eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Computación

Aprendizaje por refuerzo en Unity para entrenamiento de varios agentes en entornos competitivos

Unai Salas

Dirección
Borja Calvo Molinos

2022

Esker onak / Agradecimientos

En primer lugar, agradecerle a mi tutor Borja por todas las recomendaciones y la ayuda que me ha facilitado desde los inicios de este proyecto y la gran paciencia que ha tenido conmigo durante estos meses.

Me gustaría agradecer a mi familia por ayudarme a llegar hasta aquí y por todo el apoyo y la ayuda que me han dado desde siempre. Si he llegado hasta aquí es gracias a vuestro esfuerzo.

Y por último, pero no menos importante, a los amigos que he ido haciendo durante estos años. Especialmente a mis amigas de la cuadrilla, con las que tanto tiempo he pasado en esta ciudad y en llamadas a altas horas de la madrugada. Sois una parte fundamental en todo esto.

Resumen

En este proyecto se estudia el aprendizaje por refuerzo (reinforcement learning, abreviado como RL a lo largo del documento) de forma competitiva entre dos agentes con el fin de observar las interacciones entre ambos, junto con la viabilidad y eficiencia de diferentes algoritmos.

El proyecto está formado por tres fases incrementales a la que se enfrentan los agentes.

- La primera fase será una fase simple, diseñada para el desarrollo completo de un agente base, con todas las características funcionales con las que contará en las siguientes fases. El agente en esta fase aprenderá a llegar al objetivo en un escenario simple.
- En la segunda fase dos agentes se enfrentan entre ellos en un escenario simple con el objetivo de llegar antes que el otro agente al objetivo.
- En la tercera y última fase se introduce un laberinto generado de forma aleatoria para que los agentes puedan competir entre ellos como en la segunda fase, pero en un entorno que no conocen y que en cada ejecución cambia.

Los agentes fueron capaces de obtener unos resultados favorables tras el entrenamiento para las dos primeras fases, las cuales usaban entornos simples, mostrando incluso comportamientos e interacciones entre ellos que no eran esperados. Sin embargo, en la tercera fueron capaces de resolver laberintos de 4x4 celdas pero no pudieron hacer frente a los laberintos de 8x8 celdas.

Índice de contenidos

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

Desde los años 90 hasta hoy día los videojuegos han acompañado al mundo como una de las principales formas de ocio. Desde el que pudo considerarse el primer videojuego creado por William Nighinbottham usando un osciloscopio de laboratorio en el año 1958, llamado Tennis for Two[1], hasta los actuales juegos con gráficos elaborados y depuradas mecánicas.

En la actualidad, los videojuegos son una de las opciones de ocio predilectas en España. Según afirma el anuario de AEVI (Asociación Española de Videojuegos), basados en los datos proporcionados por GameTrack, llegaron a facturarse 1.795 millones de euros en 2021 y se le dedica una media de 8,1 horas semanales en España[1].

El sector no hace más que crecer gracias a los medios de transmisión (Twitch, HitBox, etc.) que han ido apareciendo y a la tecnología que sigue desarrollándose y ayudando a crear videojuegos nuevos y que se adaptan a las necesidades y gustos de cada persona o grupo de personas.

Por otra parte, otro de los sectores que también se está avanzando año tras año es el campo de la inteligencia artificial (IA). Este tuvo su nacimiento en 1950 de la mano de Alan Turing gracias al "Test de Turing", al que seguiría Arthur Samuel, el cual en 1952 creó un programa de ordenador capaz de aprender mientras jugaba a las damas.

La colaboración de ambos sectores en la creación de videojuegos que emplean IAs capaces de ayudar a los jugadores en sus tareas o de poner a prueba sus habilidades abre un abanico de posibilidades. La idea principal de la IAs en los videojuegos es, tal y como dice Laura E. Shummon, maximizar la participación de los jugadores y que disfruten del proceso de aprendizaje[2]. Aun así, pueden entrenarse IAs lo suficientemente sofisticadas como para que supongan un verdadero reto para los jugadores más experimentados o que busquen enfrentarse a estrategias que en un principio no serían ejecutadas por personas [2].

Puesto que existen bastantes algoritmos de aprendizaje por refuerzo[3][4], utilizarlos para desarrollar IAs capaces de enfrentar a jugadores podría resultar interesante de cara a

[1] https://www.vidaextra.com/industria/el-padre-de-los-videojuegos-william-higinbotham-su-historia-su-leyenda

[2] https://towardsdatascience.com/artificial-intelligence-in-video-games-3e2566d59c22

la obtención de nuevos resultados. Un claro ejemplo de estas IAs es AlphaGo[5], que fue entrenada mediante aprendizaje por refuerzo y fue capaz de derrotar en varias ocasiones al que era el campeon mundial, Lee Sedol [3]. Puesto que el enfrentamiento de IA se ha realizado para escenarios de tablero, se plantea el desarrollo de IAs empleando aprendizaje por refuerzo con escenarios más complejos, como podría ser un laberinto.

1.1. Conceptos básicos

Para poder comprender el proyecto y sus elementos, primero se deben comprender algunos conceptos básicos que se irán nombrando a lo largo del documento o que tienen una relación muy directa con ellas y por tanto puede ser aconsejable conocerlos.

1.1.1. Aprendizaje automático

El campo del aprendizaje automático (o *Machine Learning* en inglés, abreviado como ML durante el documento) es un área de la IA que ha cobrado fuerza en los últimos años. El ML se podría definir como un campo de estudio que otorga a los ordenadores la habilidad de aprender sin ser explícitamente programados. Es decir, que este tipo de programas aprende mediante datos que se le proporcionan como entrada con el objetivo de hacer predicciones, permitiendo que se puedan realizar tareas específicas de forma autónoma[6].

Los algoritmos del ML pueden clasificarse de diversas maneras y existe una gran cantidad de paradigmas de aprendizaje [7] y los principales son el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo.

1.1.2. Aprendizaje supervisado

El aprendizaje supervisado tiene como objetivo aprender un clasificador que se aproxime de forma fiable a una tarea de clasificación inferida a partir de un conjunto de ejemplos del problema de interés. Una vez aprendido se utiliza en una fase de predicción para anticipar la etiqueta de ejemplos que no la tengan. El término supervisado en el aprendizaje automático, por tanto, indica que los ejemplos usados en la etapa de aprendizaje siempre cuentan con la etiqueta real de su clase[7].

Existe también dentro del aprendizaje supervisado el campo de la regresión, el cual busca predecir un valor continuo dados los valores de las variables predictivas.

1.1.3. Aprendizaje no supervisado

El aprendizaje no supervisado, en cambio, utiliza datos de entrenamiento sin etiquetas, sin clasificación o sin categorizar. El objetivo principal de este tipo de aprendizaje es descubrir los patrones que se dan en estos datos para después poder agruparlos y catalogarlos. Los datos empleados para este tipo de entrenamiento no requieren de supervisión humana para ser etiquetados, por lo que hace falta poco esfuerzo para poder crearlos[8].

[3] www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol

1.1.4. Aprendizaje por refuerzo

El aprendizaje por refuerzo es un área del ML en el que se emplea un agente artificial que toma acciones en un entorno con el objetivo de maximizar todo lo posible la recompensa que recibe. Las acciones que toma influyen en el entorno y el agente recibe información del entorno y la recompensa que se le proporciona. Tras analizar estas observaciones, el agente decide la acción futura con el objetivo de maximizar la recompensa final. Como es deducible, este tipo de aprendizaje está diseñado para casos en los que solo se puede aprender a través de interacciones con el entorno. Este proceso puede verse reflejado en la Figura 1.1.

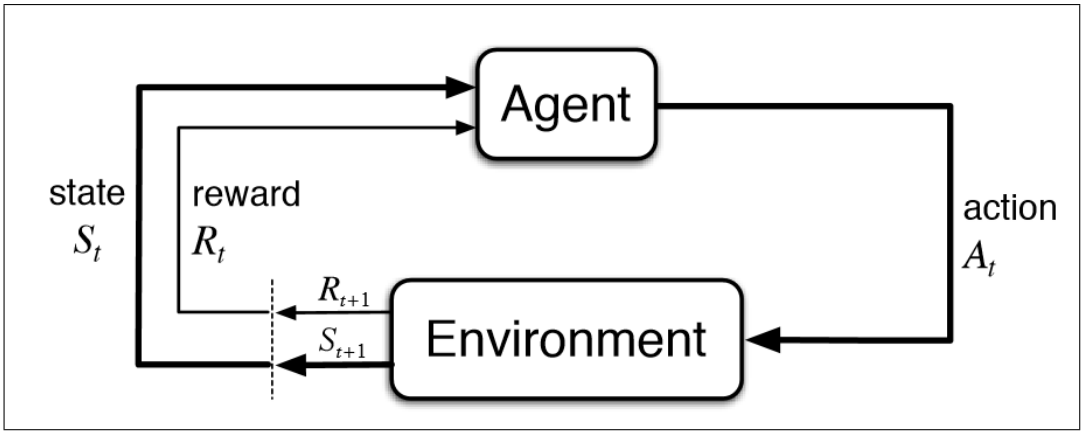


Figura 1.1: Diagrama del aprendizaje por refuerzo. Sutton & Barto (1998)

Dentro del RL existen diferentes elementos clave con los que se debe estar familiarizado. A continuación se describirán brevemente estos elementos:

1.1.4.1. El agente

Es el sujeto del aprendizaje por refuerzo. El agente tendrá una serie de observaciones sobre su entorno y sus diferentes estados (s_t) y en base a lo que interprete, ejecutará la acción (a_t) que sea conveniente.

En la Figura 1.1 se ve un agente, pero podría darse el caso de que más de un agente interactuase con el entorno de forma simultánea.

1.1.4.2. El entorno

Es la representación de un problema al que se enfrenta el agente. Se considera entorno todo aquello que no forma parte del agente. Este entorno responde a las acciones que realiza el agente con consecuencias, en forma de transición entre estados y recompensas.

La forma en la que el entorno reacciona a acciones del agente puede ser o no conocido por el agente, dividiendo estas situaciones en las conocidas como *model-based* y *model-free*. En el primer caso, el agente conoce el entorno y sus modificaciones. En el segundo caso, el agente no conoce el entorno o tiene una información incompleta sobre este.

1.1.4.3. El estado

El entorno está representado por un conjunto de variables relacionados con el problema y ese conjunto de variables y sus posibles valores se denomina espacio de estados (S). Por tanto, un estado es un conjunto de valores que toman estas variables y lo representaremos con la letra s. Los posibles estados en los que se encuentre el agente en cada instante t con respecto al entorno debe cumplir la condición $s_t \in S$.

El agente puede obtener información sobre el estado del entorno y a esta se le llama observación.

1.1.4.4. Acciones

Para cada estado, el agente debe elegir realizar una acción del conjunto de acciones (A) que tiene disponible para cada estado. Estas acciones tienen influencia sobre el entorno y este puede cambiar de estado en reacción a dichas acciones del agente. Las acciones realizadas por el agente las representaremos con la letra a y estarán previamente definidas en el conjunto de acciones, por lo que $a_t \in A$.

1.1.4.5. Recompensa

Las recompensas mencionadas anteriormente son las responsables de proporcionar al agente una respuesta directa a las acciones realizadas, siendo una retroalimentación cuantitativa para este. Estas recompensas pueden ser positivas o negativas, dependiendo de lo que se espera lograr con ellas. Estas recompensas serán representadas como r_t.

Las recompensas son entregadas por el entorno a través de una función de recompensa. Esta puede ser representada como $R(s, a)$, cuando es basada en una acción a realizada en un estado s, o también representada como $R(s, a, s')$, cuando además se considera la transición del estado s al estado s'.

La definición de unas recompensas adecuadas para el problema es una de las tareas más difíciles de su modelización, ya que se está tratando de enseñar al agente cuales son las acciones que debe tomar.

Para poder entender mejor lo que son las recompensas, podemos compararlo con el adiestramiento de un animal. Cuando el animal realiza bien la tarea que la persona que lo adiestra le ha asignado, obtendrá un premio, por lo general una golosina o algo que al animal le resulte atractivo. En el caso de que el animal haga cosas que no debería, tendrá que ser regañado, lo que puede ser comparado con una recompensa negativa. De esta forma, el animal asocia sus acciones o comportamientos a diferentes efectos.

1.1.4.6. Episodios

Los agentes y el entorno realizan interacciones en el tiempo, a lo cual llamaremos *time step (o step)* y a cada conjunto de paso desde la aparición del agente hasta que realiza la tarea o se resuelve un final no natural (llegar al máximo de pasos permitido) lo llamaremos **episodio**. Estos pasos o momentos serán representados con la letra t mientras no sea el momento final del episodio, en cuyo caso será representado como T.

En cada step, la combinación de estado, nuevo estado, acción y recompensa forman una tupla que usa el agente para aprender, y a esta le llamaremos ***experiencia***, formalmente expresado como $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$, donde

- s_t es el estado en el que se encuentra el agente en el step t
- a_t es la acción que el agente realiza en el step t estando en el estado s_t
- r_{t+1} es la recompensa que obtiene el agente tras realizar la acción a_t en el estado s_t
- s_{t+1} es el estado al que pasa el agente tras ejecutar la acción a_t en el estado s_t

En ocasiones, queremos expresar una secuencia de estados, acciones y recompensas entre agente y entorno. A cada una de las interacciones de estas secuencias las llamaremos **transiciones** y se define con la tupla (s_t, a_t, r_{t+1}) en el step t.

1.1.4.7. Factor de descuento

Los agentes tienen como objetivo maximizar las recompensas acumuladas a lo largo de un episodio, a lo cual llamaremos retorno, denominado como G_t. Sin embargo, el agente no sabe cómo de importante es obtener recompensas a lo largo del tiempo. Para ello se introducirá el parámetro **factor de descuento**, denominado con la letra γ, que emplea valores en un rango entre 0 y 1. Con ello se ajusta la importancia de las recompensas haciendo que si el valor es cercano a 0 priorice las recompensas actuales o a corto plazo, mientras que si es cercano a 1, la diferencia temporal no tendrá mucho efecto.

Sabiendo de la existencia del factor de descuento, se puede definir el retorno con descuento, expresado como G_t en el step t como:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \gamma^4 r_{t+5} + ... = \left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \right]$$

Este retorno con descuento facilitará al agente en la elección de acción para obtener la mejor recompensa. Aun así, puesto que el agente no es capaz de pronosticar el retorno con descuento esperado, necesita hacer predicciones. Es entonces donde interviene la política.

