



Entrenamiento de agentes virtuales para el aprendizaje de políticas óptimas de los jugadores durante un partido de pádel

Jia Long Ji Qiu

Director/a: Carlos Andújar Gran (Departamento de Ciencias de la Computación)

Codirector/a: Mohammadreza Javadiha (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

23/01/2024

Agradecimientos

Quisiera mostrar mi gratitud hacia aquellas personas que me han acompañado y apoyado a lo largo del desarrollo de este trabajo de fin de grado. En especial a mi director, Carlos Andújar, y codirector, Mohammadreza Javadiha, por toda la ayuda y enseñanza recibida. Agradecer también a mis amigos, quienes me han introducido al pádel y servido como fuente de inspiración, haciendo de este proyecto un proceso más ameno. Finalmente a mi familia, a quien más debo, sin la cual no hubiera podido llegar hasta aquí.

Resumen

En videojuegos o simuladores, una de las tareas más laboriosas suele ser la atribución de inteligencia a personajes no jugables, especialmente cuando se trata de entornos con un grado elevado de complejidad. En este trabajo de fin de grado se exploran alternativas a la programación tradicional para simular el comportamiento humano en entidades virtuales. En particular, se estudian diversas técnicas de aprendizaje por refuerzo y por imitación para entrenar agentes que, con un conocimiento básico de un entorno de pádel, aprendan a tomar decisiones óptimas en cuanto a posición en la pista, desplazamientos, y acción técnica a realizar.

A lo largo de este proyecto, se han estudiado conceptos clave de los métodos de aprendizaje por refuerzo sin modelo, desde métodos basados en el valor como Q-learning, hasta métodos basados en la política como REINFORCE, con la finalidad de entender la base teórica que hay detrás de algunos algoritmos del estado del arte. Estos algoritmos se han utilizado para entrenar agentes en un entorno de aprendizaje personalizado basado en el pádel implementado en Unity, utilizando el Unity ML-Agents Toolkit. Durante los experimentos, se obtuvieron resultados muy interesantes tras el entrenamiento de agentes mediante el algoritmo de Proximal Policy Optimization, en los que los agentes eran capaces de tomar decisiones muy naturales en diversos escenarios.

Resum

A videojocs o simuladors, una de les tasques més laborioses sol ser l’atribució d’intel·ligència a personatges no jugables, especialment quan es tracta d’entorns amb un grau elevat de complexitat. En aquest treball de fi de grau s’exploren alternatives a la programació tradicional per simular el comportament humà en aquests personatges. En particular, s’estudien diverses tècniques d’aprenentatge per reforç i per imitació per entrenar agents que, amb un coneixement bàsic d’un entorn de pàdel, aprenguin a prendre decisions òptimes pel que fa a la posició a la pista, els desplaçaments i l’acció tècnica a realitzar.

Al llarg d’aquest projecte, s’han estudiat conceptes clau dels mètodes d’aprenentatge per reforç sense model, des de mètodes basats en el valor com el Q-learning, fins a mètodes basats en la política com el REINFORCE, amb la finalitat d’entendre la base teòrica que hi ha darrere d’alguns algorismes de l’estat de l’art. Aquests algoritmes s’han fet servir per entrenar agents en un entorn d’aprenentatge personalitzat basat en el pàdel implementat a Unity, utilitzant l’Unity ML-Agents Toolkit. Durant els experiments, es van obtenir resultats molt interessants després de l’entrenament d’agents mitjançant l’algorisme de Proximal Policy Optimization, en els que els agents eren capaços de prendre decisions molt naturals a diversos escenaris.

Abstract

In video games or simulators, one of the most laborious tasks is usually the attribution of intelligence to non-playable characters, especially when dealing with environments with a high degree of complexity. This bachelor's degree final project explores alternatives to traditional programming to simulate human behavior in those characters. In particular, various reinforcement and imitation learning techniques are studied in order to train agents who, with a basic knowledge of the padel environment, will eventually learn to make optimal decisions regarding position on the court, movements, and technical action to be performed.

Throughout this project, many key concepts of model-free reinforced learning methods have been studied, from value-based methods like Q-learning to policy-based methods such as REINFORCE, in order to understand the underlying theory behind some of the state-of-the-art algorithms. These algorithms have then been used to train agents in a customized padel-based learning environment implemented in Unity, using the Unity ML-Agents Toolkit. During the experiments, many positive results were obtained after training agents using Proximal Policy Optimization, where agents were able to make human-like decisions on different scenarios.

Índice

1. Introducción y contextualización	1
1.1. Contexto	1
1.2. Conceptos	1
1.2.1. Aprendizaje por refuerzo	1
1.2.2. Aprendizaje por imitación	2
1.3. Identificación del problema	3
1.3.1. Diseño y desarrollo de un entorno virtual de pádel	3
1.3.2. Simulación de agentes capaces de jugar a pádel	3
1.4. Agentes implicados	4
2. Justificación	5
2.1. Estudio de soluciones existentes	5
2.1.1. Motor gráfico	5
2.1.2. Entorno de entrenamiento	5
3. Alcance	7
3.1. Objetivos	7
3.2. Requerimientos	7
3.2.1. Requerimientos funcionales	7
3.2.2. Requerimientos no funcionales	8
3.3. Obstáculos y riesgos	8
4. Metodología	9
4.1. Herramientas	9
5. Planificación temporal	10
5.1. Descripción de las tareas	10
5.1.1. Gestión del proyecto [T1]	10
5.1.2. Trabajo previo [T2]	11
5.1.3. Desarrollo del entorno virtual en Unity [T3]	11
5.1.4. Desarrollo de técnicas de aprendizaje por refuerzo profundo [T4]	12
5.1.5. Desarrollo de técnicas de aprendizaje por imitación [T5]	12
5.1.6. Desarrollo de la interfaz gráfica de usuario [T6]	12
5.2. Recursos	13
5.2.1. Recursos humanos	13
5.2.2. Recursos materiales	13
5.3. Gestión del riesgo	15
6. Gestión económica	16
6.1. Presupuesto	16
6.1.1. Costes de personal	16
6.1.2. Costes genéricos	17
6.1.3. Contingencia	18
6.1.4. Imprevistos	19
6.1.5. Coste total	19
6.2. Control de gestión	20

7. Sostenibilidad	21
7.1. Autoavaluación	21
7.2. Dimensión económica	21
7.3. Dimensión ambiental	22
7.4. Dimensión social	22
8. Estudio previo	23
8.1. Estructura de un problema de RL	23
8.1.1. Proceso de decisión de Márkov	23
8.1.2. Espacio de observaciones y estados	24
8.1.3. Espacio de acciones	24
8.1.4. Retorno, episodios y el factor de descuento	24
8.2. RL basado en modelos y sin modelos	25
8.3. Métodos sin modelos	26
8.3.1. Métodos basados en el valor	26
8.3.2. Métodos basados en la política	33
8.4. Unity ML-Agents Toolkit	37
8.4.1. Componentes clave	37
8.4.2. ML-Agents en la interfaz de Unity	39
8.4.3. Métodos de entrenamiento implementados en ML-Agents	41
8.4.4. Configuración del entrenamiento	47
9. Desarrollo	51
9.1. Experimento piloto	51
9.1.1. Reglamento del entorno virtual	51
9.1.2. Diseño inicial	52
9.1.3. Implementación inicial	53
9.1.4. Problemas de la versión inicial	54
9.2. Versión final del entorno virtual	55
9.2.1. Diseño final	55
9.2.2. Implementación final	57
9.2.3. Implementación específica para la grabación de demostraciones	67
10. Resultados	73
10.1. Aprendizaje por refuerzo profundo	73
10.1.1. Experimento 1. Entrenamiento mediante PPO	74
10.1.2. Experimento 2. Efecto de Curiosity en el entorno virtual de pádel	75
10.1.3. Experimento 3. Influencia de la recompensa por golpeo	76
10.1.4. Experimento 4. Importancia del sistema de puntuación Elo	77
10.1.5. Experimento 5. Influencia de las posiciones clave	78
10.2. Aprendizaje por imitación	79
11. Planificación final	80
11.1. Cambios en la planificación	80
11.2. Planificación final	80
12. Conclusiones	82
12.1. Trabajo futuro	82
Referencias	84

A. Diagramas de Gantt	86
A.1. Diagrama de Gantt de la planificación inicial	86
A.2. Diagrama de Gantt de la planificación final	87
B. Configuraciones del entrenamiento	88
B.1. Configuración del entrenamiento mediante PPO	88
B.2. Configuración del entrenamiento mediante PPO + Curiosity	89

Índice de figuras

1.	Flujo genérico del aprendizaje por refuerzo	2
2.	Entorno virtual de pádel	4
3.	Ciclo de desarrollo ágil	9
4.	Taxonomía de algoritmos de RL	26
5.	Diagrama de bloques del entorno de ML-Agents	38
6.	Ejemplo del componente Behavior Parameters en Unity. (Elaboración propia) .	40
7.	Ejemplo del componente Decision Requester en Unity. (Elaboración propia) .	40
8.	Ejemplo del componente Demonstration Recorder en Unity. (Elaboración propia)	40
9.	Gradiente de la función sustituto L^{CLIP}	42
10.	Módulo de Curiosidad Intrínseca (ICM)	46
11.	Entorno virtual de pádel en Unity	52
12.	Diagrama de flujo simplificado del entorno virtual de pádel	54
13.	Entorno virtual de pádel con el modo de depuración activado	56
14.	Posiciones de servicio del entorno virtual de pádel	59
15.	Cuadrícula de la pista de padel	60
16.	Posiciones de interés de devolución	63
17.	Ejemplo de una posición óptima de cobertura	64
18.	Posiciones de interés de cobertura	65
19.	Ejemplo gráfico de una única muestra del conjunto de datos utilizado para grabar demostraciones	68
20.	Conjunto de datos utilizado para la grabación de demostraciones	68
21.	Diagrama secuencial de la comunicación entre el agente, el controlador del entrenador y el entrenador	72
22.	Resultados del experimento 1	74
23.	Resultados del experimento 2	75
24.	Resultados del experimento 3	76
25.	Resultados del experimento 4	77
26.	Resultados del experimento 5	78
27.	Diagrama de Gantt de la planificación inicial, parte 1 de 2	86
28.	Diagrama de Gantt de la planificación inicial, parte 2 de 2	86
29.	Diagrama de Gantt de la planificación final, parte 1 de 2	87
30.	Diagrama de Gantt de la planificación final, parte 2 de 2	87

Índice de tablas

1.	Taxonomía de simuladores basada en la flexibilidad de la especificación del entorno, con ejemplos para cada categorí	6
2.	Resumen de las tareas con la duración, dependencias y recursos necesarios . . .	14
3.	Costes de personal	16
4.	Estimación de costes económicos por tarea	17
5.	Costes de recursos software	18
6.	Costes de recursos hardware	18
7.	Plan de contingencia con un sobrecoste de 15 % por cada tipo	18
8.	Estimación de costes y riesgos de los imprevistos	19
9.	Coste total del proyecto	19
10.	Resumen del comportamiento de la función objetivo sustituto con límite L^{CLIP} .	42
11.	Configuración común de todos los tipos de entrenamiento (PPO, SAC y POCA)	48
12.	Configuración específica de PPO	48
13.	Configuración de las recompensas extrínsecas	49
14.	Configuración de las recompensas intrínsecas procedentes de Curiosity . . .	49
15.	Configuración de las recompensas intrínsecas procedentes de GAIL	49
16.	Configuración de self-play	50
17.	Función de recompensa del experimento 1	74
18.	Función de recompensa del experimento 2	75
19.	Función de recompensa del experimento 3	76
20.	Función de recompensa del experimento 4	77
21.	Función de recompensa del experimento 5	78
22.	Resumen de la planificación final	81

1. Introducción y contextualización

En la actualidad, con el auge de la inteligencia artificial (IA), surgen sin cesar nuevas herramientas destinadas a resolver problemas cuyo grado de complejidad es cada vez más elevado. En ocasiones, dicha complejidad es tal que no basta con utilizar un conjunto de algoritmos sencillos para obtener el resultado esperado, sino que se opta por emplear otras técnicas que se basan en intentar conseguir el mejor resultado posible, respetando un cierto conjunto de reglas.

El problema principal que se plantea en este proyecto es la simulación del comportamiento humano durante un partido de pádel en un entorno virtual. Precisamente, el pádel es un deporte bastante complejo y diseñar un programa capaz de jugarlo puede llegar a complicarse bastante, teniendo en cuenta la cantidad de mecánicas de las que dispone.

Una solución a este problema es la simulación mediante agentes virtuales, los cuales son, en esencia, programas diseñados para interactuar con personas, otros agentes y/o un entorno específico que, mediante técnicas de aprendizaje automático, se les puede entrenar para atribuirles comportamientos que no son triviales de implementar.

Concretamente, en este trabajo de fin de grado (TFG) se estudian las técnicas de aprendizaje por refuerzo y por imitación, así como su aplicación en el entrenamiento de agentes virtuales en el entorno del pádel.

1.1. Contexto

El proyecto *Entrenamiento de agentes virtuales para el aprendizaje de políticas óptimas de los jugadores durante un partido de pádel* es un TFG de modalidad A (centro) que pertenece a la titulación del Grado en Ingeniería Informática, impartido por la Facultad de Informática de Barcelona (FIB) de la Universidad Politécnica de Cataluña (UPC), dentro de la mención de Computación. Este proyecto está dirigido por Carlos Andújar Gran como director y Mohammadreza Javadiha como codirector, ambos del Departamento de Ciencias de la Computación (CS).

1.2. Conceptos

1.2.1. Aprendizaje por refuerzo

El aprendizaje por refuerzo (RL) es un área del aprendizaje automático que utiliza agentes para tomar decisiones, premiando las acciones deseadas y penalizando las no deseadas. En general, un agente de RL es capaz de percibir e interpretar el entorno en el que se encuentra, tomar acciones basándose en su política y, como resultado, recibir recompensas y transiciones a nuevos estados. El objetivo del aprendizaje por refuerzo es, por lo tanto, aprender unas políticas óptimas que maximicen las recompensas acumuladas a largo plazo.

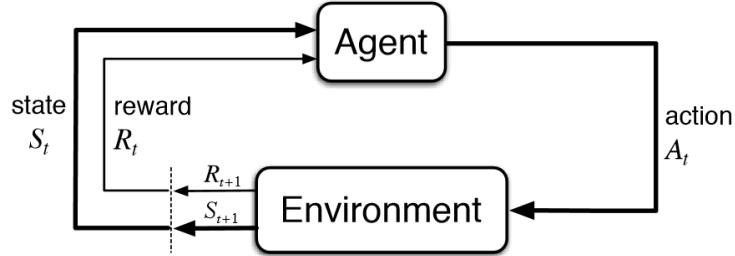


Figura 1: Flujo genérico del aprendizaje por refuerzo. (Fuente: [1])

Formalmente, un modelo básico de RL (Figura 1) consiste en: un conjunto de estados del entorno S ; un conjunto de acciones A ; un conjunto de recompensas R y una función de transición T .

En cada instante t , el agente percibe un estado $s_t \in S$ y recibe un conjunto de posibles acciones para dicho estado $A(s_t)$. Luego, selecciona una acción $a \in A(s_t)$ y recibe del entorno un nuevo estado s_{t+1} y una recompensa r_{t+1} , lo que significa que el agente hace una asignación de estados a probabilidades de seleccionar cada acción. Esta asignación es la política del agente y se denota como π_t , donde $\pi_t(a|s)$ es la probabilidad con la que $a_t = a$ si $s_t = s$, es decir, la probabilidad de seleccionar la acción a dado el estado s en el instante t .

La función de recompensa es la que define el objetivo en un problema de RL. El agente inteligente busca, a la larga, maximizar la recompensa recibida, por lo que debe explorar activamente su entorno y observar los efectos de sus acciones.

1.2.2. Aprendizaje por imitación

Aunque el aprendizaje por refuerzo funcione bien en muchos contextos, es necesario poder asumir que se conoce la función de recompensa, hecho que presenta algunos inconvenientes. Por una parte, determinar una función de recompensa adecuada que permita cumplir el objetivo deseado no resulta fácil. Por otra, las recompensas definidas podrían resultar ser demasiado escasas, dificultando el proceso de aprendizaje al no haber suficiente *feedback* con el que se pueda guiar un agente.

Por ello, surge la necesidad de introducir el enfoque de aprendizaje por imitación (IL) en el RL, donde no se asume que la función de recompensa sea conocida de antemano, sino que se asume que dicha función de recompensa se define implícitamente mediante demostraciones o *feedback* de expertos.

La formulación de un problema de IL es similar a la del RL. La principal diferencia es que, en lugar de definir explícitamente una función de recompensa, se asume que se proporciona un conjunto de demostraciones realizadas por un experto, donde cada demostración ξ consiste en una secuencia de pares de estados y acciones $\xi = \{(s_0, a_0), (s_1, a_1), \dots\}$.

En un problema de IL, en contraste a un problema de RL, el agente inteligente debe buscar la política π que más se asemeja a la política de un experto π^* . La política del experto no se puede observar directamente, sino que se trata de aproximar a través de las demostraciones.

1.3. Identificación del problema

En este trabajo se desarrolla una aplicación que permite simular partidos de pádel en un entorno virtual, por lo que el problema se ha dividido en dos grandes bloques. Por una parte está el diseño y desarrollo de un entorno virtual de pádel y, por otra, la simulación de agentes capaces de jugar a pádel.

1.3.1. Diseño y desarrollo de un entorno virtual de pádel

El entorno virtual de pádel es una pieza clave en este proyecto, ya que determina lo fieles que serán las simulaciones respecto a la realidad. Por esta razón, el diseño de este entorno se debe hacer meticulosamente, respetando lo máximo posible el entorno físico en el que se basa, para poder transmitir cierta sensación de realismo.

El diseño no implica únicamente la estructura de una pista de pádel, sino que también se ha de tener en cuenta todas las leyes físicas que hay detrás, ya sea de los impactos, los rebotes, las trayectorias de la pelota o el movimiento de los jugadores.

El proceso de desarrollo debe llevarse a cabo teniendo en cuenta las normas del pádel, lo cual implica la implementación de toda una serie de estados que tienen lugar a lo largo de un partido, como por ejemplo el inicio y final de un punto, las posiciones de saque o las condiciones para ganar un punto. Además de los estados, también se deben tener presentes todos los parámetros requeridos de cara a la integración de los agentes, ya que estos serán necesarios para su entrenamiento.

1.3.2. Simulación de agentes capaces de jugar a pádel

Una vez finalizado todo el proceso de diseño y desarrollo del entorno virtual, se integran los agentes para poder simular partidos de pádel. Estos agentes deben seguir un conjunto de reglas a la hora de decidir qué hacer durante el tránscurso de un partido: hacia dónde moverse, qué tipo de golpeo hacer y hacia qué dirección dirigir la pelota.

Atribuir un comportamiento así únicamente mediante programación imperativa, habiendo una infinidad de posibles estados, puede llegar a ser una tarea muy laboriosa, debido al grado de complejidad presente en este deporte. Es por este motivo que se explora el uso de distintas técnicas de aprendizaje por refuerzo y por imitación para el entrenamiento de agentes virtuales, para tratar de simplificar la resolución de este problema.

1.4. Agentes implicados

El producto desarrollado va dirigido a una gran variedad de usuarios dependiendo del uso que se le quiera dar, por ejemplo:

- A jugadores, entrenadores o analistas del pádel, ya que puede ser útil a la hora de estudiar los movimientos de los agentes, partiendo del análisis del entorno junto a la decisión que acaba tomando un agente, permitiendo un estudio de jugadas que quizás no se han llegado a contemplar anteriormente. En especial, el modelo entrenado con aprendizaje por imitación, que busca simular el comportamiento de jugadores profesionales, puede servir para analizar la probabilidad de que un jugador tome ciertas decisiones dada una situación en concreto.
- A desarrolladores de videojuegos y videojugadores, pues es un producto que se podría integrar perfectamente en un videojuego relacionado con el pádel. Sería interesante especialmente en un videojuego de realidad virtual donde, aprovechándose de la captura de movimientos, el videojugador pueda competir contra agentes que han aprendido a seguir unas políticas óptimas, o bien han aprendido las políticas que siguen ciertos jugadores profesionales del pádel. De esta manera, el videojugador podría entrenar sus habilidades en el pádel mediante simulaciones de partidos bastante fieles a la realidad.
- A investigadores interesados en el aprendizaje por refuerzo y/o aprendizaje por imitación, dado que puede servir como un punto de partida para una futura investigación con unos fines parecidos a los de este proyecto.

En cuanto a los beneficiarios de los resultados del producto, se puede apreciar que todos los usuarios mencionados se pueden favorecer de una manera u otra. A nivel personal también ha sido beneficioso, ya que todo el proceso de diseño, desarrollo y gestión ha servido para profundizar los conocimientos en este campo de estudio.

Además de los usuarios finales, otros agentes implicados es el equipo docente que hay detrás de este proyecto, el cual se ha encargado de orientar y supervisar las tareas realizadas durante el trámite de este TFG.

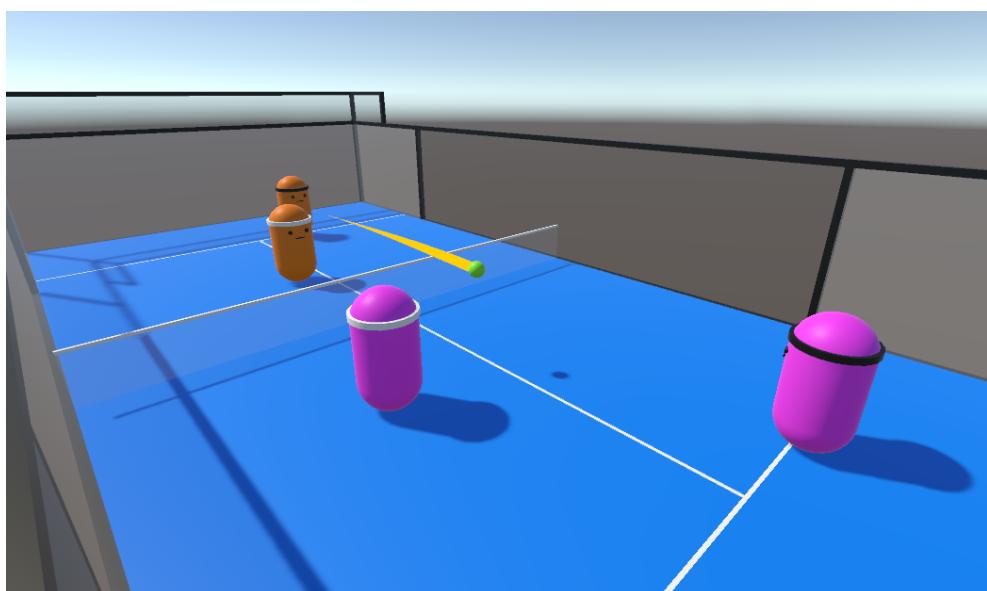


Figura 2: Entorno virtual de pádel. (Elaboración propia)

2. Justificación

Dentro del conjunto de técnicas de aprendizaje automático, la realidad es que la popularidad de las áreas de RL y IL no es tan elevada como otras como el aprendizaje profundo. De la misma manera, el pádel no es un deporte tan popular como otros gigantes como el fútbol, el baloncesto o el tenis. Por estas razones, resulta difícil encontrar recursos que combinen directamente ambas cosas. Aun así, existen soluciones y herramientas que se pueden adaptar al dominio de este proyecto, de las cuales se hablan a continuación.

2.1. Estudio de soluciones existentes

2.1.1. Motor gráfico

Para el diseño y desarrollo del entorno virtual de pádel se requiere un motor gráfico, un conjunto de herramientas para desarrollar aplicaciones visuales, es decir, renderizar escenas 2D o 3D. Además, es necesario que este motor sea compatible con otras funcionalidades, como por ejemplo un motor físico o un soporte para la integración de agentes virtuales.

Los motores gráficos más populares en la actualidad son *Unity* y *Unreal Engine*, los cuales ofrecen prestaciones similares y se diferencian principalmente en el lenguaje de programación utilizado para programar *scripts*: C# en el caso de *Unity* y C++ en el caso de *Unreal Engine*.

Entre estos dos motores gráficos, el motor por el que se ha optado finalmente es el de *Unity*, ya que como se verá más adelante, ofrece la herramienta *Unity Machine Learning Agents Toolkit* (ML-Agents) [2], un proyecto de código abierto que permite utilizar juegos y simulaciones como entornos para entrenar agentes inteligentes, lo cual cumple con lo requerido para este trabajo.

2.1.2. Entorno de entrenamiento

Según un estudio realizado por el equipo detrás de ML-Agents [3], en el panorama de simuladores, entornos y plataformas se puede diferenciar cuatro categorías:

- La categoría *Single Environment* describe entornos fijos que actúan como cajas negras desde la perspectiva de un agente.
- La categoría *Environment Suite* consiste en conjuntos de entornos agrupados y que se utilizan normalmente para hacer pruebas de rendimiento de un algoritmo o método en algunas dimensiones de interés.
- La categoría *Domain-specific Platform* define plataformas que permiten la creación de tareas a realizar dentro de un dominio en específico.
- Por último, la categoría *General Platform* consiste en una plataforma que permite a los usuarios crear entornos con tareas y interacciones visuales, físicas y sociales arbitrariamente complejos. En principio, dichas plataformas permiten definir cualquier entorno de investigación de inteligencia artificial.

Single Env	Env Suite	Domain-specific Platform	General Platform
Cart Pole	ALE	MuJoCo	Unity & ML-Agents
Mountain Car	DMLab-30	DeepMind Lab	
Obstacle Tower	Hard Eight	Project Malmo	
Pitfall!	AI2Thor	VizDoom	
CoinRun	OpenAI Retro	GVGAI	
Ant	DMControl	PyBullet	
	ProcGen		

Tabla 1: Taxonomía de simuladores basada en la flexibilidad de la especificación del entorno, con ejemplos para cada categoría. (Fuente: [3])

Entre estas cuatro categorías, la categoría de *General Platform* es la que mejor se adapta a las necesidades de este proyecto, ya que es necesario un alto grado de flexibilidad a la hora de diseñar el entorno de entrenamiento para agentes, pues se busca hacer una representación fiel del pádel. Por ello, se utilizará ML-Agents para el entrenamiento de los agentes virtuales.

3. Alcance

Definir el alcance de un proyecto es importante para establecer los límites de este desde el principio, para así evitar posibles contratiempos y poder cumplir con los plazos de las entregas. A continuación se definen los objetivos, los requerimientos funcionales y no funcionales, y los posibles obstáculos y riesgos del proyecto.

3.1. Objetivos

El objetivo principal de este trabajo es el diseño, desarrollo y evaluación de una aplicación que permita simular partidos de pádel en un entorno virtual. Se busca en especial una implementación mediante técnicas de aprendizaje automático, en concreto de aprendizaje por refuerzo y por imitación. Este objetivo se divide en los siguientes subobjetivos:

1. Diseñar e implementar un entorno virtual que sea lo más fiel posible a un entorno real de pádel, donde se harán las simulaciones de partidos de pádel mediante agentes virtuales.
2. Investigar y aplicar distintas técnicas de aprendizaje por refuerzo para entrenar agentes virtuales, los cuales acabarán siendo capaces de jugar a pádel tras aprender las políticas que mejor les recompensan.
3. Investigar y aplicar distintas técnicas de aprendizaje por imitación para entrenar agentes virtuales, los cuales acabarán aprendiendo a jugar a pádel basándose en datos de jugadores profesionales, obtenidos mediante la aplicación de técnicas de visión por computador para el seguimiento de los jugadores en vídeos de partidos.
4. Evaluar qué técnicas dan los mejores resultados, teniendo especial consideración en cómo se comparan con el comportamiento de jugadores humanos. Este subobjetivo se irá aplicando de manera iterativa, ya que será necesario hacer una evaluación para cada técnica utilizada.

3.2. Requerimientos

3.2.1. Requerimientos funcionales

Además de los requerimientos funcionales ya definidos en el objetivo, existen otros requisitos necesarios en el funcionamiento de la aplicación:

- Durante una simulación, el usuario debe poder hacer cambios entre las distintas políticas aprendidas por los agentes, e indicar en todo momento qué políticas están siendo usadas (ya sean aquellas aprendidas mediante RL o mediante IL).
- El usuario debe poder interrumpir, reiniciar, pausar o continuar una simulación.

3.2.2. Requerimientos no funcionales

Los requerimientos no funcionales, es decir, aquellos requisitos que no tienen una relación directa con el funcionamiento de la aplicación, pero que se han de tener en cuenta durante su desarrollo, son los siguientes:

- Usabilidad: La aplicación debe ser fácil de usar.
- Reusabilidad: la aplicación debe ser fácilmente integrable en otros sistemas, como por ejemplo en un videojuego.

3.3. Obstáculos y riesgos

Anticipar las posibles dificultades que pueden surgir en un proyecto puede ayudar a minimizar el impacto que presentan. Los principales obstáculos y riesgos son los siguientes:

- Falta de conocimiento: este trabajo parte de un conocimiento muy básico en el campo del aprendizaje por refuerzo y por imitación, por lo que será necesario invertir más tiempo a la investigación.
- Recursos insuficientes: al pretender desarrollar la aplicación en un ordenador personal, es posible que este no tenga la suficiente potencia para llevar a cabo el entrenamiento de agentes. Es importante disponer de alternativas para casos así, como por ejemplo tener acceso a un ordenador externo.
- Entrenamiento ineficiente: tener un ordenador con suficiente potencia no implica un entrenamiento más veloz, ya que se relaciona también con la complejidad del modelo implementado. Un modelo demasiado complejo aumenta considerablemente el tiempo necesario para obtener unos resultados mínimamente buenos.
- Costes en la obtención o insuficiencia de datos: el aprendizaje por imitación requiere conjuntos de datos de un tamaño considerable, por lo que es posible que los datos de los que se dispone en un principio sean insuficientes. La obtención de estos datos tampoco resulta trivial, por lo que es una tarea bastante costosa.
- Diseño de un entorno virtual demasiado específico: teniendo en cuenta que el entrenamiento de agentes mediante técnicas de aprendizaje por refuerzo y por imitación se quiere realizar en un mismo entorno virtual, podría ocurrir que, en una primera iteración, este acabe siendo demasiado específico para uno de los dos métodos, lo cual supondría un retraso debido a la necesidad de rediseñar dicho entorno.

4. Metodología

La naturaleza de este proyecto implica hacer cambios constantemente, ya sea por los fallos producidos durante el desarrollo, por la necesidad de estar continuamente probando distintos algoritmos y evaluar si son adecuados, o simplemente por querer hacer mejoras en el diseño del producto en general. Es por ello que se exige una metodología flexible, por lo que la metodología ágil ha sido la más adecuada.

