliza un modelo de movimiento uniformemente acelerado que no considera la fricción del aire. En el futuro, se podría introducir la fuerza de Magnus a la pelota para lograr un movimiento más realista, si bien se considera prioritario sustituir el avatar del jugador

por un modelo antropométrico más adecuado.

- Aumentar el espacio de acciones permitirá a los agentes realizar más acciones, logrando un comportamiento más similar al de un jugador real. Por ejemplo, permitir que los agentes puedan saltar para golpear la pelota en el aire. También se puede restringir que los agentes no puedan golpear la pelota que se encuentra detrás de ellos; para hacerlo, tendrían que girarse 180º, entre otras acciones.
- En este proyecto, no nos hemos centrado en estudiar los efectos de las recompensas, sino en los efectos de las nuevas funcionalidades del entorno y las acciones de los agentes. De cara al futuro, se podrían realizar experimentos con diferentes configuraciones de hiperparámetros y recompensas para estudiar sus comportamientos y optimizar el rendimiento del modelo.

Referencias

[1] *Entornos 3D de alta fidelidad para Realidad Virtual y Computación Visual: geometría, movimiento, interacción y visualización para salud, arquitectura y ciudades*. 2022-09-01. URL: https://futur.upc.edu/34220564 visted at 24-02-2024.

[2] K. Nikolopoulou. *Easy introduction to reinforcement learning*. en-US. Ago. de 2023. URL: https://www.scribbr.com/ai-tools/reinforcement-learning/ visted at 24-02-2024.

[3] J. l. Ji qiu. *entorno virtual de Pádel para el Aprendizaje por refuerzo (Trabajo de Fin de Grado)*. 2024. URL: https://github.com/jialongjq/tfg?tab=readme-ov-file visted at 12-02-2024.

[4] I. Lopez Rodríguez. *Agentes Inteligentes para imitar el comportamiento de los jugadores de Pádel*. Oct. de 2022. URL: https://upcommons.upc.edu/handle/2117/374020 visted at 23-02-2024.

[5] Mayo de 2024. URL: https://www.padelvrgame.com/.

[6] A. Howard, A. Raichuk, B. Rondepierre, C. F. Group, G. Swimer, M. Michalski, P. Stanczyk, S. Harris y tsmall. *Google Research Football with Manchester City F.C.* 2020. URL: https://kaggle.com/competitions/google-football.

[7] *Top 6 Reinforcement Learning Tools to Use*. URL: https://www.turing.com/kb/best-tools-for-reinforcement-learning visted at 24-02-2024.

[8] *Metodologías ágiles: Qué Son, Para qué sirven y tipos*. URL: https://www.esdesignbarcelona.com/actualidad/design-management/metodologias-agiles-que-son-para-que-sirven-tipos visted at 23-02-2024.

[9] *Sistema de evaluación y créditos*. URL: https://www.upc.edu/sri/es/movilidad/movilidad-estudiantes/incomings/estudiar-en-la-upc/copy_of_sistema-de-evaluacion-y-creditos visted at 29-02-2024.

[10] URL: https://es.talent.com/salary visted at 09-03-2024.

[11] Feb. de 2024. URL: https://ayudatpymes.com/gestron/cuanto-se-paga-de-seguros-sociales-por-cada-trabajador/.

[12] *OpenAI Spinning Up Documentation - Part 2: Kinds of RL Algorithms*. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html visted at 12-05-2024.

[13] S. AI. *A brief overview of imitation learning*. Sep. de 2019. URL: https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c.

[14] S. Arora y P. Doshi. «A survey of inverse reinforcement learning: Challenges, methods and progress». En: *Artificial Intelligence* 297 (ago. de 2021), pág. 103500. DOI: 10.1016/j.artint.2021.103500. URL: https://www.sciencedirect.com/science/article/abs/pii/S0004370221000515.

[15] J. Ho y S. Ermon. *Generative Adversarial Imitation Learning*. 2016. arXiv: 1606.03476 [cs.LG].

[16] *Unity ML-Agents Toolkit*. URL: https://github.com/Unity-Technologies/ml-agents.

[17] Y. Gu, Y. Cheng, C. L. P. Chen y X. Wang. «Proximal Policy Optimization With Policy Feedback». En: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52.7 (2022), págs. 4600-4610. DOI: 10.1109/TSMC.2021.3098451.