1.1.4.8. Política

La política es el plan que ejecuta el agente para determinar las acciones a tomar cuando está en un estado del entorno. Esta política va mejorando mientras se realizan episodios de entrenamiento. La política se determina con el símbolo π. Para encontrarlas y compararlas, los agentes examinan como de positivos son los estados y las acciones que pueden realizar en relación con sus objetivos, basándose en una función de valor.

Existen dos funciones de valor: la función de valor del estado y la función del valor del estado-acción.

La función de valor del estado (también conocida como **función V**) mide el retorno con descuento posible si se empieza desde un estado determinado. De la misma forma, también se puede interpretar la función de valor de estado como un valor para cada acción desde un estado, al cual llamaremos **función de valor de la acción** o **función Q**.

Si utilizamos la política para obtener el siguiente estado definimos la siguiente función:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi \left[\sum_{j=0}^{T} \gamma^j r_{t+j+1} | S_t = s \right]$$

La función del valor de la acción (o **función Q**) define el valor de la acción en un estado teniendo una política determinada. Si describimos el valor del retorno con descuento G_t con la acción a, desde el estado s en el step t con la política π obtenemos la siguiente expresión:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{j=0}^{T} \gamma^j r_{t+j+1} | S_t = s, A_t = a \right]$$

Sabiendo ambas funciones de valor se puede obtener la función de ventaja, la cual estima cómo de buena es una acción en comparación con la acción promedio de un estado:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

En caso de que la ventaja sea positiva, la acción realizada es buena y se puede obtener una recompensa buena tras la acción. En caso contrario, si la ventaja es negativa, la acción realizada no genera una buena recompensa por lo que las probabilidades de que suceda serán menores.

La política óptima es aquella política que hace que el agente tome la mejor acción en cada estado para obtener a mayor recompensa posible. Esta es representada como:

$$Q^*(s, a) = \max Q_\pi(s, a)$$

1.2. Algoritmos

Existen múltiples algoritmos y métodos para el estudio del aprendizaje por refuerzo profundo, pero para este caso particular se ha decidido estudiar los resultados que generaban cuatro de ellos y continuar con aquel/aquellos que obtenga los mejores resultados.

Los algoritmos que se van a comprobar se pueden dividir en dos grupos: los *on-policy* y los *off-policy*. Los algoritmos *on-policy* interactúan con el entorno y la política para realizar una mejora de la misma. Sin embargo, los algoritmos *off-policy* hacen uso de la experiencia de los agentes con el entorno para mejorar la política. Tanto PPO como SARSA son algoritmos *on-policy*, mientras SAC y Q-learning son algoritmos *off-policy*.

1.2.1. PPO

El algoritmo PPO (Proximal Policy Optimization) [9] es un algoritmo *on-policy*, sucesor del algoritmo TRPO (Trust Region Policy Optimization)[10]. El algoritmo PPO tiene como objetivo maximizar las recompensas esperadas modificando la política mediante la búsqueda

de una función que haga de límite inferior de la que se desea obtener y actualizarla de forma ascendente, haciendo uso de un paso de gradiente ascendente. Es decir, el objetivo es actualizar la política de forma gradual y sabiendo que va a mejorar.

Este algoritmo es altamente costoso, ya que implica calcular una derivada de segundo orden y su inversa, por lo que para buscar su viabilidad se utilizan restricciones.

En PPO se define una relación de probabilidad entre la nueva política y la anterior, a la cual se le llamará $r(\theta)$ (siendo θ el parámetro de la política) y que se define como:

$$r(\theta) = \frac{\pi_\theta(a \mid s)}{\pi_{\theta old}(a \mid s)}$$

Siendo la función objetivo del algoritmo TRPO (siendo este antecesor de PPO)

$$J(\theta)^{TRPO} = E\left[r(\theta)\hat{A}_{\theta old}(s,a)\right],$$

se puede formular la función objetivo de PPO como:

$$J^{CLIP}(\theta) = \mathbb{E}\left[\text{mín}\left(r(\theta)\hat{A}_{\theta_{\text{antigua}}}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{antigua}}}(s,a)\right)\right]$$

donde

- θ es un parámetro de la política.
- $\mathbb{E}$ denota la esperanza empírica.
- La función clip trunca el índice de política entre el rango $1-\epsilon, 1+\epsilon$, donde ϵ determina como de alejada puede estar una política de la anterior.
- $\hat{A}$ denota la ventaja estimada

Tal y como se comentó anteriormente, una ventaja positiva indica que la acción tomada es buena y una ventaja negativa indica lo contrario. Para PPO, independientemente del valor de la ventaja, se asegura que la actualización no sea demasiado grande.

1.2.2. SAC

El algoritmo SAC (Soft Actor Critic)[11] es otro algoritmo de aprendizaje por refuerzo, pero que a diferencia de PPO, es *off-policy*. Forma parte de los métodos *actor-critic*, los cuales son un paradigma que se caracteriza por usar dos redes neuronales para su entrenamiento. Cada una de estas redes neuronales tiene una función con respecto a la otra: la primera hace de actor y da la orden al agente para que realice determinadas acciones mientras la segunda toma el papel de crítico, evaluando las acciones que ha decidido tomar la primera. Estos métodos actualizan las políticas tras las decisiones del actor y no tras analizar el resultado final del episodio.

Este algoritmo se basa en la entropía máxima de RL y tiene como objetivo encontrar la política óptima que maximiza la recompensa esperada y la entropía[12] [3]. Tener una entropía alta en la política anima a realizar una exploración con probabilidades muy similares para la realización de cada acción disponible. De esta forma el modelo no colapsara con acciones repetitivas, pudiendo ocasionar una inconsistencia en la función aproximada de Q.

El objetivo de entropía puede ser representado como

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t} r\left(\mathbf{s}_t, \mathbf{a}_t\right) - \alpha \log\left(\pi\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\right]$$

donde α corresponde a la importancia relativa de la entropía con respecto a la recompensa. Es así que la política maximiza el rendimiento esperado junto con la entropía.

Dentro de este algoritmo existen diferentes formas de optimización a través de tres funciones. Dentro de estas tres funciones, se puede encontrar una función de valor de estado V parametrizada por ψ, una función Q suave parametrizada por θ y una función de política parametrizada por ϕ. Para obtener más información sobre estas y profundizar sobre este algoritmo se recomienda la lectura del estudio [13].

1.2.3. Q-Learning

El algoritmo Q-Learning[14] [4] es un algoritmo *off-policy*, o lo que es lo mismo, usa una política distinta para aprender la política óptima, por lo que se realizará una diferenciación entre la política empleada para aprender y la que está aprendiendo. Esto se traduce a que la estimación de la función de valor no depende de la acción escogida, y por tanto de la política.

Para entender el funcionamiento de este algoritmo se debe conocer la existencia de la matriz Q. Esta matriz guarda para cada combinación de estados y acciones un valor esperado, siendo las acciones las columnas y los estados las filas. Este valor esperado corresponde a la recompensa que se espera obtener tras realizar una acción determinada en un estado concreto. Esto se traduce a la recompensa que se debería recibir si, tras realizar esa acción en ese estado concreto, todas las decisiones son acertadas, la recompensa máxima que se puede obtener para ese estado.[15]

Dicha matriz se inicializa con valores nulos. Siendo m el número de estados y n el número de acciones, entonces:

$$Q_{t=0}(s,a) = 0 \qquad \forall s \in [1,\ldots,m]; \forall a \in [1,\ldots,n]$$

Para realizar la actualización del valor esperado en cada acción por estado se usa la siguiente fórmula, donde $Q(s,a)$ es el valor Q actual, α es la tasa de aprendizaje, $R(s,a)$ es la recompensa por realizar la acción a en el estado s, γ es el factor de descuento y $\max Q'\left(s',a'\right)$ es la recompensa máxima esperada:

$$Q_{t+1}(s,a) = Qt(s,a) + \alpha\left[R(s,a) + \gamma \max Q\left(s',a'\right) - Qt(s,a)\right]$$

Tanto la tasa de aprendizaje como el factor de descuento son hiper parámetros que deben ser configurados antes de realizar cualquier aprendizaje, teniendo que ser valores entre 0 y 1.

1.2.4. SARSA

El algoritmo SARSA[16] es un algoritmo variado del algoritmo Q-Learning y un algoritmo de aprendizaje por diferencia temporal. A diferencia del algoritmo Q-Learning, este algoritmo es *on-policy*, por lo que la estimación de función de valor depende de la política y por tanto, de la explotación. El nombre está directamente ligado a las variables empleadas para la actualización de valor (s, a, r, s', a').

Para la actualización de SARSA se usa el valor de la función para la acción escogida en el estado, a diferencia de Q-Learning, donde se usaba el valor máximo de Q en el estado s':

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \cdot \left[R(s,a) + \gamma \cdot Q\left(s', a'\right) - Q_t(s,a)\right]$$

Desde un inicio, este método comienza con una política y una función Q generadas de forma aleatoria, que irán se actualizando a medida que se avance con el entrenamiento. La actualización en este algoritmo se realiza en cada transición de un estado s_t a un estado s_{t+1}. Tras finalizar un episodio, se seleccionan las acciones con mayor valor en cada estado para actualizar la política y que formen parte de ella. Aun así, puesto que todos los pares estado-acción deben ser visitados, se utiliza una política ϵ-greedy, la cual asegura que la probabilidad de usar todas las acciones sea mayor que 0[15][6].

1.3. Aprendizaje por refuerzo profundo

Una variante del RL que tiene muy buenos resultados es el Aprendizaje por Refuerzo Profundo. La idea principal detrás de este concepto es la integración de las redes neuronales en el aprendizaje por refuerzo. Esta integración aporta gran utilidad ya que el aprendizaje por refuerzo es útil cuando se trata de entornos pequeños, pero resulta impensable para espacios más grandes como podría ser el ajedrez, cuya complejidad del árbol de búsqueda de 10^{120} de acuerdo a lo calculado por Claude Shannon [17]. En este apartado se hará una introducción al Aprendizaje Profundo y su utilidad en el RL.

1.3.1. Redes Neuronales

Las Redes Neuronales son una herramienta de modelado estadístico que toma como inspiración el cerebro humano, que permiten modelar relaciones no lineales entre entradas y salidas. Estas están formadas por unas neuronas artificiales que en la literatura se conocen como perceptrones. Estos perceptrones simulan el funcionamiento de una neurona "artificial", teniendo entradas y salidas numéricas.

En la Figura 1.2 se muestra una representación visual del perceptrón, el cual tiene varias señales de entrada (x_n), cada una con un peso asignado (w_i) y que realiza una suma ponderada de ellas (Z) junto con la constante que se aplica llamada *bias* (b). Una vez se ha obtenido esta suma ponderada, se pasa por la función de activación (f), cuyo objetivo es el

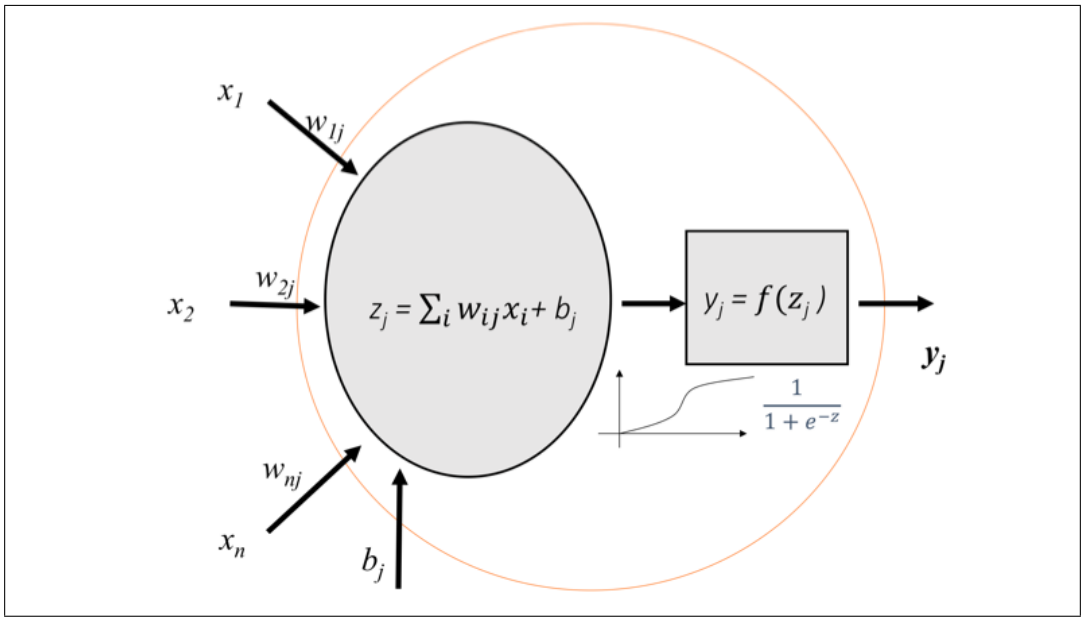


Figura 1.2: Esquema de perceptrón

de modificar el valor resultado o impone un límite que se debe sobrepasar para poder seguir a otra neurona. En cualquiera de los casos se obtiene un resultado de esta función (y).

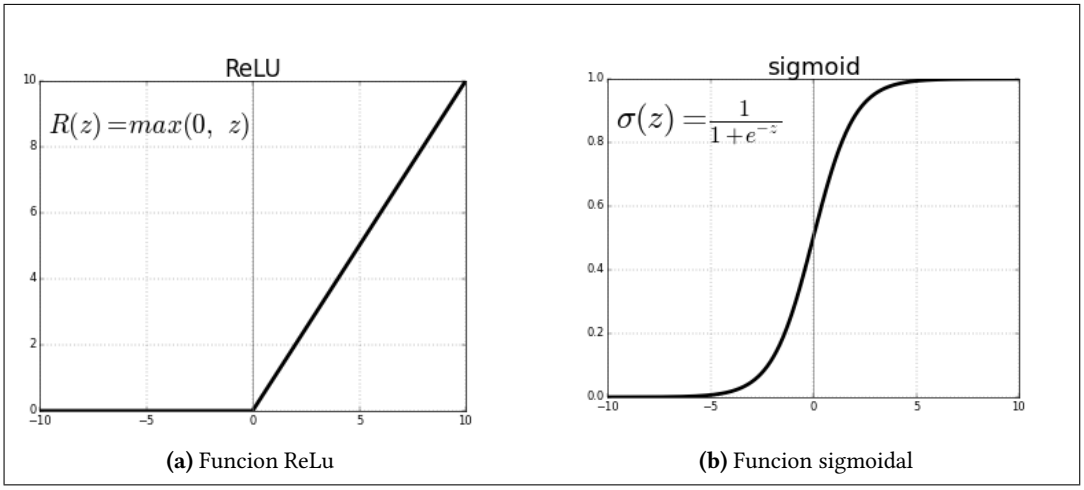
Existen múltiples tipos de funciones de activación, y se usan dependiendo del problema con el que se esté lidiando o con el objetivo de obtener unos resultados concretos. La función que puede verse en la Figura 1.3a transforma el valor que obtiene al rango $[0, \infty]$, mientras que la función sigmoidal de la Figura 1.3b transforma el valor al rango $[0, 1]$.