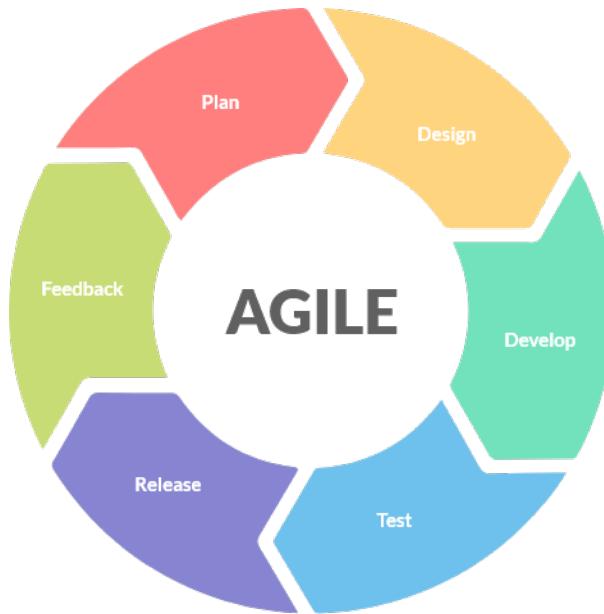


Figura 3: Ciclo de desarrollo ágil (Fuente: [4])

Siguiendo la metodología ágil, la idea ha sido que durante el trascurso del TFG se fuera definiendo periódicamente (cada una o dos semanas) un pequeño ciclo de trabajo, consistente en diseñar, implementar y probar una funcionalidad.

Para adaptarse a esta metodología, se han ido haciendo reuniones periódicas con el director, con la finalidad de verificar que el proyecto estuviera progresando correctamente, recibiendo *feedback* e indicaciones, y que cumpliera con las competencias técnicas que se exigen.

4.1. Herramientas

Para hacer el seguimiento de todas las tareas que han surgido a lo largo del desarrollo de este proyecto, así como permitir una clara visualización de su estado, se ha utilizado la aplicación *Trello*, la cual permite la creación de tareas en forma de bloques cuyos estados se pueden distinguir entre pendientes, en proceso o finalizados.

Además, con la finalidad de mantener un control de versiones, guardar copias de seguridad y compartir el código con el director del proyecto, se ha utilizado la plataforma *Github* para guardar el proyecto en un repositorio.

5. Planificación temporal

El TFG consta de una carga lectiva de 18 créditos ECTS y, según la normativa académica de la FIB, cada crédito equivale a 30 horas de trabajo, por lo que se espera una dedicación de un total de 540 horas. Este proyecto empezó el 18 de septiembre de 2023 y se espera su finalización para el día 19 de enero de 2024, un total de 120 días en los que se espera una dedicación mínima de 4.5 horas por día.

Con la finalidad de mantener una buena organización durante el trámite de este proyecto, y terminar este proyecto en la fecha prevista, se propuso una planificación temporal donde se describen las distintas tareas en las que se descompone.

5.1. Descripción de las tareas

Las tareas a realizar se han agrupado en bloques para distinguir a qué fase pertenecen. Se ha elaborado un resumen de tareas en la Tabla 2, además de un diagrama de Gantt que se puede consultar en el Apéndice A.1.

5.1.1. Gestión del proyecto [T1]

Este bloque agrupa todas aquellas tareas relacionadas con la gestión del proyecto, engloba la definición del alcance del proyecto, su planificación, estimación de un presupuesto, informe de sostenibilidad, además de las reuniones semanales. En total, se ha estimado una duración de 100 horas.

- Alcance [T1.1]: consiste en definir cuál es el objetivo de este proyecto, qué se va a desarrollar, a quién va dirigido y qué medios serán necesarios. Se estima una duración de 30 horas.
- Planificación [T1.2]: se propone una planificación inicial del proyecto, los recursos necesarios y una gestión del riesgo. Se estima una duración de 15 horas.
- Presupuesto [T1.3]: se realiza un presupuesto para calcular el coste necesario para llevar a cabo este proyecto. Se estima una duración de 7.5 horas.
- Informe de sostenibilidad [T1.4]: se analiza el impacto medioambiental, económico y social del proyecto. Se estima una duración de 7.5 horas.
- Reuniones [T1.5]: consiste en reuniones que se harán cada una o dos semanas con el director del proyecto, de aproximadamente 1 hora por sesión. Se estima una duración de 25 horas.
- Documentación [T1.6]: la documentación es una tarea que en realidad está implícita en todos los bloques, no solo del bloque de gestión del proyecto, ya que es un proceso que se hace en paralelo al resto. Por lo tanto, el tiempo estimado que se dedicará a ello queda presente en cada tarea.
- Presentación [T1.7]: se preparará una presentación del proyecto. Incluye el material de soporte para la presentación, el guion y ensayos. Se estima una duración de 15 horas.

5.1.2. Trabajo previo [T2]

El bloque de trabajo previo define la preparación requerida antes de empezar con el desarrollo del proyecto, y se estima una duración total de 40 horas.

- Estudio del estado del arte [T2.1]: se debe hacer un estudio de los distintos algoritmos de aprendizaje por refuerzo profundo (DRL) y aprendizaje por imitación (RI), en particular de aquellos que se puedan implementar con las herramientas que ofrece ML-Agents [2]. Se hará una comparación entre estos algoritmos para analizar sus ventajas, desventajas y se analizará la viabilidad que tienen para este proyecto. Se estima una duración de 30 horas.
- Familiarización con ML-Agents [T2.2]: se hará un primer contacto con ML-Agents, que consistirá en la instalación y configuración del paquete en Unity, y probar algunos de los ejemplos disponibles. Se estima una duración de 10 horas.

5.1.3. Desarrollo del entorno virtual en Unity [T3]

La fase de desarrollo del entorno virtual en Unity consiste en la preparación del *Game Scene*, donde se llevarán a cabo las simulaciones de partidos de pádel. Dicha preparación se deberá hacer teniendo en cuenta los requerimientos de los entornos de entrenamiento mediante DRL y IL. Se estima una duración total de 180 horas.

- Diseño [T3.1]: se planteará qué *Game Objects* son necesarios, y cómo será la interacción entre estos. Se estima una duración de 40h.
- Preparación del entorno de entrenamiento DRL [T3.2]: se listarán los parámetros y requisitos necesarios para el entrenamiento mediante DRL. Se estima una duración de 30 horas.
- Preparación del entorno de entrenamiento IL [T3.3]: se listarán los parámetros y requisitos necesarios para el entrenamiento mediante IL. Se estima una duración de 30 horas.
- Implementación [T3.4]: se codificará el entorno virtual teniendo en cuenta el diseño y los requisitos ideados. Se estima una duración de 60 horas, ya que lo más probable es que acabe siendo un proceso iterativo donde se acabarán haciendo cambios.
- Testing [T3.5]: se comprobará que la implementación sea correcta y cumpla con las expectativas deseadas. Se estima una duración de 20 horas.

5.1.4. Desarrollo de técnicas de aprendizaje por refuerzo profundo [T4]

En este bloque se llevará a cabo el entrenamiento de agentes mediante DRL, así que dependerá del bloque T3. Se estima una duración total de 100 horas.

- Diseño del modelo [T4.1]: se ideará un modelo de DRL para el entrenamiento de agentes. Se estima una duración de 40 horas.
- Implementación [T4.2]: se codificará el modelo diseñado. Se estima una duración de 40 horas.
- Testing [T4.3]: probar que el comportamiento de los agentes sea correcto y tenga un funcionamiento esperado. Se estima una duración de 20 horas.

5.1.5. Desarrollo de técnicas de aprendizaje por imitación [T5]

En este bloque se llevará a cabo el entrenamiento de agentes mediante IL, por lo que también dependerá del bloque T3. Se estima una duración total de 100 horas.

- Procesamiento de datos [T5.1]: se analizarán los datos de los que se dispone y se adaptarán a lo requerido en el entrenamiento de IL de ML-Agents. Se estima una duración de 20 horas.
- Diseño del modelo [T5.2]: se ideará un modelo de IL para el entrenamiento de agentes. Se estima una duración de 20 horas.
- Implementación [T5.3]: se codificará el modelo diseñado. Se estima una duración de 40 horas.
- Testing [T5.4]: probar que el comportamiento de los agentes sea correcto y tenga un funcionamiento esperado. Se estima una duración de 20 horas.

5.1.6. Desarrollo de la interfaz gráfica de usuario [T6]

Una vez finalice el desarrollo del bloque principal del proyecto, compuesto por los bloques T3, T4 y T5, se integrará a la aplicación una interfaz gráfica de usuario. Este bloque se compone por el diseño, la implementación y el testing de la interfaz gráfica, y se estima una duración total de 20 horas.

- Diseño [T6.1]: se ideará una interfaz estética, intuitiva y fácil de usar para el usuario final. Incluye el diseño de iconos y funcionalidades. Se estima una duración de 10 horas.
- Implementación [T6.2]: después del diseño, se implementarán los iconos y las funcionalidades en la aplicación. Se estima una duración de 5 horas.
- Testing [T6.3]: se comprobará que tras la integración toda la aplicación funciona correctamente. Se estima una duración de 5 horas.

5.2. Recursos

5.2.1. Recursos humanos

Los roles presentes en este proyecto son los de director de proyecto, analista de datos, programador, diseñador de la experiencia de usuario e ingeniero de control de calidad. A excepción del rol de analista de datos, en el que también participa el codirector Mohammadreza Javadiha, proporcionando datos de jugadores de pádel, todos los roles son asumidos por el autor del proyecto.

- Jefe del proyecto: se encarga de la planificación y el desarrollo del proyecto, establecer una coordinación entre los miembros del equipo y escribir la documentación.
- Analista de datos: se encarga de recopilar, limpiar e interpretar conjuntos de datos, con la finalidad de extraer información útil para el desarrollo del proyecto.
- Programador: se encarga de la implementación de la aplicación.
- *UX/UI Designer*: se encarga de asegurar que la interacción entre el usuario y el producto final sea sencilla e intuitiva.
- *QA Engineer*: se encarga de asegurar que el producto final funcione correctamente y cumpla con las expectativas deseadas.

5.2.2. Recursos materiales

Los recursos materiales que se necesitan para este proyecto son:

- Ordenador de sobremesa de altas prestaciones: equipo principal donde se llevarán a cabo todo o gran parte del desarrollo del proyecto.
- (Opcional) Ordenador portátil: equipo que se puede utilizar para realizar aquellas tareas relacionadas con la gestión del proyecto. De ser posible, también puede ser útil para agilizar el proceso de entrenamiento de agentes.
- Unity: motor gráfico que se utilizará como entorno de desarrollo para el proyecto.
- Gantter: herramienta web para la creación de diagramas de Gantt.
- Trello: aplicación que permite organizar tareas.
- Overleaf: editor de LaTeX colaborativo basado en la nube utilizado para la documentación.
- GitHub: plataforma utilizado para el control de versiones.
- Google Meet: aplicación utilizada para hacer reuniones en remoto.

ID	Tarea	Tiempo	Dependencias	Recursos	Rol
T1	Gestión del proyecto	100h	-	-	-
T1.1	Alcance	30h	[]	PC, Overleaf	JP
T1.2	Planificación	15h	[T1.1]	PC, Overleaf, Gantter	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, Trello	JP
T1.6	Documentación	-	[]	PC, Overleaf	JP
T1.7	Presentación	15h	[T1.6]	PC, Overleaf	JP
T2	Trabajo previo	40h	-	-	-
T2.1	Estudio del estado del arte	30h	[]	PC	JP
T2.2	Familiarización con ML-Agents	10h	[T2.1]	PC	JP
T3	Desarrollo del entorno virtual en Unity	180h	-	-	-
T3.1	Diseño	40h	[]	PC, Unity	JP
T3.2	Preparación del entorno de entrenamiento DRL	30h	[T3.1]	PC, Unity	P
T3.3	Preparación del entorno de entrenamiento IL	30h	[T3.1]	PC, Unity	P
T3.4	Implementación	60h	[T3.2, T3.3]	PC, Unity	P, QA
T3.5	Testing	20h	[T3.4]	PC, Unity	QA
T4	Desarrollo de técnicas de DRL	100h	-	-	-
T4.1	Diseño del modelo	40h	[T3]	PC, Unity	JP
T4.2	Implementación	40h	[T4.1]	PC, Unity	P, QA
T4.3	Testing	20h	[T4.2]	PC, Unity	QA
T5	Desarrollo de técnicas de IL	100h	-	-	-
T5.1	Procesamiento de datos	20h	[T3]	PC	AD
T5.2	Diseño del modelo	20h	[T5.1]	PC, Unity	JP
T5.3	Implementación	40h	[T5.2]	PC, Unity	P, QA
T5.4	Testing	20h	[T5.3]	PC, Unity	QA
T6	Desarrollo de la interfaz gráfica de usuario	20h	-	-	-
T6.1	Diseño	10h	[T4, T5]	PC, Unity	UX/UI
T6.2	Implementación	5h	[T6.1]	PC, Unity	P, QA
T6.3	Testing	5h	[T6.2]	PC, Unity	QA
-	Total	540h	-	-	-

Tabla 2: Resumen de las tareas con la duración, dependencias y recursos necesarios. Roles: JP - Jefe del proyecto, P - Programador, QA - QA Engineer, UX/UI - UX/UI Designer, AD - Analista de datos. (Elaboración propia)

5.3. Gestión del riesgo

Además de la planificación, también es necesario hacer una previsión de los posibles riesgos que puedan surgir y proponer posibles alternativas para reducir el impacto de estos. Los riesgos más relevantes son los siguientes:

- Recursos materiales insuficientes: en caso de no disponer de un ordenador de sobremesa con suficientes prestaciones, se puede optar por utilizar un equipo disponible en el laboratorio del grupo de investigación ViRVIG, ubicado en la Facultad de Matemáticas y Estadística de la UPC. Como último recurso, también existe la posibilidad de llevar a cabo, en particular, el entrenamiento de agentes virtuales en un equipo ofrecido por el director Carlos Andújar. Este riesgo, aun teniendo un impacto mínimo, supone una adición en el coste temporal debido al tiempo requerido para el desplazamiento de ida y vuelta, el cual es de un total de 2 horas por día.
- Costes en la obtención o insuficiencia de datos: si la obtención de datos de jugadores profesionales de pádel, destinados al aprendizaje por imitación, acaba siendo inviable, se podría optar añadir datos generados desde el propio entorno. En este caso, sería el propio autor el que, controlando un agente mediante unos controles básicos, grabaría datos representando un comportamiento inteligente destinados al entrenamiento de agentes. Esto supondría invertir más horas en las tareas T3.3 y T5.1, en particular 20 horas para la implementación de controles suficientemente complejos y 10 horas para la generación de datos manual, por lo que se puede considerar como un riesgo de impacto medio.
- Diseño de un entorno virtual demasiado específico: podría ocurrir que el entorno virtual de Unity diseñado, en una primera iteración, acabe siendo demasiado específico para el aprendizaje profundo por refuerzo, resultando en un diseño incompatible para el aprendizaje por imitación. La mejor manera de evitar este riesgo sería teniéndolo en cuenta desde el principio y adaptar un diseño más genérico que admita la implementación de ambos métodos. En el peor de los casos, haría falta un rediseño del entorno virtual, afectando a toda la aplicación debido a las dependencias con el bloque de tareas T3, y se podría producir un retraso de hasta 100 horas, por lo que se considerará como un riesgo de impacto alto.
- Entrenamiento de agentes demasiado ineficiente: a la hora de entrenar agentes, puede pasar que el modelo implementado sea demasiado complejo y, como consecuencia, requiera de demasiado tiempo de entrenamiento para obtener unos resultados mínimamente buenos. Para evitarlo, se debe analizar continuamente la evolución del proceso de entrenamiento, para evaluar si vale la pena seguir con el entrenamiento o sería mejor optar por hacer un ajuste de hiperparámetros, o directamente un cambio de diseño y/o modelo. A pesar de que estos cambios suelen ser pequeños, una ocurrencia reiterada podría suponer un aumento de tiempo considerable a largo plazo, el cual podría sumar un total de hasta 50 horas. Por este motivo, se considerará como un riesgo de impacto medio.

6. Gestión económica

En esta sección, tras haber propuesto una planificación inicial, se estima el coste asociado al desarrollo de este proyecto con la finalidad de analizar su viabilidad económica. En particular, se identifican los costes asociados al personal, a los espacios de trabajo y a los recursos materiales que se utilizarán. Además, se propone un plan de contingencia, una partida de imprevistos y los mecanismos de control de gestión para poder hacer frente a las desviaciones que puedan surgir durante el desarrollo.

6.1. Presupuesto

6.1.1. Costes de personal

Los costes de personal consisten en el salario de los roles definidos previamente en la sección de planificación: jefe de proyecto, analista de datos, programador, *UX/UI Engineer* y *QA Engineer*. En la Tabla 3 se muestra cada rol con su respectivo salario bruto por hora, basado en la guía de mercado laboral de Indeed [5]. También se estima el coste de la seguridad social, el cual se ha obtenido multiplicando el salario bruto por 1.3.

Rol	Salario bruto por hora	SS	Salario + SS
Jefe de proyecto	21.05 €	6.32 €	27.37 €
Analista de datos	18.98 €	5.69 €	24.67 €
Programador	14.36 €	4.31 €	18.67 €
UX/UI Engineer	14.39 €	4.32 €	18.71 €
QA Engineer	18.09 €	5.43 €	23.52 €

Tabla 3: Costes de personal. (Elaboración propia)

En la Tabla 4 se detalla el coste de cada tarea de acuerdo a los salarios definidos. Se estima un coste total de 16,015.86 € en el personal.

ID	Tarea	Tiempo	Rol	Coste
T1	Gestión del proyecto	100h	-	2,737.01 €
T1.1	Alcance	30h	JP	821.10 €
T1.2	Planificación	15h	JP	410.55 €
T1.3	Presupuesto	7.5h	JP	205.28 €
T1.4	Informe de sostenibilidad	7.5h	JP	205.28 €
T1.5	Reuniones	25h	JP	684.25 €
T1.7	Documentación	-	JP	-
T1.8	Presentación	15h	JP	410.55 €
T2	Trabajo previo	40h	-	1,094.80 €
T2.1	Estudio del estado del arte	30h	JP	821.10 €
T2.2	Familiarización con ML-Agents	10h	JP	273.70 €
T3	Desarrollo del entorno virtual en Unity	180h	-	5,216.80 €
T3.1	Diseño	40h	JP	1,094.80 €
T3.2	Preparación del entorno de entrenamiento DRL	30h	P	560.10 €
T3.3	Preparación del entorno de entrenamiento IL	30h	P	560.10 €
T3.4	Implementación	60h	P, QA	2,531.40 €
T3.5	Testing	20h	QA	470.40 €
T4	Desarrollo de técnicas de DRL	100h	-	3,252.80 €
T4.1	Diseño del modelo	40h	JP	1,094.80 €
T4.2	Implementación	40h	P, QA	1,687.60 €
T4.3	Testing	20h	QA	470.40 €
T5	Desarrollo de técnicas de IL	100h	-	3,198.80 €
T5.1	Procesamiento de datos	20h	AD	493.40 €
T5.2	Diseño del modelo	20h	JP	547.40 €
T5.3	Implementación	40h	P, QA	1,687.60 €
T5.4	Testing	20h	QA	470.40 €
T6	Desarrollo de la interfaz gráfica de usuario	20h	-	515.65 €
T6.1	Diseño	10h	UX/UI	187.10 €
T6.2	Implementación	5h	P, QA	210.95 €
T6.3	Testing	5h	QA	117.60 €
-	Total	540h	-	16,015.86 €

Tabla 4: Estimación de costes económicos por tarea. Roles: JP - Jefe del proyecto, P - Programador, QA - QA Engineer, UX/UI - UX/UI Designer, AD - Analista de datos. (Elaboración propia)

6.1.2. Costes genéricos

Tal y como se ha especificado previamente, el espacio de trabajo será principalmente en el hogar durante toda la semana, y puntualmente en una biblioteca pública. Al ser espacios compartidos y situados en Barcelona, se estima un coste de 300 € mensuales de acuerdo a la tarifa de un espacio de *coworking* en Barcelona, lo que sumaría un total de 1200 € para 4 meses.

En cuanto a los costes de recursos *software*, se ha optado por utilizar las versiones gratuitas siempre que sea posible. Por lo tanto, tal y como se puede observar en la Tabla 5, el único programa con costes será Gantter, el cual tendrá un coste total de 20 €.

Software	Coste mensual	Meses	Coste total
Unity	0.00 €	4	0.00 €
Gantter	5.00 €	4	20.00 €
Trello	0.00 €	4	0.00 €
Overleaf	0.00 €	4	0.00 €
GitHub	0.00 €	4	0.00 €
Google Meet	0.00 €	4	0.00 €
Total	-	-	20.00 €

Tabla 5: Costes de recursos software. (Elaboración propia)

Finalmente, en la Tabla 6 se estima el cálculo de costes de recursos *hardware*, donde se hace un cálculo de la amortización mediante la siguiente fórmula:

$$A = \frac{P * Dp}{V * Dl * H} \quad (1)$$

donde A es la amortización, P el precio del producto, Dp la duración del proyecto, V la vida útil, Dl el número de días laborales en un año (se considera que hay 220 días laborales en un año) y H el número de horas de trabajo al día.

Hardware	Precio	Vida útil	Duración del proyecto	Amortización
Ordenador de sobremesa	1,500.00 €	4 años	540 horas	115.05 €
Ordenador portátil	2,300.00 €	4 años	540 horas	176.42 €
Total	-	-	-	291.47 €

Tabla 6: Costes de recursos hardware. (Elaboración propia)

6.1.3. Contingencia

Debido a la posibilidad de que surjan complicaciones no esperadas durante el desarrollo del proyecto, lo cual puede encarecer el coste del trabajo, es importante añadir un sobrecoste que permita hacerles frente. Los valores típicos del nivel de contingencia en un proyecto de desarrollo de software se suelen situar entre el 10 % y 20 %, por lo que se ha decidido fijar un 15 % de sobrecoste. El plan de contingencia de este proyecto se detalla en la Tabla 7.

Tipo	Coste	Contingencia
Personal	16,015.86 €	2,402.38 €
Espacio	1,200.00 €	180.00 €
Software	20.00 €	3.00 €
Hardware	291.47 €	43.72 €
Total	-	2,629.10 €

Tabla 7: Plan de contingencia con un sobrecoste de 15 % por cada tipo. (Elaboración propia)

6.1.4. Imprevistos

Por último, también se calcula el coste necesario para afrontar los obstáculos e imprevistos que puedan surgir a lo largo de este proyecto. En la Tabla 8 se cuantifica el riesgo y el coste que pueden suponer, y se explican a continuación:

- Aumento del tiempo de desarrollo: este imprevisto recoge la mayoría de obstáculos que se han descrito previamente. En caso de necesitar más tiempo de desarrollo, se añadirá a la planificación inicial 20 horas de implementación y 10 horas de testing, lo cual supone un aumento de 1079 € en el coste total de personal. El riesgo de este imprevisto se ha cuantificado con un 30 %, el cual es bastante elevado ya que durante el proyecto se trabajará con tecnologías que no se han utilizado previamente.
- Fallo en un dispositivo hardware: el riesgo de que se produzca algún fallo en los dispositivos hardware es bastante bajo, ya que se trata de dispositivos relativamente nuevos, por lo que se ha cuantificado un riesgo del 10 %. En el caso de una avería en el ordenador de sobremesa, se buscaría reemplazar el componente defectuoso. Una avería en el ordenador portátil no sería muy grave, ya que su uso es opcional.

Imprevisto	Coste	Riesgo	Coste total
Aumento del tiempo de desarrollo	1,079.00 €	30 %	323.70 €
Fallo de ordenador de sobremesa	1,500.00 €	10 %	150.00 €
Fallo de ordenador portátil	2,300.00 €	10 %	230.00 €
Total	-	-	703.70 €

Tabla 8: Estimación de costes y riesgos de los imprevistos. (Elaboración propia)

6.1.5. Coste total

Una vez calculados los distintos costes del proyecto, en la Tabla 9 se resume el coste total, el cual es de 20,860.13 €.

Tipo	Coste
Personal	16,015.86 €
Espacio	1,200.00 €
Software	20.00 €
Hardware	291.47 €
Contingencia	2,629.10 €
Imprevistos	703.70 €
Coste total	20,860.13 €

Tabla 9: Coste total del proyecto. (Elaboración propia)

6.2. Control de gestión

Tras haber definido el presupuesto inicial, a continuación se definen los mecanismos para controlar las desviaciones que puedan surgir en relación al presupuesto, además de indicadores numéricos de cálculo que facilitan este control.

Cada vez que una tarea finalice, se actualizará el presupuesto con las horas reales invertidas y se comparará con las horas estimadas. De cara a los imprevistos, también se tendrá en cuenta el gasto extra que haya sido necesario, y se contrastará con la previsión de imprevistos y la contingencia.

Las métricas que se utilizarán para controlar las desviaciones son las siguientes:

1. Desviación del coste = (coste estimado hora - coste real hora) × horas reales
2. Desviación del consumo = (horas estimadas - horas reales) × coste estimado hora
3. Desviación total de horas = horas estimadas - horas reales

7. Sostenibilidad

7.1. Autoavaluación

Tras realizar la encuesta sobre la sostenibilidad, me he dado cuenta de que poseo unas nociones muy básicas sobre esta, especialmente cuando se trata de mi ámbito profesional. A pesar de tener una mínima base teórica, a lo largo de mi carrera universitaria apenas he tenido la oportunidad de aplicarla en un proyecto cuya magnitud sea crítica.

Si bien es cierto que en ocasiones he pensado en el impacto social que tienen o podrían tener ciertos productos relacionados con la informática, es distinto cuando se trata de la dimensión económica y ambiental, las cuales no se han tenido tanto en cuenta. Esto se debe en gran parte a que los proyectos en los que he participado no van más allá de ser aplicaciones sencillas, las cuales requieren poca inversión en recursos y no trascienden del ámbito académico, por lo que el impacto económico y ambiental es mínimo.

La realización de un trabajo de fin de grado es una buena oportunidad para cambiar esta situación, ya que permite poner estas nociones en práctica durante la realización del proyecto, incitando a abordar una solución sostenible tanto en la dimensión económica como la ambiental y social.

7.2. Dimensión económica

Sobre el coste estimado para la realización del proyecto, teniendo en cuenta la finalidad que tiene en los ámbitos de entretenimiento e investigación, además de ser una manera bastante innovadora de abordar el problema planteado, pienso que es un coste justificado en relación a los beneficios que aporta.

Generalmente, desarrollar un simulador de pádel en el que los jugadores puedan adquirir comportamientos complejos suele ser una tarea bastante laboriosa, sobre todo cuando no se emplean las técnicas adecuadas, lo que supone un aumento en el coste de personal. La solución planteada en este proyecto, por lo contrario, busca reducir este coste simplificando el proceso de desarrollo.

Si bien es cierto que el entrenamiento de un modelo resulta costoso, una vez finalizado e integrado en el producto final, transferir el modelo y utilizarlo resulta ser un proceso muy eficiente. Una contraparte es, sin embargo, su coste de mantenimiento. Esto se debe específicamente a que se trata de una aplicación que debe mantenerse al tanto de la normativa del pádel, aunque sea poco probable que esta cambie. Además, hay que tener en cuenta que las estrategias de los jugadores de pádel también van evolucionando a lo largo del tiempo, por lo que podría surgir la necesidad de reentrenar agentes con datos actualizados.

7.3. Dimensión ambiental

El desarrollo de este proyecto se ha llevado a cabo principalmente con un ordenador de sobremesa, y se ha utilizado ocasionalmente un portátil cuando ha sido conveniente. Por lo tanto, el impacto ambiental se ha reducido a la fabricación del equipo de trabajo y su consumo.

Respecto al espacio de trabajo, se ha trabajado principalmente en un hogar dotado de paneles solares, por lo que parte del consumo eléctrico ha provenido de energías renovables. Puntualmente, se ha optado por trabajar en bibliotecas públicas a causa de algunos inconvenientes en el hogar.

Para los desplazamientos necesarios para las reuniones, las cuales han sido mayoritariamente remotas, se ha utilizado exclusivamente el transporte público para tratar de reducir la huella de carbono. Además, se ha trabajado únicamente en digital, por lo que el uso de recursos naturales ha sido mínimo.

En cuanto al impacto ambiental del producto desarrollado, aunque el entrenamiento de agentes haya requerido un consumo ligeramente más elevado, la ejecución supone un consumo energético parecido al de un programa cualquiera, y la transferencia de los modelos entrenados no supone ningún impacto negativo.

7.4. Dimensión social

A nivel personal, como alguien que ha crecido alrededor de los videojuegos y dado mi interés por el aprendizaje automático, rama en la que me gustaría especializarme, este proyecto ha sido el escenario perfecto para, por una parte, tener una idea general sobre cómo funcionan los videojuegos internamente y, por otra, profundizar mis conocimientos en el ámbito del aprendizaje por refuerzo.

En cuanto a la vida útil del producto, actualmente existen muy pocos simuladores o videojuegos de pádel, en los que generalmente no se utilizan técnicas de aprendizaje por refuerzo para atribuir inteligencia artificial a los personajes no jugadores. Es por ello que el producto desarrollado permite darle otro enfoque al uso de estas técnicas, ofreciendo una alternativa que logre comportamientos más complejos en un menor tiempo de desarrollo.

Si bien es cierto que este proyecto no es una necesidad prioritaria en la sociedad, para aquellas personas interesadas en el mundo del pádel se podría considerar como otra forma de entretenimiento y/o aprendizaje. Después de todo, un simulador permite facilitar la enseñanza aportando un apoyo visual, en el que se podría mostrar situaciones que no se suelen dar en el pádel y se podría hacer un análisis de estas. Además, para otros investigadores interesados en el aprendizaje por refuerzo, este proyecto puede servir como referencia para entender cómo funciona y experimentar con ello.

8. Estudio previo

Este capítulo se divide en dos partes. En la primera parte se estudian los fundamentos del aprendizaje por refuerzo, donde se definen conceptos clave y se analiza cómo se aplican en la práctica estudiando los métodos pertinentes, con el objetivo de entender cómo funcionan los algoritmos que se utilizarán finalmente en el desarrollo del proyecto.

Es posible que para entender algunos de estos conceptos en su totalidad sea necesario tener un conocimiento previo sobre el aprendizaje profundo, por lo que se recomienda el libro *Deep Learning* [6] como recurso adicional.

En la segunda parte se exploran las características de *Unity ML-Agents Toolkit*, dando énfasis a aquellas herramientas y funcionalidades que son relevantes para la fase de desarrollo. En particular, se comentan los componentes clave de ML-Agents y los métodos de entrenamiento que se ponen en práctica en este proyecto.

8.1. Estructura de un problema de RL

8.1.1. Proceso de decisión de Márkov

En la Sección 1.2 se propuso una definición tanto formal como informal del proceso por el que pasa un agente de RL. En matemáticas, este proceso se conoce como el proceso de decisión de Márkov (MDP). De hecho, cuando un problema puede ser descrito como un MDP, un algoritmo de RL puede ser una buena elección para resolverlo. Normalmente, un MDP se define como una 5-tupla (S, A, R, P, γ) donde:

- S es el conjunto de todos los estados del entorno.
- A es el conjunto de todas las acciones que puede realizar un agente.
- $R : S \times A \times S \rightarrow \mathbb{R}$ es la función de recompensa $r_t = R(s_t, a_t, s_{t+1})$ donde r_t es la recompensa recibida tras pasar del estado s_t al estado s_{t+1} debido a la acción a_t .
- $P : S \times A \rightarrow P(S)$ es la función de probabilidad de transición, siendo $P(s'|s, a)$ la probabilidad de llegar al estado s' si se empieza en el estado s y se toma la acción a .
- γ es el factor de descuento que determina la importancia de las recompensas futuras en relación con las más cercanas.