[18] Y. Bai, C. Jin y T. Yu. «Near-Optimal Reinforcement Learning with Self-Play». En: *Advances in Neural Information Processing Systems*. Ed. por H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan y H. Lin. Vol. 33. Curran Associates, Inc., 2020, págs. 2159-2170. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/172ef5a94b4dd0aa120c6878fc29f70c-Paper.pdf.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford y O. Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[20] Unity. *Training Configuration File - Unity ML-Agents Toolkit*. URL: https://unity-technologies.github.io/ml-agents/Training-Configuration-File/.

Apéndices

A. Diagramas de Gantt

A.1. Diagrama de Gantt de la planificación inicial

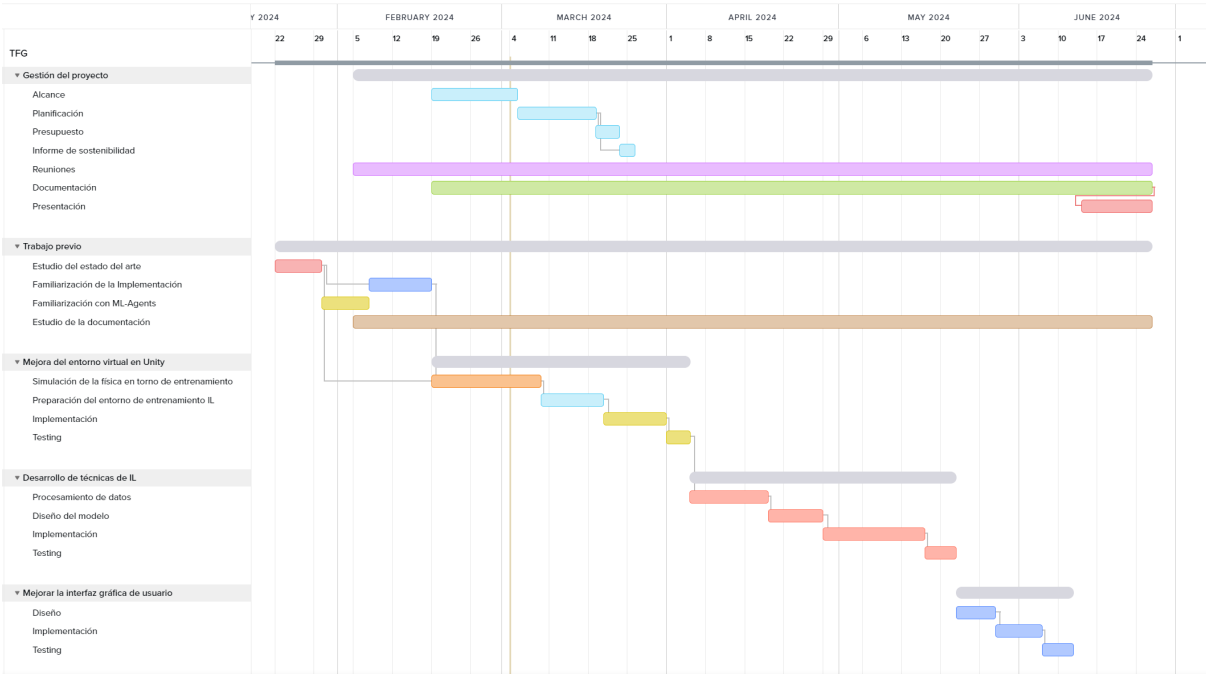


Figura 26: Diagrama de Gantt de la planificación inicial. (Elaboración propia)