Estos perceptrones se organizan en capas, conectando cada perceptrón con los perceptrones de las capas anteriores y posteriores. Cada capa realiza la interpretación de un patrón de datos concreto, por lo que la capacidad variará dependiendo de la cantidad de capas que se tengan. En la literatura, las redes neuronales que tienen una capa de entrada, una o más capas compuestas por perceptrones (capas ocultas) y una capa de salida se conocen como perceptrón multicapa. Por lo general cuando se hace una referencia a aprendizaje profundo, el modelo de red neuronal está compuesto por varias capas ocultas. En la Figura 1.4 puede verse el esquema representado de forma visual.

1.4. Unity

Unity es uno de los motores gráficos actuales en el mercado de los videojuegos. Empleado por muchas empresas a la hora de realizar múltiples videojuegos, cuenta con gran renombre en el sector no solo entre los desarrolladores, sino también entre los jugadores. Fue creado por Unity Technologies en el año 2001 y permite la realización de videojuegos tanto en 2D como en 3D, con la capacidad de exportar a múltiples plataformas.

Una de las principales ventajas de este motor es que cuenta con una interfaz gráfica muy sencilla de usar, tal y como puede verse en la Figura 1.5, que cuenta con mucho potencial y que facilita el desarrollo con múltiples ayudas al desarrollador. Además, cuenta con una gran comunidad de desarrolladores que crea y comparte multitud de assets, pudiendo



(a) Funcion ReLu **(b)** Funcion sigmoidal

Figura 1.3: Funciones de activación más frecuentes

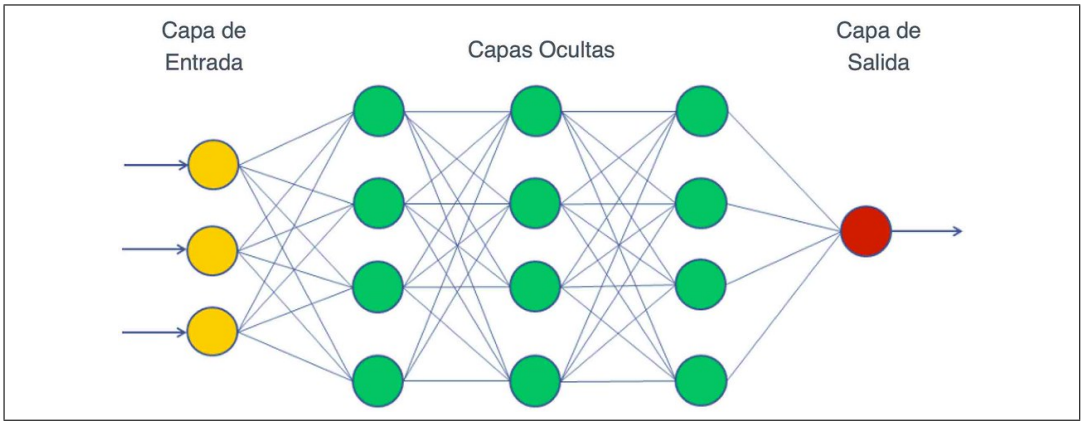


Figura 1.4: Esquema de perceptrón multicapa

encontrar desde texturas y modelos hasta proyectos completos o tutoriales sobre diferentes tecnologías o métodos.

Este motor utiliza C# como lenguaje principal para sus *scripts*, aunque existe la posibilidad de utilizar otros lenguajes (Python, por ejemplo) a través de comunicadores externos.

1.5. Tensorflow

Tensorflow es una librería de código abierto desarrollada por Google y abierta en noviembre de 2015. Esta librería ofrece la capacidad de construir y entrenar redes neuronales con múltiples objetivos, como la detección de correlaciones y descifrado de patrones, entre otros.

Esta librería cuenta con la posibilidad de usar más de una GPU y CPU al mismo tiempo, lo cual ayuda a realizar las ejecuciones más rápidas. Pone a disposición una API desarrollada en Python con la que se podrán entrenar a los agentes con diferentes algoritmos.

A la hora de realizar el entrenamiento con los agentes, se deben detallar los parámetros que el algoritmo va a usar, también conocidos como hiper parámetros. Existe la posibilidad de realizar una Grid Search para realizar todas las combinaciones posibles de hiperparametros,

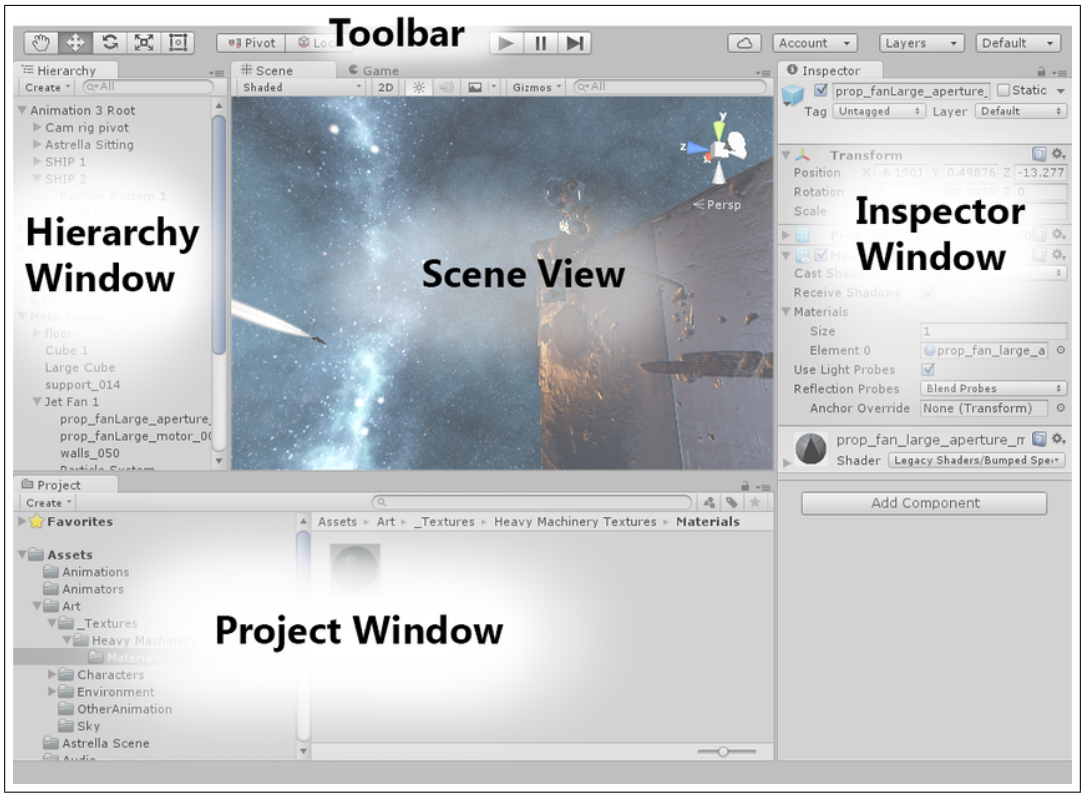


Figura 1.5: Interfaz grafica de Unity

pero esto requiere de una gran capacidad de cómputo, o por el contrario, se puede realizar una búsqueda manual de los hiperparametros adecuados para el modelo. Esta segunda opción es menos costosa a nivel computacional pero resulta imposible sin un método para observar el aprendizaje del entrenamiento realizado. Afortunadamente, dentro de la librería Tensorflow existe una herramienta para ello.

Dentro de esta librería, se encuentra la herramienta Tensorboard, que se usará para la visualización de gráficas que reflejan el proceso de aprendizaje de los agentes. Teniendo esta visualización, es posible una rápida comprensión, depuración y optimización del modelo que se entrena.

1.6. ML-Agents

El kit de herramientas ML-Agents[18] es un proyecto de código abierto que permite entrenar agentes inteligentes a través de juegos y simulaciones como entornos. Principalmente tiene como objetivo el aprendizaje por refuerzo, aprendizaje por imitación, neuroevolución y otros métodos de aprendizaje automático en agentes dentro de Unity. Para eso hace uso de una API de Python y algoritmos implementados en PyTorch.

Este kit de herramientas cuenta con cinco componentes clave:

- Learning Environment : contiene la escena de Unity y los diferentes personajes del juego. Esta escena proporciona el entorno en el que los agentes observan, actúan y

aprenden. Esta escena será desarrollada de forma arbitraria dependiendo del problema al que se quiera enfrentar.

- API de Python: Contiene una interfaz a bajo nivel para interactuar y manipular el entorno de aprendizaje. Esta no forma parte de Python y se comunica con este a través de un comunicador. Se puede usar bien para comunicarse con el Academy, el cual contiene los cerebros del entorno, o bien para usar algoritmos de aprendizaje automático propios. Toda la información adicional sobre estos elementos puede encontrarse en el Apéndice del documento.
- External Communicator: es el encargado de conectar el entorno de aprendizaje con la API de Python.
- Python Trainers: contiene los algoritmos de aprendizaje automático que se usan para entrenar a los agentes. Estos se implementan en Python y actúan con el comunicador externo.
- Gym Wrapper: Una forma adicional de interactuar con el entorno proporcionada por OpenAI. Es increíblemente útil pero todavía cuenta con limitaciones.

En la Figura 1.6 puede verse la estructura simplificada de los componentes descritos. Cabe destacar que dentro del entorno de aprendizaje entran más elementos en juego, los cuales pueden ser consultados en el anexo 6 de este documento.

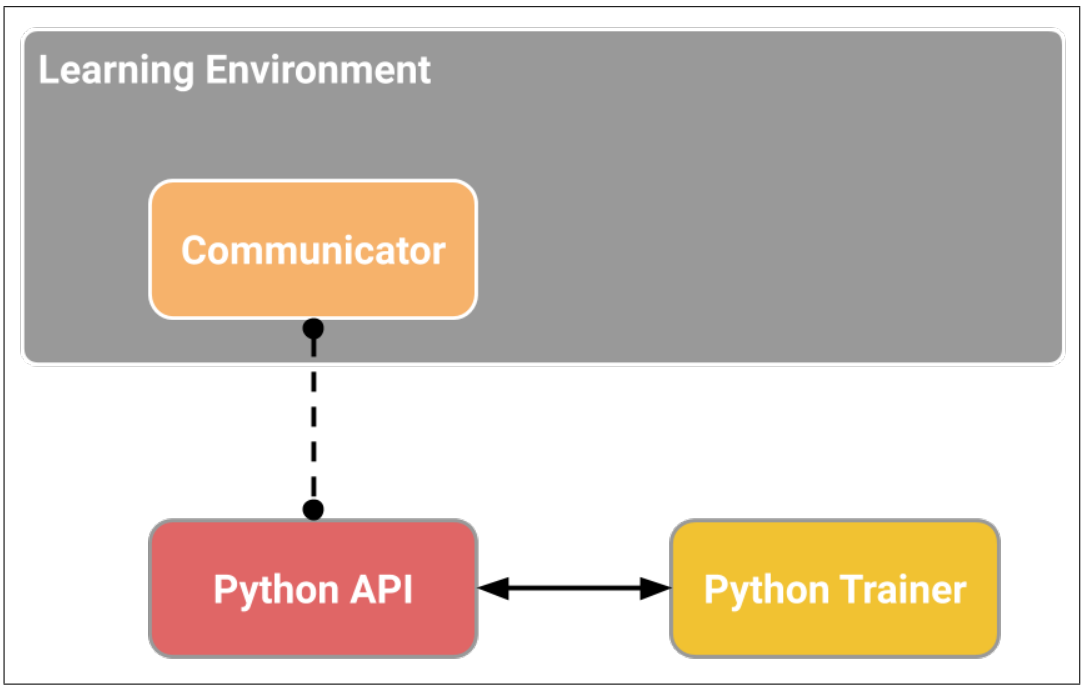


Figura 1.6: Diagrama simplificado de ML-Agents

CAPÍTULO 2

Gestión del proyecto

2.1. Objetivos y alcance del proyecto

El objetivo principal es comparar el comportamiento de los algoritmos PPO, SAC, Q-Learning y SARSA en escenarios de complejidad creciente. Con ello se pretende ver cual es el desarrollo de los agentes empleando los algoritmos en entornos competitivos con más de un agente. Para la visualización de dichos entornos se implementará realidad virtual (RV), haciendo los escenarios más intuitivos y realistas.

El alcance valorado abarca la creación de un entorno controlado y cambiante (un laberinto) que dos agentes deben ser capaces de explorar tras ser entrenados a través de diferentes algoritmos de aprendizaje por refuerzo. Cuenta con varias fases de desarrollo que tienen como objetivo la supervisión y comparación del avance de los agentes en su labor de aprendizaje y la creación de un entorno viable para el desarrollo de estos.

2.1.1. Descripción de los objetivos técnicos del proyectos

El objetivo principal es comparar el comportamiento de diferentes algoritmos en escenarios de complejidad creciente. Los objetivos principales del proyecto se resumen en:

- Creación y desarrollo de dos agentes capaces de recorrer un laberinto en busca de la meta antes que su rival.
- Implementación de Raycast en los agentes para la obtención de observaciones del entorno.
- Desarollo de sistema para la creación de laberintos de forma aleatoria.

Las tareas principales de los agentes son:

- La exploración independiente del laberinto, teniendo como objetivo la meta situada en un punto de este.
- Llegar a la meta designada antes que el otro agente para obtener así la recompensa.
- Llegar lo antes posible a la meta, intentando evitar dar pasos innecesarios.

2.2. Descripción de las tareas a realizar

El proyecto se realizará en tres fases de desarrollo, omitiendo la gestión de proyecto y la recopilación de información, las cuales irán aumentando gradualmente la complejidad y que se realizarán de forma escalonada, tal y como puede comprobarse en el diagrama de la Figura 2.2.

2.2.1. Primera fase

La primera fase tendrá como objetivo ser una primera toma de contacto con el software, aprendiendo así como utilizarlo correctamente y que posibilidades puede permitirle al usuario. Para que esta fase sea sencilla, se utilizará un entorno simple que consta de cuatro paredes, un suelo, un agente y un punto objetivo. El agente deberá llegar hasta ese punto objetivo a través de diferentes acciones.