Dependiendo del contexto de un problema, en ocasiones se utilizan variaciones de esta tupla en las que se sustituye el factor de descuento γ con una distribución del estado inicial p_0 , o bien se define únicamente con las variables S, A, R y P . Por ejemplo, en entornos en los que, dependiendo del estado inicial, la secuencia de acciones óptima puede variar notablemente, sería relevante definir la distribución del estado inicial.

El nombre de este proceso hace referencia a la propiedad de Márkov: las transiciones dependen únicamente del estado y la acción más reciente, y no del historial de estados y acciones pasadas. Esta propiedad es fundamental para los problemas de RL, ya que permite simplificar la representación de estados y, de esta manera, reducir la complejidad computacional.

8.1.2. Espacio de observaciones y estados

El espacio de observaciones y estados es la información que recibe un agente del entorno. Un estado s es una descripción completa del estado del mundo, sin información oculta. Una observación o , en cambio, es una descripción parcial del mundo la cual puede omitir alguna información, dependiendo de cómo se haya modelado el problema.

Los estados y las observaciones se suelen representar con vectores, matrices o tensores. Por ejemplo, el estado de un juego de Atari se puede representar como una matriz RGB con el valor de cada píxel; el estado de un brazo robot se puede representar como un vector con las posiciones y velocidades angulares de sus ejes. En la práctica, la notación estándar s hace referencia a observaciones y estados indistintamente, ya que se sobrentiende de qué se trata realmente a partir del contexto del problema.

8.1.3. Espacio de acciones

El espacio de acciones es el conjunto de acciones válidas que se pueden realizar en un entorno determinado. En algunos entornos, como Atari o el ajedrez, el espacio de acciones es discreto, ya que el número de movimientos que puede realizar un agente es limitado. En otros, como por ejemplo el de un agente que controla un robot que se mueve en un mundo físico, el espacio de acciones es continuo, ya que acciones como por ejemplo el ángulo de giro se expresan mediante valores reales y por lo tanto es infinito.

8.1.4. Retorno, episodios y el factor de descuento

El objetivo de un agente es maximizar la recompensa acumulada que recibe a largo plazo. Dada una secuencia de recompensas recibidas a partir de un instante t , denotada $R_{t+1}, R_{t+2}, R_{t+3}, \dots$, en general, el aspecto que se busca maximizar es el retorno esperado (o recompensa acumulada esperada), donde el retorno G_t se define como:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T = \sum_{k=t+1}^T R_k \quad (2)$$

donde T es el último instante de la secuencia.

Este planteamiento tiene sentido cuando existe una noción temporal de inicio y fin en el entorno, es decir, cuando la interacción entre el agente y el entorno puede dividirse en subsecuencias, conocidas como episodios. Cada episodio termina en un estado final, seguido de un reinicio hacia un estado inicial estándar o una instancia de una distribución de estados iniciales.

Por otra parte, hay entornos en los que la interacción entre el agente y el entorno no se puede dividir en episodios, sino que continúa infinitamente, conocidos como entornos continuos.

En los entornos continuos, el planteamiento de la Ecuación 2 resulta ser un problema ya que $T = \infty$, por lo que el objetivo a maximizar podría ser infinito. El concepto que se introduce para solucionar este problema es el factor de descuento $\gamma \in [0, 1]$, dando lugar al retorno con descuento con horizonte infinito.

El retorno con descuento con horizonte infinito consiste en la suma de todas las recompensas obtenidas por el agente, pero descontadas según cómo de lejos en el futuro se hayan obtenido:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (3)$$

La intuición detrás del uso del factor de descuento consiste en que es más probable obtener aquellas recompensas más cercanas que no las más lejanas, debido a la incertidumbre acerca del futuro en un sistema dinámico, por lo que se le atribuye más valor a estas primeras. Cuando $\gamma = 0$, el agente solo tiene en cuenta las recompensas inmediatas. En cambio, cuando γ se approxima a 1, también da relevancia a las recompensas futuras, pero no tanta como a las inmediatas.

Matemáticamente, cuando $\gamma < 1$, la suma infinita de la Ecuación 3 puede converger hacia un número finito, lo cual simplifica su uso.

8.2. RL basado en modelos y sin modelos

Una vez entendido cómo se estructura un problema de RL, lo siguiente es estudiar cómo se utilizan estos conceptos en el entrenamiento de agentes, y saber qué clase de algoritmos se utilizan para este fin.

En la clasificación de algoritmos de RL, la principal distinción parte de preguntarse si el agente tiene acceso a un *modelo del entorno*. Un modelo del entorno consiste en la representación de la dinámica de un entorno, es decir, una función capaz de predecir la transición de estados y la recompensa que conlleva llegar a ciertos estados. Aunque este TFG se centra exclusivamente en el RL sin modelos, por completitud, conviene tener también una idea general sobre el RL basado en modelos.

En el RL basado en modelos, un agente planifica por adelantado su trayectoria, explorando todo un espacio de posibles estados y acciones, es decir, el modelo, para elegir de manera explícita qué opciones generan una recompensa acumulada óptima. El modelo puede ser dado o aprendido, pero debido a que usualmente es complicado construir un modelo exacto del entorno, suele ser el agente quien aprende una aproximación del modelo durante el entrenamiento.

Aunque los métodos basados en modelos necesitan pocas muestras, aprender un modelo no es sencillo y es intensivo desde un punto de vista computacional, ya que requieren más capacidad de procesamiento y memoria. Estos métodos suelen utilizarse en entornos bien definidos y no cambiantes. Un ejemplo de métodos basados en modelos es AlphaZero, un algoritmo capaz de jugar a ajedrez, shogi y go, desarrollado por DeepMind.

La contraparte de los métodos basados en modelos son los métodos sin modelos, los cuales no requieren un modelo del entorno para funcionar. En su lugar, utilizan un enfoque de prueba y error para tratar de lograr una estrategia óptima. Durante el proceso de aprendizaje, mediante la experiencia (es decir, la interacción del agente con el entorno), el agente estima una “función de valor” o una “política” (conceptos que se explican más adelante) que una vez entrenadas las utiliza para determinar qué acciones realizar.

Los métodos sin modelos, a pesar de requerir una gran cantidad de muestras, tienden a ser más fáciles de implementar y entrenar. Suelen utilizarse en entornos grandes, cambiantes y que no se pueden describir fácilmente.

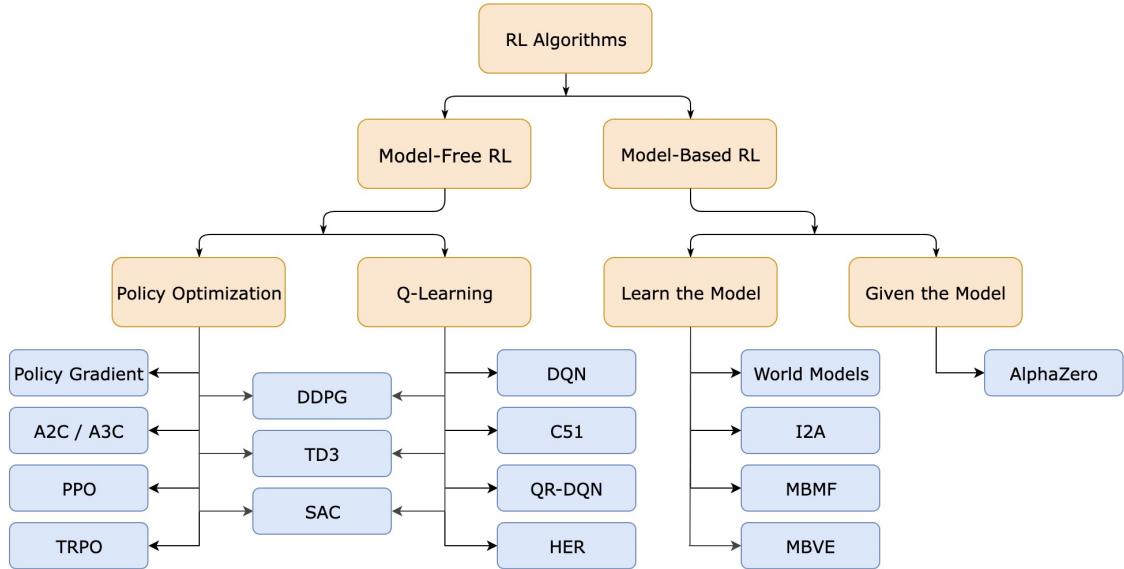


Figura 4: Taxonomía de algoritmos de RL. (Fuente: [7])

8.3. Métodos sin modelos

Recordando la definición formal de RL de la Sección 1.2, cuando un agente percibe un estado $s_t \in S$, selecciona una acción $a \in A(s_t)$ y pasa al estado s_{t+1} . Esta selección se realiza de acuerdo a la política del agente π , una función que indica qué acción realizar cuando el agente se encuentra en un determinado estado. El objetivo final en RL es, por lo tanto, encontrar una política óptima π^* , es decir, la política que maximiza el retorno esperado cuando el agente actúa de acuerdo a esta.

La política óptima se puede obtener de dos maneras:

- Directamente, cuando el agente aprende una política, es decir, qué acción realizar dado el estado actual: métodos basados en la política.
- Indirectamente, cuando el agente aprende una función de valor, es decir, cuán valioso es un estado, y elige las acciones que le lleven al mayor número de estados valiosos: métodos basados en el valor.

Por lo tanto, dentro de los métodos sin modelos, es posible hacer otra distinción según si se basan en el valor o en la política.

8.3.1. Métodos basados en el valor

En los métodos basados en el valor, el agente aprende una función de valor que devuelve el retorno esperado que puede obtener cuando empieza en cierto estado y actúa de acuerdo a una política hasta el final.

La política en los métodos basados en el valor no es algo que se pueda aprender, a diferencia de la política en los métodos basados en la política, como se verá más adelante. En los métodos basados en el valor, la política se trata de una función predefinida a mano y suele consistir en una estrategia sencilla. Por ejemplo, una vez aprendida la función de valor, una política que podría adoptar el agente sería el de elegir siempre la acción que más valor le genere. A esta política se le llama política *greedy*.

La idea de los métodos basados en el valor es que el agente, a medida que interactúa con el entorno, vaya afinando esta función de valor para que, una vez termine el entrenamiento, pueda utilizarla como guía para determinar la trayectoria a seguir.

Hay dos tipos de funciones de valor:

- La función de valor del estado $v_\pi(s)$, la cual devuelve el retorno esperado cuando el agente empieza en el estado s y actúa siguiendo una política π hasta el final.

$$v_\pi(s) = \mathbb{E}_\pi [G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \quad (4)$$

- La función de valor de la acción $q_\pi(s, a)$, la cual devuelve el retorno esperado cuando el agente empieza en el estado s , realiza una acción a , y sigue una política π hasta el final.

$$q_\pi(s, a) = \mathbb{E}_\pi [G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \quad (5)$$

Las funciones de valor introducen un orden parcial sobre las distintas políticas: una política π es mejor o igual que otra política π' cuando su retorno esperado es mayor o igual que el de la política π' para todos los estados. Formalmente, para una función de valor del estado $v_\pi(s)$, y de manera similar para una función de valor de la acción $q_\pi(s, a)$, $\pi \geq \pi'$ si y solo si $v_\pi(s) \geq v_{\pi'}(s)$ para todos los estados $s \in S$. Por lo tanto, siempre habrá al menos una política que es mejor o igual que el resto de políticas, es decir, una política óptima π^* .

Encontrar la función de valor óptima, es decir, la función que maximiza el retorno esperado, daría lugar a obtener una política óptima:

$$v^*(s) = \max_{\pi} v_\pi(s) \quad (6)$$

$$q^*(s, a) = \max_{\pi} q_\pi(s, a) \quad (7)$$

Ecuación de Bellman

En la práctica, estimar las funciones de valor con las Ecuaciones 4 y 5 tal y como estan definidas puede llegar a ser algo tedioso: primero porque para el cálculo del valor de un solo estado es necesario hacer un recorrido de toda una secuencia de recompensas y estados; y segundo porque son funciones que deben estar definidas para todos los estados posibles, por lo que muchos de estos cálculos se acaban repitiendo, ya que algunos recorridos son subconjuntos de otros.

Para tratar de simplificar las funciones de valor, se siguen los principios de la programación dinámica para dividir el problema de calcular el retorno G_t , el cual es bastante complejo, en subproblemas más pequeños y manejables.

A partir de la función de retorno G_t (Ecuación 3), se puede definir una relación recursiva entre la recompensa inmediata y las recompensas futuras:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \quad (8)$$

Sustituyendo esta relación en la función de valor del estado $v_\pi(s)$:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi [G_t | S_t = s] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \sum_a \pi(a | s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_\pi [G_{t+1} | S_{t+1} = s'] \right] \\ &= \sum_a \pi(a | s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_\pi(s') \right] \end{aligned} \quad (9)$$

Similarmente, para la función de valor de la acción $q_\pi(s, a)$:

$$q_\pi(s, a) = \sum_{s', r} p(s', r | s, a) \left[r + \gamma \sum_{a'} \pi(a' | s') q_\pi(s', a') \right] \quad (10)$$

Estas son las ecuaciones de Bellman para las funciones de valor del estado y de la acción. Siguiendo una estructura recursiva, se descompone el valor de un estado o un par estado-acción en la suma de una recompensa inmediata más el valor descontado del siguiente estado. De esta manera, para calcular el retorno esperado de un estado o un par estado-acción basta con tener información del estado actual y del estado siguiente, en lugar de toda una secuencia de estados, resultando en un proceso mucho más eficiente.

Estrategias de aprendizaje: Monte Carlo y Diferencia Temporal

Hasta ahora se ha hablado de qué son las funciones de valor, para qué se utilizan y cómo se calculan. El cálculo, sin embargo, no es un proceso que se pueda ejecutar directamente haciendo un recorrido exhaustivo sobre todas las posibles secuencias de estados, hasta hallar la función de valor que mejor describa la bondad de cada estado o par estado-acción. Más bien, se trata un proceso iterativo en el que el agente, a medida que interactúa con el entorno, actualiza el valor de cada estado según la cantidad de recompensas que haya obtenido. A este proceso se le llama entrenamiento.

Durante el entrenamiento, se utilizan distintas estrategias de aprendizaje que, en esencia, determinan la frecuencia con la que se actualizan los valores de cada estado. En esta sección se estudian las estrategias de aprendizaje de Monte Carlo y Diferencia Temporal.

En la estrategia de Monte Carlo, el agente actualiza la función de valor al final de un episodio, utilizando el retorno del episodio G_t como objetivo:

$$v(s_t) \leftarrow v(s_t) + \alpha[G_t - v(s_t)] \quad (11)$$

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha[G_t - q(s_t, a_t)] \quad (12)$$

donde α es la tasa de aprendizaje (*learning rate*). De esta manera, tras cada episodio, el valor de los estados recorridos se van actualizando dependiendo de cuán mejor o peor haya sido el retorno obtenido a partir del estado correspondiente, incrementándose o decrementándose dependiendo de la tasa de aprendizaje α , hasta converger a una función de valor óptima.

En la estrategia de Diferencia Temporal, a diferencia de la estrategia de Monte Carlo, el agente actualiza la función de valor en cada paso (o iteración). Esta vez, el agente no dispone de ningún retorno G_t , ya que no experimenta un episodio entero. En su lugar, estima el retorno G_t con la suma de la recompensa inmediata R_{t+1} más el valor descontado del siguiente estado:

$$v(s_t) \leftarrow v(s_t) + \alpha[R_{t+1} + \gamma v(s_{t+1}) - v(s_t)] \quad (13)$$

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha[R_{t+1} + \max_a q(s_{t+1}, a) - q(s_t, a_t)] \quad (14)$$

Utilizar la estimación del retorno se conoce como *bootstrapping*, ya que se aproxima la función de valor utilizando estimaciones de valores ya existentes, $v(s_{t+1})$ o $q(s_{t+1}, a_{t+1})$, en lugar de una muestra completa G_t . La diferencia entre la aproximación del retorno y el valor del estado o par estado-acción actual se denomina *TD error* (*Temporal Difference error*).

A la hora de comparar el efecto del uso de una estrategia u otra, a menudo se habla sobre el sesgo y la varianza que se produce según qué valor se utiliza como objetivo durante el entrenamiento, si el retorno G_t obtenido a partir de una muestra real o una aproximación de este, utilizando valores *bootstrap*.

Respecto al sesgo, recordando que el objetivo final es aproximar las funciones de valor reales definidas por las Ecuaciones 4 y 5, en la estrategia de Monte Carlo, se utilizan muestras de recompensas obtenidas directamente del entorno como objetivo para aproximar la función de valor, sin necesidad de utilizar estimaciones intermedias, evitando de esta manera cualquier sesgo en la estimación de valores de estados o acciones. En la estrategia de Diferencia Temporal, en cambio, al utilizarse valores *bootstrap* que se definen inicialmente de manera arbitraria como una parte del objetivo, los cuales no están relacionados con los valores reales, se genera un sesgo en las estimaciones al depender de esta inicialización.

En cuanto a la varianza, en la estrategia de Monte Carlo, al utilizar muestras episódicas, el retorno puede variar mucho de un episodio a otro, provocando una alta varianza en la actualización de las funciones de valor. En la estrategia de Diferencia Temporal, en cambio, al hacerse actualizaciones en cada paso, no suele producirse mucho cambio en las recompensas, por lo que la varianza es menor.

En la práctica, el aprendizaje mediante el uso de estrategias de Diferencia Temporal suele converger más eficientemente, siempre y cuando el sesgo no suponga ningún problema. La convergencia utilizando estrategias de Monte Carlo, en cambio, suele ser más lenta al requerir más muestras debido a la alta varianza entre episodios, pero es conceptualmente más sencillo, robusto y fácil de implementar.

On-policy vs. off-policy

Cuando se habla sobre los algoritmos de RL, aparecen recurrentemente los conceptos de *on-policy* y *off-policy*. Un método es *off-policy* cuando, durante el entrenamiento, hay una distinción entre la política que sigue el agente para actuar y para aprender. En cambio, cuando ambas políticas son iguales, es decir, se actúa con la política que se está aprendiendo, se considera *on-policy*.

Otra manera de entenderlo es desde el punto de vista de cómo se obtienen de muestras. Cuando las muestras se obtienen con la política objetivo, es decir, la política que se pretende optimizar, se considera un método *on-policy*. En cambio, si las muestras se obtienen con cualquier otra política, la cual no guarda relación con la política objetivo, se trata de un método *off-policy*.

El algoritmo Q-learning, por ejemplo, es un algoritmo *off-policy* porque para actuar utiliza una política *ϵ -greedy*, mientras que para afinar la función de valor utiliza una política *greedy*. El algoritmo REINFORCE, por otro lado, es un algoritmo *on-policy* ya que durante el entrenamiento el agente actúa de acuerdo a la política que se está entrenando para obtener las muestras.

Q-learning

El algoritmo Q-learning es un método basado en el valor, *off-policy*, que utiliza la Diferencia Temporal como estrategia de aprendizaje. En particular, se utiliza para entrenar una función de valor de la acción, denominada Q-function.

Internamente, la Q-function no es nada más que una tabla, denominada Q-table, que hace una asignación de todos los posibles pares estado-acción a su respectivo valor Q-value. La idea es que se inicialice esta tabla arbitrariamente (generalmente, se inicializa a 0) para que, tras el entrenamiento, acabe convergiendo hacia una tabla óptima, es decir, una tabla que asigna más valor a aquellos pares estado-acción que realmente contribuyen a conseguir un retorno óptimo. Teniendo esta Q-function óptima, el agente puede utilizarla como guía para determinar cómo actuar en un estado determinado.

Como ya se había mencionado anteriormente, el algoritmo Q-learning se trata de un método *off-policy* ya que la política que utiliza para actuar es distinta a la que utiliza para afinar la Q-function. Haciendo referencia al pseudocódigo de Q-learning (1), para obtener una recompensa R y el estado siguiente S' , se escoge una acción A dado el estado actual S utilizando una política *ϵ -greedy*. Para actualizar el valor del par estado-acción $Q(S, A)$, en cambio, utiliza una política *greedy* para calcular el *TD error*.

Utilizando una política *ϵ -greedy*, la idea es que con una probabilidad de ϵ el agente escoja una acción aleatoria, y con una probabilidad de $1 - \epsilon$ escoja la mejor acción disponible, según el valor de un determinado par estado-acción.

La política ϵ -greedy se utiliza para controlar la compensación entre exploración y explotación. La exploración consiste en ejecutar acciones aleatorias con tal de hallar más información sobre el entorno. La explotación consiste en aprovechar la información conocida para maximizar las recompensas. Cuando no se controla la compensación entre exploración y explotación, es posible que el agente caiga en la trampa de realizar ciertas acciones en bucle, consiguiendo recompensas muy pequeñas aunque efectivas, pero perdiendo la oportunidad de ejecutar acciones que le conlleven a conseguir una recompensa mucho más favorable, por no haber explorado suficientemente el entorno.

Algorithm 1 Q-learning

```

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\epsilon > 0$ 
Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$ 
for each episode do
    Initialize  $S$ 
    for each step of episode do
        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
        Take action  $A$ , observe  $R, S'$ 
         $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
         $S \leftarrow S'$ 
    end for
end for

```

Deep Q-learning

El algoritmo Deep Q-learning [8], desarrollado por DeepMind, es una variante de Q-learning que utiliza una red neuronal profunda, denominada Q-network, en lugar de una tabla. Este algoritmo se utiliza, en particular, para aprender políticas de control a través de observaciones en un espacio de alta dimensionalidad.

El problema de Q-learning es que se trata de un método tabular: su uso se limita a entornos cuyo espacio de estados y acciones es lo suficientemente pequeño como para guardar todos los pares estado-acción posibles en una tabla, por lo que no es escalable. Al introducir redes neuronales profundas para aproximar la Q-function, es posible aprender asignaciones complejas utilizando información de estados como datos de entrada, que es precisamente de lo que trata Deep Q-learning.

Para entender cómo funciona este algoritmo, es importante entender el tipo de contextos para los que fue concebido inicialmente: los juegos de Atari 2600. El objetivo de Deep Q-learning es entrenar políticas de control para superar juegos arcade como *Space Invaders* o *Pong*, utilizando una combinación de redes neuronales convolucionales (C-NN) y redes neuronales completamente conectadas (FC-NN), de manera que la C-NN procesa frames del juego para atribuirles Q-values mediante la FC-NN.

A partir de este planteamiento, surgen nuevas funcionalidades con respecto al algoritmo de Q-learning, que tratan de resolver algunos inconvenientes específicos a este entorno:

- Preprocesamiento de secuencias: en los juegos de Atari 2600, los colores y los bordes de los frames son irrelevantes, por lo que se puede transformar cada frame a tensores de tamaño 84×84 utilizando una escala de grises. Además, debido a que con un frame individual es complicado extraer información como la trayectoria o la velocidad de algunos elementos del juego, se apila el frame actual con los 3 frames anteriores, formando una secuencia de 4 frames. En el pseudocódigo, este preprocesamiento se define con ϕ , y consiste en un tensor de tamaño $84 \times 84 \times 4$.
- Reproducción de experiencias: durante el entrenamiento, las experiencias (o muestras) se guardan en una memoria de reproducción D para utilizarse más tarde en la actualización de los parámetros θ , esto se conoce como reproducción de experiencias. La reproducción aleatoria de experiencias permite entrenar eficientemente un modelo más robusto, ya que evita la correlación entre muestras y el olvido de muestras pasadas.

El objetivo del entrenamiento es, en esencia, igual que el de Q-learning: que la diferencia entre el valor real de un par estado-acción y el Q-value sea mínima. Sin embargo, al tratarse esta vez de una red neuronal profunda, se define una función de pérdida basada en el *TD error*, sobre la que se aplica descenso de gradiente para actualizar los parámetros θ mediante propagación hacia atrás.

Algorithm 2 Deep Q-learning with Experience Replay

```

Initialize replay memory  $D$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights
for episode = 1,  $M$  do
    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$ 
    for  $t = 1, T$  do
        With probability  $\epsilon$  select a random action  $a_t$ 
        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ 
        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$ 
        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ 
        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$ 
        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$ 
        Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$ 
        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  w.r.t  $\theta$ 
    end for
end for

```

8.3.2. Métodos basados en la política

En la sección anterior se ha hablado de cómo un agente, mediante el aprendizaje de una función de valor que cuantifica cuán valioso es un estado o un par estado-acción, puede llegar a adquirir consecuentemente una política óptima. Lo siguiente a estudiar son los métodos basados en la política, donde en lugar de aprender una función de valor, se aprende directamente la política que debe seguir el agente. En particular, se estudian los métodos de gradiente de política (*policy gradient*), los cuales son un subconjunto de los métodos basados en la política.

En los métodos de gradiente de política, el agente utiliza una política parametrizada para seleccionar qué acción tomar, sin necesidad de consultar ninguna función de valor. Dado el vector de parámetros de una política $\theta \in \mathbb{R}^{d'}$, se define

$$\pi(a|s, \theta) = \Pr \{A_t = a | S_t = s, \theta_t = \theta\} \quad (15)$$

como la probabilidad de escoger la acción a , dado el estado s y los parámetros θ en un instante t .

Cuando el espacio de acciones es discreto y no demasiado grande, una manera común y natural de parametrizar la política es definir una función de preferencia numérica parametrizada $h(s, a, \theta) \in \mathbb{R}$ para cada par estado-acción. De esta manera, las acciones con una mayor preferencia dado un cierto estado obtienen una mayor probabilidad de ser seleccionadas siguiendo, por ejemplo, la distribución *softmax*:

$$\pi(a|s, \theta) = \frac{e^{h(s, a, \theta)}}{\sum_b e^{h(s, b, \theta)}} \quad (16)$$

La función de preferencia numérica parametrizada se puede definir arbitrariamente. Por ejemplo, se puede calcular la preferencia para cada par estado-acción mediante redes neuronales profundas, donde θ es el vector de todos los pesos de la red neuronal.

Utilizando la política parametrizada $\pi(a|s, \theta)$, el objetivo es aprender los parámetros θ que dan lugar a conseguir el mejor retorno posible. Para ello, se define una medida escalar de rendimiento $J(\theta)$, la cual se busca maximizar mediante gradiente ascendiente respecto al parámetro θ :

$$\theta_{t+1} \leftarrow \theta_t + \alpha \widehat{\nabla J(\theta_t)} \quad (17)$$

donde $\widehat{\nabla J(\theta_t)}$ es una estimación estocástica cuya esperanza aproxima el gradiente de la medida de rendimiento con respecto a su argumento θ_t .

En los métodos basados en el valor, la idea era afinar las funciones de valor hasta converger a un valor que pudiera aproximar el retorno esperado real, que luego utilizarían los agentes para decidir qué acción realizar. Esta vez, en los métodos basados en la política, se afinan los parámetros de la política para controlar la distribución de probabilidad de las acciones, favoreciendo directamente aquellas acciones que permitan maximizar el retorno esperado.

Teorema del gradiente de política

La incógnita que queda por resolver es cómo se define exactamente la medida de rendimiento $J(\theta)$, de manera que se pueda aplicar gradiente ascendiente para afinar el parámetro θ , y todo ello con la finalidad que favorecer aquellas acciones que permiten obtener una mayor recompensa.

La medida de rendimiento $J(\theta)$ se define de manera distinta según si se trata de un entorno continuo o episódico. En este TFG, se estudia el caso episódico ya que es el tipo de entorno que se utilizará en el desarrollo de la aplicación. En el caso episódico, se define el rendimiento como:

$$J(\theta) = v_{\pi_\theta}(s_0) \quad (18)$$

donde v_{π_θ} es la función de valor real para π_θ , la política determinada por θ .

Utilizando una aproximación de la función de valor real, a primera vista, es complicado afinar los parámetros de la política de manera que se garantice una mejora en la política en cuestión. El problema es que el rendimiento depende tanto de la selección de acciones como de la distribución de estados (Ecuación 9), ambos afectados por el parámetro de la política. Dado un estado, es sencillo calcular el efecto que tienen los parámetros de la política sobre las acciones y las recompensas. El efecto sobre la distribución de estados, sin embargo, no es diferenciable debido a que la función de la dinámica del entorno normalmente se desconoce (se trata de un método sin modelo).

Para solventar este problema, se utiliza una expresión analítica para el gradiente del rendimiento con respecto al parámetro de la política, la cual no implica derivar la distribución de estados, conocida como el teorema del gradiente de política. El teorema del gradiente de política establece que:

$$\begin{aligned} \nabla J(\theta) &\propto \sum_s \mu_\pi(s) \sum_a q_\pi(s, a) \nabla_\theta \pi(a|s, \theta) \\ &= \sum_s \mu_\pi(s) \sum_a \pi(a|s, \theta) q_\pi(s, a) \frac{\nabla_\theta \pi(a|s, \theta)}{\pi(a|s, \theta)} \\ &= \mathbb{E}_\pi [q_\pi(s, a) \nabla_\theta \ln \pi(a|s, \theta)] \end{aligned} \quad (19)$$

donde $\mu_\pi(s)$ es una distribución estacionaria de los estados sobre la política π , y \mathbb{E}_π es la esperanza $\mathbb{E}_{s \sim \mu_\pi, a \sim \pi_\theta}$ cuando tanto la distribución de estados como acciones siguen la política π_θ (*on-policy*).

Para profundizar más sobre el teorema del gradiente de política se recomiendan los artículos de Lilian Weng [9] y Cheng Xi Tsou [10], en los que se da una explicación más detallada sobre la demostración propuesta por Sutton y Barto [1] (Sección 13.2).

REINFORCE

Un algoritmo clásico que utiliza el gradiente de política *vanilla*, es decir, la forma básica del gradiente de política (descrita en el apartado anterior), es REINFORCE. Este algoritmo sigue una estrategia de Monte Carlo, ya que utiliza muestras episódicas para actualizar el parámetro de la política θ .

REINFORCE funciona porque dada la relación de la Ecuación 5, es posible sustituir la función de valor $q_\pi(s, a)$ del teorema del gradiente de política (Ecuación 19) por el retorno G_t , el cual se puede calcular a partir de muestras episódicas reales:

$$\begin{aligned} \nabla J(\theta) &= \mathbb{E}_\pi [q_\pi(s, a) \nabla_\theta \ln \pi(a|s, \theta)] \\ &= \mathbb{E}_\pi [G_t \nabla_\theta \ln \pi(a|s, \theta)] \end{aligned} \quad (20)$$

Algorithm 3 REINFORCE: Monte Carlo Policy-Gradient Control (episodic)

Input: a differentiable policy parametrization $\pi(a|S, \theta)$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ (e.g., to 0)

for each episode **do**

Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|S, \theta)$

for each episode step $t = 0, 1, \dots, T - 1$ **do**

$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$

$\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \theta)$

end for

end for

Métodos actor-crítico

Dentro de los métodos de gradiente de política, hay una clase de algoritmos en los que además de aprender una política, se aprende también una función de valor que se utiliza como refuerzo durante la actualización de la política. Esta función de valor sustituye, precisamente, a la función de valor presente en la Ecuación 19, y permite resolver problemas como la varianza del gradiente en los métodos *vanilla*. Estos métodos se conocen como los métodos actor-crítico.