A.2. Diagrama de Gantt de la planificación final

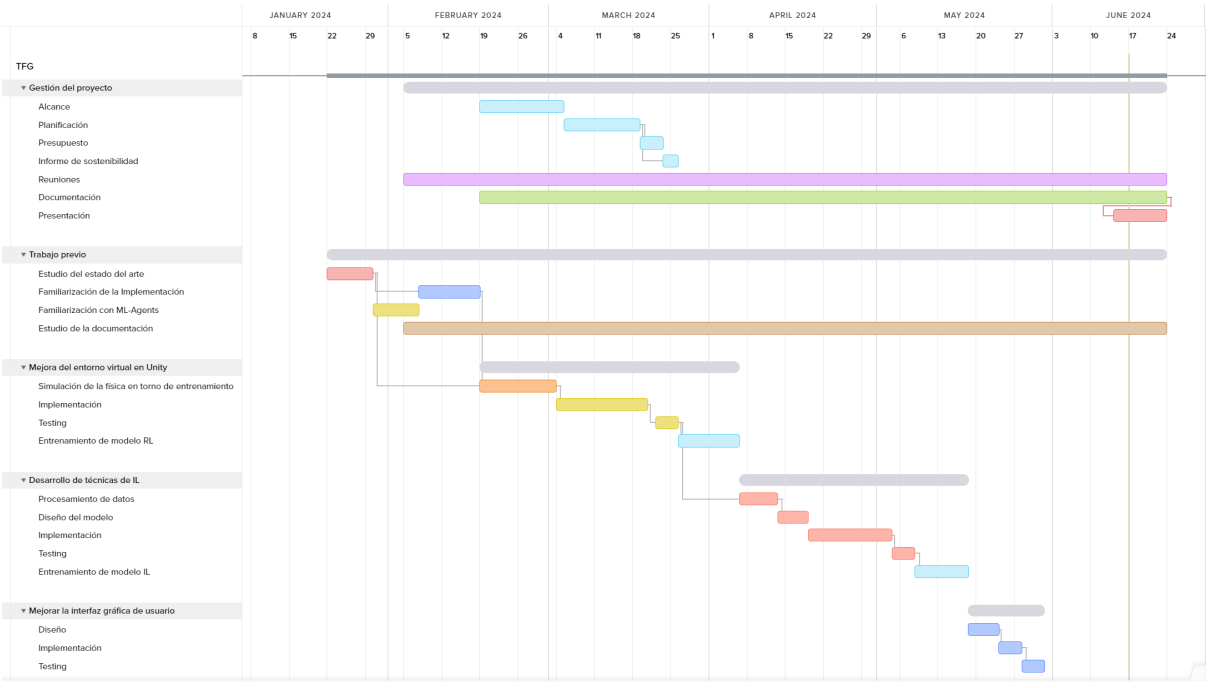


Figura 27: Diagrama de Gantt de la planificación final(Elaboración propia)

B. Configuraciones del entrenamiento

B.1. Configuración del RL

```
behaviors:
  PadelAgent:
    trainer_type: ppo
    summary_freq: 25000
    max_steps: 10000000
    keep_checkpoints: 5
    hyperparameters:
      learning_rate: 0.0003
      learning_rate_schedule: linear
      batch_size: 512
      buffer_size: 20480
      beta: 0.005
      beta_schedule: linear
      epsilon: 0.2
      epsilon_schedule: linear
      num_epoch: 3
    network_settings:
      hidden_units: 64
      num_layers: 4
    reward_signals:
      extrinsic:
        gamma: 0.95
        strength: 1.0
    self_play:
      save_steps: 50000
      swap_steps: 50000
      team_change: 100000
      window: 15
      play_against_latest_model_ratio: 0.5
      initial_elo: 1200.0
```

B.2. Configuración del IL

```
behaviors:
  PadelAgent:
    trainer_type: ppo
    summary_freq: 25000
    max_steps: 10000000
    keep_checkpoints: 5
    hyperparameters:
      learning_rate: 0.0003
      learning_rate_schedule: linear
      batch_size: 512
      buffer_size: 20480
      beta: 0.005
      beta_schedule: linear
      epsilon: 0.2
      epsilon_schedule: linear
      num_epoch: 3
    network_settings:
      hidden_units: 64
      num_layers: 4
    reward_signals:
      extrinsic:
        gamma: 0.95
        strength: 1.0
      gail:
        gamma: 0.99
        strength: 0.01
        learning_rate: 0.0003
        use_actions: false
        use_vail: false
        demo_path: ./demon
    behavioral_cloning:
      demo_path: ./demon
      steps: 50000
      strength: 1.0
      samples_per_update: 0
    self_play:
      save_steps: 50000
      swap_steps: 50000
      team_change: 100000
      window: 15
      play_against_latest_model_ratio: 0.5
      initial_elo: 1200.0
```