2.2.2. Segunda fase

La segunda fase buscará explorar los comportamientos e interacciones mutuas de los agentes en un entorno parecido al utilizado en la primera fase. Ambos agentes tendrán que competir por llegar al punto objetivo antes que el otro agente. Se espera que ambos agentes sean capaces de lograr llegar al punto objetivo.

2.2.3. Tercera fase

La tercera fase será la última fase del proyecto. El entorno pasará a ser un laberinto generado de forma aleatoria, el cual los agentes deberán explorar para llegar al objetivo antes que su contrincante. El laberinto es modificable bajo ciertas circunstancias, por lo que los agentes podrán usar su dinamismo para lograr su objetivo.

Paquete de Trabajo Recopilación de Información (RI)

- Recopilación de literatura metodológica. En esta tarea se recopila información acerca de los diferentes algoritmos que se pueden utilizar, los elementos clave que tiene un agente, etc.
- Recopilación de literatura tecnológica. En esta tarea se recopila información sobre las diferentes tecnologías disponibles y cuales son sus ventajas y desventajas con respecto a las otras, teniendo que decidir cuál o cuáles se usarán para el desarrollo del proyecto.
- Recopilación de fuentes para la bibliografía. En esta tarea se recopilaran y guardaran las diferentes fuentes empleadas para la obtención de información a lo largo del proyecto, de forma que a la hora de reunirlas en la documentación final sea más fácil realizarlo.

Paquete de Trabajo Diseño de Agentes (DA)

- Creación y diseño de modelo físico del agente. Se debe hacer un diseño inicial sobre el modelo físico que va a tener el agente, es decir, que forma va a tener (esférica,

cuadrada, cilíndrica, humanoide...), definir su velocidad de desplazamiento y rotación y cuáles serán sus físicas.

- Diseño de observaciones para el agente. Puesto que el agente irá adquiriendo cada vez más complejidad, se deben tener en cuenta cuáles serán las observaciones que recibirán los agentes, ya que es un elemento muy importante para que puedan aprender correctamente. Además, es importante definir cómo las va a recibir, ya que será un posible limitante para el agente en caso de ser muy restrictivos con sus observaciones.
- Diseño de Sistema de Recompensas. Otro de los elementos que destaca dentro del aprendizaje por refuerzo es la recompensa que se utiliza para las acciones de los agentes. Es importante realizar una investigación sobre cómo funcionan estos sistemas de recompensas y comenzar con un sistema de recompensas aparentemente funcional, que se irá modificando a medida que avance la fase de experimentación, obteniendo así mejores resultados.

Paquete de Trabajo Elaboración de entorno (EE)

- Diseño y desarrollo de los diferentes entornos necesarios. Durante el proyecto se emplearán diversos escenarios, por lo que será necesario realizar el diseño de los diferentes entornos a utilizar para que sean viables y útiles para el desarrollo.
- Diseño y desarrollo de elementos aleatorios. En nuestros entornos finales se dan elementos generados de forma aleatoria, por lo que tendremos que realizar un diseño de estos para poder desarrollar posteriormente un sistema eficiente.

Paquete de Trabajo Experimentación (E)

- Experimentación con parámetros del algoritmo a ejecutar. Cuando se entrenan a los agentes, se realizan diferentes pruebas de experimentación en los que se deben realizar diferentes ejecuciones con diferentes parámetros para poder obtener los mejores resultados posibles. Como no es viable realizar pruebas con todas las combinaciones de los parámetros posibles, se realizarán una cantidad determinada de pruebas en un inicio para ver los rendimientos obtenidos y trabajar sobre los mejores resultados.
- Experimentación con parámetros de recompensas. Después de diseñar e implementar un sistema inicial de recompensas, haremos diferentes pruebas con las diferentes recompensas que recibirán los agentes con la intención de encontrar un sistema que funcione correctamente y maximice la recompensa positiva obtenida.

Paquete de Trabajo Obtención de Resultados (OR)

- Recopilación de resultados. A medida que vayamos realizando pruebas con diferentes parámetros, recopilaremos los diferentes resultados para realizar comparaciones y poder obtener información de las simulaciones realizadas.
- Análisis y obtención de conclusiones. Tras obtener la recopilación de los resultados y realizar un análisis sobre estos, podremos obtener información sobre el comportamiento que ha ido teniendo y si los resultados son los esperados.

- Desarrollo de conclusiones y líneas de trabajo futuras. Se establecerán líneas de investigación que pueden desarrollar el trabajo desarrollado para encontrar formas más eficientes de realizar el trabajo o que pueda concluir en investigaciones relacionadas con Inteligencia Artificial.

2.3. Dedicación inicial de horas

Como puede verse en la Figura 2.1, se estima que las tareas que más tiempo lleven serán la experimentación de diferentes parámetros, en base a que las simulaciones cada vez son más complejas y necesitan más tiempo para poder realizarse correctamente. Este proceso puede acelerarse a través de la ejecución simultánea de agentes, incrementando el número de agentes que se entrenan al mismo tiempo y haciendo que el modelo se entrene más rápido. También puede observarse que la segunda tarea más costosa es la recopilación de literatura, ya que empezar el proyecto con una buena base y entender que es lo que se quiere lograr y a través de qué medios podemos llegar a ello es crucial, puesto que sin una buena base no se podría realizar un trabajo de calidad.

2.4. Diagrama de Gantt del proyecto

En la Figura 2.2 de la página 20 se presenta el diagrama de Gantt diseñado para este proyecto, en el que se comienza con la obtención de información las primeras semanas de diciembre y se estima que debería estar terminado para finales de mayo, dejando un margen de error en junio por si se diera el caso de necesitar más tiempo para terminar el proyecto.

2.5. Análisis de riesgos

En el momento que se plantea el proyecto, se deben valorar los riesgos y sus posibles soluciones a estos problemas para poder comenzar. De los riesgos encontrados, se mencionan los más relevantes y los planes de contingencia que se buscan a estos.

- **Errores o problemas graves con las herramientas del proyecto.** En caso de que se diera este escenario y no se pudiera continuar o empezar, se ha recopilado información sobre diferentes herramientas para el desarrollo de este, por lo que se podría probar a realizarlo en cualquiera de las herramientas investigadas en la recopilación de literatura tecnologica.

- **Incumplimiento de las fechas establecidas.** La probabilidad de no seguir el ritmo estipulado y necesitar más tiempo para poder realizar el proyecto nunca es nula, por lo que se ha dejado un tiempo prudencial en los plazos de las tareas (definidas en la Figura 2.2) de forma que queden entre dos y tres semanas libres para realizar aquellas tareas que queden por hacer.

- **Los resultados finales obtenidos no son los esperados.** En caso de que los resultados no sean los esperados, estos se podrían estudiar y valorar como información útil y de valor para el desarrollo de líneas de investigación futuras. Los errores obtenidos en un proyecto de este tipo pueden aportar información que podría ser de utilidad en el futuro.

Tarea a realizar	Horas estimadas
Paquete de trabajo Recopilación de Información (RI)	
Recopilación de literatura metodológica	30
Recopilación de literatura tecnológica	30
Recopilación de fuentes para la bibliografía	5
Paquete de trabajo Diseño de Agentes (DA)	
Creación y diseño de modelo físico del agente	10
Diseño de observaciones para el agente	5
Diseño de Sistema de Recompensas	5
Paquete de trabajo Elaboración de Entorno (EE)	
Diseño y desarrollo de los diferentes entornos necesarios	25
Diseño y desarrollo de elementos aleatorios	25
Paquete de trabajo Experimentación (E)	
Experimentación con parámetros de algoritmos a ejecutar	50
Experimentación con parámetros de recompensas	50
Paquete de trabajo Obtención de Resultados (OR)	
Recopilación de resultados	10
Análisis y obtención de conclusiones	10
Desarrollo de conclusiones y líneas de trabajo futuras	10
Paquete de Gestión de Proyecto (GP)	
Gestión de proyecto	10
Redacción de memoria	25
Suma total:	**300**

Figura 2.1: Estimación de horas dedicadas a las tareas del proyecto

- **Falta de capacidad para desarrollar alguna tarea específica.** En caso de no ser capaz de realizar alguna de las tareas, la realización de algún algoritmo o la puesta en marcha de un escenario que satisfaga las necesidades del proyecto, se hablaría con el tutor en busca de una solución que pueda hacer avanzar el estado del proyecto,

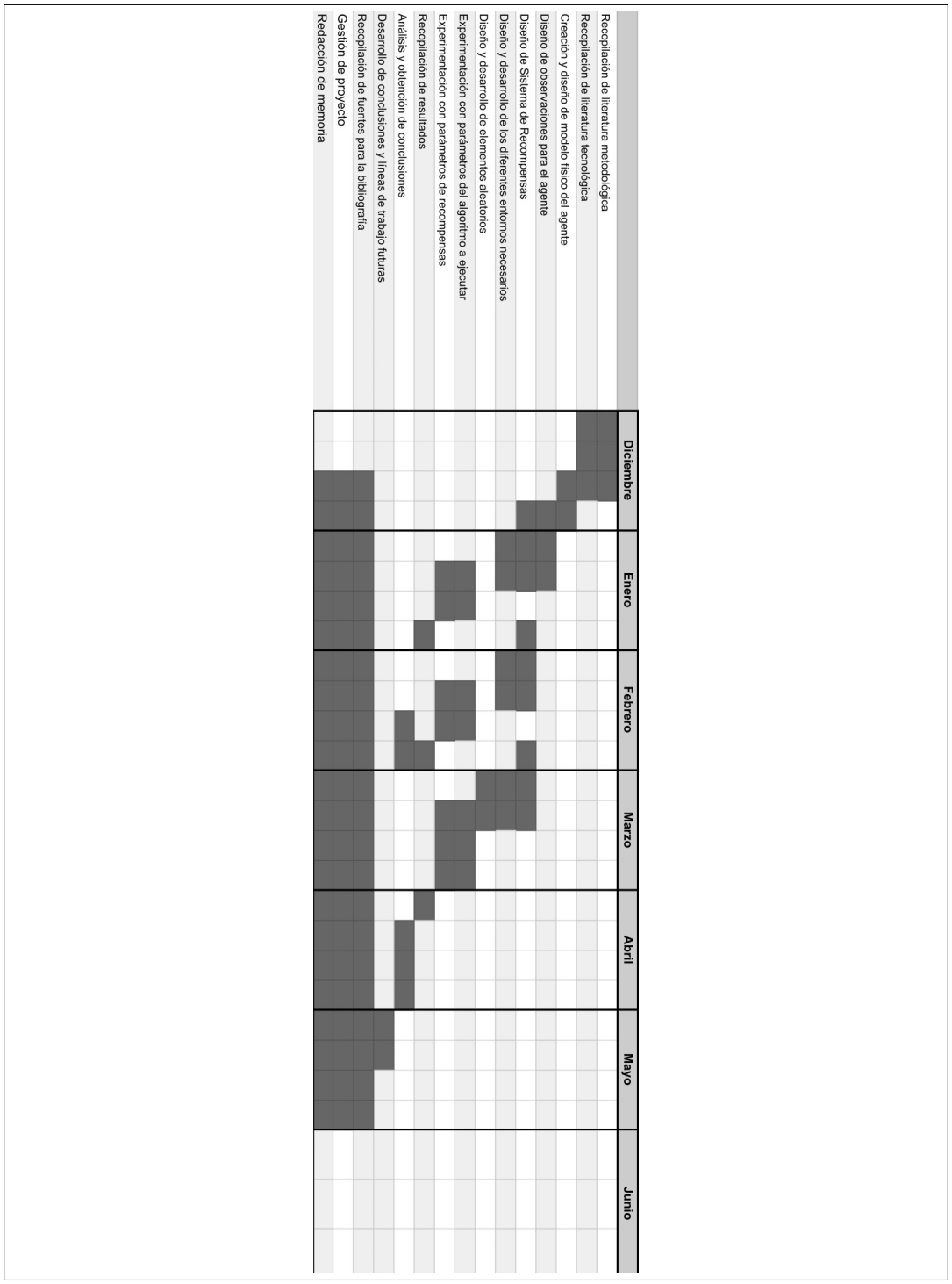


Figura 2.2: Diagrama de Gantt del proyecto

procurando mantener los objetivos iniciales y variando los métodos.

2.6. Herramientas

En esta sección se proporciona un listado de las herramientas que se han empleado para el desarrollo del proyecto.

- Unity. Es uno de los motores de creación de videojuegos más usados en la actualidad. Sus funcionalidades más utilizadas son animaciones, sonidos, Inteligencia artificial, motor físico que simula las leyes de la física... En este proyecto se ha usado como herramienta principal para la creación del entorno virtual con el que se trabaja.

- TensorBoard/ Tensorflow. Tensorboard es un kit de herramientas de visualización de la biblioteca de código abierto Tensorflow. Este kit proporciona la visualización y las herramientas necesarias para experimentar con el aprendizaje automático, por lo que gracias a esta herramienta se puede monitorizar el entrenamiento de los agentes con gráficas de la recompensa, entropía o longitud del entrenamiento entre otras.

- ML-agents. Es un proyecto de código abierto que permite que los juegos y las simulaciones sirvan como entornos para entrenar agentes inteligentes. Estos agentes pueden utilizarse para múltiples propósitos, como el control del comportamiento de los NPCs (*non-player character* o personaje no jugador), evaluaciones de diseño de juegos o incluso pruebas automatizadas sobre juegos compilados.

    - Raycast. Es una función física que permite proyectar unos rayos que surgen desde un agente en unas direcciones determinadas y nos indicará si colisiona con algún objeto. Si estos rayos colisionan con un objeto, darán información sobre la colisión, como la distancia o si reconoce el objeto[1].

      Para este proyecto en concreto, y gracias a que ML-Agents dispone de tal tecnología, se usa el componente Ray Perceptor Sensor 3D, el cual puede lanzar más de un rayo. Además, puede configurarse para determinar tanto el número de rayos, el grado máximo de los rayos o incluso su longitud.

- Anaconda. Anaconda Navigator es una interfaz gráfica de escritorio incluida en el paquete de distribución de Anaconda, permitiendo iniciar aplicaciones y administrar paquetes, entornos y canales conda sin usar comandos. Además, permite instalar Tensorflow, que se usará para el entrenamiento de los agentes con los algoritmos seleccionados.

- Github. Github es un portal creado para alojar proyectos con código de aplicaciones y herramientas, con el objetivo de descargarlo, visualizarlo o colaborar con su desarrollo. Esta herramienta usa un control de versiones que permite a los usuarios mantener copias de cada uno de los estados anteriores de los proyectos con el objetivo de no perder estos estados al actualizar. Se usará para mantener un control del proyecto y como sistema de seguridad.