En los métodos actor-crítico, al disponer de una aproximación de la función de valor, es posible seguir la estrategia de Diferencia Temporal gracias a los valores *bootstrap*. De esta manera, al poder realizar actualizaciones paso por paso, se reduce significativamente la varianza entre muestras. En REINFORCE, por ejemplo, es necesario seguir la estrategia de Monte Carlo y utilizar muestras episódicas para poder calcular el retorno G_t , lo cual introduce varianza en el gradiente, como ya se había comentado previamente.

Los métodos actor-crítico consisten en dos modelos (o aproximaciones de funciones), generalmente redes neuronales profundas:

- El crítico, quien actualiza el vector de pesos $w \in \mathbb{R}^d$ de la función de valor, la cual puede ser una función de valor del estado $\hat{v}(s, w)$ o de la acción $\hat{q}(s, a, w)$ dependiendo del algoritmo.
- El actor, quien actualiza el parámetro de la política θ para $\pi(a|s, \theta)$, según el valor aproximado por el crítico.

La idea es, por lo tanto, que dado dos modelos inicializados arbitrariamente, el actor y el crítico, mientras el agente interactúa con el entorno y se procesan las muestras, por un lado el crítico va aprendiendo la función de valor según las recompensas obtenidas, mientras que por otro, el actor va aprendiendo la política utilizando la estimación del valor del estado o par estado-acción actual proporcionada por el crítico para realizar el ascenso de gradiente.

Algorithm 4 Actor-Critic with action-value function

Input: a differentiable policy parameterization $\pi(a|s, \theta)$
 Input: a differentiable action-value function parametrization $\hat{q}(s, a, w)$
 Parameters: learning rates $\alpha^\theta > 0$, $\alpha^w > 0$, discount factor γ
 Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$ (e.g., to 0)
for each episode **do**
 Initialize S (first state of episode)
 $I \leftarrow 1$
for each time step t in episode **do**
 $A \sim \pi(\cdot|S, \theta)$
 Take action A , observe S', R
 $\theta \leftarrow \theta + \alpha^\theta I \hat{q}(S, a, w) \nabla_\theta \ln \pi(A|S, \theta)$
 $\delta \leftarrow R + \gamma \hat{q}(S', a, w) - \hat{q}(S, a, w)$ \triangleright (if S' is terminal, then $\hat{q}(S', a, w) = 0$)
 $w \leftarrow w + \alpha^w \delta \nabla_w \hat{q}(S, a, w)$
 $I \leftarrow \gamma I$
 $S \leftarrow S'$
end for
end for

Advantage Actor Critic

Para reducir aún más la varianza, es posible modificar el teorema del gradiente de política añadiendo un *baseline* $b(s)$, el cual puede ser cualquier función que dependa del estado s , que no altera la esperanza del gradiente:

$$\begin{aligned}\nabla J(\theta) &= \mathbb{E}_\pi [\nabla_\theta \ln \pi(a|s, \theta) q_\pi(s, a)] \\ &= \mathbb{E}_\pi [\nabla_\theta \ln \pi(a|s, \theta) (q_\pi(s, a) - b(s))]\end{aligned}\tag{21}$$

Utilizando el *baseline*, es posible reducir el incremento o decremento del gradiente, afectado por el valor $q_\pi(s, a)$.

Recordando la diferencia entre las funciones de valor: una función de valor de la acción $q_\pi(s, a)$ indica el retorno esperado cuando se realiza la acción a en el estado s , y se sigue la política π hasta el final; una función de valor del estado $v_\pi(s)$, en cambio, solo indica el retorno esperado cuando se empieza en el estado s y se sigue la política π hasta el final. Si hubiera alguna ventaja $A_\pi(s, a)$ en escoger una determinada acción a sobre la acción que sigue la política $a \sim \pi(a|s)$ en el estado s , esta se podría calcular mediante la diferencia de ambas funciones de valor:

$$A_\pi(s, a) = q_\pi(s, a) - v_\pi(s)\tag{22}$$

Al introducir la función de ventaja en el teorema del gradiente de política, se obtiene el gradiente utilizado en el algoritmo Advantage Actor Critic (A2C):

$$\nabla J(\theta) = \mathbb{E}_\pi [\nabla_\theta \ln \pi(a|s, \theta) A_\pi(s, a)]\tag{23}$$

La intuición detrás del uso de la función de ventaja es la siguiente: cuando escoger una acción a resulta ser mejor que seguir la política π , entonces $A_\pi(s, a) > 0$, por lo que se incrementa la probabilidad logarítmica de la acción a asociada a la ventaja; contrariamente, cuando es peor que seguir la política, $A_\pi(s, a) < 0$, su probabilidad logarítmica se decrementa; en el caso de que a sea justamente la acción sugerida por la política π , la ventaja sería $A_\pi(s, a) = 0$, por lo que no se produciría ningún cambio en la política.

Gracias al uso de la ventaja en el cálculo del gradiente, es posible reducir la varianza de la actualización de la política al hacer pasos proporcionales a cómo de mejor o peor es una determinada acción respecto al comportamiento que se hubiera seguido normalmente, en lugar de utilizar directamente el retorno esperado, el cual puede cambiar mucho entre muestras.

Para conocer más detalles sobre por qué el *baseline* no altera la esperanza del gradiente, y qué métodos existen para aproximar las funciones de valor del estado y de la acción para calcular la ventaja, se recomienda el artículo de Alexander Van de Kleut [11].

8.4. Unity ML-Agents Toolkit

El *toolkit* de ML-Agents de Unity ofrece un kit de desarrollo de software (SDK), desarrollado por Unity, diseñado para usarse junto al editor del propio motor para crear entornos de aprendizaje automático. También ofrece implementaciones de algoritmos de aprendizaje automático del estado del arte, basados en PyTorch, que permiten al usuario entrenar agentes y experimentar con distintos métodos de una manera muy sencilla.

8.4.1. Componentes clave

Una aplicación desarrollada sobre el Unity ML-Agents Toolkit está formada por:

- Un entorno de aprendizaje, el cual engloba la escena de Unity y todos los objetos del juego. Desde la escena de Unity, es posible implementar un entorno en el que unos agentes pueden observar, actuar y aprender. Este entorno puede implementarse con la finalidad de entrenar y probar agentes en una misma escena, o bien para entrenar el agente de una manera más genérica, para finalmente utilizarlos en un escenario más específico. Utilizando el SDK de ML-Agents, se puede transformar cualquier escena de Unity en un entorno de aprendizaje tras definir unos agentes y sus comportamientos.
- Una interfaz de programación de aplicaciones (API) de bajo nivel de Python, la cual permite interactuar y manipular un entorno de aprendizaje. Este componente es externo a Unity y se comunica con el entorno de aprendizaje mediante un comunicador. Esta API se utiliza principalmente para controlar y comunicarse con la *Academy*, un objeto de Unity que orquesta todos los agentes y comportamientos de una escena de Unity.
- Un comunicador externo, el cual que conecta el entorno de aprendizaje con la API de bajo nivel de Python.
- Un entrenador basado en Python, el cual ofrece todos los algoritmos de aprendizaje automático que ofrece ML-Agents. Es el programa que se ejecuta para iniciar el entrenamiento de los agentes, dado un cierto método de entrenamiento y su configuración. El entrenador interactúa únicamente con la API de bajo nivel de Python.

- Unos *wrappers* de Gym y PettingZoo, los cuales son interfaces que se utilizan en la implementación de los algoritmos de aprendizaje automático.

En el entorno de aprendizaje, los principales componentes que provienen de ML-Agents son:

- Una clase *Agent*, la cual se puede vincular a cualquier *GameObject* de Unity, y es el componente que permite generar observaciones, realizar las acciones según el comportamiento que tenga asignado, y recibir las recompensas que estén definidos en el entorno de entrenamiento.
- El comportamiento de un agente, el cual define los atributos específicos del agente en cuestión, como por ejemplo el número de observaciones y acciones de las que dispone. Es equivalente a una función que, dado unas observaciones y unas recompensas, devuelve las acciones que debe realizar el agente. Hay tres tipos de comportamiento: aprendizaje, cuando el comportamiento aún debe ser entrenado; heurística, cuando el comportamiento está definido a mano, mediante reglas implementadas en el propio código del agente; e inferencia, cuando el comportamiento infiere un modelo de redes neuronales, obtenido a partir de un comportamiento entrenado.

En un mismo entorno de aprendizaje puede haber varios agentes con distintos comportamientos al mismo tiempo. Mediante la *Academy*, el entorno de aprendizaje asegura que todos los agentes estén sincronizados. Además de los agentes y sus comportamientos, también están los *Side Channels*, utilizados para intercambiar datos con la API de Python, pero no se tienen en consideración en este proyecto. En la Figura 5 se muestra el diagrama de bloques del entorno de ML-Agents, formado por todos los componentes mencionados.

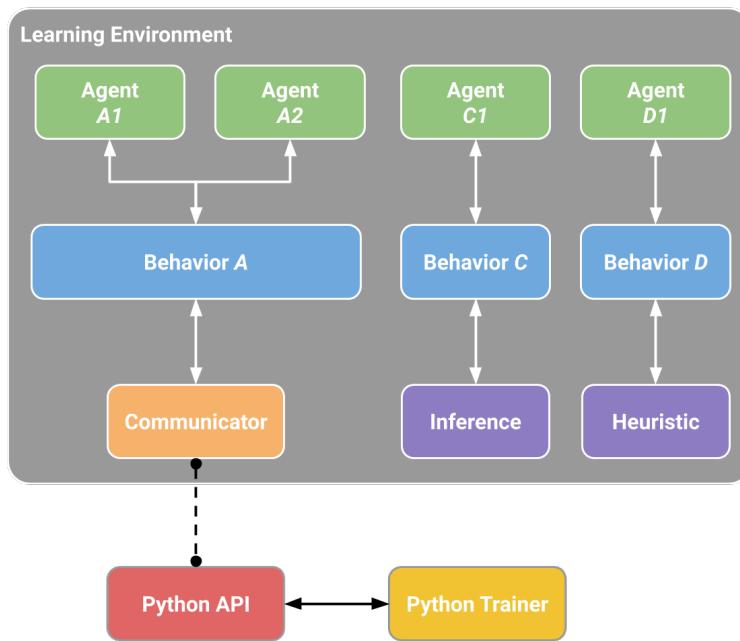


Figura 5: Diagrama de bloques del entorno de ML-Agents. (Fuente: [2])

8.4.2. ML-Agents en la interfaz de Unity

En esta sección se resumen los componentes que se deben implementar para crear un entorno de aprendizaje desde la interfaz de Unity. Este TFG no pretende ser una guía de Unity, por lo que no se explica la interfaz en sí. En caso de necesitar más información sobre Unity, se recomienda consultar la documentación oficial [12].

Lo primero es crear un entorno con el que interactuará el agente. Este entorno no necesita implementar nada específico de ML-Agents, por lo que el usuario es libre de diseñar cualquier tipo de entorno, utilizando las herramientas habituales de Unity.

El elemento que realmente permite formar un entorno de aprendizaje es la clase `Agent`. Los `Game Objects` que se quieran utilizar como agentes deben tener vinculado un `C# Script` que extienda esta clase, e implementar los siguientes métodos:

- `OnEpisodeBegin()`, utilizado para restablecer el agente a unas condiciones iniciales de cada episodio.
- `CollectObservations()`, utilizado para añadir determinadas observaciones del agente a un vector de observaciones.
- `OnActionReceived()`, utilizado para especificar el comportamiento del agente en un paso, basado en las acciones recibidas como parámetro, las cuales proceden ya sea del modelo que se está entrenando, de la inferencia de un modelo ya entrenado o de una heurística definida por el usuario.
- `Heuristic()`, utilizado para especificar el comportamiento del agente manualmente, modificando las acciones pertinentes.
- `AddReward()`, utilizado para definir la función de recompensa.

Además de la clase `Agent`, también se requieren los componentes `Behavior Parameters` y `Decision Requester`. En caso de querer grabar demostraciones para utilizarlas en GAIL, es necesario utilizar el componente `Demonstration Recorder`.

Los parámetros relevantes del componente `Behavior Parameters` (Figura 6) son:

- `Behavior Name`, el nombre de la política.
- `Vector Observation`, donde se define el tamaño del espacio de observaciones.
- `Actions`, donde se define el tamaño del espacio de acciones, tanto continuas como discretas. Cada acción continua consiste en un valor $a \in \mathbb{R}$ dentro del intervalo $[-1, 1]$. Cada acción discreta consiste en un valor discreto cuyo rango se define con `Branch Size`. Si por ejemplo una acción discreta tiene `Branch Size = 3`, los valores posibles son 0, 1 y 2.
- `Model`, donde se añade el modelo obtenido tras el entrenamiento.
- `Behavior Type`, utilizado para especificar si el agente debe seguir el comportamiento definido por defecto, por heurística, o por inferencia.
- `Team Id`, utilizado para especificar a qué equipo pertenece un agente, en caso de que haya varios equipos.

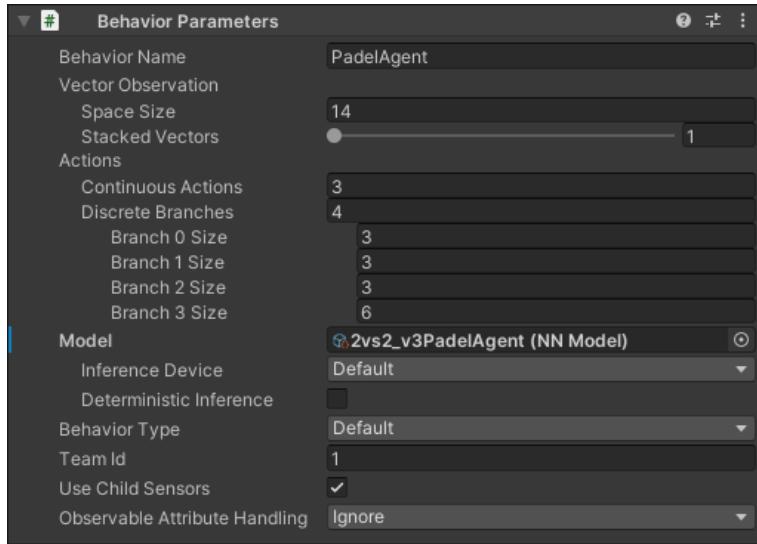


Figura 6: Ejemplo del componente Behavior Parameters en Unity. (Elaboración propia)

El Decision Requester (Figura 7) se utiliza para definir la frecuencia en la que se efectúan las decisiones sobre qué acción realizar tras procesar las observaciones. Por ejemplo, si Decision Period = 5, el agente observa y recibe acciones de la política cada 5 Fixed Updates de Unity. Normalmente, el valor del intervalo de tiempo fijo es de 0.02 segundos, por lo que la frecuencia sería de 0.1 segundos.

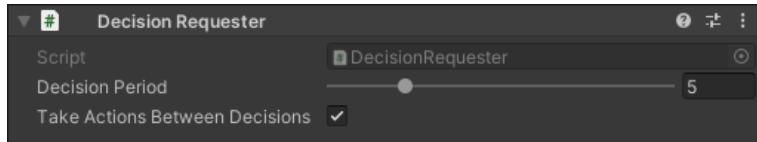


Figura 7: Ejemplo del componente Decision Requester en Unity. (Elaboración propia)

El componente Demonstration Recorder (Figura 8) permite la grabación de las demostraciones necesarias para el aprendizaje por imitación. Cuando se graba una demostración, se asume que las acciones son realizadas por expertos, dadas ciertas observaciones. En este componente se define el número de pasos que se quieren grabar, y es posible indicar un número indefinido de pasos mediante el valor 0. También es necesario especificar el nombre del archivo y el directorio en el que se quiera guardar la demostración. Cuando se graban demostraciones en un entorno multi-agente, basta con añadir todas las demostraciones en un mismo directorio, ya que los métodos de aprendizaje por imitación utilizan todas las demostraciones incluidas en el directorio especificado.

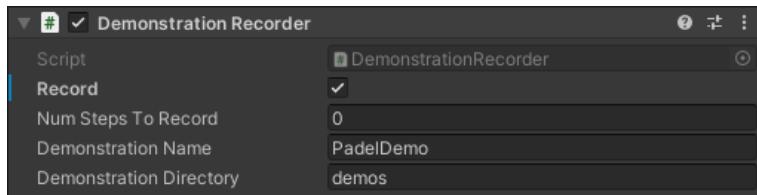


Figura 8: Ejemplo del componente Demonstration Recorder en Unity. (Elaboración propia)

8.4.3. Métodos de entrenamiento implementados en ML-Agents

Los principales algoritmos de aprendizaje por refuerzo disponibles en ML-Agents son Proximal Policy Optimization (PPO) y Soft Actor-Critic (SAC). Entre estos dos, se ha optado por utilizar PPO, ya que se ha demostrado que es un método más genérico y estable que otros algoritmos de RL. Además, citando directamente desde la documentación de ML-Agents: «*Self-play can be used with our implementations of both Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC). However, from the perspective of an individual agent, these scenarios appear to have non-stationary dynamics because the opponent is often changing. This can cause significant issues in the experience replay mechanism used by SAC. Thus, we recommend that users use PPO*», PPO parece funcionar mejor con *Self-Play*, una funcionalidad que se utiliza para entrenar agentes en juegos adversariales.

Para la parte de aprendizaje por imitación, que en realidad no deja de ser aprendizaje por refuerzo, pero con una extensión en la definición de recompensas, se explora el uso de Generative Adversarial Imitation Learning, el cual utiliza un enfoque adversarial para atribuir recompensas al agente por tener un comportamiento similar a un conjunto de demostraciones. También se ha considerado el uso de Behavioral Cloning, pero al ser un método que se limita a enseñar al agente a copiar exactamente las demostraciones proporcionadas, y al tener un entorno de pádel un espacio de estados muy complejo, no es posible ofrecer un conjunto de demostraciones lo suficientemente grande como para poder generalizar el comportamiento del agente, por lo que se ha descartado.

ML-Agents también dispone de métodos adicionales que pueden ser de utilidad para entrenar agentes en un tipo de entorno específico. Entre estos, los métodos relevantes son Self-Play, utilizado en entornos multi-agente competitivos, y MultiAgent Posthumous Credit Assignment (MA-POCA) [13], utilizado en entornos multi-agente cooperativo, al ser el pádel un deporte que combina cooperación con competición. Sin embargo, se ha descartado el uso de MA-POCA al ser un algoritmo implementado en el contexto de entornos multi-agente, cuya teoría no se cubre en este TFG. Self-Play, por otra parte, es una funcionalidad compatible con PPO por lo que no es necesario orientar la implementación del entorno hacia un entorno multi-agente.

Aunque esto último parece ser contradictorio, ya que tiene sentido que el pádel sea un entorno multi-agente, en realidad, este TFG se centra más bien en entornos de agentes individuales, en los que la cooperación entre agentes acaba surgiendo como una consecuencia del entrenamiento y de la manera en la que se asignan las recompensas, a nivel individual o grupal, según cómo se hayan definido.

Proximal Policy Optimization

Proximal Policy Optimization (PPO) [14], desarrollado por OpenAI, es un método de gradiente de política que busca mejorar la estabilidad del entrenamiento de agentes, evitando actualizaciones demasiado grandes en la política.

Se sabe, empíricamente, que es más probable converger hacia una solución óptima cuando la política varía poco entre actualizaciones. Cuando el paso del gradiente es demasiado grande, se corre el riesgo de empeorar la política, haciendo que sea difícil o incluso imposible recuperar su bondad.

Para actualizar la política de manera conservativa, PPO introduce una función que limita el cambio de la política, llamada función objetivo sustituta con límite (*clipped surrogate objective function*), denominada sustituta en el sentido de que es una aproximación de la función objetivo real (maximizar el retorno esperado), ya que no aplica directamente el gradiente, sin límite (como en los métodos *vanilla*), que teóricamente conllevaría a conseguir una política óptima.

En esta función, se sustituye la probabilidad logarítmica de la función objetivo del teorema del gradiente de política con ventaja (Ecuación 23), por una relación de probabilidad definida como

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \quad (24)$$

la cual calcula la relación entre la política actual $\pi_\theta(a_t|s_t)$ y la política anterior $\pi_{\theta_{\text{old}}}(a_t|s_t)$. Sustituyendo esta relación en la función objetivo, y restringiendo el cambio de la política en un pequeño rango, se obtiene la función objetivo sustituta con límite:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}} \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t) \right] \quad (25)$$

donde ϵ es un hiperparámetro, generalmente pequeño (e.g $\epsilon = 0.2$). El primer término dentro de la función de minimización, $r_t(\theta)\hat{A}_t$, es el objetivo sustituto sin límite. El segundo término, $\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t$, modifica el objetivo sustituto limitando la relación $r_t(\theta)$ dentro del intervalo $[1 - \epsilon, 1 + \epsilon]$. Al escoger finalmente el mínimo entre estos dos términos, el objetivo final acaba siendo una cota inferior del objetivo sustituto sin límite, evitando de esta manera cambios excesivamente grandes en la política.

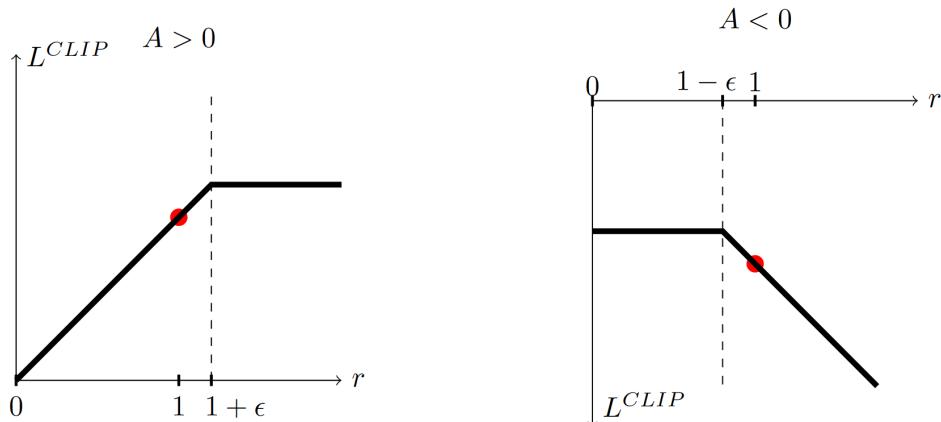


Figura 9: Paso del gradiente de la función sustituto L^{CLIP} en función de la relación $r_t(\theta)$, para ventajas positivas (izquierda) y negativas (derecha). (Fuente: [14])

$r_t(\theta) > 0$	\hat{A}_t	Valor de \min	Objetivo limitado	Signo del objetivo	Gradiente
$r_t(\theta) \in [1 - \epsilon, 1 + \epsilon]$	+	$r_t(\theta)\hat{A}_t$	no	+	✓
$r_t(\theta) \in [1 - \epsilon, 1 + \epsilon]$	-	$r_t(\theta)\hat{A}_t$	no	-	✓
$r_t(\theta) < 1 - \epsilon$	+	$r_t(\theta)\hat{A}_t$	no	+	✓
$r_t(\theta) < 1 - \epsilon$	-	$(1 - \epsilon)\hat{A}_t$	sí	-	0
$r_t(\theta) > 1 + \epsilon$	+	$(1 + \epsilon)\hat{A}_t$	sí	+	0
$r_t(\theta) > 1 + \epsilon$	-	$r_t(\theta)\hat{A}_t$	no	-	✓

Tabla 10: Resumen del comportamiento de la función objetivo sustituto con límite L^{CLIP} . (Fuente: [15])

En la Figura 9 se muestra el cambio en el gradiente del valor de L^{CLIP} en función de la relación $r_t(\theta)$, donde se dan seis casos no triviales (esto es, cuando $r_t(\theta) > 0$), distinguidos en la Tabla 10:

1. Cuando la relación se encuentra dentro del intervalo del límite y la ventaja es positiva, se aplica un gradiente positivo para aumentar la probabilidad de una determinada acción.
2. Igual que en el primer caso, cuando se encuentra dentro del intervalo del límite y la ventaja es negativa, se aplica un gradiente negativo para decrementar la probabilidad de una determinada acción.
3. Cuando la relación es inferior al intervalo del límite y la ventaja es positiva (actualmente cierta acción es menos probable que en la política anterior pero genera más recompensas), se aplica un gradiente positivo para aumentar la probabilidad de una determinada acción.
4. Cuando la relación es inferior al intervalo del límite y la ventaja es negativa (actualmente cierta acción es menos probable que en la política anterior y genera menos recompensas), al ya tener la política actual una menor probabilidad de realizar una determinada acción, no es necesario decrementar aún más su probabilidad, por lo que el gradiente es 0.
5. Cuando la relación es superior al intervalo del límite y la ventaja es positiva (actualmente cierta acción es más probable que en la política anterior y genera más recompensas), al ya tener la política actual una mayor probabilidad de realizar una determinada acción, no es necesario aumentar aún más su probabilidad, por lo que el gradiente es 0.
6. Cuando la relación es superior al intervalo del límite y la ventaja es negativa (actualmente cierta acción es más probable que en la política anterior pero genera menos recompensas), se aplica un gradiente negativo para decrementar la probabilidad de una determinada acción.

En resumen, dentro del intervalo del límite (casos 1 y 2), el cambio de la política es pequeño así que es seguro realizar el gradiente según la ventaja. Cuando la relación está fuera del intervalo del límite: si la política actual se contradice con la ventaja (casos 3 y 6), se aplica el gradiente según la ventaja; si la política actual coincide con la ventaja (casos 4 y 5), no se aplica ningún gradiente ya que no es necesario modificarla.

Cabe destacar que PPO se puede utilizar junto a métodos actor-crítico. Cuando las arquitecturas de redes neuronales del actor (política) y el crítico (función de valor) comparten parámetros θ , como es en el caso de la implementación de PPO en ML-Agents, se utiliza una función de pérdida que combina la función objetivo sustituto con límite y un término de error de la función de valor. Para tratar de incentivar la exploración, también es posible añadir un bono de entropía, de manera que cuanta más incertezza haya en la distribución de probabilidades, mayor será el bono. Combinando estos términos, se obtiene la siguiente función objetivo, la cual se debe maximizar:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 H[\pi_\theta](s_t) \right] \quad (26)$$

donde c_1 y c_2 son coeficientes, $L_t^{VF}(\theta)$ es una función de pérdida del error cuadrático ($V_\theta(s_t) - V_t^{targ}$)², y H el bono de entropía.

Recompensas extrínsecas e intrínsecas

Las recompensas que puede recibir un agente están definidas, típicamente, en el propio entorno, y se asocian con alcanzar un cierto objetivo. En general, estas recompensas se conocen como recompensas extrínsecas desde el punto de vista de los algoritmos de aprendizaje.

Además de las recompensas extrínsecas, las recompensas también se pueden definir fuera del entorno, denominadas recompensas intrínsecas, y son aquellas que provienen el propio algoritmo de aprendizaje, para incentivar al agente a comportarse de una cierta manera. GAIL, por ejemplo, define una recompensa intrínseca que el agente obtiene dependiendo de la similitud entre el comportamiento del agente y las demostraciones proporcionadas. Durante el entrenamiento, el agente busca maximizar la combinación de recompensas extrínsecas e intrínsecas.

Curiosity

En la primera parte de este capítulo, en el estudio del aprendizaje por refuerzo, se asumía que cada interacción del agente con el entorno llevaba a conseguir una recompensa. Sin embargo, hay entornos en los que el agente raramente recibe recompensas, por lo que le es imposible saber si una acción ha sido buena o no. A este problema se le conoce como *reward-sparsity* (escasez de recompensas).

Cuando un agente no obtiene *feedback* sobre cómo está actuando, es muy probable que acabe moviéndose sobre un mismo conjunto de estados durante mucho tiempo hasta hallar, si hay suerte, un estado que le proporcione recompensas. En escenarios donde hay escasez de recompensas, el uso de recompensas intrínsecas puede ser de gran ayuda.

Curiosity [16] es un algoritmo que ayuda al agente a descubrir nuevos estados de un entorno por pura curiosidad, otorgándole una recompensa intrínseca cuando las recompensas extrínsecas son escasas o incluso inexistentes. Internamente, consiste en un Módulo de Curiosidad Intrínseca (ICM), mostrado en la Figura 10, el cual entrena dos modelos de redes neuronales profundas:

- El *Inverse Model*, cuyo fin es predecir la acción a_t que ha realizado el agente para moverse de un estado s_t al estado siguiente s_{t+1} , utilizando una red neuronal para aprender la función g , definida como

$$\hat{a}_t = g(s_t, s_{t+1}; \theta_I) \quad (27)$$

donde \hat{a}_t es la estimación prevista y θ_I son los parámetros de la red neuronal que se entranan para optimizar una función de pérdida L_I , la cual mide la discrepancia entre la acción estimada y la acción real:

$$\min_{\theta_I} L_I(\hat{a}_t, a_t) \quad (28)$$

Además, este modelo tiene un paso intermedio en el que se codifican los estados s_t y s_{t+1} en vectores de características $\phi(s_t)$ y $\phi(s_{t+1})$. De esta manera, al minimizar la discrepancia entre las acciones a_t y \hat{a}_t , se obtiene un submódulo capaz de extraer las características más relevantes de cada estado, es decir, aquellas características que realmente han llevado al agente a moverse de un estado s_t al estado siguiente s_{t+1} realizando la acción a_t .

- El *Forward Model*, cuyo fin es predecir la codificación del estado siguiente $\phi(s_{t+1})$ dada una acción a_t y la codificación de un estado actual $\phi(s_t)$, también utilizando una red neuronal para aprender la función f , definida como

$$\hat{\phi}(s_{t+1}) = f(\phi(s_t), a_t; \theta_F) \quad (29)$$

donde $\hat{\phi}(s_{t+1})$ es la estimación prevista de $\phi(s_{t+1})$ y θ_F los parámetros de la red neuronal que se optimizan minimizando la función de pérdida L_F , definida como la norma L^2 (norma euclídea) al cuadrado entre la codificación estimada y la codificación real:

$$L_F(\phi(s_{t+1}), \hat{\phi}(s_{t+1})) = \frac{1}{2} \|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2 \quad (30)$$

Optimizando ambos modelos conjuntamente, el *Inverse Model* aprende un espacio de características que codifica únicamente la información relevante para predecir las acciones del agente, y el *Forward Model* hace predicciones en este espacio de características.

Utilizando las funciones entrenadas g y f se genera una recompensa intrínseca basada en la curiosidad r_t^i , la cual se define también como la norma L^2 al cuadrado entre la codificación estimada y la codificación real:

$$r_t^i = \frac{\eta}{2} \|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2 \quad (31)$$

donde $\eta > 0$ es un factor de escala. De esta manera, cuanto menos se parezca la codificación del estado siguiente esperado $\hat{\phi}(s_{t+1})$ a la codificación del estado siguiente real $\phi(s_{t+1})$, mayor será la recompensa intrínseca. Gracias a que el espacio de características no tiene incentivos para codificar elementos del estado que no tienen influencia en las acciones del agente, el agente no recibe ninguna recompensa cuando se encuentra en estados que en su totalidad parecen distintos, pero esencialmente son iguales. Esto hace que la estrategia de exploración sea robusta frente a la presencia de ruido en el entorno, ya sea por cambios de iluminación, objetos distractores, etc.