[1] https://gamedevbeginner.com/raycasts-in-unity-made-easy/

2.7. Cambios respecto a la planificación

2.7.1. Diagrama de Gantt final tras el proyecto

Tras realizar el proyecto, se corrigió el diagrama de Gantt, Figura 2.3 que inicialmente se planteó y se indicó el proceso real que ha atravesado el proyecto durante estos meses. Además, se recogió el número de horas utilizadas para el proyecto para poder compararlas con las estimadas inicialmente, recogidas en la Figura 2.4.

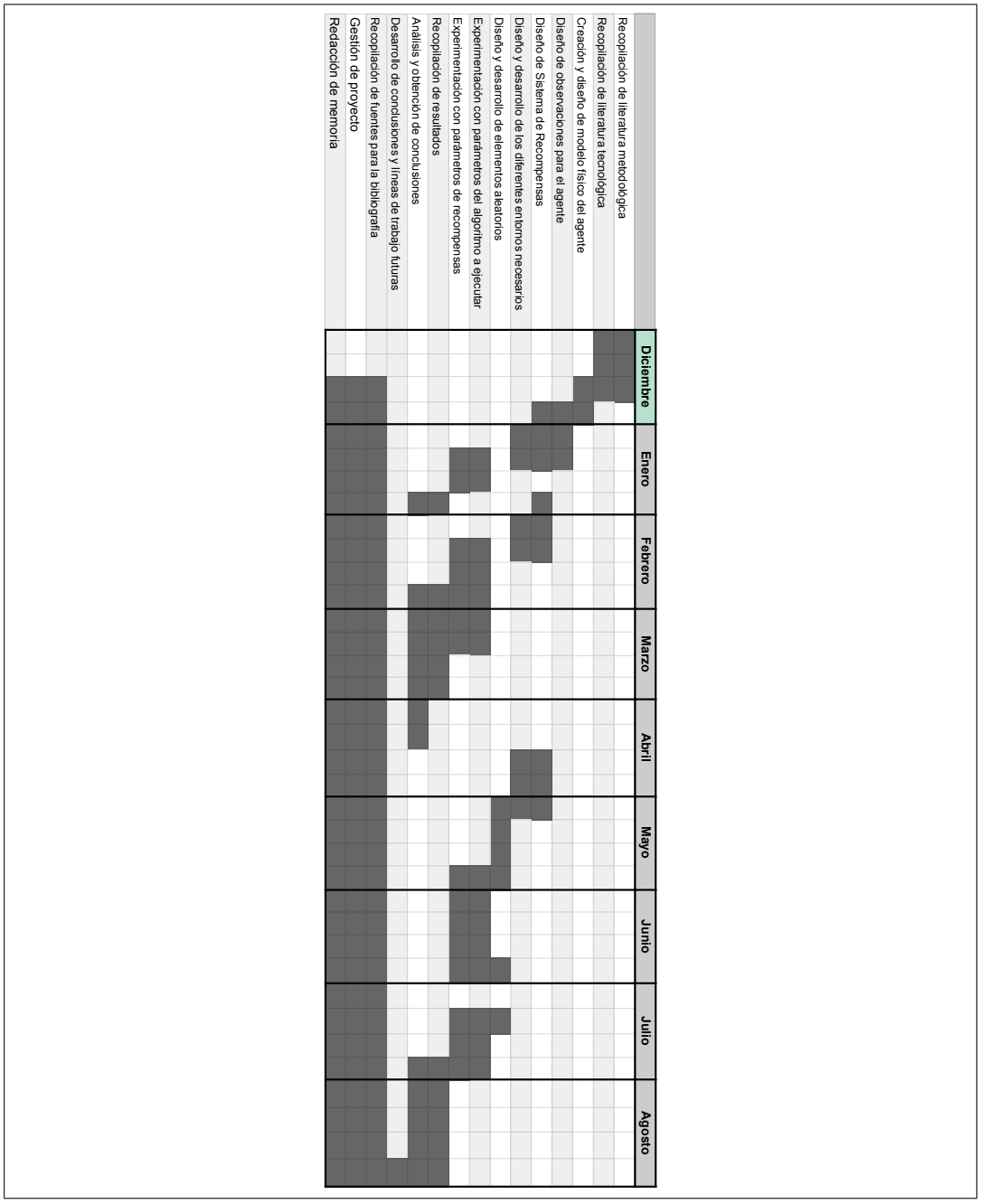


Figura 2.3: Diagrama real de Gantt del proyecto

Tarea a realizar	Horas estimadas
Paquete de trabajo Recopilación de Información (RI)	
Recopilación de literatura metodológica	30
Recopilación de literatura tecnológica	30
Recopilación de fuentes para la bibliografía	10
Paquete de trabajo Diseño de Agentes (DA)	
Creación y diseño de modelo físico del agente	5
Diseño de observaciones para el agente	5
Diseño de Sistema de Recompensas	10
Paquete de trabajo Elaboración de Entorno (EE)	
Diseño y desarrollo de los diferentes entornos necesarios	20
Diseño y desarrollo de elementos aleatorios	25
Paquete de trabajo Experimentación (E)	
Experimentación con parámetros de algoritmos a ejecutar	40
Experimentación con parámetros de recompensas	60
Paquete de trabajo Obtención de Resultados (OR)	
Recopilación de resultados	5
Análisis y obtención de conclusiones	10
Desarrollo de conclusiones y líneas de trabajo futuras	5
Paquete de Gestión de Proyecto (GP)	
Gestión de proyecto	10
Redacción de memoria	35
Suma total:	**300**

Figura 2.4: Horas reales dedicadas al proyecto

2.7.2. Complicaciones o problemas durante el desarrollo

Durante el desarrollo del proyecto han surgido distintos problemas, entre ellos problemas para el desarrollo de los algoritmos tras las diferentes actualizaciones de versiones realizadas, ya que la documentación al respecto no estaba del todo completa y era difícil

encontrar los métodos de cada clase.

Además, también surgieron problemas a la hora de hacer que las versiones de las diferentes herramientas fueran compatibles, por lo que hubo que probar con varias versiones distintas hasta encontrar la adecuada. Sobre la complicación referente a las versiones de las herramientas usadas se hablará en el próximo capítulo, en el apartado complicaciones durante el desarrollo.

Las diferencias entre ambos diagramas de Gantt se deben principalmente a la falta de tiempo bien por la obtención de un trabajo a tiempo completo o falta de tiempo por asuntos personales.

CAPÍTULO 3

Fase 1: Entorno simple con un agente

En este capítulo se explicará el inicio práctico del proyecto, las complicaciones que fueron surgiendo a medida que este avanzaba y como se fueron abordando, así como los resultados obtenidos tras numerosas ejecuciones y pruebas.

3.1. Construcción del escenario

El objetivo de esta primera fase es conocer y aprender a usar el entorno de aprendizaje, por lo que se decidió diseñar un entorno sencillo con el que experimentar y comenzar a obtener conclusiones. El problema diseñado para esta fase es el de un agente que ha de llegar a un objetivo situado en un punto aleatorio del entorno. Para este propósito haremos uso de las técnicas de aprendizaje por refuerzo.

Como puede observarse en la Figura 3.1, el entorno cuenta con una estructura básica. Entre sus componentes se pueden encontrar un agente (el cuadrado rojo), un objetivo (el cilindro verde), un suelo y cuatro paredes. Las cuatro paredes se sitúan siempre a los bordes del suelo, delimitando el espacio por el que el agente puede moverse libremente. Este espacio tiene unas dimensiones de 22 unidades de alto y 22 unidades de alto.

Otra de las mecánicas implementadas para este modelo es el cambio de color del suelo para poder saber si el agente ha llegado al objetivo o si se ha chocado con una pared. En caso de que el agente llegue al objetivo, el suelo cambiara a un color verde, tal y como puede verse en la Figura 3.2a, y en caso de colisionar con una pared, cambiará a color rojo, Figura 3.2b. Esta mecánica está pensada para poder llevar un registro visual de lo que está pasando en el escenario.

Una vez el escenario estaba preparado y sus componentes bien configurados, se replicó el escenario hasta contar con 25 escenarios diferentes e independientes, que ayudarían a acelerar el ritmo de aprendizaje del agente (ver Figura 3.3). Para poder obtener estos 25 escenarios se generó un prefab, un elemento de Unity que permite almacenar objetos manteniendo las propiedades del escenario original, y se introdujeron 24 copias de ese prefab en el entorno (ver Figura 3.4). Al contar con tantos escenarios realizando episodios

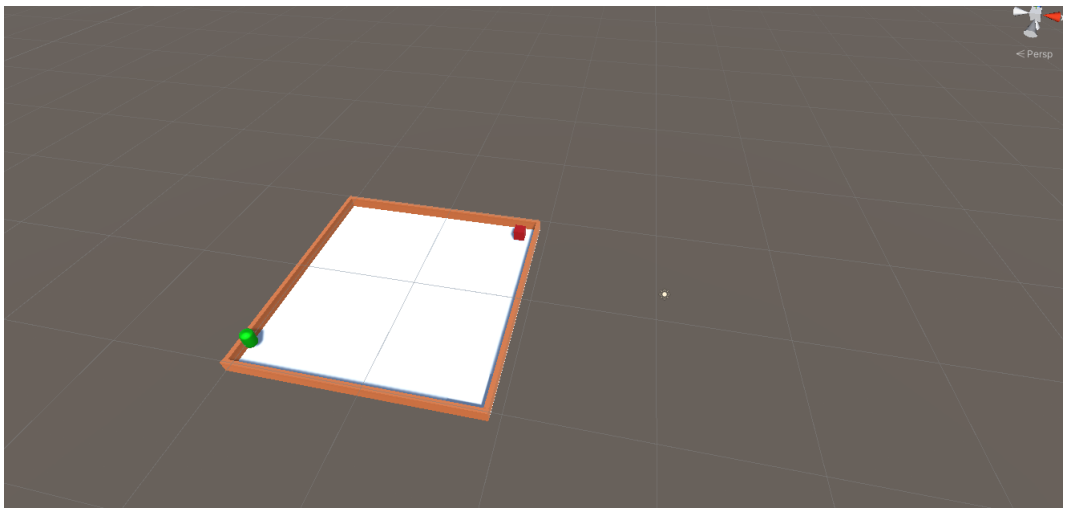
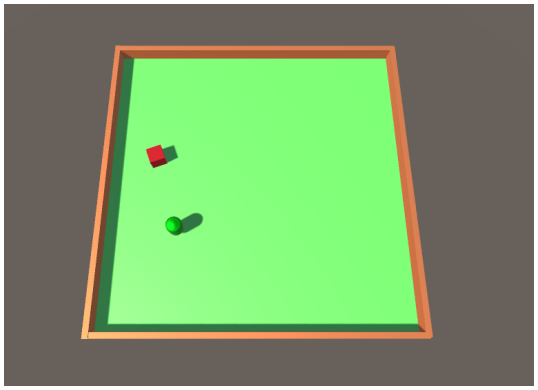
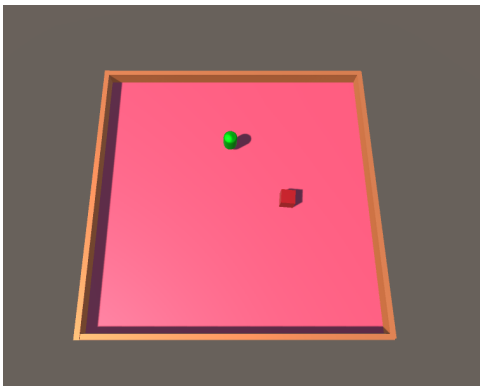


Figura 3.1: Fase 1: Escenario



(a) Suelo verde indicando que ha llegado al objetivo



(b) Suelo rojo indicando que ha colisionado con una pared

Figura 3.2: Fase 1: Cambios de colores de escenario

de forma conjunta, las simulaciones se realizan mucho más rápido a la hora de comprobar la eficiencia de los algoritmos.

3.2. Observaciones

Uno de los elementos clave para el aprendizaje por refuerzo son las observaciones, ya que otorgan información sobre el entorno. Esta información se utilizará para determinar las acciones a tomar por el agente, ya que serán recibidas como inputs para decidir la acción a realizar en la red neuronal.

Las observaciones que se hacen del entorno en esta fase vienen dadas por el Raycast. El Raycast es una función de Unity que permite proyectar un rayo en la escena que devolverá un booleano si colisiona con los objetos que puede identificar. En estos casos los agentes obtendrán información sobre la distancia, posición o que objeto es a través de los Raycast. En este caso serán capaces de detectar e identificar los muros y el objetivo. Estos ayudarán

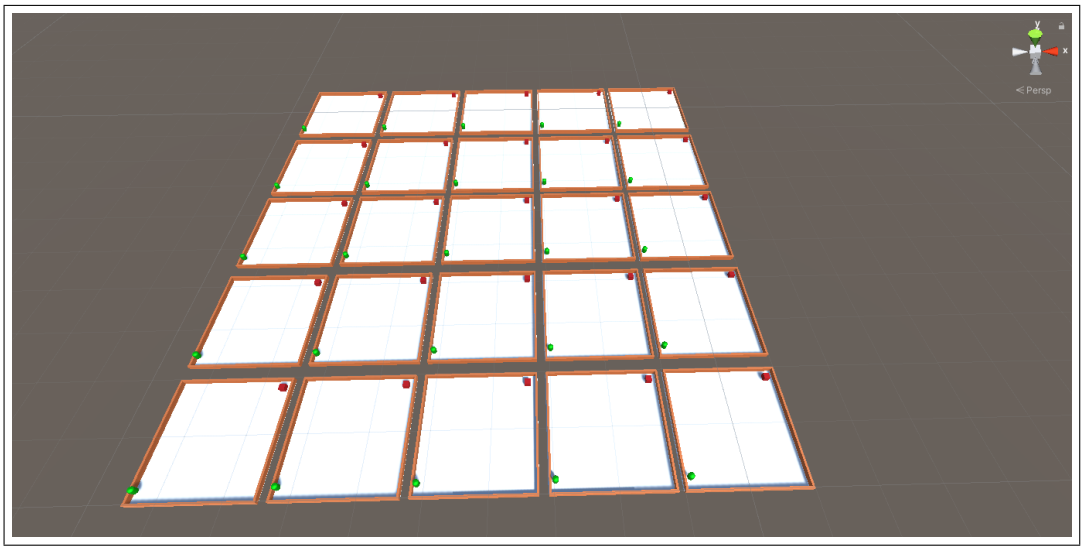


Figura 3.3: Fase 1: Entrenamiento simultáneo

Figura 3.4: Fase 1: Prefab del escenario

al agente a detectar tanto los muros como el objetivo y de esa forma poder explorar el entorno.