El problema conjunto de optimización se compone por las funciones de pérdida de las Ecuaciones 30 y 28, las cuales se deben minimizar conjuntamente, junto al objetivo principal de maximización del retorno:

$$\min_{\theta_P, \theta_I, \theta_F} \left[-\lambda \mathbb{E}_{\pi(s_t; \theta_P)} \left[\sum_t r_t \right] + (1 - \beta)L_I + \beta L_F \right] \quad (32)$$

donde $0 \leq \beta \leq 1$ es un escalar que pondera la pérdida del modelo inverso frente a la pérdida del modelo directo, y $\lambda > 0$ es un escalar que pondera la importancia del gradiente de la política frente a la importancia de la recompensa intrínseca.

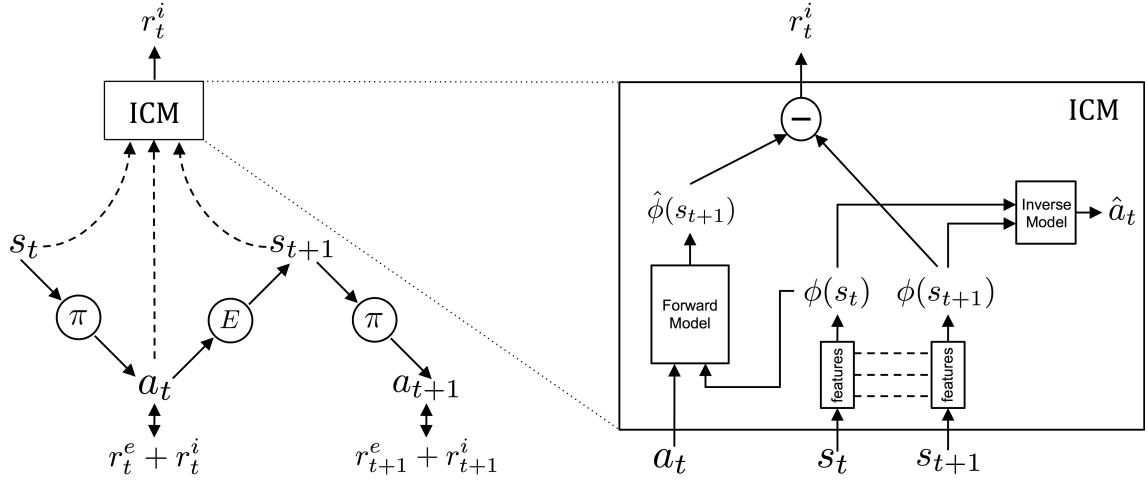


Figura 10: Módulo de Curiosidad Intrínseca (ICM). (Fuente: [16])

Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning (GAIL) [17] es un algoritmo que combina las ideas del aprendizaje por refuerzo inverso (IRL), uno de los enfoques del aprendizaje por imitación, y las redes generativas antagónicas (GAN). Debido a que tanto IRL como GAN son temas que están fuera del marco teórico de este TFG, en este apartado no se hace un estudio exhaustivo pero sí se da una intuición sobre GAIL.

En el aprendizaje por refuerzo tradicional, el objetivo es aprender una política para maximizar una función de recompensa predefinida en el entorno. En el aprendizaje por refuerzo inverso, se le da una vuelta a este problema y el objetivo pasa a ser definir una función de recompensa a partir de la observación del comportamiento de un agente.

El IRL puede ser útil en entornos en los que es difícil predefinir una función de recompensa, debido a la complejidad del entorno. Para ello, se introduce la idea de utilizar muestras de agentes expertos, denominadas demostraciones, las cuales se asumen que siguen un comportamiento óptimo en un determinado entorno. Utilizando estas muestras, el agente adopta el rol de aprendiz y trata de aprender una política lo más parecida posible a la del experto. Para guiar al agente aprendiz durante el proceso de entrenamiento, el modelo de IRL determina en qué situaciones debe recompensar al agente aprendiz, según el grado de similitud con el comportamiento del agente experto.

Uno de los enfoques de IRL consiste en utilizar GAN. En una arquitectura de GAN se entrena dos modelos simultáneamente: un modelo generativo G , que aprende una distribución de datos, y un modelo discriminador D , que predice si una muestra de entrada procede de conjunto de muestras reales, o bien del generador G . Mientras que el discriminador se entrena para hacer una distinción correcta entre si una muestra de entrada proviene del generador o del conjunto de muestras reales, el generador intenta engañar al discriminador haciéndole pensar que las muestras que genera son reales. Entrenando ambos modelos simultáneamente, el objetivo final es conseguir que el generador sea capaz de crear muestras cuya distribución de datos sea la misma que la de las muestras reales.

El uso de GAN en el contexto de IRL da lugar a GAIL. En GAIL, el discriminador debe aprender a distinguir si una observación/acción procede de una demostración o del generador, es decir, del agente aprendiz. De esta manera, durante el entrenamiento, mientras el agente va mejorando cada vez más, recibiendo recompensas intrínsecas según cuánto se acerque su comportamiento a las demostraciones expertas, el discriminador se va volviendo cada vez más estricto haciendo que el agente se deba esforzar más para engañarle.

Una de las ventajas de GAIL es que, en lugar de buscar replicar exactamente el comportamiento del agente experto, permite aprender una política que produzca acciones y estados similares a las demostraciones, es decir, una aproximación, requiriendo menos demostraciones para conseguir una política más genérica.

Self-play

En entornos en los que varios agentes compiten entre sí, entrenar una política correctamente no resulta fácil. Por una parte, es necesario tener un oponente bien entrenado para jugar contra el agente a ser entrenado. Por otra, cuando un oponente es demasiado bueno, es difícil que el agente mejore su política cuando está perdiendo constantemente y no tiene control sobre lo que está ocurriendo.

Una forma de hacer frente a este problema es utilizar oponentes con el mismo nivel que el agente, y que vayan mejorando simultáneamente a medida que compiten. En el contexto de RL, a este mecanismo se le llama *self-play* [18].

En self-play, la idea es que los agentes compartan copias de la misma política, ya que todos tratan de resolver el mismo problema. En un entorno de uno contra uno, uno de los agentes fija su política (agente a entrenar) mientras que el otro la va cambiando (agente oponente). A medida que entranan, el oponente va cambiando de política a copias más recientes de la política que está aprendiendo el agente a entrenar. De esta manera, la política va mejorando progresivamente a la vez que el oponente va incrementando su complejidad. Por supuesto, este mecanismo se puede extender a entornos de más agentes, por ejemplo de dos contra dos, mediante el uso de equipos.

8.4.4. Configuración del entrenamiento

Por último, en sección se recopila la configuración relevante del entrenamiento de agentes en ML-Agents, explicando el funcionamiento de los parámetros e hiperparámetros que se utilizan en los métodos de entrenamiento presentados en la sección anterior. Una explicación más detallada de todos los parámetros, incluidos aquellos que se omiten en esta sección, puede consultarse en la documentación de ML-Agents [19].

Configuración común

Ajuste	Descripción
trainer_type	Tipo de entrenamiento a utilizar: ppo, sac o poca.
summary_freq	Número de experiencias que deben ser procesadas antes de generar y mostrar estadísticas del entrenamiento.
max_steps	Número total de pasos que deben realizarse en el entorno antes de terminar el entrenamiento.
keep_checkpoints	Número de puntos de guardado del modelo que se deben guardar.
even_checkpoints	Indica si los puntos de guardado deben distribuirse uniformemente, dependiendo de los valores de keep_checkpoints y max_steps
hyperparameters/learning_rate	Factor de aprendizaje inicial utilizado en el descenso de gradiente.
hyperparameters/learning_rate_schedule	Determina cómo cambia el factor de aprendizaje con el tiempo. Para PPO se recomienda una decadencia lineal, para una convergencia más estable. Los valores posibles son linear o constant
hyperparameters/batch_size	Número de experiencias utilizadas en cada iteración del descenso de gradiente.
hyperparameters/buffer_size	En PPO, corresponde al número de experiencias que deben ser guardadas antes de actualizar la política.
network_settings/num_layers	Número de <i>hidden layers</i> de la red neuronal, tanto para la red neuronal del actor como la del crítico.
network_settings/hidden_units	Número de neuronas de cada <i>hidden layer</i> , tanto para la red neuronal del actor como la del crítico.

Tabla 11: Configuración común de todos los tipos de entrenamiento (PPO, SAC y POCA).
(Fuente: [19])

Configuración específica de PPO

Ajuste	Descripción
hyperparameters/beta	Factor de escala del bono de entropía.
hyperparameters/epsilon	Determina el límite de divergencia entre la política actual y la anterior durante el cálculo de descenso de gradiente.
hyperparameters/beta_schedule	Determina cómo cambia beta con el tiempo.
hyperparameters/epsilon_schedule	Determina cómo cambia epsilon con el tiempo.
hyperparameters/num_epoch	Número de veces que se recorre el búfer de experiencias al realizar la optimización mediante gradiente descendiente.

Tabla 12: Configuración específica de PPO. (Fuente: [19])

Recompensas extrínsecas

Ajuste	Descripción
extrinsic/strength	Factor de escala de las recompensas extrínsecas.
extrinsic/gamma	Factor de descuento de las recompensas futuras procedentes del entorno.

Tabla 13: Configuración de las recompensas extrínsecas. (Fuente: [19])

Recompensas intrínsecas de Curiosity

Ajuste	Descripción
curiosity/strength	Factor de escala de las recompensas generadas por el módulo de curiosidad intrínseca.
curiosity/gamma	Factor de descuento de las recompensas futuras.
curiosity/network_settings	Especificaciones de la red neuronal utilizada por el modelo de curiosidad intrínseca.
curiosity/learning_rate	Factor de aprendizaje utilizado para actualizar el módulo de curiosidad intrínseca.

Tabla 14: Configuración de las recompensas intrínsecas procedentes de Curiosity. (Fuente: [19])

Recompensas intrínsecas de GAIL

Ajuste	Descripción
gail/strength	Factor de escala de las recompensas de GAIL.
gail/gamma	Factor de descuento de las recompensas futuras.
gail/demo_path	Ruta del directorio de los archivos .demo.
gail/network_settings	Especificaciones de la red neuronal del discriminador de GAIL.
gail/learning_rate	Factor de aprendizaje utilizado para actualizar el discriminador.
gail/use_actions	Determina si el discriminador debe predecir basándose en acciones y observaciones o únicamente observaciones.
gail/use_vail	Habilita el <i>variational bottleneck</i> del discriminador de GAIL. Esta opción fuerza al discriminador a aprender una representación más genérica y reducir la tendencia a ser “demasiado bueno” prediciendo.

Tabla 15: Configuración de las recompensas intrínsecas procedentes de GAIL. (Fuente: [19])

Configuración de Self-Play

Ajuste	Descripción
save_steps	Número de pasos entre los que se hacen copias de la política que se está entrenando. Un valor grande de save_steps permite cubrir un rango de niveles de habilidad más amplio.
team_change	Número de pasos entre cambios del equipo que aprende. Cuanto más grande es el valor de team_change, más capaces se vuelven los agentes del equipo que aprende de vencer a los oponentes. Sin embargo, entrenar contra los mismos oponentes durante demasiado tiempo puede causar <i>overfitting</i> contra las estrategias de esos oponentes en particular.
swap_steps	Número de <i>pasos fantasma</i> entre cambios de la política del oponente. Un <i>paso fantasma</i> es un paso dado por un agente con la política fija (que no está aprendiendo), relevante para entornos asimétricos.
play_against_latest_model_ratio	Probabilidad con la que el agente se enfrentará contra la copia más reciente de la política. Con una probabilidad de $1 - \text{play_against_latest_model_ratio}$, el agente se enfrentará contra una de las copias pasadas, seleccionada aleatoriamente.
window	Tamaño de la ventana deslizante de copias pasadas de la política, desde donde se obtiene la política del oponente. Cada vez que se hace una copia nueva, se descarta la más antigua. Un valor grande implica guardar una mayor diversidad de comportamientos del agente. Aprender una política capaz de enfrentarse a una diversidad de oponentes es más difícil, pero a su vez permite obtener una política más robusta.

Tabla 16: Configuración de self-play. (Fuente: [19])

9. Desarrollo

En este capítulo se dan detalles sobre el proceso de desarrollo del proyecto, el cual se ha dividido en dos fases. La primera fase ha consistido en hacer un experimento piloto, en el que se ha puesto a prueba una propuesta inicial del entorno de pádel para estudiar su viabilidad. En la segunda fase se ha implementado la versión definitiva de este entorno, la cual ha derivado de la resolución de los problemas surgidos durante la primera fase.

9.1. Experimento piloto

9.1.1. Reglamento del entorno virtual

Para diseñar el entorno virtual de pádel, el primer paso fue definir una normativa lo más parecida posible al entorno real:

- Un partido de pádel se divide normalmente en tres sets. En el entorno virtual, al tratarse de una simulación, consiste en un único set que se juega indefinidamente, es decir, se ignora la puntuación de juegos.
- A diferencia de un partido real, en el entorno virtual no se permite jugar fuera de la pista, por lo que si la pelota se sale fuera se considera falta.
- Cada juego se forma con puntos: el primer punto que gana una pareja suma 15 puntos, el segundo 30, el tercero 40 y con el cuarto se gana un juego. Cuando se produce un empate a 40 puntos, gana el juego la pareja que sume dos puntos de ventaja.
- Durante un juego el servicio lo realiza siempre el mismo jugador. El servicio empieza en el lateral derecho de la pista, y se va alternando de lado tras cada punto.
- Tras un juego se produce una rotación en el servicio. Dado los jugadores A, B, C y D, formando las parejas AB y CD, el orden de saque sería A → C → B → D → A.
- En el servicio, el jugador debe botar la pelota detrás de la línea de saque y enviar la pelota de manera que el primer rebote sea en el lado contrario del campo rival, sin sobrepasar la línea de saque del rival. Tras el primer rebote en el suelo, la pelota no puede tocar la malla. No cumplir estas condiciones se considera falta.
- A la hora de devolver la pelota, se permite golpear la pelota contra la pared del propio campo. Tras un golpeo contra la pared o un golpeo directo, si la pelota rebota por primera vez en el propio campo o en la pared se considera falta.
- Tras devolver la pelota, esta puede rebotar como máximo una vez en el suelo. El número de rebotes en la pared es indiferente. Si una pelota rebota dos veces en el suelo, el equipo que haya enviado esa pelota gana un punto.
- Tras devolver la pelota, si la pelota impacta en el compañero o la red es punto para el rival. Si la pelota impacta en alguno de los dos oponentes, es punto para el propio equipo.

9.1.2. Diseño inicial

Una vez establecidas las reglas, lo siguiente fue diseñar la pista de pádel en Unity, partiendo del diseño de una pista oficial de World Padel Tour. Al no tratarse de un TFG enfocado en gráficos, bastaba con un diseño simple pero funcional. En la Figura 11 se muestra la pista de pádel del entorno virtual, la cual sigue las medidas oficiales de una pista de pádel, y tiene paredes compuestas por materiales que representan vidrio templado (en blanco transparente) y mallas metálicas (en negro transparente), además de una red en el centro. Las paredes tienen un *Physic Material* con la propiedad *Dynamic Friction* = 0.5, para que la pelota pierda velocidad con cada rebote.

El origen de coordenadas local (0,0) para un entorno se sitúa en el centro de la pista, de manera que el rango de la pista es $x \in [-5, 5]$ en el eje X, y $z \in [-10, 10]$ en el eje Z.

En cuanto a pelota, la idea original era utilizar medidas reales, pero finalmente se optó por utilizar medidas más grandes por cuestiones de visibilidad. La pelota tiene un *Physic Material* con *Bounciness* = 0.77, de manera que tras caer desde una altura de 2.5 metros, la pelota rebota y alcanza una altura de 1.5 metros. También se le ha atribuido un *Trail Renderer* utilizado para visualizar su trayectoria y saber qué equipo ha sido el último en golpearla.

Los jugadores, sobre los cuales se implementan los agentes, tienen forma de cápsula y tienen el color correspondiente a su equipo: naranja para el equipo denominado T1, y morado para el equipo denominado T2. Para poder diferenciar entre los compañeros del mismo equipo, se les ha añadido una cinta blanca para el Jugador 1 y una cinta negra para el Jugador 2. Los agentes tienen un alcance de 2 metros de radio para golpear la pelota.

Además, a cada *Game Object* se le han añadido los componentes necesarios para simular las físicas mediante el motor de físicas de Unity (*Colliders* y *Rigid Bodies*).

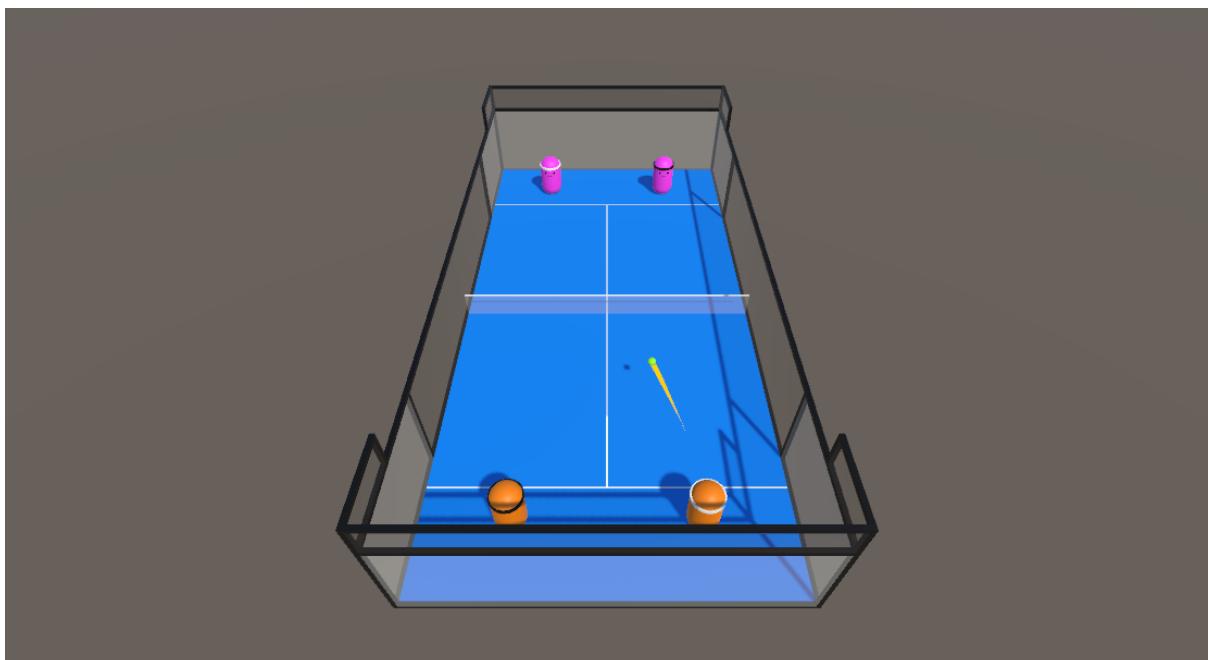


Figura 11: Entorno virtual de pádel en Unity. (Elaboración propia)

9.1.3. Implementación inicial

La implementación de la lógica detrás de los elementos definidos consiste principalmente en tres *Scripts*, uno para controlar el entorno, otro para la pelota y otro para definir el agente vinculado a cada jugador. En esta sección se da una idea general sobre cómo funcionan estos componentes, ya que se trata de una implementación inicial que acabó pasando por muchos cambios.

Controlador del entorno

El controlador del entorno se encarga de la lógica detrás del flujo de un partido de pádel: puntuaciones, servicio y rotaciones de los jugadores, cumpliendo con las reglas establecidas anteriormente. En la Figura 12 se muestra un diagrama de flujo simplificado del tránscurso de un partido de pádel en el entorno virtual.

Se optó por implementar el servicio a mano, utilizando valores fijos para la posición y la fuerza de golpeo, ya que aprender estrategias para el saque no es muy relevante para este proyecto.

Controlador de la pelota

El controlador de la pelota gestiona el estado de la pelota: qué equipo ha golpeado, cuántas veces ha rebotado, dónde ha rebotado, además de definir métodos para golpear la pelota. Este controlador se comunica con el controlador del entorno para incrementar la puntuación de un equipo, ya sea debido a una falta o el doble rebote en el suelo de la pelota.

Agente de pádel

El agente, el cual controla el jugador con el que está vinculado, tiene definidos los elementos que requiere ML-Agents para utilizar algoritmos de aprendizaje por refuerzo. Estos elementos son el espacio de observaciones, el espacio de acciones, las recompensas y el principio y fin de un episodio.

El espacio de observaciones de un agente consiste en un total de 14 parámetros:

- Posición local 2D del propio agente, su compañero y los dos oponentes.
- Posición local y velocidad 3D de la pelota.

El espacio de acciones está formado por:

- 3 acciones continuas, utilizadas para controlar la fuerza de golpeo.
- 4 acciones discretas, utilizadas para el tipo de golpeo, el movimiento lateral, el movimiento frontal y el giro. En ML-Agents, una rama consiste.

Las recompensas definidas son +1 por cada pelota devuelta y -1 por cada punto que se pierde. El comportamiento que se quería buscar con esta configuración era que los agentes pudieran mantener la pelota en juego.

En el entorno virtual, un episodio consiste en un punto de un juego: empieza con el servicio y termina con la obtención de un punto para cualquier equipo.

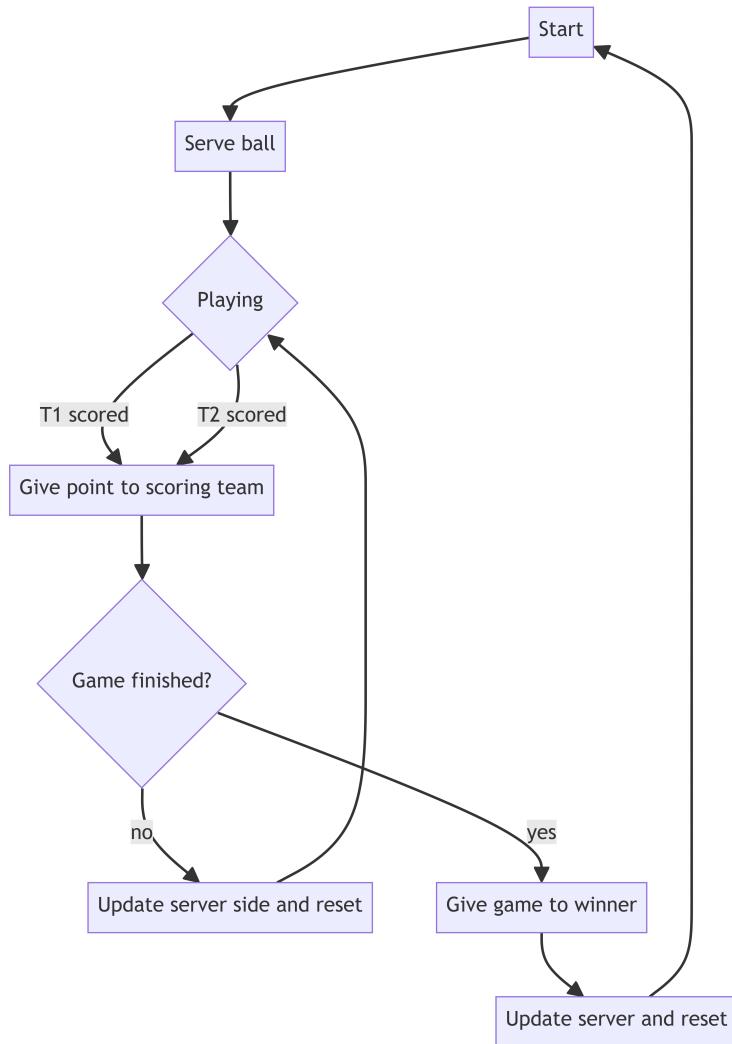


Figura 12: Diagrama de flujo simplificado del entorno virtual de pádel. (Elaboración propia)

9.1.4. Problemas de la versión inicial

Al principio de esta memoria, uno de los objetivos principales que se definió para este proyecto era la simplicidad en la implementación de la lógica detrás de un partido de pádel. Por lo tanto, la filosofía que se siguió durante los primeros experimentos fue “cuanto más simple, mejor”.

En un modelo ideal, los agentes deberían ser capaces de aprender a golpear la pelota con la fuerza necesaria para enviarla a la posición que maximice la probabilidad de ganar un punto. En una primera prueba, en la cual los agentes eran libres de emplear una fuerza cualquiera, siempre y cuando estuviera dentro de un rango válido, por muchas horas de entrenamiento que se hacían, los agentes no eran siquiera capaces de enviar la pelota al campo rival.

Tras esta primera prueba, se hizo evidente que era necesario hacer cambios en el comportamiento del agente, simplificando el espacio de acciones. Esta vez, la idea era hacer que el agente aprendiera a modificar fuerzas de golpeo predefinidas, según el tipo de golpeo (globo, derecha, remate...), mediante acciones continuas, como si añadieran un cierto margen de error en la fuerza. Aunque los resultados fueron positivos, el comportamiento de los agentes no era el deseado, ya que en realidad los agentes no podían elegir voluntariamente hacia dónde enviar la pelota, sino que debían partir siempre de una fuerza fija, resultando en un entorno muy poco realista.

Llegados hasta ese punto, la propuesta final fue deshacerse de las acciones continuas, ya que estaba claro que era más sencillo elegir sobre un rango discreto que no sobre uno continuo, por lo que este cambio facilitaría bastante el aprendizaje. Pero, ¿cómo se debería modelar el espacio de acciones, limitándose a acciones discretas, para conseguir un comportamiento realista? La solución más directa fue dividir la pista en posiciones equidistantes, de manera que el agente pudiera elegir hacia qué posición enviar la pelota, y dejar el cálculo de la fuerza al propio entorno. Esto se asemeja al comportamiento de un jugador de pádel: primero decide el tipo de acción a realizar, que incluye tipo de golpe y dirección del mismo (aspectos tácticos), y luego realiza los movimientos necesarios para ejecutar esa decisión (aspectos técnicos).

Otra cuestión importante a resolver era el posicionamiento de los agentes. En la primera versión del entorno, las recompensas definidas eran demasiado escasas. Esto hacía que los agentes, al no recibir ninguna retroalimentación entre devoluciones de la pelota, se movieran hacia la pelota aunque el compañero ya se encontrara cerca, y mantenían muy poca distancia entre sí, dejando demasiados espacios libres en la pista. El comportamiento ideal hubiera sido que los jugadores cubrieran posiciones óptimas dada cada situación de un partido de pádel. Por este motivo, de cara a la versión definitiva, era necesario cambiar la función de recompensas para poder guiar a los agentes durante los momentos en los que no disponen de una valoración de cómo están actuando.

Al requerir más funcionalidades, era inviable mantener toda la lógica necesaria en los tres *scripts* definidos. Debido a esto, de cara a la versión definitiva, otro objetivo que se planteó fue repartir cada funcionalidad en un Script específico. Por ejemplo, el control de la puntuación, el servicio y la rotación de los jugadores recaía únicamente en el controlador del entorno. Tras el rediseño, tendría que haber un Script para cada funcionalidad, y se utilizaría el controlador del entorno como intermediario entre estos componentes.

9.2. Versión final del entorno virtual

9.2.1. Diseño final

Tras los problemas de la versión inicial, los cambios que se hicieron fueron los siguientes:

- El primer cambio consistió en reestructurar la implementación de los controladores ya existentes. El controlador del entorno debía servir como mediador entre el resto de controladores. A partir de ahí, cada funcionalidad debía tener su propio controlador: control de la pelota, el servicio y la puntuación.
- La funcionalidad principal que se añadió fue el cálculo de fuerzas para enviar la pelota hacia una posición en concreto. Para discretizar estas posiciones, se optó por dividir el campo de cada equipo en una cuadrícula de tamaño 5×5 .

- Recordando que el origen de coordenadas local (0,0) de una pista se sitúa en el centro de la pista, y al ser el pádel un juego simétrico, es posible simplificar el espacio de observaciones invirtiendo los ejes X y Z de los agentes de uno de los dos equipos. De esta manera, la política tendría que aprender a jugar únicamente en uno de los dos campos. Esto es válido ya que la decisión óptima en un campo debería ser la misma en el otro, cuando el estado es equivalente. Por ello, el espacio de observaciones se modificó de manera que los agentes del equipo T2 perciben ahora las posiciones con los ejes invertidos.
- En cuanto al espacio de acciones de los agentes, en la primera versión se utilizaba un Rigidbody y el movimiento se basaba en aplicarle fuerzas y rotaciones. Esto no parecía funcionar muy bien, por lo que el Rigid Body se sustituyó por un Character Controller, y las acciones consisten ahora en moverse hacia delante, detrás, izquierda o derecha.
- Para evitar la escasez de recompensas, se han añadido unas guías denominadas “posiciones clave”. Estas posiciones consisten en posiciones de interés que recompasan a los agentes cuando están cerca o bien se están acercando a la posición en cuestión.
- Las posiciones clave introducen la necesidad de asignar roles a cada agente, ya que la función de recompensa cambia según la situación del partido. Por ejemplo, en un mismo campo, las posiciones de interés cambian según si los agentes deben devolver una pelota, o si están esperando a que los oponentes la devuelvan.
- Cuantas más funcionalidades tiene una aplicación, más probabilidades hay de que surjan errores durante el desarrollo. Por ello, también se añadieron herramientas para realizar una depuración de una manera cómoda.

El diseño del entorno se mantuvo igual que en la versión inicial, pero se añadieron todos los elementos visuales necesarios para crear un modo de depuración, mostrado en la Figura 13. Esto incluye el uso de textos para mostrar la puntuación y la información de los agentes, los roles y la pelota; Prefabs para materializar los puntos clave; marcas en los jugadores que indican los roles que tienen asignados; y un Line Rendererer que permite simular la trayectoria de la pelota.