Como puede verse en la Figura 3.5, se han configurado un total de 15 rayos con una longitud de 20 unidades (uno que siempre mira hacia el frente, 7 en el lado derecho y otros 7 en el lado izquierdo). Uno de los parámetros a configurar es el grado máximo de los rayos, que determina el tamaño del cono para los rayos. El uso de 90 grados arrojará rayos a la izquierda y a la derecha. En caso de que el ángulo es mayor que 90, los rayos irán hacia atrás.

Cada uno de los rayos recoge varios datos del entorno y son enviados a la red neuronal con un formato *float*. La cantidad de *floats* se calcula en base a la cantidad de objetos que puede detectar más dos *floats*, uno que indica si el rayo ha colisionado con un objeto y otro que indica si la distancia a la que ha colisionado. Es decir, el número de entradas para la

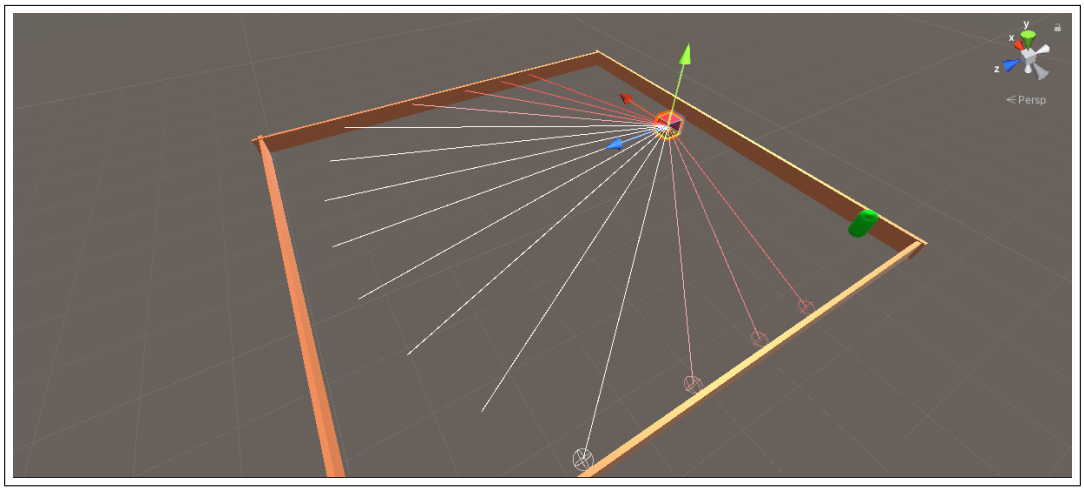


Figura 3.5: Raycast en el agente

red neuronal que genera cada rayo son el número de objetos que puede detectar más 2.

Las entradas para la red neuronal generados por el rayo funcionan de la siguiente forma:

- Entrada 1: 1.0 si el rayo colisiona con una pared, 0.0 si no lo hace.
- Entrada 2: 1.0 si el rayo colisiona con el objetivo, 0.0 si no lo hace.
- Entrada 3: 1.0 si el rayo no ha colisionado con nada, 0.0 si no lo hace.
- Entrada 4: 1.0 La distancia a la que está el objeto con el que ha colisionado (1.0 si no colisiona con nada).

Sabiendo que el agente posee 15 rayos, el cálculo de entradas para la red neuronal se haría multiplicando el número de entradas por rayo calculado anteriormente por 15 rayos, siendo esto un total de 60 entradas.

3.3. Movimiento

El agente a lo largo de su entrenamiento realizará diferentes acciones para poder llegar al objetivo. Para ello, contará con una lista de acciones de las cuales ejecuta una por cada paso. Estas acciones se ejecutarán dependiendo de los valores que devuelva la red neuronal. Dichos valores se establecen en un vector de acción para el agente en el que cada uno de estos valores de este vector es una variable de tipo entero, que más tarde se traduce a una acción.

Cada uno de los pasos el agente tomará una de las siguientes acciones: moverse hacia delante, hacia atrás, desplazarse hacia la derecha, desplazarse hacia la izquierda, rotar hacia la izquierda o rotar hacia la derecha. Este movimiento será de una unidad en cualquiera de los sentidos y para las rotaciones se realizan rotaciones de 1 grado por movimiento. La idea del movimiento viene determinada por ser la forma que se suele ofrecer a los jugadores para desplazarse en los videojuegos (siendo esta la forma estándar establecida para los videojuegos actualmente). La rotación se ha establecido de esta forma para que los agentes

puedan realizar la rotación siendo capaces de observar el entorno correctamente en sus rotaciones.

Cada episodio, los agentes podrán realizar un máximo de 50000 pasos antes de volver a iniciar el episodio desde el principio.

3.4. Recompensas

La función de recompensa es quizá el elemento más complejo de esta fase, ya que de esta depende que el agente aprenda correctamente lo que debe hacer o no. Un sistema mal balanceado de recompensas puede hacer que los agentes obtengan resultados erráticos. Cada vez que el agente llegue al objetivo, obtendrá una recompensa positiva, lo cual estimula al agente a que la probabilidad de que se de determinada conducta aumente. Por el contrario, con una recompensa negativa, se intenta eliminar un evento insatisfactorio, aunque en determinadas ocasiones se interpreta como un incentivo para terminar el episodio lo antes posible, ya que pierde puntos[19].

Para esta primera fase, se han implementado tres recompensas, una positiva y dos negativas. La recompensa positiva se proporciona cuando el agente consigue llegar al objetivo, por ende, cada vez que llega se le da una recompensa positiva de +1. Se dice que el agente ha llegado al objetivo cuando éste colisiona con el objeto que representa al objetivo. De la misma manera, cuando el agente toca o colisiona con una de las paredes del entorno, se le aplica una recompensa negativa de -1, buscando así que el agente evite en cualquier circunstancia tocar las paredes. La otra recompensa negativa, la cual se aplica cada vez que el agente realiza una acción, se implementó por dos motivos y tiene un valor de -10^{-4}. La primera era evitar que las recompensas vinieran dadas por colisiones con elementos del escenario, obligando al agente a tomar decisiones que no implicarán quedarse estático. El segundo motivo era instar al agente a llegar al objetivo lo antes posible, minimizando la cantidad de pasos que daba intentando llegar al objetivo y, por ende, la cantidad de recompensas negativas.

3.5. Algoritmos

El objetivo era probar los algoritmos PPO, SAC, Q-Learning y SARSA, probando diferentes valores para algunos de sus parámetros en ambos algoritmos y comprobar así las variaciones que pudieran surgir. Sin embargo, como se explicará al final del capítulo, solo ha sido posible probar PPO y SAC.

3.5.1. Algoritmo PPO

En el caso del algoritmo PPO se han realizado 9 simulaciones en las que se modificaron los parámetros epsilon (corresponde al umbral de divergencia aceptable entre la política antigua y la nueva en el descenso de gradiente) y beta (corresponde a la fuerza de regularización de la entropía, lo cual hace que la política usada tenga una mayor aleatoriedad). Estas nueve simulaciones se realizaron con valores de 0.1, 0.2, 0.3 para epsilon y $0{,}0001$, $0{,}001$ y $0{,}01$ para beta. La configuración de parámetros empleada para este algoritmo puede ser consultada en la sección de 6 del Apéndice .

Todas las simulaciones han tenido una convergencia en valores de recompensa de aproximadamente 0.9 alrededor de los 600000-850000 pasos (siendo los pasos totales de cada simulación 1000000) y han tenido una duración de 20-40 minutos aproximadamente, aunque la mayoría de las ejecuciones han tenido una duración de cerca de 20 minutos. La Figura 3.6 puede servir como referencia gráfica sobre la evolución que han presentado las diferentes ejecuciones de este algoritmo. Todos los resultados son positivos y el resultado es el esperado en todas las simulaciones. Se puede considerar un éxito de cara a los objetivos tempranos de este proyecto.

Tanto la gráfica de la Figura 3.6 como el resto de gráficas del documento presentan el mismo formato. El eje vertical representa la recompensa obtenida y el eje horizontal representa los pasos a lo largo de la ejecución, es decir, es una representación del tiempo a través del entrenamiento mediante pasos. Las gráficas cuentan con periodos en los que esta recompensa crece o decrece, haciendo clara así la evolución de la recompensa obtenida por los agentes a lo largo del entrenamiento. En los periodos en los que decrece la gráfica, la recompensa obtenida es menor a la que se obtuvo con respecto a los pasos directamente anteriores, bien porque ha colisionado con la pared o porque no ha sido capaz de llegar al objetivo y esto a hecho que acumule una gran cantidad de pequeñas recompensas negativas por movimiento realizado, o bien la combinación de ambas. En la Figura 3.6 se puede observar que en las primeros 50000 pasos se puede ver una gráfica que desciende rápido, pues el agente no es capaz de resolver el problema planteado y se centra en las primeras exploraciones de este. A medida que avanza el entrenamiento, las recompensas van teniendo picos de subida y bajada entre los 100000 y 350000 pasos aproximadamente. A partir de este momento estas comienzan a subir, fruto de una política que empieza a ser apta para la resolución del problema planteado, mejorando los resultados obtenidos a medida que avanzan los pasos, tal y como se puede ver por la recompensa ascendente a partir de los 350000 (salvando pequeños picos descendentes que terminan corrigiendose). También puede observarse que esta subida comienza en una recompensa negativa y pasa el umbral de 0, pasando a ser positiva, siendo evidente una clara mejora en los resultados. Esta subida se da hasta los 600000 pasos, momento en el que el agente es capaz de resolver el problema sin aparente complicación y obtener una recompensa muy cercana a 1.

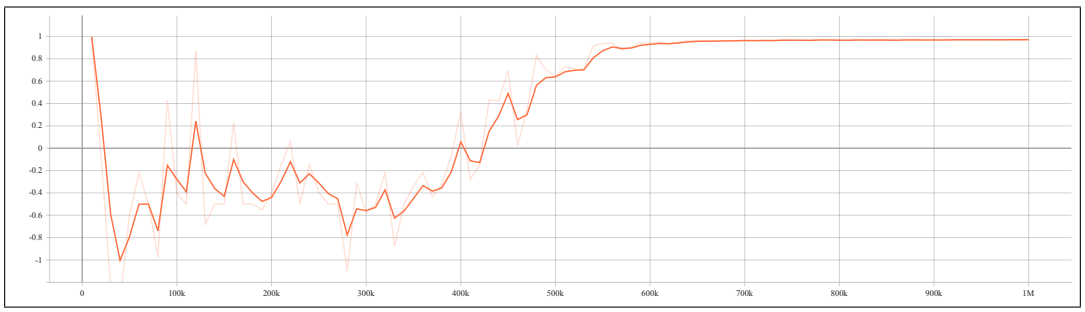


Figura 3.6: Gráfico de recompensa de ejecución de PPO en la primera fase

Para obtener información completa sobre los resultados finales de las ejecuciones del algoritmo PPO en la primera fase, véase la Tabla 1 situada en el anexo 6 del documento.

3.5.2. Algoritmo SAC

Por otro lado, con el algoritmo SAC se han realizado también 9 simulaciones, modificando los valores de los parámetros tau (la magnitud de actualización de Q durante la ejecución) y learning rate (corresponde a la fuerza de actualización de descenso de gradiente).

Tras analizar los resultados de las simulaciones, se comprobó que tres de las nueve simulaciones obtuvieron buenos resultados, teniendo como resultado un agente que era capaz de llegar al objetivo sin problema. Como puede verse en la Figura 3.7, la recompensa obtiene valores positivos tras realizar el proceso de entrenamiento de estas simulaciones. Sin embargo, también se obtuvieron simulaciones, concretamente seis de las nueve realizadas, con resultados en los que el agente no era capaz de llegar al objetivo. En las gráficas obtenidas de estas simulaciones, Figura 3.8, se podía comprobar que el agente se mantenía con una recompensa negativa durante casi toda la ejecución.

En la Figura 3.7 el comienzo se realiza con una caída de la recompensa, por la falta de eficacia que se tiene ante el problema. Esta caída es visible entre los primeros pasos y los 50000 pasos, manteniéndose en niveles bajos de recompensa hasta aproximadamente los 400000 pasos, siendo este el momento en el que las recompensas obtenidas por el agente comienzan a subir hasta cerca de los 450000 pasos, donde se mantiene variando entre unas recompensas positivas de 0.3 y 0.7

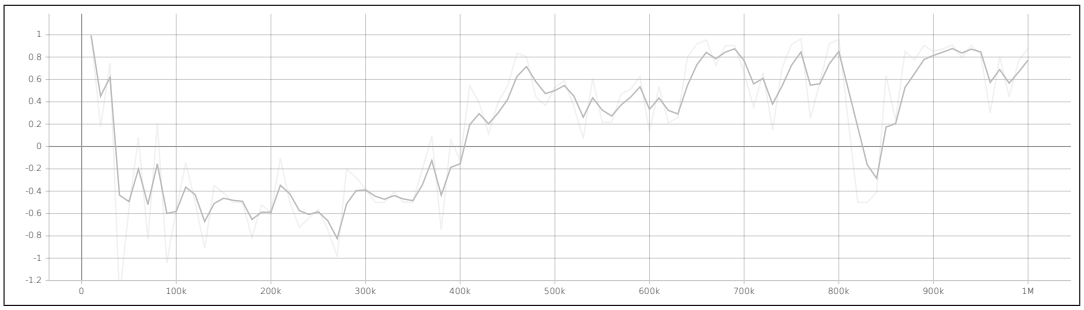


Figura 3.7: Gráfico de recompensa de ejecución de SAC en la primera fase

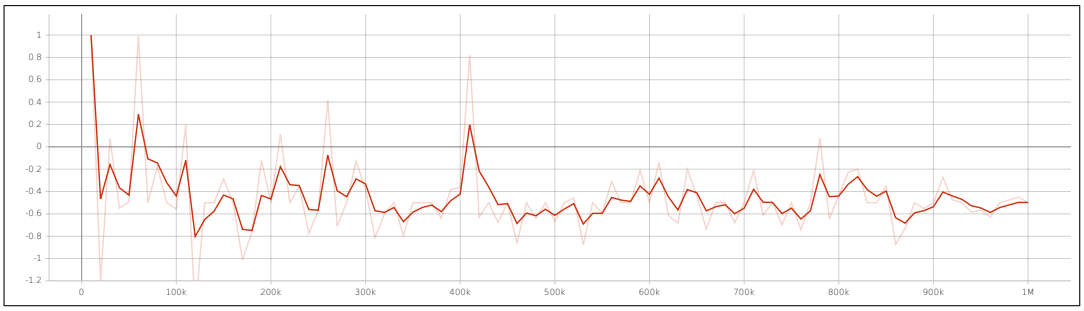


Figura 3.8: Grafico de recompensa de ejecución de SAC en la primera fase

Para obtener información completa sobre los resultados finales de las ejecuciones del algoritmo SAC, véase la Tabla 2 situada en los anexos del documento.

3.6. Conclusiones

Tras observar y analizar los resultados obtenidos por los algoritmos empleados, se considera un éxito a pesar de que uno de los algoritmos no ha dado los resultados deseados. El agente ha sido capaz de aprender cuál era su tarea y a realizarla correctamente. Ha sido un aprendizaje rápido por parte del algoritmo PPO, que ha logrado que el agente sea capaz de observar su entorno a través de realizar rotaciones para poder visualizar el objetivo y desplazarse hacia él directamente, lo cual corresponde a la reacción lógica esperada para este tipo de escenarios.

Puesto que el entrenamiento de los agentes se hace con varias réplicas y son ejecuciones que se realizan rápido, el entrenamiento a través de algoritmos on-policy, como podría ser el ejemplo de PPO, obtiene mejores resultados que a través de off-policy, SAC como ejemplo. Puesto que los algoritmos off-policy necesitan recopilar información sobre experiencias pasadas, este escenario de ejecución no es adecuado para su comprobación. Ya que la cantidad de tiempo que necesita para realizar el entrenamiento es alta por los recursos limitados de los que se disponen, no es posible realizar el entrenamiento de los agentes sin réplicas. Es por esto que se decide no continuar usando los algoritmos off-policy para las siguientes dos fases del proyecto.

Aun así, para comprobar que el resultado de lo obtenido es fiable, se le asigna el modelo entrenado con el algoritmo PPO al agente y se ejecuta esperando que sepa llegar al objetivo. Tras comprobar que esto sucede, se puede confirmar el éxito del experimento abordado en la fase 1.

El enlace al video con la demostración de la ejecución del agente usando el modelo entrenado se encuentra en el anexo 6.

3.7. Complicaciones durante el desarrollo

Durante el desarrollo surgieron algunas complicaciones con respecto a los algoritmos que se querían comprobar en el proyecto. Los algoritmos en cuestión son SARSA y Q-Learning.

En el paquete ML-Agents que se ha usado en este proyecto vienen implementados los algoritmos SAC y PPO, por lo que ejecutarlos es relativamente sencillo, ya que tan solo hay que configurar los parámetros con los que se van a ejecutar los algoritmos para poder usarlos. En cambio, para poder ejecutar los algoritmos SARSA y Q-Learning se debía usar un comunicador con Python.

El problema surgio a raiz de este comunicador y las funciones implementadas en las clases de ML-Agents, ya que en repetidas ocasiones y a pesar de realizar varios intentos con diferentes entornos de prueba y distintas versiones del paquete ML-Agents, las funciones no eran compatibles y la documentación de los desarrolladores no era del todo clara sobre los cambios realizados entre versiones. Es por ello que se pasó a mirar el funcionamiento del paquete y las funciones que contenía, pero al ser tantas y no contar con una documentación clara ni con el tiempo suficiente para poder investigarlo bien, se decidió que el coste que podía suponer hacer funcionar los algoritmos de forma manual era demasiado elevado en cuanto al tiempo que podía tomar y se dejó en segundo plano.

Existía la posibilidad de intentar usar el entorno de Gym, una API creada por OpenAI,

pero cuenta con el inconveniente de que solo permite realizar el entrenamiento con un solo agente. Esta limitación hace que únicamente se pueda realizar la primera fase y no se pueda continuar con la comprobación en caso de quererlo hacer competitivo. Es por esto que se ha decidido no usarlo, ya que no se podría avanzar a la segunda fase y quedaría incompleto.

La idea original era poder comparar el desempeño entre los algoritmos PPO, SAC, SARSA y Q-Learning, pero como no se consiguieron solucionar los problemas ocasionados, se dejaron en segundo plano los algoritmos SARSA y Q-learning y se continuó el proyecto con PPO y SAC.

CAPÍTULO 4

Fase 2: Entorno simple con dos agentes

En este capítulo se explicará el avance del proyecto con la implicación de dos agentes compartiendo un escenario y teniendo el mismo objetivo, las interacciones que puedan surgir por ello y las conclusiones obtenidas.

4.1. Construcción del escenario

La fase anterior seguía una estrategia más introductoria, mientras que la segunda fase busca observar cómo pueden interaccionar dos agentes cuando tienen que competir en un mismo escenario. Para ello, se adaptará el escenario de la primera fase, haciendo que este sea más grande y pase a tener unas dimensiones de 50 unidades de alto y 50 de ancho. El propósito de la ampliación es que los agentes tengan más terreno por el que desplazarse y exista la probabilidad de que estos en determinado momento se crucen.

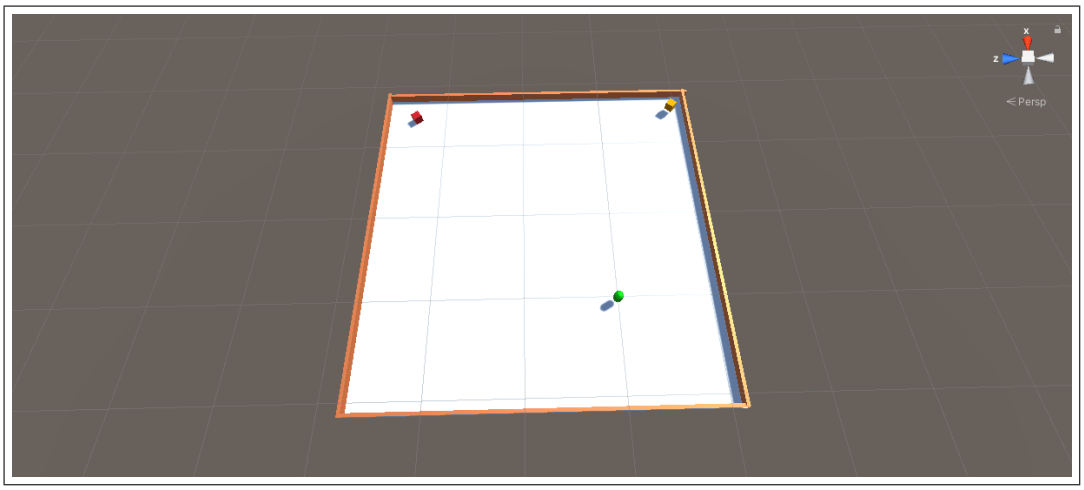
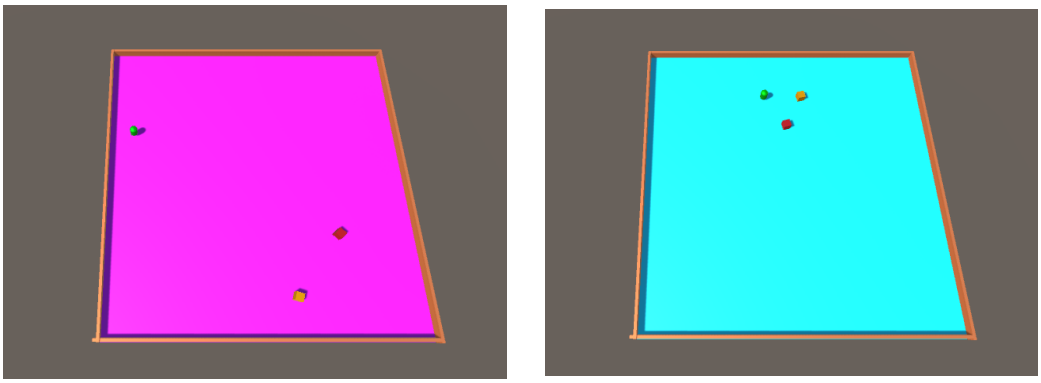


Figura 4.1: Fase 2: Escenario

Además de estos cambios de escenario, se añade otro agente (también con forma de cuadrado), al cual llamaremos oponente de ahora en adelante (para hacer más sencilla la diferenciación entre ambos agentes). El agente, de color rojo, es el situado en el margen izquierdo superior del escenario, mientras que el oponente, de color amarillo, se situa en el margen derecho superior. El objetivo sigue adoptando la misma forma que en la primera fase.

El oponente cuenta con las mismas características que el agente, para evitar que sea una disputa desequilibrada en cuanto a capacidades se refiere.

De la misma forma que se hizo en la primera fase, cuando el oponente colisiona con el objetivo o contra una pared, el suelo cambia de color con el objetivo de poder entender de forma visual que es lo que ha sucedido en el escenario. En caso de que el oponente colisione con el objetivo, el suelo cambiará a color morado. En caso de que colisione con la pared, cambiará a color cyan.



(a) Suelo morado indicando que el oponente ha llegado al objetivo

(b) Suelo cyan indicando que el oponente ha colisionado con una pared

Figura 4.2: Fase 1: Cambios de colores de escenario

4.2. Subfases de testeo

En esta fase se ha querido comprobar cómo interaccionan dos agentes que tras aparecer en un escenario han de llegar al objetivo primero. Para ello se han diseñado dos subfases para explorar diferentes escenarios.

La primera subfase tiene como característica que el agente, el oponente y el objetivo tienen la aparición en una zona aleatoria del escenario, siendo posible la distancia de aparición del agente al objetivo menor que la del oponente al objetivo, o viceversa. En esta subfase se busca observar las interacciones que surgen entre los dos agentes tras ser entrenados al mismo tiempo en un entorno compartido y competitivo.

Por otro lado, la segunda subfase está diseñada para ser equitativa. Tanto el agente como el oponente hacen su aparición en el escenario a la misma distancia del objetivo, es

decir, en puntos aleatorios en la circunferencia de radio aleatorio con respecto al objetivo. De esta subfase se espera que ambos agentes sean capaces de explorar su entorno lo más rápido posible para adelantarse a su oponente.

Tal y como se explica en el apartado de recompensas de esta fase, se añadió una subfase adicional ya que las dos subfases anteriores no obtuvieron los resultados esperados y se quiso comprobar si realizando el entrenamiento de forma independiente para los dos agentes en un entorno compartido los resultados pasarían a ser mejores.

4.3. Observaciones

Para esta fase se mantienen las mismas entradas de observación que en la primera fase, es decir, los rayos. El oponente cuenta con un componente que otorga los mismos rayos con la misma longitud y mismos ángulos que los del agente, es decir, contará con los 15 rayos con los que cuenta el agente. El agente es capaz de detectar las paredes, el objetivo y al oponente y de la misma manera, el oponente es capaz de detectar las paredes, el objetivo y al agente. La identificación entre agente y oponente obliga a añadir un dato de entorno más a cada uno de los rayos, incrementando el número de entradas de la red neuronal.

Las entradas para la red neuronal generados por cualquiera de los rayos del agente quedarían así:

- Entrada 1: 1.0 si el rayo colisiona con una pared, 0.0 si no lo hace.
- Entrada 2: 1.0 si el rayo colisiona con el objetivo, 0.0 si no lo hace.
- Entrada 3: 1.0 si el rayo colisiona con el oponente, 0.0 si no lo hace.
- Entrada 4: 1.0 si el rayo no ha colisionado con nada, 0.0 si no lo hace.
- Entrada 5: 1.0 La distancia a la que está el objeto con el que ha colisionado (1.0 si no colisiona con nada).

Por otro lado, las entradas generadas por cada rayo del oponente quedan así:

- Entrada 1: 1.0 si el rayo colisiona con una pared, 0.0 si no lo hace.
- Entrada 2: 1.0 si el rayo colisiona con el objetivo, 0.0 si no lo hace.
- Entrada 3: 1.0 si el rayo colisiona con el agente, 0.0 si no lo hace.
- Entrada 4: 1.0 si el rayo no ha colisionado con nada, 0.0 si no lo hace.
- Entrada 5: 1.0 La distancia a la que está el objeto con el que ha colisionado (1.0 si no colisiona con nada).

Siguiendo el esquema planteado en la primera fase para el calculo de entradas de la red neuronal y puesto que ambos agentes son capaces de detectar un objeto mas que en la fase anterior, ahora se cuentan con 75 entradas (15 rayos x (3 objetos detectables para cada uno de los agentes más dos entradas)), ya que el hecho de que puedan detectarse entre ellos es importante puesto que puede influir en las acciones que se realizan por parte de ambos agentes, ya sea evitando o bien bloquearse el camino o la visión hacia el objetivo.

4.4. Movimiento

En esta fase el agente cuenta con los mismos movimientos que en la primera fase y el oponente adopta estos mismos movimientos también, ejecutados a la misma velocidad para que ninguno de los dos agentes tenga ventaja por la capacidad de movimiento. En este caso, tanto el agente como el oponente cuentan con diferentes redes neuronales, es decir, cada agente tiene una propia. Estas serán las responsables de hacer que tanto el agente como el oponente realicen los movimientos.

Ambos agentes cuentan con el mismo número de pasos máximo con el que contaba el agente en la fase anterior antes de volver a iniciar de 0 el episodio, es decir, un total de 50000.

4.5. Recompensas

La incorporación de un nuevo agente que interactúa con el entorno y debe realizar sus acciones de forma competitiva implica que se debe diseñar una función de recompensa que estimule la competitividad entre ambos agentes. Como idea principal, se implementó una recompensa negativa cuando el contrincante llegase al objetivo. En otras palabras, cuando el agente colisiona con el objetivo, el oponente recibe una recompensa negativa, y cuando el oponente colisiona con el objetivo, el agente recibe una recompensa negativa.

La recompensa negativa que se ha implementado para ambos agentes cuando el contrincante llega al objetivo es de -1, lo cual busca que los agentes prioricen más el llegar lo antes posible al objetivo, ya que al no tratarse de una recompensa negativa que se aplica por cada step y llega únicamente cuando el contrincante ha llegado a la meta, los agentes pueden tratar de llegar a la meta procurando que no pase el tiempo suficiente para que esa recompensa negativa llegue.

Las recompensas mencionadas en la fase anterior se siguen manteniendo para esta fase, es decir, cada agente obtendrá una recompensa con valor 1 cada vez que llegue al objetivo y una recompensa con valor -1 cada vez que colisionaban con la pared. La recompensa negativa de valor 10^{-4} también se mantiene, con el fin de que minimice los pasos necesarios para llegar al objetivo.

4.6. Conclusiones

Tras observar las diferentes ejecuciones que se realizaron con los agentes, pudo observarse que en ocasiones uno de los agentes, bien el oponente o bien el agente, no eran capaces de aprender cuál era la tarea que debian realizar, ya que uno de los dos aprendía a realizar la tarea muy rapido y el otro únicamente recibía recompensas negativas sin posibilidad de tener la opción de mejorar. Este tipo de entrenamiento supone un problema ya que en principio no parece viable un aprendizaje simultáneo si el éxito de uno entorpece el aprendizaje del otro.