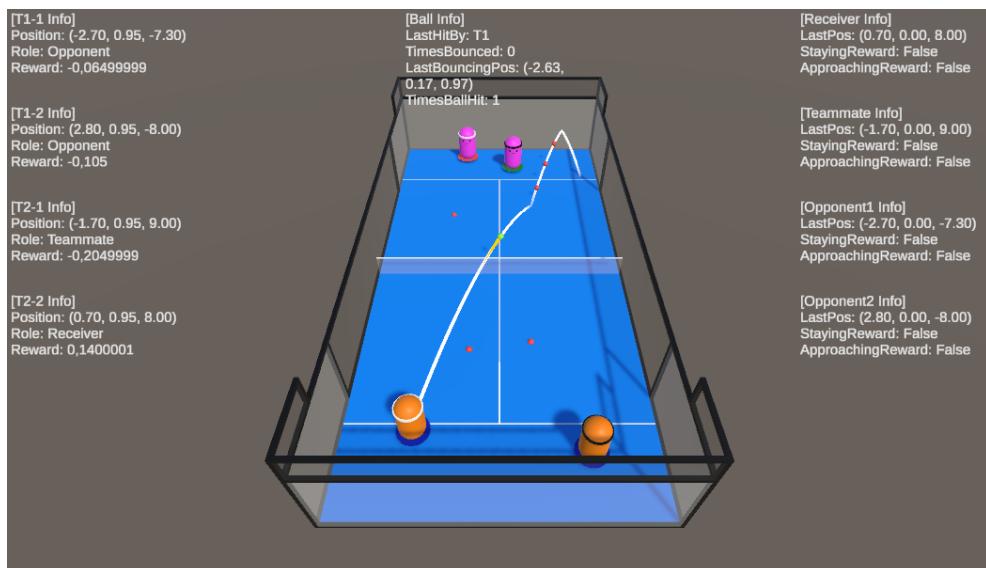


Figura 13: Entorno virtual de pádel con el modo de depuración activado. (Elaboración propia)

9.2.2. Implementación final

En la implementación final se ven reflejados los cambios mencionados previamente. Además de los controladores ya existentes (del entorno, de la pelota, del servicio y de la puntuación), se han añadido más controladores para la implementación de las nuevas funcionalidades. El código fuente de este proyecto está disponible en Github [20].

Controlador del entorno

Como ya se ha mencionado previamente, el controlador del entorno hace de mediador entre el resto de controladores, de manera que cuando un controlador necesita información de otro controlador, debe enviar una solicitud al controlador del entorno para que se encargue de resolverla. Utilizando este patrón de diseño se eliminan las dependencias entre controladores, haciendo que la funcionalidad de cada controlador esté aislada y el mantenimiento sea más sencillo.

También se definen enumeraciones y variables públicas que comparten varios elementos del entorno. Las enumeraciones definidas son:

- `PlayerId`, para dar un identificador a cada jugador.
- `Side`, para representar el lado derecho e izquierdo.
- `Team`, para especificar a qué equipo pertenece cada jugador.
- `Role`, para asignar al agente un rol según sea `Receiver`, `Teammate` o `Opponent`.

Las variables por otra parte, se utilizan para definir las recompensas del entorno:

- `WinningReward` es la recompensa por ganar un punto.
- `LosingReward` es la recompensa por perder un punto.
- `ApproachingKeyPositionReward` es la recompensa por acercarse a una posición de interés.
- `StayingAroundKeyPositionReward` es la recompensa por quedarse cerca de una posición de interés.
- `HittingBallReward` es la recompensa por darle a la pelota.

Otras variables que tienen un uso más específico son `DebugMode`, utilizado para activar o desactivar el modo depuración mediante la tecla asignada; `EnvironmentId`, para asignar un identificador al entorno, ya que se utilizan varios a la vez para el entrenamiento; y `SimulationCompleted`, para saber si la simulación de la trayectoria de la pelota ha finalizado.

El resto de variables y métodos definidos son aquellos relacionados con los controladores, necesarios para delegar las solicitudes recibidas al controlador correspondiente.

Controlador del servicio

El controlador del servicio se encarga de mantener un control sobre el estado del servicio en el comienzo de cada punto. La principal funcionalidad que tiene consiste en hacer botar la pelota y acto seguido realizar el saque.

Durante el servicio, el primer paso es botar la pelota aplicándole una fuerza vertical negativa. Una vez la pelota haya alcanzado una altura superior a 1 metro tras el rebote, el controlador procede a enviar una solicitud al controlador de la pelota para enviar la pelota a una determinada posición de la cuadrícula de la pista, dependiendo de qué equipo realiza el servicio y desde qué lado.

Como en esta versión aprender a realizar el servicio no es parte del objetivo de aprendizaje, se trata de un proceso determinista en el que, al principio de cada punto, todos los agentes se sitúan en las mismas posiciones, sin moverse, y la pelota se coloca dependiendo de desde dónde se hace el servicio, pero siempre se sitúa a la misma distancia del servidor.

Tras haber realizado el saque, los agentes ya pueden moverse libremente y empieza el punto, el cual se controla a través del controlador de la pelota. Cada vez que finaliza un punto, se realiza la rotación del servicio actualizando las variables pertinentes y se repite el proceso.

Controlador de la puntuación

El controlador de la puntuación gestiona los puntos conseguidos por los equipos T1 y T2. Cuando un equipo gana un punto, suma los puntos correspondientes según la puntuación actual (siguiendo el orden 0, 15, 30, 40). Cuando un equipo gana un punto teniendo una puntuación de 40, si el equipo rival no ha llegado a 40 puntos, gana el juego. En el caso de que el equipo rival también tenga 40 puntos, si el equipo rival había ganado un punto de ventaja tras los iguales, ambos equipos vuelven a estar iguales. Un equipo gana un punto de ventaja cuando el equipo rival no tiene ninguno, y gana el juego tras dos puntos de ventaja seguidos.

Cuando termina un punto, el controlador de la puntuación envía una solicitud al controlador del entorno para cambiar el lado del servicio (izquierda o derecha del equipo correspondiente), y seguidamente otra solicitud para reiniciar el entorno. Cuando termina un juego, envía una solicitud al controlador del entorno para efectuar la rotación del servicio, el cual empieza desde el lado derecho del equipo correspondiente y siguiendo el orden establecido en el reglamento, además de la solicitud para reiniciar el entorno.

Controlador de los agentes

El controlador de los agentes gestiona los cuatro jugadores del entorno, implementando las funcionalidades necesarias para colocar a los jugadores en la posición correspondiente, asignar recompensas a nivel grupal (por equipos), y aplicar cambios al estado de los agentes a nivel global, para permitir el movimiento o para terminar un episodio.

Las posiciones de los agentes se alternan dependiendo del lado del servicio (izquierda o derecha) y el jugador que realiza el servicio. Las posiciones locales entre las que se alternan son fijas, mostradas en la Figura 14.

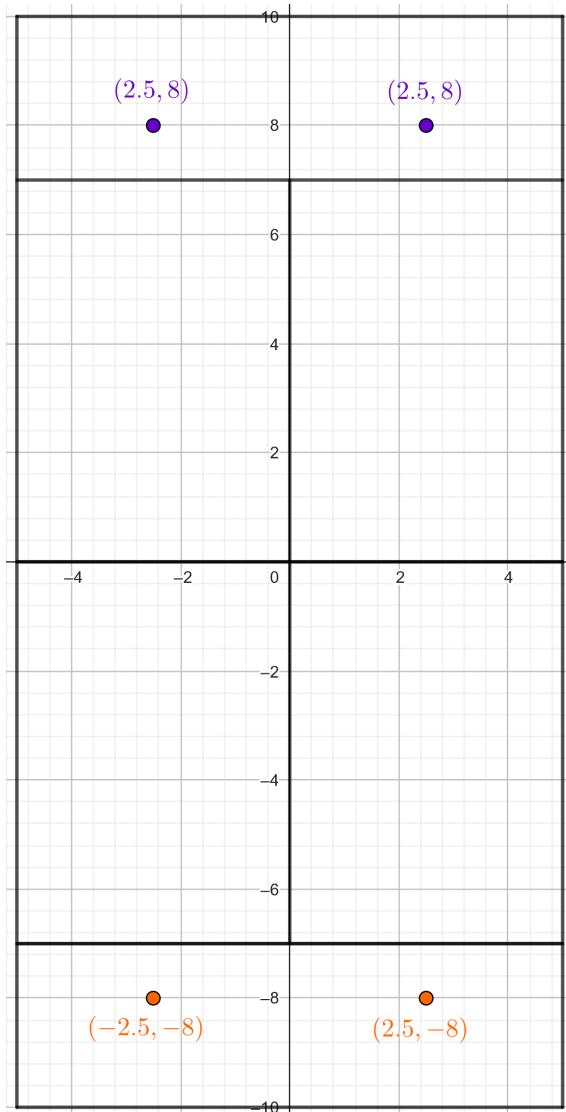


Figura 14: Posiciones de servicio del entorno virtual de pádel. (Elaboración propia)

Controlador de la pelota

En el controlador de la pelota se han implementado todas aquellas funcionalidades relacionadas con la pelota, como el golpeo de la pelota, ya sea para saques o devoluciones, o el control sobre impactos y rebotes, tanto en el suelo como en la pared, entre otros.

Para golpear la pelota se calcula el cambio de velocidad necesario para que impacte en una determinada posición, dada una posición inicial, una altura máxima y el equipo del jugador que pretende golpearla. La posición de impacto es una selección de una de las posiciones de la cuadrícula de posiciones la pista. Esta cuadrícula tiene una resolución de 5×5 , dejando una separación equidistante de $\frac{10}{6}$ metros entre cada posición, tal y como se muestra en la Figura 15.

×	×	×	×	×	×	×	×	×
×	×	×	×	×	×	×	×	×
×	*	*	*	*	*	*	*	*
×	*	*	*	*	*	*	*	*
×	*	*	*	*	*	*	*	*

Figura 15: Cuadrícula de la pista de padel. Cada cruz representa una posición de impacto, equidistantes entre sí a una distancia de $\frac{10}{6}$ metros. (Elaboración propia)

Desde el punto de vista de cada equipo, la posición $(0, 0)$ de la cuadrícula corresponde a la esquina superior izquierda del campo contrario, la posición $(4, 0)$ a la esquina superior derecha, la posición $(0, 4)$ a la esquina inferior izquierda, cerca de la red, y la posición $(4, 4)$ a la esquina inferior derecha, también cerca de la red. Esto permite reducir la complejidad del aprendizaje, aprovechando una vez más la simetría del pádel. Por ejemplo, para la misma colocación de los oponentes desde el punto de vista de un jugador, la posición óptima de impacto debería ser la misma independientemente de si es de un equipo u otro. Por supuesto, esto es válido siempre y cuando las observaciones de los agentes también estén invertidas.

Para el cálculo de la fuerza se utilizan las ecuaciones de movimiento parabólico:

$$x = x_0 + v_x t \quad (33)$$

$$y = y_0 + v_{y_0} t + \frac{1}{2} a t^2 \quad (34)$$

$$v_x = v_{x_0} \quad (35)$$

$$v_y = v_{y_0} + a t \quad (36)$$

donde x y v_x son las ecuaciones de movimiento rectilíneo uniforme (MRU) en el eje X, e y y v_y las ecuaciones de movimiento rectilíneo uniformemente acelerado en el eje Y. Nótese que para el eje Z también se utilizan las ecuaciones de MRU, pero se omite por simplificación al ser equivalentes a las del eje X.

El uso de las ecuaciones de movimiento parabólico se trata de una simplificación respecto al comportamiento real, ya que se ignora la fricción de la pelota con el aire, así como los diferentes efectos que pueden acompañar al golpeo (cortado, liftado, spin lateral).

Los datos disponibles son la posición inicial (x_0, y_0, z_0) , la posición final $(x_f, 0, z_f)$ (es decir, el objetivo para la pelota que ha elegido el agente), la aceleración $a = -9,8 \text{ m/s}^2$ y la altura máxima y_{max} . Bajo estas condiciones, es posible calcular las velocidades iniciales $(v_{x_0}, v_{y_0}, v_{z_0})$ desarrollando las ecuaciones de movimiento parabólico.

Dado que cuando la altura es máxima, la velocidad $v_y = 0$, y por lo tanto $t = -\frac{v_{y_0}}{a}$:

$$\begin{aligned}
 y_{max} &= y_0 + v_{y_0}t + \frac{1}{2}at^2 \\
 &= y_0 + v_{y_0} \left(-\frac{v_{y_0}}{a} \right) + \frac{1}{2}a \left(-\frac{v_{y_0}}{a} \right)^2 \\
 &= y_0 - \frac{v_{y_0}^2}{2a} \\
 v_{y_0} &= \sqrt{(y_0 - y_{max})2a}
 \end{aligned} \tag{37}$$

Para calcular las velocidades iniciales en los ejes X y Z, es necesario calcular el tiempo total desde el instante en el que se golpea la pelota hasta que cae al suelo. Este tiempo se puede calcular sustituyendo v_{y_0} , obtenido a partir de la Ecuación 37, en la Ecuación 34. Resolviendo la ecuación de segundo grado para $t > 0$, se obtiene el tiempo necesario para calcular v_{x_0} y v_{z_0} :

$$v_{x_0} = \frac{x_f - x_0}{t} \tag{38}$$

$$v_{z_0} = \frac{z_f - z_0}{t} \tag{39}$$

En cuanto a las colisiones de la pelota, las cuales engloban el rebote y el impacto en otros objetos del entorno, se gestionan de la siguiente manera:

- Cuando la pelota impacta en la red, si ha sido un impacto directo, el equipo que haya golpeado la pelota pierde el punto. Si ha sido un impacto después de que la pelota haya botado en el suelo del campo contrario, entonces este equipo gana el punto.
- Cuando la pelota colisiona con otros jugadores tras la devolución de un cierto jugador, si la pelota impacta en un jugador del mismo equipo (ya sea a sí mismo o a su compañero), su equipo pierde el punto. Si impacta en un jugador del equipo contrario, se considera que los oponentes no han sido capaces de devolverla, por lo que su equipo gana el punto.
- Para los rebotes en la pared tras la devolución de un cierto jugador, si ha sido un impacto directo en la pared del campo contrario, su equipo pierde el punto. Si durante el servicio toca la malla tras el primer rebote en el suelo, también pierde el punto.
- Para los rebotes en el suelo, si la pelota ha rebotado por segunda vez sin haberse cometido ninguna falta, gana el punto el equipo que la haya golpeado. Si se trata del primer rebote, si es durante el servicio y la pelota bota fuera del área correspondiente, el equipo que haya realizado el servicio pierde el punto. Si no es durante el servicio, un equipo pierde el punto cuando golpea la pelota pero esta rebota en el campo propio.

Cada vez que se gana un punto, el controlador de la pelota reinicia el estado de la pelota y solicita al controlador del entorno que sume un punto al equipo correspondiente. Es entonces cuando el controlador del entorno hace las llamadas pertinentes para reiniciar todo el entorno.

Otro aspecto a tener en cuenta es la sincronización de la pelota con el entorno, ya que muchas de las funciones se ejecutan de forma concurrente, en respuesta a eventos. En ocasiones, tras la devolución de la pelota, mientras se calcula predicción de la trayectoria de la pelota para el modo depuración, es posible que un jugador del equipo contrario devuelva la pelota antes de que haya finalizado el cálculo, creando incoherencias en las fuerzas aplicadas. Para evitar esto, cuando empieza el cálculo de la predicción se bloquea la pelota durante un margen de 0.1 segundos.

Otra situación problemática se da cuando, durante el reinicio del entorno, un jugador se encuentra justamente en la posición donde se debe colocar la pelota, haciendo que se juegue un punto indeseado. Para impedir que pase esto, tras finalizar un punto, la pelota se coloca en una posición fuera de la pista y se mantiene ahí durante un margen de 0.1 segundos, para que todos los jugadores puedan recolocarse correctamente, y entonces colocar la pelota en la posición correspondiente.

El resto de funcionalidades sirven para gestionar el estado de la pelota, por ejemplo para reiniciar el estado de la pelota, cambiar el color del Trail Renderer tras cada golpeo, colocar la pelota delante del servidor, etc.

Controlador de la trayectoria

El controlador de la trayectoria se utiliza para simular la trayectoria de la pelota tras cada golpeo. Este controlador crea una escena de físicas de Unity (esto es, una escena de Unity que utiliza únicamente el motor físico de Unity y permite hacer simulaciones de interacciones físicas entre objetos) para cada entorno, replicando los Colliders de la pista del entorno correspondiente, y utiliza una réplica de la pelota para simular su trayectoria con un número determinado de frames de antelación.

En cada simulación, se instancia una réplica de la pelota, denominada GhostBall, en una determinada posición inicial, y se le aplica un determinado cambio de velocidad. La escena de físicas simula un frame cada vez que se hace una llamada al método PhysicsScene.Simulate(float step), dado un cierto intervalo de tiempo, normalmente definido por un intervalo de tiempo fijo Time.fixedDeltaTime. La idea es, por lo tanto, hacer llamadas reiterativamente hasta un número máximo de frames, o bien hasta cuando la pelota haya llegado a un estado terminal (es decir, que haya rebotado dos veces en el suelo). Esto permite simplificar la predicción de la trayectoria de la pelota, eliminando la necesidad de calcular los cambios de velocidad tras los rebotes.

Durante la simulación, las posiciones de la réplica de la pelota en cada frame de la escena de físicas se envía al controlador de posiciones clave, ya que algunas posiciones de interés que se han definido tienen relación con la trayectoria de la pelota. Concretamente para el modo de depuración, cuando está activado, estas mismas posiciones se utilizan también para determinar los puntos que conectan un LineRenderer, permitiendo de esta manera visualizar la trayectoria de la pelota. El LineRenderer se elimina cada vez que empieza una simulación nueva o bien cuando se desactiva el modo de depuración.

Controlador de las posiciones clave

El controlador de las posiciones clave define todas las posiciones de interés del entorno, las cuales pueden ser de devolución o de cobertura, como se verá más adelante. Estas posiciones permiten asignar recompensas a los agentes cuando se encuentran cerca o están acercándose a las posiciones en cuestión. Utilizando estas recompensas durante el entrenamiento, el objetivo es que los agentes puedan aprender a posicionarse correctamente durante un partido de pádel siguiendo un criterio definido manualmente.

Las posiciones de interés de devolución son relevantes para el rol *Receiver*, y se definen como aquellas posiciones por las que pasará la pelota y son alcanzables por uno de los jugadores del equipo correspondiente, antes de que la pelota sobrepase esa posición. Al haber muchas posiciones que cumplen este criterio, se ha decidido limitar el número de posiciones a un máximo de cinco posiciones.

Utilizando la misma distancia equidistante de la cuadrícula de posiciones, de $\frac{10}{6}$ metros, se divide un campo en intervalos en los que puede haber como mucho una única posición de interés. Cuando se envía la pelota del campo de T1 hasta el de T2, por ejemplo, dado un índice $0 \leq i < 5$, los intervalos se calculan como $z \in [(i + 1)\frac{10}{6}, (i + 2)\frac{10}{6}]$, empezando el primer intervalo a $\frac{10}{6}$ metros después la red, y terminando el último intervalo justo en la pared de la pista, a 10 metros después de la red. Si es una pelota que se envía del campo de T2 hasta el de T1, se hace exactamente lo mismo pero invirtiendo el eje Z. La posición de interés que se asigna a un intervalo es la primera que cumpla el criterio establecido, siguiendo el orden de la trayectoria, de más próximo a más lejano.

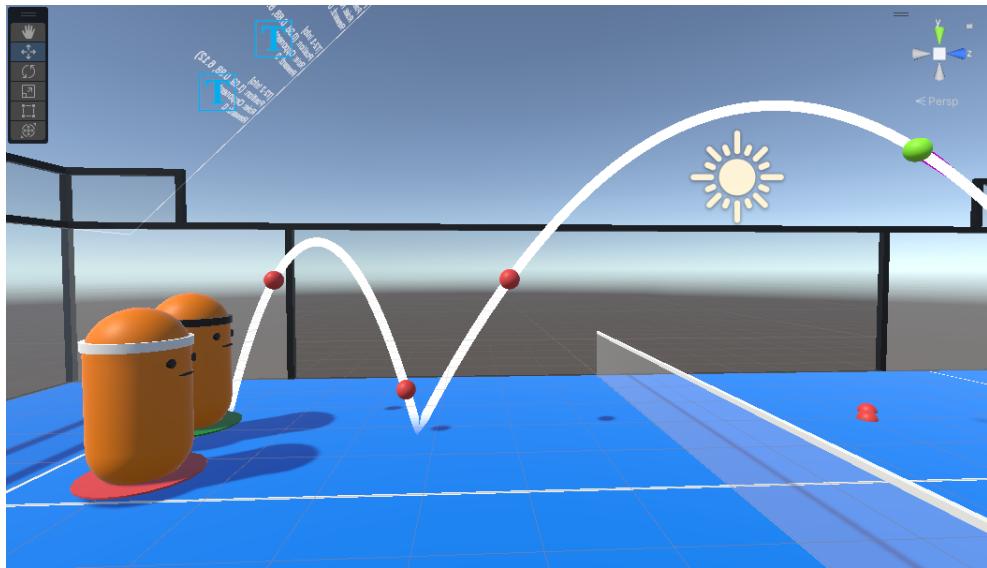


Figura 16: Posiciones de interés de devolución, representadas con un punto rojo en la línea de la trayectoria. En esta situación, el jugador 2 del equipo T1 (cinta negra) ha recibido el rol de *Receiver*, representado con la marca verde, ya que tiene más posiciones clave cerca. (Elaboración propia)

Estas posiciones se calculan en el momento que se simula una trayectoria, ya que las posiciones futuras de la pelota provienen del controlador de la trayectoria. Una vez calculadas, se determina a quién se debería asignar el rol de *Receiver*, según qué jugador tiene más posiciones de interés de devolución más cercanas que el compañero. Las posiciones deben actualizarse constantemente, ya que a medida que pasa el tiempo pueden pasar a ser inalcanzables, ya sea porque la pelota haya sobrepasado la posición en cuestión, o bien porque ya no haya ningún jugador que pueda alcanzarla en el margen de tiempo restante. Como consecuencia, es posible que el rol de *Receiver* se vaya alternando dinámicamente.

Mediante las posiciones clave de devolución, es posible definir una función de recompensa que incentiva al agente con mayor probabilidad de poder realizar una devolución a posicionarse correctamente para la devolución, ayudándole de cierta manera a predecir dónde caerá la pelota.

Las posiciones de interés de cobertura son relevantes para los roles Teammate y Opponent. El rol Teammate se asigna al compañero del agente que tenga el rol Receiver, y el rol Opponent se asigna a los agentes del equipo que acaba de devolver la pelota.

Una posición de cobertura consiste en la posición que cubre la zona más descubierta de un jugador. Dada la situación ilustrada en la Figura 17, donde se sitúa el jugador *Player* en la posición $(x, z) = (2, 7)$, en un área equivalente al campo del equipo T2, la posición de cobertura óptima se calcula como el punto medio de las distancias más largas entre el jugador y los límites del área:

$$(x, z)_{KP} = (x, z)_{Player} + 0.5 \arg \max(\|\vec{x}_1\|, \|\vec{x}_2\|) + 0.5 \arg \max(\|\vec{z}_1\|, \|\vec{z}_2\|) \quad (40)$$

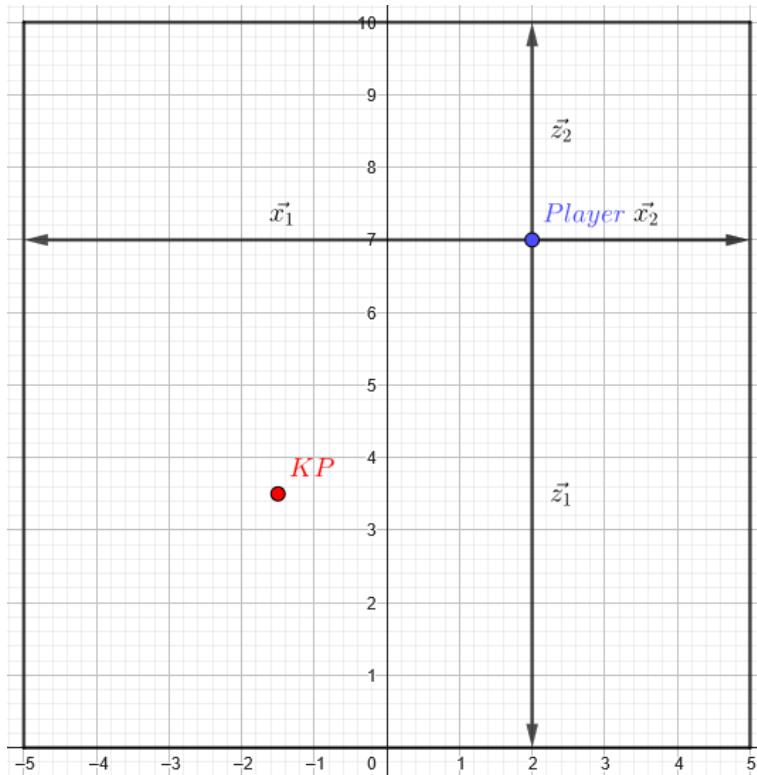


Figura 17: Ejemplo de una posición óptima de cobertura *KP*, generada por el jugador *Player*.
(Elaboración propia)

Estas posiciones de cobertura permiten incentivar a los agentes a distribuirse correctamente en la pista, evitando que los agentes de un mismo equipo se junten demasiado. En el caso del agente con el rol Teammate asignado, tendería a cubrir la zona que deja descubierta el agente Receiver. Para los agentes Opponent el comportamiento es distinto, ya que ambos deberían cubrir el espacio descubierto que deja el otro agente. En este caso, los agentes tenderían a buscar un punto de equilibrio en el que ambos puedan cubrir su campo de manera óptima.

Por supuesto, en un partido real de pádel el punto óptimo para cubrir el espacio depende de diversos aspectos, no únicamente de las posiciones de los jugadores en el propio campo, pero se ha optado por utilizar una funcionalidad más simple para no aumentar demasiado la complejidad de la solución. El fin de utilizar estas posiciones de cobertura es, principalmente, para que los agentes aprendan que deben mantenerse alejados entre sí, pero sin que sea demasiado exagerado.

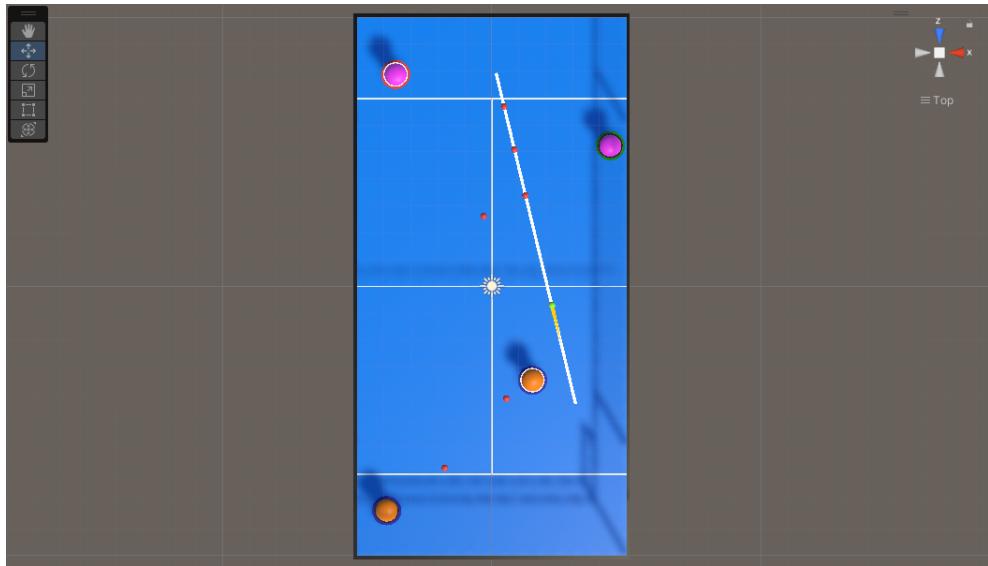


Figura 18: Posiciones de interés de devolución, representadas con un punto rojo en el suelo de la pista. Ambos jugadores con el rol Opponent generan una posición de cobertura para el otro. El jugador con el rol Receiver genera una posición de cobertura para su compañero, con el rol Teammate. (Elaboración propia)

En el controlador de las posiciones clave se implementan también los métodos utilizados para asignar roles y recompensas a los agentes. Para los roles, el rol de Opponent se asigna en el momento del golpeo, mientras que los roles Receiver y Teammate se asignan dependiendo de los puntos de devolución que se hayan calculado, y pueden ir alternándose dinámicamente.

Como ya se ha mencionado previamente, las recompensas se asignan a un agente cuando se está acercando o está cerca de un punto clave. Para comprobar si se está acercando, guarda internamente las posiciones del paso anterior y compara con el paso actual las distancias que hay entre ambas posiciones y la posición clave. Las posiciones de cobertura son únicas para los agentes correspondientes, y en el caso de las posiciones de devolución, se elige la posición más cercana al agente Receiver. Se considera que un agente está cerca de una posición clave cuando está a una distancia inferior a 1.5 metros.

Agente de pádel

El nuevo espacio de observaciones del agente de pádel consiste ahora en:

- Las posiciones 2D de los cuatro agentes del entorno.
- La posición y la velocidad 3D de la pelota.
- Su propio rol, el cual puede ser Opponent, Receiver o Teammate.
- Un valor booleano que indica si puede golpear la pelota.

Sumando un total de 16 observaciones. En el caso de los agentes del equipo T2, se invierten las coordenadas del eje X e Z, como ya se había mencionado en los cambios realizados.

Un agente puede golpear la pelota cuando:

- La pelota esta está a su alcance, dentro de un radio de 2 metros.
- La pelota tiene una altura superior a 0.25 metros.
- El último equipo en golpear la pelota ha sido el equipo contrario al del agente.
- La pelota no está bloqueada debido al cálculo de simulaciones.

El nuevo espacio de acciones está formado únicamente por acciones discretas, con un total de 5 ramas, donde cada rama consiste en un conjunto de subacciones de tamaño variable:

- La rama 0, de tamaño 3, utilizada para definir el movimiento vertical: 0 para quedarse quieto, 1 para moverse hacia adelante, y 2 para moverse hacia atrás, dependiendo siempre de la orientación del agente (en esta versión, el agente solo puede tener dos orientaciones: 0° y 180° , que no cambian durante un juego).
- La rama 1, de tamaño 3, utilizada para definir el movimiento horizontal: 0 para quedarse quieto, 1 para moverse hacia la derecha, y 2 para moverse hacia la izquierda, dependiendo también de la orientación del agente.
- La rama 2, de tamaño 5, utilizada para especificar hacia qué columna de la cuadrícula enviar la pelota (representan cambios en el eje X).
- La rama 3, de tamaño 5, utilizada para especificar hacia qué fila de la cuadrícula enviar la pelota (representan cambios en el eje Z).
- La rama 4, de tamaño 4, utilizada para especificar qué tipo de golpeo realizar: 0 para no golpear, 1 para ejecutar una derecha, 2 para un globo y 3 para un remate. Para una derecha se define una altura máxima $y_{max} = 2$ metros, para el globo $y_{max} = 4$ metros y para el remate $y_{max} = -1$ metros (ya que en el cálculo de la velocidad inicial necesaria, cuando $y_{max} < y_0$, se asigna $y_{max} = y_0$ de manera que se aplica una velocidad paralela al plano XZ).

El movimiento tanto vertical como horizontal se realiza con una velocidad de 5 m/s y depende del tiempo entre frames Time.deltaTime. Para el golpeo de la pelota se comprueba que el agente pueda golpearla dadas las condiciones descritas previamente. Tras el golpeo de la pelota, se añade una recompensa a nivel de equipo.

Este espacio de acciones no tiene por qué ser necesariamente este, pero debido a la limitación de tiempo del TFG, se ha dado por bueno y no se ha explorado más alla de este espacio. Sería posible aumentar la resolución de la cuadrícula y añadir más tipos de golpeos, aunque esto aumentaría la complejidad del entrenamiento y requeriría más pasos de entrenamiento.

El inicio de un episodio comienza con el servicio y termina con la obtención de un punto. Al principio de cada episodio, los agentes no pueden moverse hasta que se haya realizado el servicio.

Las funciones de recompensa se han mencionado a lo largo de esta sección pero, resumidamente, un agente puede recibir una recompensa (o penalización, si es negativa) tras ganar un punto, perder un punto, acercarse a una posición clave, quedarse cerca de una posición clave o golpear a la pelota.

En el agente de pádel también se han implementado funcionalidades para modificar el estado del agente, por ejemplo para asignar los roles, cambiar el color de la marca según el rol, etc.

9.2.3. Implementación específica para la grabación de demostraciones

Además del entorno en todo su conjunto, también se han implementado las funcionalidades necesarias para la grabación de demostraciones, las cuales son necesarias para entrenar agentes mediante GAIL.

Con ML-Agents, las demostraciones normalmente son sencillas de obtener cuando se trata de entornos en los que hay un único agente. Durante la grabación, suele ser el propio usuario quien proporciona las demostraciones expertas mediante una heurística definida manualmente (por ejemplo, mover el agente mediante controles de teclado). En este entorno, sin embargo, al haber cuatro agentes que deben actuar como expertos, es difícil controlarlos a la vez a través de un teclado, y más aún si deben moverse de manera distinta, cuando todos comparten la misma heurística.

La solución que se ha propuesto para grabar las demostraciones es utilizar un conjunto de datos de partidos profesionales de pádel, obtenidos mediante la aplicación de técnicas de visión por computador para el seguimiento de los jugadores en vídeos de partidos [21].

En esta solución, un *entrenador* se encarga de procesar los datos relevantes del conjunto de datos, formando, en particular, muestras con información sobre una situación posicional y las acciones que hayan realizado los jugadores profesionales en cada respectiva situación. La idea es que un agente pueda solicitar al entrenador las acciones que debería realizar enviándole la información relevante del estado del entorno virtual. El entrenador, a partir de esta información, busca la muestra con la situación posicional más parecida posible dentro del conjunto de datos. La acción obtenida a partir de esta muestra se envía finalmente al agente para que la ejecute. Repitiendo esto sucesivamente, en cada paso, el objetivo es poder reproducir, en los agentes, un comportamiento similar a los jugadores profesionales en un partido de pádel.

Al igual que en el resto del entorno, la comunicación entre el agente y el entrenador se hace a través de un controlador del entrenador, con tal de evitar introducir dependencias entre estas clases.

Entrenador

El entrenador es un script de Python que funciona como un servidor, en el que se define un socket para establecer una comunicación con la aplicación de Unity. Este script carga el conjunto de datos especificado, lo procesa eliminando aquella información innecesaria y se queda en espera para recibir solicitudes. Estas solicitudes pueden ser de dos tipos, según si es para solicitar un movimiento, o bien para solicitar un golpeo. Cada solicitud llega acompañada de información sobre el estado del entorno virtual de pádel, y se utiliza para obtener la muestra más parecida del conjunto de datos según cómo de parecida sea la situación posicional con el estado del entorno virtual. A partir de esta muestra, se extrae la acción asociada y se devuelve finalmente al cliente.

El conjunto de datos que procesa el entrenador es, en particular, un fichero Excel (.xlsx) que contiene 22,615 filas, donde cada fila consiste en la información posicional de los jugadores y de la pelota en una pista de pádel, además de otros valores calculados durante el posprocesamiento.

En el dominio de estos datos, el origen de coordenadas $(0, 0)$ se sitúa en la esquina inferior izquierda de la pista, el ancho de la pista está definido por $x \in [0, 10]$, y el largo por $y \in [0, 20]$. Los jugadores situados en la mitad superior de la pista ($y > 10$) forman el equipo T, mientras que aquellos situados en la mitad inferior ($y < 10$) forman el equipo B. Dentro de un mismo equipo también se hace una distinción entre los propios jugadores, según en qué lado han empezado un punto. El jugador que está más a la izquierda se denomina TL, mientras que el jugador situado más a la derecha se denomina TR. Los jugadores del equipo B también se distinguen de esta manera, denominados BL y BR. En la Figura 19 se representa una muestra individual en una gráfica, limitada por las medidas de una pista de padel.

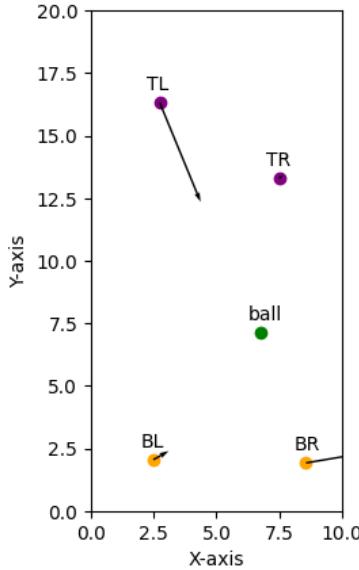


Figura 19: Ejemplo gráfico de una única muestra del conjunto de datos utilizado para grabar demostraciones. (Elaboración propia)

Los valores relevantes de este conjunto de datos, además de algunos ejemplos de muestras, se presentan en la Figura 20. De izquierda a derecha, cada par (x, y) consiste en la posición de un jugador TL, TR, BL, BR, respectivamente; $(\text{ball } x, \text{ ball } y)$ es la posición de la pelota; cada par $(\text{speed}x, \text{ speedy})$ es la velocidad de cada respectivo jugador; shot es el tipo de golpeo que se ejecuta, según si es un golpeo normal, un globo o un remate, definido únicamente en aquellas muestras en las que un jugador golpea la pelota; y last hit es una variable que indica qué equipo ha sido el último en golpear la pelota.

	x	y	x.1	y.1	x.2	y.2	x.3	y.3	ball x	ball y	speedx	speedy	speedx.1	speedy.1	speedx.2	speedy.2	speedx.3	speedy.3	shot	target x	target y	last hit
0	1.89	18.28	7.49	13.35	2.50	1.60	7.87	1.66	1.89	18.28	-8.17e-01	-3.90e-01	-5.09e-03	-0.04	-0.14	-0.31	1.49e-01	-1.20e+00	normal	8.98	2.11	T
1	1.87	18.27	7.49	13.35	2.50	1.59	7.88	1.62	2.14	17.72	7.00e-01	-2.61e+00	-5.44e-02	-0.20	-0.22	1.27	3.90e-01	-6.59e-01	undef	0.00	0.00	T
2	1.89	18.18	7.49	13.35	2.49	1.63	7.89	1.60	2.38	17.17	1.36e+00	-2.46e+00	-2.42e-02	-0.05	-0.76	2.13	3.45e-01	-8.45e-01	undef	0.00	0.00	T
3	1.94	18.10	7.49	13.34	2.47	1.70	7.90	1.57	2.63	16.61	8.56e-01	-4.29e+00	-7.52e-03	-0.03	-0.25	0.92	9.74e-01	1.03e-01	undef	0.00	0.00	T
4	1.96	17.96	7.49	13.34	2.46	1.74	7.93	1.58	2.87	16.05	7.56e-03	0.00e+00	0.00	0.00	0.00	1.50e-05	4.80e-05	undef	0.00	0.00	T	
...
21610	2.71	17.63	8.32	18.75	3.57	6.26	6.16	6.99	8.02	16.79	6.76e-01	5.20e-01	2.32e-01	-0.17	0.74	3.45	1.19e+00	-1.75e+00	undef	0.00	0.00	B
21611	2.73	17.64	8.33	18.74	3.60	6.38	6.20	6.93	8.11	17.30	5.01e-01	-4.95e-01	9.88e-01	0.19	0.78	0.81	2.34e+00	1.43e+00	undef	0.00	0.00	B
21612	2.75	17.63	8.36	18.75	3.63	6.41	6.27	6.98	8.21	17.80	1.17e-01	1.04e+00	-8.57e-02	-0.65	0.48	-1.00	-2.41e-02	1.22e-01	undef	0.00	0.00	B
21613	2.75	17.66	8.36	18.72	3.64	6.37	6.27	6.98	8.30	18.31	7.27e-02	8.94e-03	1.28e+00	2.74	0.04	3.57	2.06e+00	4.98e-01	undef	0.00	0.00	B
21614	2.76	17.66	8.40	18.82	3.64	6.49	6.34	7.00	8.40	18.82	7.27e-02	8.94e-03	1.28e+00	2.74	0.04	3.57	2.06e+00	4.98e-01	undef	0.00	0.00	B

Figura 20: Conjunto de datos utilizado para la grabación de demostraciones. (Elaboración propia)

A partir de estos valores, se define un subconjunto de valores que definen una situación posicional, formado por las posiciones de cada jugador y de la pelota:

```
[x, y, x.1, y.1, x.2, y.2, x.3, y.3, ball_x, ball_y]
```

De esta manera, es posible definir que la muestra del conjunto de datos más parecida a un estado del entorno virtual es aquella cuyas posiciones son lo más cercanas posibles a las posiciones del estado del entorno virtual, dentro de un dominio común para ambos conjuntos y utilizando la distancia Euclídea, dado que todos estos datos están en metros.

Debido a que el entrenador puede recibir solicitudes distintas según si es para solicitar un movimiento o un golpeo, también se hace una separación en los espacios de búsqueda. Esto se ha decidido así porque para extraer información sobre el golpeo de una muestra, una condición necesaria es que el golpeo esté definido, es decir, que el valor de shot sea distinto a undef. Además de la separación entre datos de movimiento y datos de golpeo, dentro de cada conjunto es necesario hacer otra división según el valor categórico last_hit. Esta división es importante para mantener la coherencia de los datos, ya que el movimiento de los jugadores es distinto según si la pelota está yendo al propio campo o al campo rival. De la misma manera, el golpeo también es distinto según si lo realiza un jugador del equipo T o del equipo B:

1. Datos de movimientos cuando la pelota ha sido golpeada por el equipo T.
2. Datos de movimientos cuando la pelota ha sido golpeada por el equipo B.
3. Datos de golpeos realizados por el equipo T.
4. Datos de golpeos realizados por el equipo B.

Un algoritmo adecuado para realizar búsquedas de muestras bajo estas condiciones es mediante árboles decisionales, concretamente árboles k -dimensionales. El entrenador, después de haber creado los subconjuntos de datos correspondientes aplicando los filtros necesarios, crea cuatro árboles k -dimensionales, uno para cada subconjunto. Estos árboles permiten realizar búsquedas eficientes de los vecinos más próximos a una muestra proporcionada (en este caso, un vector 10-dimensional con el formato del subconjunto de valores que definen una situación posicional).

Tras completar la búsqueda, el entrenador envía al entorno virtual de Unity una respuesta en un formato determinado, según el tipo de acción que se haya buscado. En el caso de haber consistido en una búsqueda de un golpeo, la trama contiene el tipo de golpeo realizado, además de la posición de impacto:

```
[shot, target_x, target_y]
```

Para una búsqueda de un movimiento, la trama consiste en cuatro posiciones, una para cada jugador, y el vector de velocidad de cada posición:

```
[x, y, x.1, y.1, x.2, y.2, x.3, y.3, speedx, speedy, speedx.1, speedy.1, speedx.2, speedy.2, speedx.3, speedy.3]
```

Específicamente para la comunicación entre el cliente y el servidor, las tramas que se intercambian también contienen otros valores que indican el tipo de solicitud (SHOT_REQUEST, MOVEMENT_REQUEST), el tipo de respuesta (SHOT_RESPONSE, MOVEMENT_RESPONSE), y el identificador PlayerId del jugador que haya enviado la solicitud o al que se quiera enviar la respuesta. Esto permite tanto al cliente como al servidor poder mantener un control sobre la comunicación, pudiendo diferenciar qué tipo de información tiene cada trama y hacia quién va dirigida.

En las solicitudes de movimiento enviadas desde el entorno virtual al entrenador, también se especifica el último equipo que haya golpeado la pelota, según si ha sido T1 o T2. Esto permite al entrenador determinar qué árbol utilizar para la búsqueda, asociando los equipos T1 y T2 del entorno virtual a los equipos B y T de los datos, respectivamente.

Para las solicitudes de golpeo no es necesario especificar el último equipo que haya golpeado la pelota, ya que se asume que el jugador que haya enviado la solicitud será el jugador que golpeará la pelota. Por lo tanto, basta con asociar los jugadores T1_1 y T1_2 con los datos de golpes realizados por el equipo B, y los jugadores T2_1 y T2_2 con los datos de golpes realizados por el equipo T.

Heurística del agente

Las solicitudes de movimientos y golpes las envía el agente a través de la implementación de la heurística. Para moverse, se envían solicitudes de movimientos continuamente. Para el golpeo de la pelota, se envían solicitudes de golpes siempre y cuando la pelota esté dentro de un rango de golpeo. El agente realiza las acciones proporcionadas por el entrenador en el mismo instante que las recibe.

Tanto en las solicitudes de movimientos como de golpes se envían su identificador, las posiciones locales del propio agente, de su compañero, de sus dos oponentes y de la pelota. En las solicitudes de movimientos se envía también la información sobre qué equipo ha golpeado la pelota por última vez.

Para no colapsar el intercambio de mensajes entre el agente y el entrenador, el agente tiene implementado un sistema de control no bloqueante para el envío de solicitudes y la ejecución de acciones. Un agente puede enviar una solicitud siempre y cuando no esté esperando a que llegue la respuesta de una solicitud pasada. En cuanto un agente recibe la acción que debe realizar, se considera que la solicitud pasada, que estaba pendiente a ser resuelta, ya se ha resuelto, por lo que el agente puede proceder a enviar la siguiente solicitud.

Controlador del entrenador

El controlador del entrenador sirve como intermediario entre el agente y el entrenador. Dado que el dominio de las posiciones en el entorno virtual de pádel es distinto al dominio del conjunto de datos de partidos profesionales de pádel, es necesario preprocesar los datos enviados desde el agente al entrenador, y posprocesar la información recibida del entrenador para enviársela al agente.

En el preprocesamiento de datos, es necesario respetar el orden de los valores que utiliza el entrenador para la búsqueda de movimientos o golpes. Por ello, las posiciones recibidas por el agente se ordenan siguiendo el orden TL, TR, BL y BR, identificando primero qué posiciones forman parte del equipo de la parte superior T ($z > 0$), y de la parte inferior B ($z < 0$), para luego determinar, dentro de cada equipo, qué posición está más a la izquierda del otro, sin tener en cuenta orientaciones, para ordenar según los lados T y R. Además, se elimina el eje Y de las posiciones ya que no se dispone de información sobre la altura en los datos del entrenador. Por último, se aplica una traslación a cada posición con un vector $\vec{v} = (x, z) = (5, 10)$, para transformar el dominio de las posiciones del estado del entorno virtual al dominio de los datos del entrenador.

Por otro lado, en el posprocesamiento, lo primero que se hace es la traslación inversa $-\vec{v} = -(x, z) = (-5, -10)$ para proceder a adaptar la acción recibida del entrenador a una acción del entorno virtual.

Para el posprocesamiento de golpeos, la información de la que se dispone es el tipo de golpeo y la posición de impacto. La traducción del tipo de golpeo shot, los cuales pueden ser normal, lob o smash, es directa: los valores de la rama 4 serían 1, 2 o 3, respectivamente. Para la selección de la posición dentro de la cuadrícula de posiciones, la idea es que se asocie la posición de impacto con el punto de la cuadrícula que minimiza la distancia entre la posición de impacto y la posición local real de la cuadrícula. Para ello, se inicializa una matriz tanto para T1 como para T2, para asignar la posición local real correspondiente a cada posición de la cuadrícula. Haciendo un recorrido sobre la matriz correspondiente, se calculan los índices de la cuadrícula que minimicen la distancia entre la posición de impacto y la posición local real de la cuadrícula, y de esta manera se obtienen los valores necesarios para la rama 2 y 3 de las acciones discretas del agente.

Antes de definir el posprocesamiento de movimientos, cabe mencionar que el envío de solicitudes de movimientos tienen un comportamiento especial. En el entorno virtual, si los cuatro agentes enviaran constantemente solicitudes de movimientos, prácticamente de manera simultánea, el entrenador recibiría la misma información y devolvería la misma muestra para cada agente, solo que con un vector de velocidad distinto. Esto no es un funcionamiento deseado, ya que es bastante ineficiente. En su lugar, cuando el controlador del entrenador recibe una solicitud de movimiento por parte del agente, bloquea el envío del resto de solicitudes de movimientos hasta que se resuelva la primera solicitud recibida. La respuesta a esta solicitud no se trata únicamente del vector de velocidad del agente que envió la solicitud originalmente, sino que contiene información sobre cuatro posiciones y sus respectivas velocidades. Utilizando toda esta información, es posible calcular las acciones para todos los agentes con una única búsqueda.

Para el posprocesamiento de movimientos, por lo tanto, dadas las cuatro posiciones y sus respectivas velocidades, el primer paso es asociar cada posición al agente correspondiente. Esto se hace fácilmente calculando qué agente está más cerca de cada posición. Asociando a un agente una determinada posición, también se asocia indirectamente la velocidad que debería tener, y esta velocidad es justamente lo que define el movimiento del agente. Dependiendo de la orientación de cada agente, se asignan algunos valores u otros dependiendo del signo de la velocidad en los ejes correspondientes. Por ejemplo, dado un agente del equipo T1, el cual tiene vectores Forward y Right orientados hacia los ejes positivos Z y X, respectivamente, según el signo de la primera componente de la velocidad (eje X), debería moverse hacia la izquierda si fuera negativo, quedarse quieto si fuera 0, o moverse hacia la derecha si fuera positivo. Similarmente para la segunda componente, hacia adelante si fuera positivo, quedarse quieto si fuera 0, o hacia detrás si fuera negativo.

En la Figura 21 se muestra un diagrama secuencial de una ciclo completo de la comunicación entre el agente, el controlador del entrenador y el entrenador.

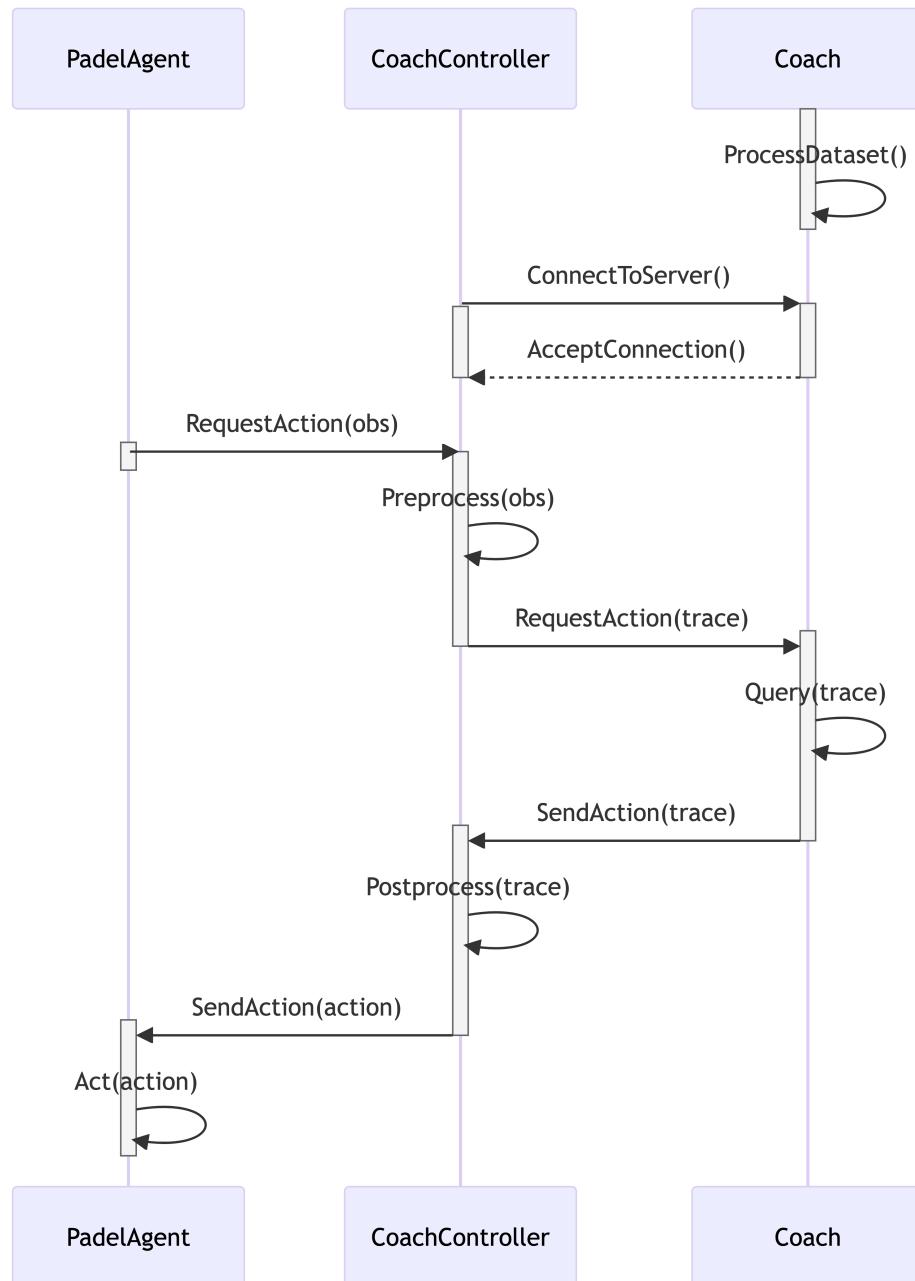


Figura 21: Diagrama secuencial de la comunicación entre el agente, el controlador del entrenador y el entrenador. (Elaboración propia)

10. Resultados

En el repositorio del proyecto [20] se pueden encontrar vídeos que muestran el comportamiento de los agentes entrenados, además de los resultados y modelos obtenidos tras el entrenamiento.

10.1. Aprendizaje por refuerzo profundo

El objetivo de la experimentación utilizando técnicas de aprendizaje por refuerzo profundo ha sido, principalmente, demostrar empíricamente que el entrenamiento de agentes virtuales en el entorno virtual de pádel es factible.

Otros aspectos que se han estudiado son el efecto que tiene utilizar Curiosity en este entorno, además de cómo afectan al entrenamiento la modificación de la función de recompensa, definida inicialmente en la prueba inicial.

En realidad, hay otros parámetros que se deberían haber estudiado durante la experimentación. Por ejemplo, hubiera sido relevante haber hecho más pruebas modificando los hiperparámetros de las redes neuronales utilizadas. Sin embargo, esto hubiera extendido demasiado la parte de experimentación, por lo que se optó por dar prioridad a experimentar con los elementos definidos directamente en el entorno virtual.

A la hora de comparar experimentos, las medidas que se han utilizado son las relacionadas con el entorno y la puntuación Elo:

- Dentro de las recompensas del entorno están la recompensa acumulada y la duración de un episodio, y son relevantes para determinar si la política que se está entrenando correctamente. Normalmente, al principio del entrenamiento los episodios suelen ser cortos y la recompensa acumulada es usualmente muy variada, ya que la política aún no sabe mucho del entorno. A medida que va aprendiendo, la idea sería que la duración de los episodios y la recompensa acumulada de cada episodio aumentaran cada vez más. Específicamente en este entorno virtual de pádel, sin embargo, un aumento en la duración del episodio no necesariamente implicaría una mejora en la política. Por ejemplo, los agentes podrían aprender a mantener la pelota en juego realizando siempre golpes muy fáciles de devolver, lo cual no es deseado.
- La puntuación Elo de la política resulta ser una medida más fiable que el resto, al tratarse de un entorno competitivo. Esta puntuación aumenta o decrece en función de cómo está rindiendo la política que está siendo entrenada contra las instancias de políticas obtenidas con *self-play*. La variación de la puntuación siempre depende del oponente, si por ejemplo se ganara a un oponente con una puntuación Elo mucho más alta, el propio agente ganaría más puntos. En cambio, si la puntuación del oponente fuera baja, ganaría muy pocos puntos. Si la puntuación Elo aumenta considerablemente durante el transcurso del entrenamiento, significaría que el agente está consiguiendo ganar a oponentes más hábiles que el propio agente.

Aunque estas medidas ya aportan mucha información sobre cómo ha ido el entrenamiento, para saber si realmente se ha podido conseguir un modelo robusto, también es importante probar el modelo directamente a través de Unity, mediante inferencia, para comprobar si realmente ha podido aprender a jugar a pádel.

10.1.1. Experimento 1. Entrenamiento mediante PPO

El primer experimento ha consistido en hacer una prueba sencilla en la que solo se definen recompensas por ganar y perder puntos, mostradas en la Tabla 17. Para el entrenamiento se ha utilizado PPO, y la configuración del entrenamiento se puede consultar en el Apéndice B.1.

Recompensa	Valor
WinningReward	10
LosingReward	-10
ApproachingKeyPositionsReward	0
StayingAroundKeyPositionsReward	0
HittingBallReward	0

Tabla 17: Función de recompensa del experimento 1. (Elaboración propia)

Tras un millón de pasos, las medidas obtenidas han sido las mostradas en la Figura 22. Como es notable, la recompensa acumulada no ha dejado de variar hasta las últimas iteraciones del entrenamiento, en los que parece que uno de los dos equipos es capaz de ganar varios puntos seguidos. Eso se debe a que, al principio del entrenamiento, como que los agentes apenas han tenido tiempo para aprender nada, y al haber rotaciones en el servicio de la pelota, la obtención de puntos se va alternando constantemente. A medida que aumenta la duración de los episodios, cada juego se va volviendo más reñido, hecho que contribuye mucho en el proceso de aprendizaje. En cuanto a la puntuación Elo, se puede observar que ha ido subiendo a medida que avanzaba el entrenamiento, y parece que la política aún tenía margen de mejora, ya que en los últimos episodios el aumento seguía siendo bastante considerable.

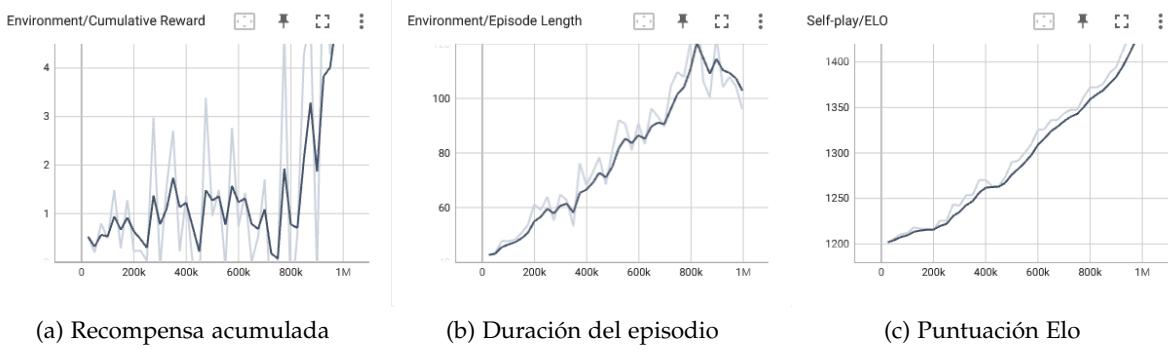


Figura 22: Resultados del experimento 1. (Elaboración propia)

Finalmente, probando el modelo obtenido directamente en el entorno de Unity, se ha observado que en esta primera prueba el posicionamiento no era muy bueno. Durante el partido, los agentes acababan juntándose frecuentemente para ir hacia la pelota a la vez. En realidad, este comportamiento era el esperado, ya que las únicas recompensas que se habían definido fueron las obtenidas tras un punto, por los que los agentes no reciben ningún feedback sobre cómo se deberían posicionar. Otro aspecto destacable es que la política parece ser capaz de enviar la pelota hacia espacios libres, lo cual se podría considerar como un comportamiento muy competitivo, lo cual es bueno. Por ejemplo, en varias ocasiones, los agentes envían la pelota hacia el fondo cuando los oponentes se encuentran cerca de la red, dificultando el alcance de la pelota a los oponentes.

10.1.2. Experimento 2. Efecto de Curiosity en el entorno virtual de pádel

En este experimento se ha puesto a prueba el efecto que tendría añadir la recompensa intrínseca Curiosity a este entorno. Para hacer una comparación justa con los resultados obtenidos en el experimento anterior, se han definido las mismas recompensas extrínsecas. En cuanto a la configuración del entrenamiento, la cual se puede consultar en el Apéndice B.2, todos los parámetros se han mantenido igual salvo la adición de la configuración específica de Curiosity.

Recompensa	Valor
WinningReward	10
LosingReward	-10
ApproachingKeyPositionsReward	0
StayingAroundKeyPositionsReward	0
HittingBallReward	0

Tabla 18: Función de recompensa del experimento 2. (Elaboración propia)

Las medidas obtenidas en este experimento (y en los experimentos de ahora en adelante) se han contrastado con los resultados obtenidos en el experimento sin Curiosity (de color azul marino para todas las gráficas). Observando la Figura 23, resulta evidente que el uso de Curiosity no solo no ha aportado ninguna mejora, sino que además ha empeorado el rendimiento de los agentes. La recompensa acumulada ha tenido mucha varianza durante todo el entrenamiento, y la duración de los episodios no ha aumentado mucho. La puntuación Elo empeoraba cada vez más durante la primera mitad del entrenamiento, y después se la primera mitad prácticamente se mantuvo constante.

La conclusión sacada tras estas pruebas es que parece que condicionar a los agentes incentivándoles a explorar nuevos estados no es muy adecuado en este tipo de entornos de aprendizaje. Esto se debe muy posiblemente a que se trata de un entorno cerrado, en el que el objetivo es muy claro: dirigirse hacia la pelota y devolverla al campo contrario. Con Curiosity, los agentes son recompensados aunque se estén alejando de la pelota, y necesitarían muchísimos más pasos de entrenamiento hasta que aprendan cuál es exactamente el objetivo que tienen, una vez el peso de las recompensas intrínsecas Curiosity sea prácticamente nulo.

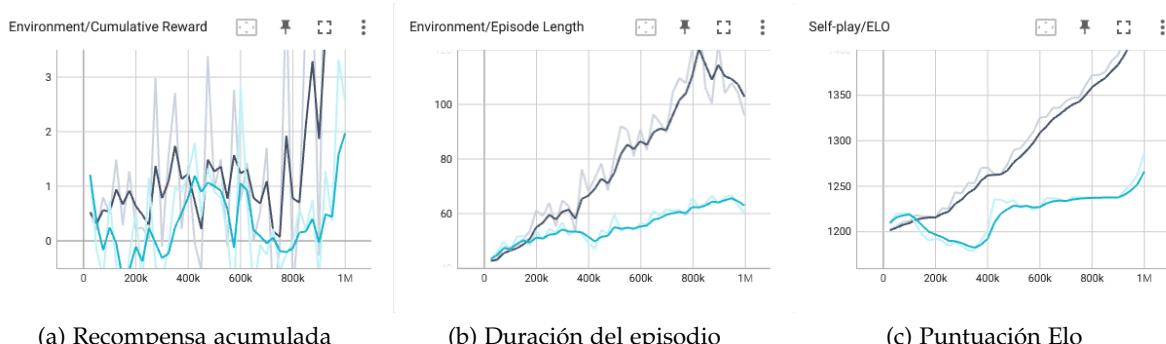


Figura 23: Resultados del experimento 2, en azul. (Elaboración propia)

Desde el entorno de Unity, enfrentando el modelo sin Curiosity contra el modelo obtenido en este experimento, se puede observar que los agentes entrenados con Curiosity la mayoría de veces ni siquiera son capaces de dirigirse a la pelota.

10.1.3. Experimento 3. Influencia de la recompensa por golpeo

En el entorno de pádel, una manera de evitar la escasez de recompensas es añadiendo una recompensa por golpear la pelota. Con esta recompensa, los agentes deberían poder aprender más rápidamente que uno de los objetivos que tienen es devolver la pelota hacia el campo rival. En el tercer experimento, se ha utilizado esta recompensa para determinar si realmente influye en el entrenamiento de los agentes. Tanto en este como en los experimentos restantes se han utilizado la configuración del entrenamiento utilizada en el primer experimento, ya que Curiosity no ha resultado ser efectivo.

Recompensa	Valor
WinningReward	10
LosingReward	-10
ApproachingKeyPositionsReward	0
StayingAroundKeyPositionsReward	0
HittingBallReward	1

Tabla 19: Función de recompensa del experimento 3. (Elaboración propia)

En los resultados de este experimento, el cambio más notable ha sido en la recompensa acumulada. Al haber una recompensa por golpear la pelota, en general, los agentes acaban obteniendo más recompensas en cada episodio, por lo que este variación era de esperarse. La diferencia entre la duración de los episodios no es tan clara, ya que aunque durante la mitad del entrenamiento la duración estuviera por debajo del primer experimento, al final del entrenamiento se acabó superando. En la puntuación Elo también pasa algo similar, parece que la política de este experimento tiende a mejorar más rápidamente, pero al final del entrenamiento apenas aumenta. En principio, parece que la recompensa por golpear la pelota no ha sido de gran utilidad. Quizás se debe mayoritariamente a que, dentro de lo que cabe, las recompensas que se dan tras cada punto no están tan alejadas entre sí.

Como apunte, es importante tener en consideración que las recompensas deben definirse en proporción a lo relevantes que sean respecto al objetivo final. En este caso, por ejemplo, la recompensa que se ha definido por golpear la pelota es, quizás, demasiado alta en comparación a la recompensa por ganar un punto: tras diez golpes en un punto, la recompensa por ganar ese punto acaba perdiendo relevancia. Esto puede conllevar a que los agentes adopten comportamientos que les permita explotar estas recompensas más pequeñas, haciendo que ambos equipos cooperen para sumar puntos en lugar de ser competitivos.

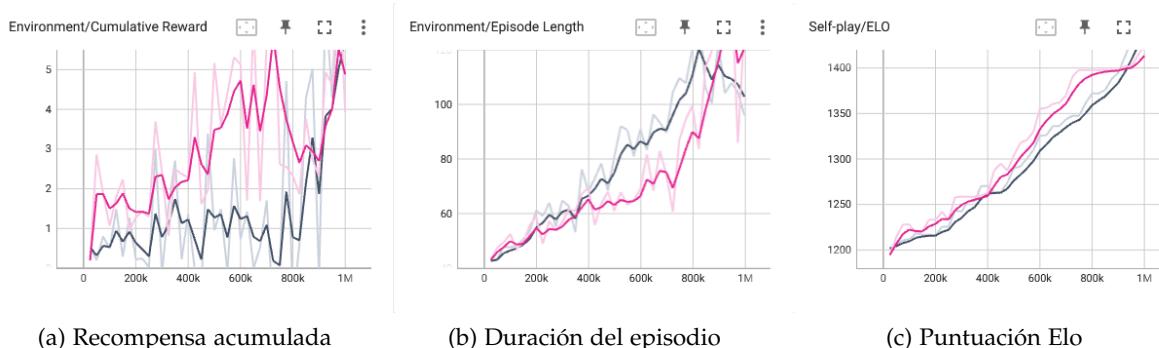


Figura 24: Resultados del experimento 3, en magenta. (Elaboración propia)

10.1.4. Experimento 4. Importancia del sistema de puntuación Elo

Derivado del tercer experimento, en el cuarto experimento se ha forzado el problema de la explotación eliminando la recompensa por ganar un punto. Nótese que aunque esta recompensa pase a ser cero, sería equivalente si la recompensa por golpear la pelota fuera excesivamente grande.

Recompensa	Valor
WinningReward	0
LosingReward	-10
ApproachingKeyPositionsReward	0
StayingAroundKeyPositionsReward	0
HittingBallReward	1

Tabla 20: Función de recompensa del experimento 4. (Elaboración propia)

Tal y como se esperaba, en los resultados de este experimento se puede observar cómo tanto la recompensa acumulada como la duración de un episodio han aumentado drásticamente, a partir de la segunda mitad del entrenamiento. Al principio, las recompensas acumuladas son negativas debido a que los agentes aún no son capaces de golpear la pelota. A medida que van aprendiendo, ambos equipos acaban descubriendo que son capaces de prácticamente anular la penalización por perder un punto, si cooperan entre sí para devolverse la pelota constantemente. La duración de un episodio aumenta precisamente por esta razón.

Sin embargo, cabe destacar que la variación en la puntuación Elo ha sido justo lo contrario. Aunque a primera vista parecía que los agentes habían aprendido una buena política, la puntuación Elo indica que apenas están mejorando. Esto se debe precisamente a que, al final de cada episodio, realmente no hay ningún equipo que pierda, ya que la recompensa acumulada de ambos equipos acaba siendo positiva. Es precisamente por este motivo que el sistema de puntuación Elo es, a veces, más relevante que las otras medidas, ya que ofrece una mejor valoración del rendimiento en los entornos competitivos.

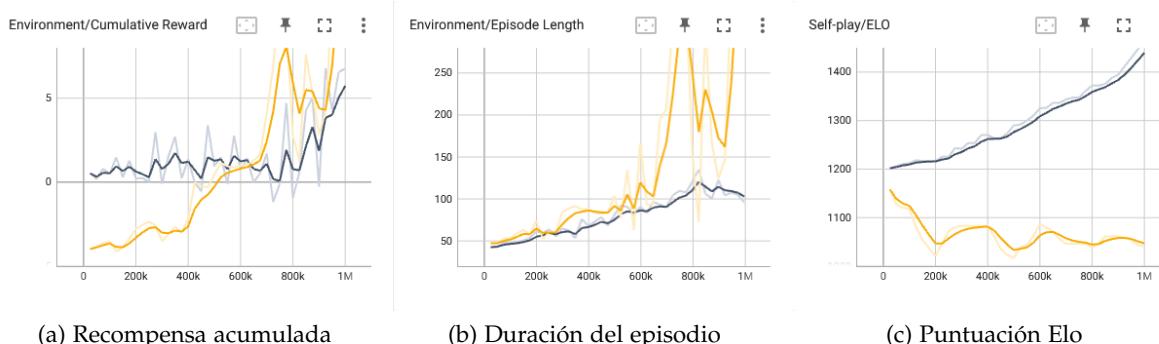


Figura 25: Resultados del experimento 4, en amarillo. (Elaboración propia)

Infiriendo el modelo obtenido en los agentes desde Unity, se puede observar que la mayoría de golpes que realizan son globos. Esto es así porque los agentes ya saben que se trata de la opción más segura para garantizar una devolución constante entre ambos equipos. También cabe mencionar que, pese a que no son agentes muy ofensivos, cuando se trata de dirigirse hacia la pelota y devolverla, actúan bastante bien.

10.1.5. Experimento 5. Influencia de las posiciones clave

Para el último experimento se ha analizado el efecto que tienen las recompensas asociadas a los puntos clave. Como son recompensas que se pueden conseguir de manera continua durante un episodio, estas deben ser lo suficientemente pequeñas como para no sobrepasar las recompensas principales.

Recompensa	Valor
WinningReward	10
LosingReward	-10
ApproachingKeyPositionsReward	0.005
StayingAroundKeyPositionsReward	0.01
HittingBallReward	0

Tabla 21: Función de recompensa del experimento 5. (Elaboración propia)

En los resultados del experimento 5, se puede apreciar una ligera mejora en el rendimiento de los agentes, justamente al final del entrenamiento. Aun así, esta mejora no es tan distinguida como se esperaba.

Mediante las posiciones clave, la idea era que los agentes pudieran ser capaces de aprender mucho antes dónde se deberían posicionar. Es probable que esto no haya sido posible debido a factores como:

- La magnitud de las recompensas, ya que al ser tan pequeñas, los agentes no le dan demasiada relevancia.
- Falta de coherencia con las observaciones, ya que los agentes, aunque ganen estas recompensas, difícilmente las pueden asociar con las posiciones de interés, ya que no disponen información de estas.
- Las circunstancias en las que se obtienen, porque las posiciones de interés quizás son demasiado específicas, y habría que probar a hacer cambios para generalizarlo. Por ejemplo, escalar la recompensa por acercarse a una recompensa, según cuánto se haya acercado realmente dentro de un intervalo de tiempo, penalizar si un agente está demasiado tiempo en una posición poco deseada, etc.

Aunque el uso de las posiciones de interés no haya sido muy efectiva en esta versión, su implementación sirve como base para definir funciones de recompensas más complejas.

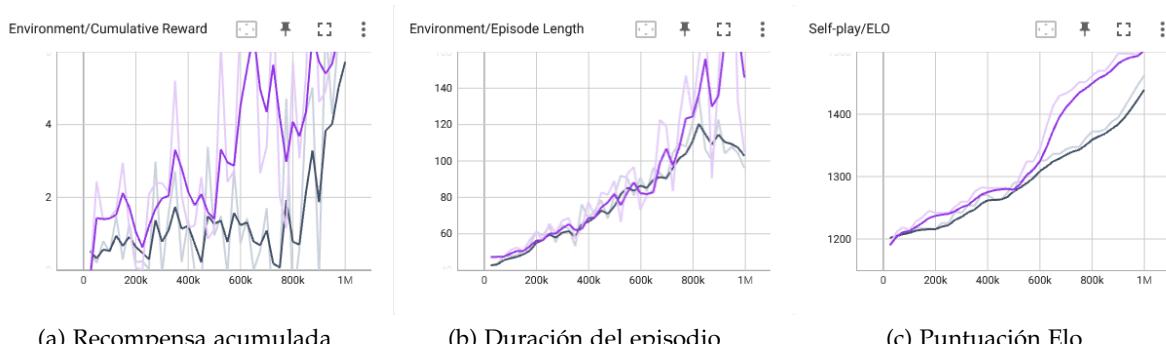


Figura 26: Resultados del experimento 5, en morado. (Elaboración propia)

10.2. Aprendizaje por imitación

La experimentación con el aprendizaje por imitación ha revelado una serie de obstáculos que limitan considerablemente el aprendizaje de los agentes. Dichos obstáculos están relacionados con las dificultades surgidas durante la grabación de demostraciones.

El principal problema es la latencia resultante de la decisión de establecer una conexión cliente-servidor para el intercambio de datos.

Aunque el envío de solicitudes se haya implementado de manera eficiente, esto no sirve de mucho cuando es la propia conexión la que introduce un tiempo de espera. Teniendo en cuenta que el entorno se va ejecutando continuamente, donde el intervalo de tiempo entre un fotograma y otro es mínimo, resulta inviable que los agentes puedan moverse únicamente cada 0.2 segundos, por ejemplo.

Solucionar el problema de la latencia hubiera implicado rediseñar la manera en la que se obtienen los datos. Por ejemplo, se podría haber considerado implementar los árboles decisionales directamente en el entorno de Unity, aunque esto podría causar problemas de rendimiento, afectando a la tasa de fotogramas. Sin embargo, esto no ha sido posible debido a las limitaciones de tiempo.

Aun así, observando cómo se comportaban los agentes guiados por el entrenador, fueron notables algunos problemas que se deberían tener en cuenta de cara a una futura implementación:

- El conjunto de datos, al ser muy escaso teniendo en cuenta que la búsqueda se realiza en un espacio 10-dimensional, es muy probable que los agentes acaben realizando secuencias de acciones de manera cíclica (sobre todo cuando los estados iniciales están predefinidos, como es en el caso de este entorno). Una solución a esto sería, en vez de obtener únicamente la muestra más cercana, que se obtengan $n > 1$ muestras más cercanas, para escoger una acción entre estas.
- Al haber coherencia temporal en los datos, muchas posiciones están muy próximas a otras. Esto hace que los agentes no puedan obtener demostraciones de muchos estados desconocidos, por lo que sería difícil generalizar la política del agente.
- Es importante tener información sobre la velocidad de la pelota, ya que las decisiones son distintas según si la pelota va hacia un lado u otro, o cómo de rápido esté llegando. Este último aspecto sería fácil de incorporar, a partir de los datos de partidos reales, si únicamente se considera la velocidad 2D (ignorando la altura).

Pese a estos problemas, ignorando el hecho de que los agentes realizaban acciones con demasiada lentitud, se podía apreciar que los movimientos eran menos erráticos, ya que en ocasiones eran capaces de acercarse a la pelota y devolverla. Esto ha servido para confirmar que el planteamiento ha sido correcto, por lo que en el futuro se planea entrenar los agentes durante mucho más tiempo y evaluar los resultados, tras optimizar el rendimiento del entrenador y la comunicación entre procesos.

11. Planificación final

11.1. Cambios en la planificación

A continuación se comentan los cambios que se han producido en la planificación, tras haber finalizado el desarrollo del proyecto.

El bloque T2 se había planificado con una duración breve, ya que no pretendía ser un estudio demasiado extenso sobre el estado del arte. Sin embargo, con el estudio de los algoritmos que se pretendían utilizar, surgió la necesidad de profundizar mucho más en la teoría del aprendizaje por refuerzo, por lo que la duración se acabó extendiendo bastante.

Los bloques T4 y T5 se habían planificado anticipando que quizás requerirían una implementación específica para los algoritmos que se querían utilizar. En realidad, el uso de los algoritmos de RL e IL en el entorno de ML-Agents es realmente sencillo, ya que se pueden aplicar en el mismo entorno virtual. Salvo la parte del procesamiento de datos del bloque T5, el resto ha consistido finalmente en hacer distintos experimentos en el entorno virtual. Dados estos cambios, lo ideal sería aumentar las horas del bloque T3, ya que engloba toda la implementación del entorno virtual, y reducir la duración de los bloques T4 y T5.

En el bloque T6, la idea era crear una aplicación con una interfaz gráfica propia que permitiera al usuario final hacer pruebas en el simulador. Sin embargo, esto no era viable ya que para hacer cambios específicos en el entorno virtual es necesario abrir el proyecto desde Unity. Como alternativa, se acabó implementando un modo de depuración, para que el usuario pueda comprobar que todo funcione correctamente durante la ejecución del simulador.

11.2. Planificación final

Tras los cambios producidos en la planificación, se ha actualizado el resumen de tareas y el diagrama de Gantt, los cuales se pueden consultar en la Tabla 22 y el Apéndice A.2, respectivamente.

Los bloques T4, T5 y T6 se han renombrado para que sigan un orden cronológico, ya que el desarrollo de la interfaz de depuración se ha realizado antes de la experimentación. Por lo tanto, el bloque T4 consiste ahora en el desarrollo de la interfaz de depuración, mientras que los bloques T5 y T6 en la experimentación con DRL e IL, respectivamente.

ID	Tarea	Tiempo	Dependencias
T1	Gestión del proyecto	100h	-
T1.1	Alcance	30h	[]
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	-	[]
T1.7	Presentación	15h	[T1.6]
T2	Trabajo previo	100h	-
T2.1	Estudio del estado del arte	80h	[]
T2.2	Familiarización con ML-Agents	20h	[T2.1]
T3	Entorno virtual en Unity	200h	-
T3.1	Diseño del entorno	50h	[T2]
T3.2	Implementación	100h	[T3.1]
T3.3	Testing	50h	[T3.2]
T4	Interfaz de depuración	50h	-
T4.1	Diseño de la interfaz	10h	[T3.1]
T4.2	Implementación	30h	[T4.1]
T4.3	Testing	10h	[T4.2]
T5	Aprendizaje por refuerzo profundo	30h	-
T5.1	Experimentación	30h	[T3]
T6	Aprendizaje por imitación	60h	-
T6.1	Procesamiento de datos	30h	[T3]
T6.2	Experimentación	30h	[T6.1]
-	Total	540h	-

Tabla 22: Resumen de la planificación final. (Elaboración propia)

12. Conclusiones

Estudiar un ámbito tan denso como es el aprendizaje por refuerzo (o la inteligencia artificial en general) puede llegar a ser bastante abrumador, especialmente cuando hay tanta variedad de recursos accesibles a través de Internet, en los que cada libro o artículo propone distintas definiciones o notaciones para un mismo concepto, haciendo que el contraste de información sea confuso. Por este motivo, el enfoque que se le ha dado a este trabajo de fin de grado ha sido que pueda servir como referencia para aquellas personas interesadas en el aprendizaje por refuerzo.

Para ello, en el tránscurso de este proyecto, se ha hecho un recorrido sobre distintos métodos de aprendizaje por refuerzo sin modelos, recogiendo de manera coherente los conceptos teóricos que han ido apareciendo y relacionándolos entre sí. Concretamente, se ha hecho un estudio extenso sobre los métodos basados en el valor y en la política, mostrando algunos ejemplos de algoritmos como Q-learning o REINFORCE para una mejor comprensión, para finalmente entender cómo funcionan los métodos actor-crítico.

Finalmente, para poner en práctica los conocimientos adquiridos y resolver el problema planteado inicialmente, se ha implementado en Unity un entorno de aprendizaje basado en el pádel, donde se han integrado agentes proporcionados por el Unity ML-Agents Toolkit. Estos agentes se definieron en el contexto de aprendizaje por refuerzo, especificando sus espacios de observaciones, acciones y funciones de recompensa, y se entrenaron mediante algoritmos como Proximal Policy Optimization o Curiosity, para dotarles un comportamiento humano y simular las decisiones óptimas que haría un jugador profesional de pádel.

En general, los resultados obtenidos han sido positivos, por lo que se considera que se ha cumplido satisfactoriamente con los objetivos propuestos inicialmente. Aun así, debido a los contratiempos surgidos durante el desarrollo, como consecuencia de la aparición de errores en la implementación, o la inviabilidad de ciertas funcionalidades, algunas pruebas como el uso del aprendizaje por imitación no se pudieron llevar a cabo. Por esta razón, se plantean a continuación algunas propuestas de trabajo futuro.

12.1. Trabajo futuro

Aunque los resultados obtenidos en este proyecto hayan sido satisfactorios, al tratarse precisamente de una primera versión, la cual ha servido sobre todo como introducción al aprendizaje por refuerzo, hay aún mucho margen de mejora en relación al diseño del entorno de aprendizaje. Algunas propuestas de mejora de cara al futuro son:

- Refinamiento de la función de recompensa: un problema con las recompensas que se han definido en este entorno es que son muy sencillas. Los valores de estas recompensas, las cuales son constantes durante todo el entrenamiento, podrían variar en función del tiempo transcurrido, por ejemplo, para incentivar a los jugadores a terminar un punto lo antes posible.

- Aplicar un margen de error en la trayectoria de la pelota: sería interesante que se añadiera en el entorno virtual un margen de error en la trayectoria de la pelota. Por ejemplo, la trayectoria se podría desviar en función de parámetros como la velocidad de la pelota o un tiempo de reacción del agente. Así, durante el entrenamiento, los agentes deberían aprender a adoptar estrategias que traten de minimizar este error, aprendiendo, por ejemplo, a dejar botar primero la pelota. Una precisión perfecta es poco realista, ya que en realidad hay muchos factores que influyen en la ejecución de un golpeo.
- Reimplementación del proceso de grabación de demostraciones: la aplicación de técnicas de aprendizaje por imitación puede resultar muy útil para facilitar el entrenamiento de agentes. El módulo utilizado para grabar demostraciones se podría mejorar reduciendo la latencia en la comunicación. En particular, se podría probar de implementar todo el proceso en el mismo entorno de Unity, haciendo que la obtención de datos sea mucho más rápida.
- Experimentación con otros espacios de observaciones y/o acciones: en el entorno de aprendizaje que se ha implementado, el tamaño del espacio de observaciones y acciones se podría aumentar, planteando un problema aún más complejo. Por ejemplo, se podrían añadir más tipos de golpeo, como la contrapared o las dejadas, y aumentar las dimensiones de la cuadrícula de posiciones. En cuanto al espacio de observaciones, sería interesante añadir otras variables que permitan al agente asociar algunas recompensas a ciertas observaciones en concreto. Por ejemplo, se podrían añadir las posiciones de interés de devolución para que los agentes puedan hacer una mejor predicción sobre la trayectoria de la pelota.
- Rediseño de la interfaz de depuración: aunque la interfaz de depuración es funcional, al mostrar todos los datos como un conjunto, a la larga se complicaría la depuración de nuevas variables. Por lo tanto, se propone dividir la interfaz de depuración en diversos menús, manipulables por el usuario, para que el seguimiento las variables sea más cómodo.

Referencias

- [1] R. S. Sutton y A. G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: <http://incompleteideas.net/book/the-book-2nd.html>.
- [2] Unity ML-Agents Toolkit. URL: <https://github.com/Unity-Technologies/ml-agents>.
- [3] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar y D. Lange. «Unity: A general platform for intelligent agents». En: *arXiv preprint arXiv:1809.02627* (2020). URL: <https://arxiv.org/pdf/1809.02627.pdf>.
- [4] Zuci Systems. *What is Agile Methodology?* URL: <https://www.zucisystems.com/software-development/agile-methodology/>.
- [5] Indeed. URL: <https://es.indeed.com/career/salaries>.
- [6] I. Goodfellow, Y. Bengio y A. Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [7] OpenAI Spinning Up Documentation - Part 2: Kinds of RL Algorithms. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.
- [8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra y M. Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].
- [9] Policy Gradient Algorithms. URL: <https://lilianweng.github.io/posts/2018-04-08-policy-gradient/>.
- [10] Introduction to Policy Gradients. URL: <https://medium.com/nerd-for-tech/reinforcement-learning-introduction-to-policy-gradients-aa2ff134c1b>.
- [11] Actor-Critic Methods, Advantage Actor-Critic (A2C) and Generalized Advantage Estimation (GAE). URL: <https://avandekleut.github.io/a2c/#advantage-function>.
- [12] Unity Documentation. URL: <https://docs.unity.com/>.
- [13] A. Cohen, E. Teng, V.-P. Berges, R.-P. Dong, H. Henry, M. Mattar, A. Zook y S. Ganguly. *On the Use and Misuse of Absorbing States in Multi-agent Reinforcement Learning*. 2022. arXiv: 2111.05992 [cs.LG].
- [14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford y O. Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].
- [15] D. Bick. *Towards Delivering a Coherent Self-Contained Explanation of Proximal Policy Optimization*. 2021. URL: https://fse.studenttheses.ub.rug.nl/25709/1/mAI_2021_BickD.pdf.
- [16] D. Pathak, P. Agrawal, A. A. Efros y T. Darrell. *Curiosity-driven Exploration by Self-supervised Prediction*. 2017. arXiv: 1705.05363 [cs.LG].
- [17] J. Ho y S. Ermon. *Generative Adversarial Imitation Learning*. 2016. arXiv: 1606.03476 [cs.LG].
- [18] A. Cohei. *Training intelligent adversaries using self-play with ML-Agents*. 2020. URL: <https://blog.unity.com/engine-platform/training-intelligent-adversaries-using-self-play-with-ml-agents>.
- [19] Unity. *Training Configuration File - Unity ML-Agents Toolkit*. URL: <https://unity-technologies.github.io/ml-agents/Training-Configuration-File/>.
- [20] J. L. Ji. *Entorno virtual de pádel para el aprendizaje por refuerzo*. URL: <https://github.com/jialongjq/tfg>.
- [21] M. Javadiha, C. Andujar, E. Lacasa, A. Ric y A. Susin. «Estimating Player Positions from Padel High-Angle Videos: Accuracy Comparison of Recent Computer Vision Methods». En: *Sensors* 21.10 (2021). URL: <https://www.mdpi.com/1424-8220/21/10/3368>.

Apéndices

A. Diagramas de Gantt

A.1. Diagrama de Gantt de la planificación inicial

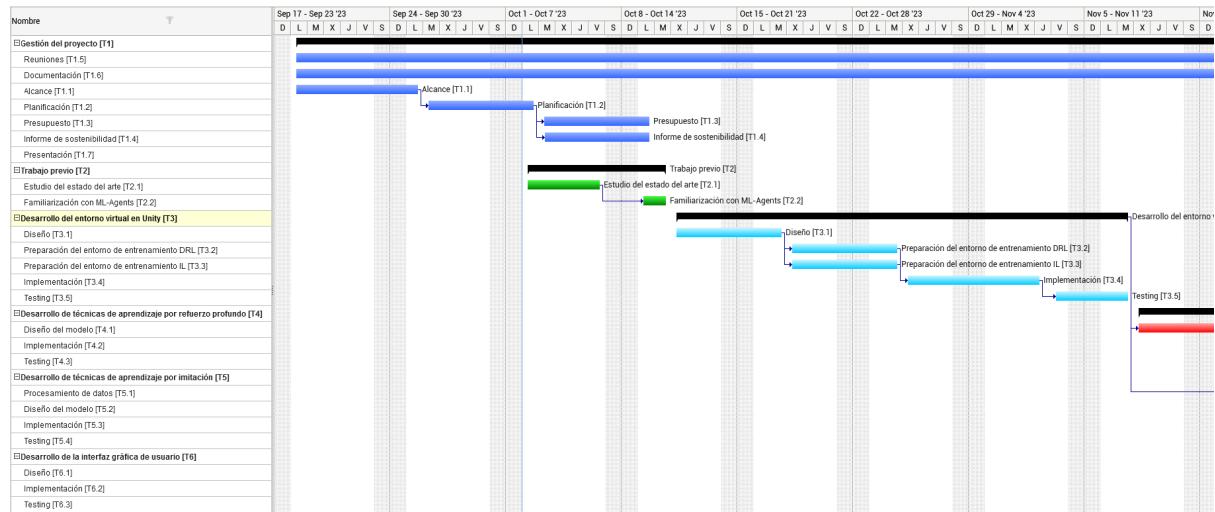


Figura 27: Diagrama de Gantt de la planificación temporal, parte 1 de 2. (Elaboración propia)



Figura 28: Diagrama de Gantt de la planificación temporal, parte 2 de 2. (Elaboración propia)

A.2. Diagrama de Gantt de la planificación final

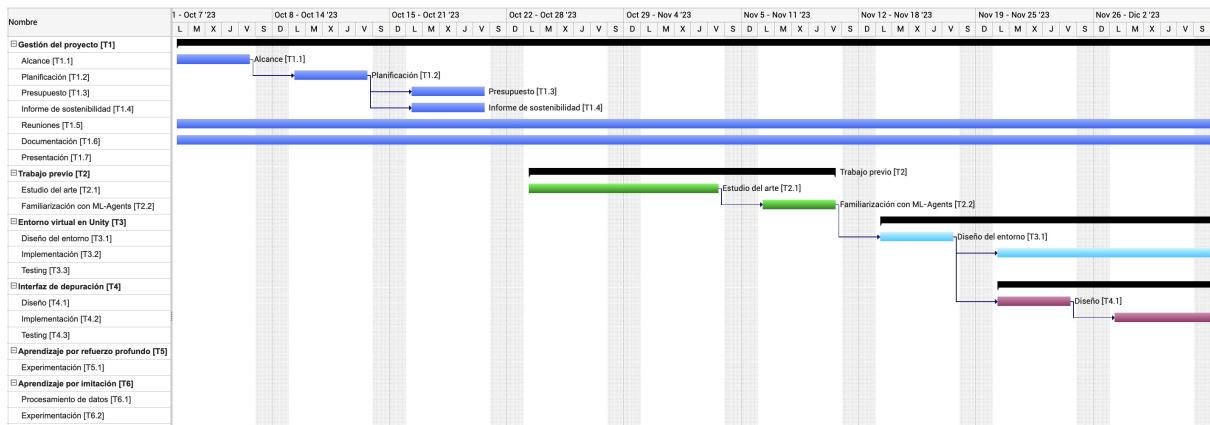


Figura 29: Diagrama de Gantt de la planificación final, parte 1 de 2. (Elaboración propia)

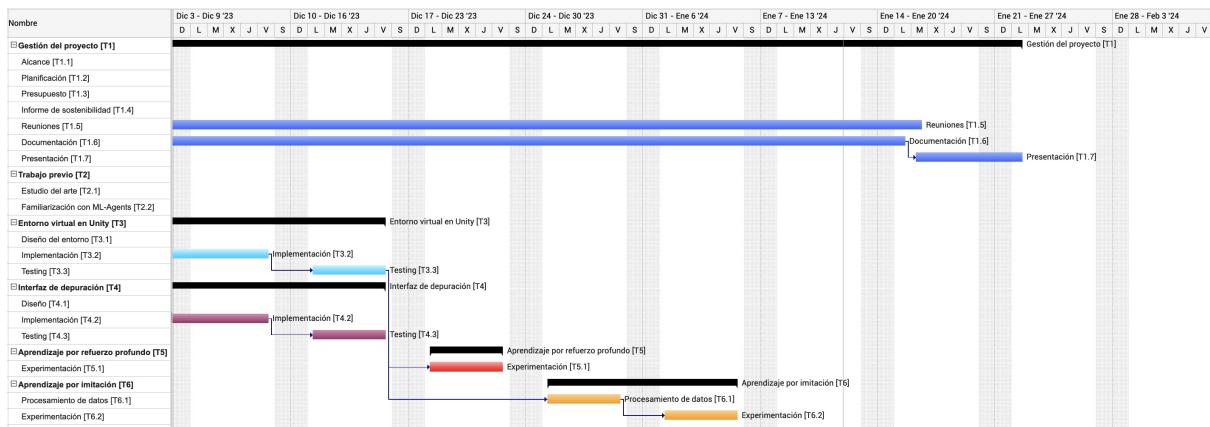


Figura 30: Diagrama de Gantt de la planificación final, parte 2 de 2. (Elaboración propia)

B. Configuraciones del entrenamiento

B.1. Configuración del entrenamiento mediante PPO

```
behaviors:  
  PadelAgent:  
    trainer_type: ppo  
    summary_freq: 25000  
    max_steps: 1000000  
    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: 25000  
      team_change: 200000  
      window: 15  
      play_against_latest_model_ratio: 0.5  
      initial_elo: 1200.0
```

B.2. Configuración del entrenamiento mediante PPO + Curiosity

```
behaviors:  
  PadelAgent:  
    trainer_type: ppo  
    summary_freq: 25000  
    max_steps: 1000000  
    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  
      curiosity:  
        gamma: 0.95  
        strength: 0.05  
        learning_rate: 1e-4  
      network_settings:  
        hidden_units: 128  
        num_latters: 2  
    self_play:  
      save_steps: 50000  
      swap_steps: 25000  
      team_change: 200000  
      window: 15  
      play_against_latest_model_ratio: 0.5  
      initial_elo: 1200.0
```