Tanto en la primera como en la segunda subfase, se obtuvieron gráficas en las que se podían observar como uno de los agentes era capaz de obtener una recompensa positiva y el otro no. Esto se traduce a que uno de los agentes sabe llegar a la meta mientras que el otro no es capaz de ello. El agente que llega a la meta obtiene de forma constante un

estímulo positivo por realizar bien la tarea, mientras que el agente que no es capaz de llegar solo obtiene recompensas negativas seguramente sin poder relacionar los eventos por falta de información sobre estos.

Como puede verse en la Figura 4.3, el oponente no es capaz de obtener una recompensa positiva, mientras que la Figura 4.4 deja ver que el agente sí que muestra unos resultados positivos. La gráfica del oponente se mantiene durante toda la ejecución por debajo del 0, dejando ver que claramente no es capaz de resolver el problema con el entrenamiento recibido, obteniendo valores que oscilan entre -0.05 y -0.4. Sin embargo, en el caso del agente se puede ver un periodo inestable durante los primeros 1500000 pasos, debido a un entrenamiento temprano y una política que no ha sido actualizada lo suficiente, haciendo que los valores no se mantengan en una zona en concreto y aparezcan picos notables hacia arriba y hacia abajo. Tras ese periodo comienza a obtener recompensas con valores más altos y los valores de la gráfica suben hasta oscilar la mayoría de los episodios entre 0 y 1, llegando así a los 3000000 pasos. A partir de los 3200000 de pasos aproximadamente los valores vuelven a desplomarse, fruto de un proceso de exploración de posibles acciones a ejecutar para obtener mayores recompensas, y se ven picos hacia abajo que llegan a alcanzar los -0.8 en recompensa negativa, en un periodo que durará hasta los 4000000 pasos. A partir de ahí vuelven a subir las recompensas y hasta el final se mantienen la gran mayoría del tiempo en valores comprendidos entre 1 y 0.6.

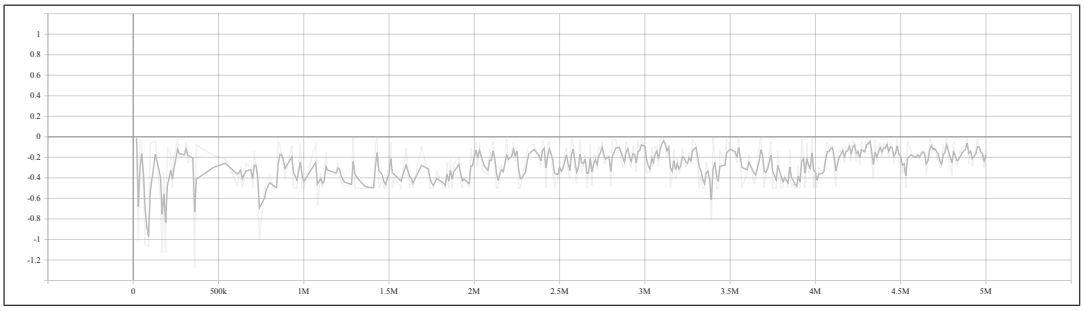


Figura 4.3: Gráfica de recompensa del oponente en las primeras dos subfases

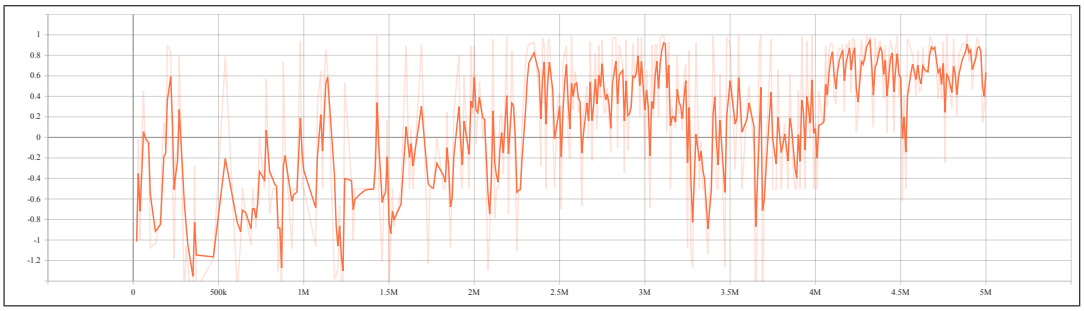


Figura 4.4: Gráfica de recompensa del agente en las primeras dos subfases

También se dieron unos resultados en algunas ejecuciones, en los que ninguno de los dos agentes fue capaz de aprender a llegar a la meta tras el entrenamiento. Estos conseguían en contadas ocasiones realizar correctamente la tarea pero no resultaba el comportamiento habitual para ninguno de los dos. Tanto en la Figura 4.5 como en la Figura 4.6 se puede ver que ninguno de los dos agentes ha obtenido resultados positivos. A pesar de que ofrecía una recompensa positiva para que los agentes aprendieran a llegar al objetivo, no se obtuvieron apenas resultados positivos durante el proceso.

En ambas gráficas puede verse que los resultados son negativos durante todo el proceso, llegando a casi no traspasar siquiera el umbral de -0.2 para el caso del agente. Para el oponente, los resultados varían un poco aunque en conjunto también resultan ser bastante negativos. A pesar de que se vea mucha inestabilidad en la gráfica y aparezcan muchos picos hacia arriba o hacia abajo por las constantes diferencias en la recompensa obtenida, la mayoría de los picos se dan en valores negativos de la escala de recompensa, por lo que se puede deducir que los picos en los que la recompensa sube de forma espontánea vienen dados por apariciones cercanas al objetivo ligadas a movimientos aleatorios que han hecho que el oponente llegue al objetivo sin estar previsto como tal.

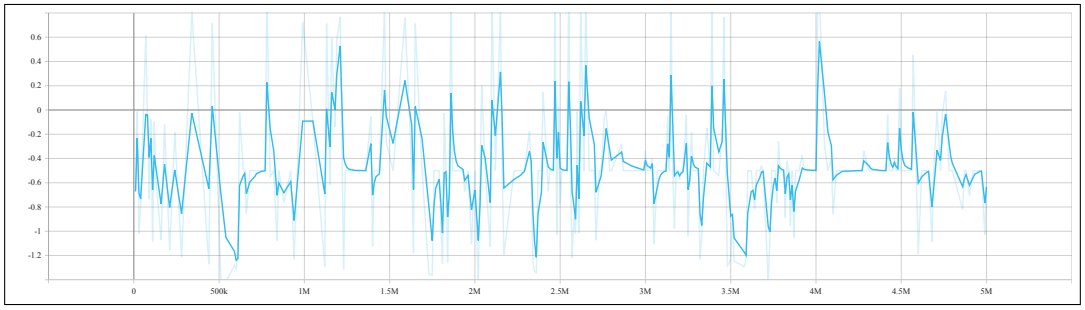


Figura 4.5: Gráfica de recompensa del oponente en la primera subfase

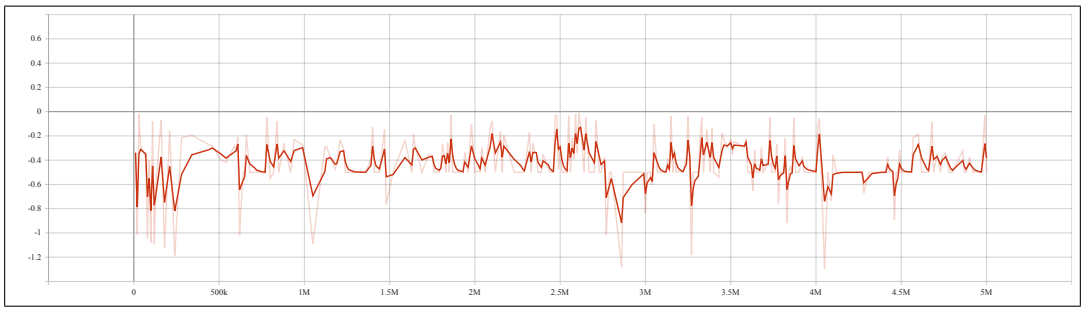


Figura 4.6: Gráfica de recompensa del agente en la primera subfase

Dentro de las ejecuciones de estas dos subfases, llegaron a encontrarse resultados que eran desfavorables para ambos agentes, lo cual dejaba ver que el modelo de recompensas no era adecuado para el escenario, por lo que se podría llegar a pensar que las recompensas negativas tienen una influencia mayor que las positivas.

Por este motivo se decidió desarrollar una variante en la que ambos agentes realizasen el entrenamiento de forma independiente y sin aplicarse recompensas negativas entre ellos, con el objetivo de comprobar si con este método eran capaces de realizar un entrenamiento positivo para ambos. El agente y el oponente compiten por llegar a la meta los primeros y para ello normalmente suelen inspeccionar el entorno que les rodea, procurando localizar el objetivo, a su adversario o incluso ambos al mismo tiempo. A través de un entrenamiento independiente pueden tener el tiempo necesario para realizar esta exploración. En caso de solo localizar al adversario, podía darse el caso de que se dirigieran hacia él para obstaculizar y empujar, lo cual no era una interacción esperada y que ha resultado toda una sorpresa. Si los agentes tenían localizado el objetivo, se dirigían a colisionar con él como en la primera fase, aunque en algunas ocasiones puede que hasta obstaculizando al adversario para hacerse con la ventaja de la carrera si ambos estaban llegando al objetivo. Este caso de

comportamientos solo se ha dado en pruebas con entornos más pequeños, ya que con el principal contaban con mucho espacio y se centraban en la exploración.

Los resultados son positivos, tal y como puede verse en la Figura 4.7 y en la Figura 4.8, pues ambos agentes son capaces de interactuar entre ellos a pesar de realizar un entrenamiento independiente (posiblemente debido a que realizaron el entrenamiento en el mismo entorno) y también aprenden a cumplir la tarea. Se tiene en cuenta para el desarrollo de la siguiente fase las interacciones que han tenido los agentes entre sí en este escenario en concreto.

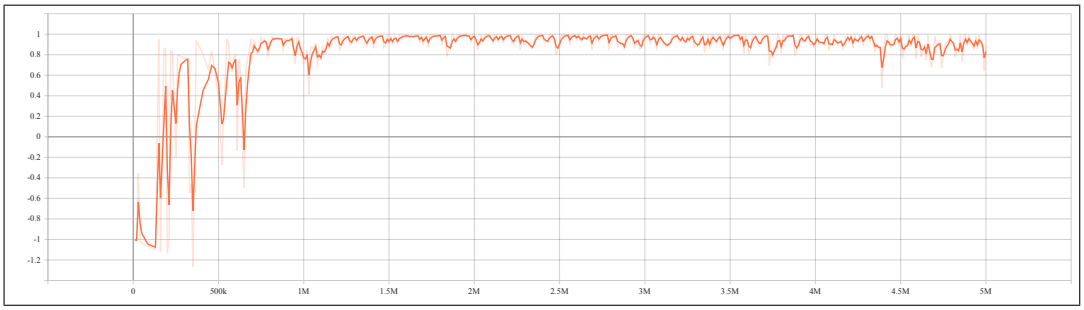


Figura 4.7: Gráfica de recompensa del agente en la ejecución independiente

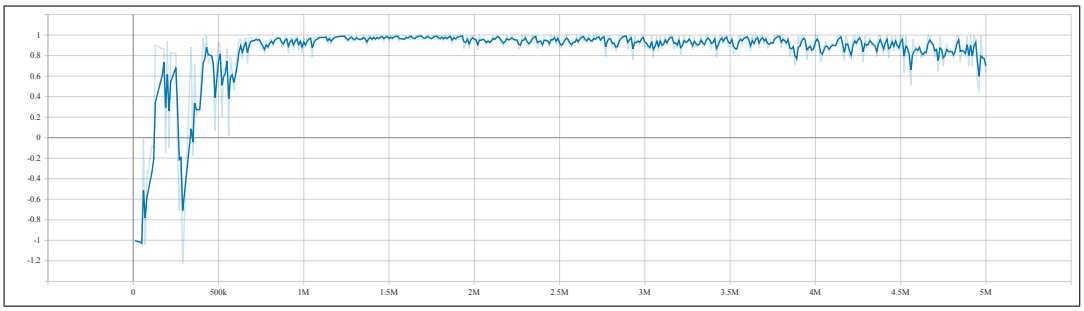


Figura 4.8: Gráfica de recompensa del oponente en la ejecución independiente

El enlace al video con la demostración de la ejecución del agente usando el modelo entrenado se encuentra en el anexo 6.

CAPÍTULO 5

Fase 3: Entorno complejo con dos agentes

En este capítulo se explicará la fase final del proyecto, siendo este un laberinto que los dos agentes deben ser capaces de explorar para llegar al objetivo.

5.1. Construcción del escenario

La fase anterior tuvo como propósito poder observar cómo podían interaccionar los agentes entre ellos, mientras que en esta fase se busca ver la capacidad que muestran para resolver un laberinto cuando comparten entorno y cuando además son enfrentados entre ellos. En el caso del escenario, se genera de forma procedural en todas las ejecuciones, por lo que cada ejecución genera un laberinto de forma aleatoria. El laberinto se diseñó para que tuviera en cuenta la entrada de alto y ancho que se deseaba para este, por lo que pueden generarse laberintos de diferentes tamaños dependiendo del escenario que se quiera obtener.

Para la generación del laberinto se ha empleado *backtracking* recursivo. El algoritmo empleado para la generación del algoritmo tiene un funcionamiento que se puede resumir en cuatro pasos:

- Se selecciona una celda inicial para comenzar el proceso.
- Se selecciona una de las celdas adyacentes a esta, y se comprueba si esta está visitada o no. En caso de no estar visitada, se crea una unión entre la celda anterior y esta.
- Se repite este proceso hasta que no queden más celdas adyacentes que visitar.
- Se hace *backtracking* hasta llegar a una celda que tenga alguna celda con la que crear una unión.
- El algoritmo termina cuando se vuelve a la celda inicial haciendo *backtracking* y no queda ninguna celda con la que crear una unión.

Este algoritmo está pensado para ser ejecutado en laberintos de tamaño pequeño y mediano, ya que su ejecución realiza muchas llamadas recursivas y supone un coste considerable de memoria realizarlo sobre laberintos de gran tamaño.

Tanto los agentes como el objetivo de la fase anterior se mantienen y no tienen ningún tipo de modificación en sus propiedades.

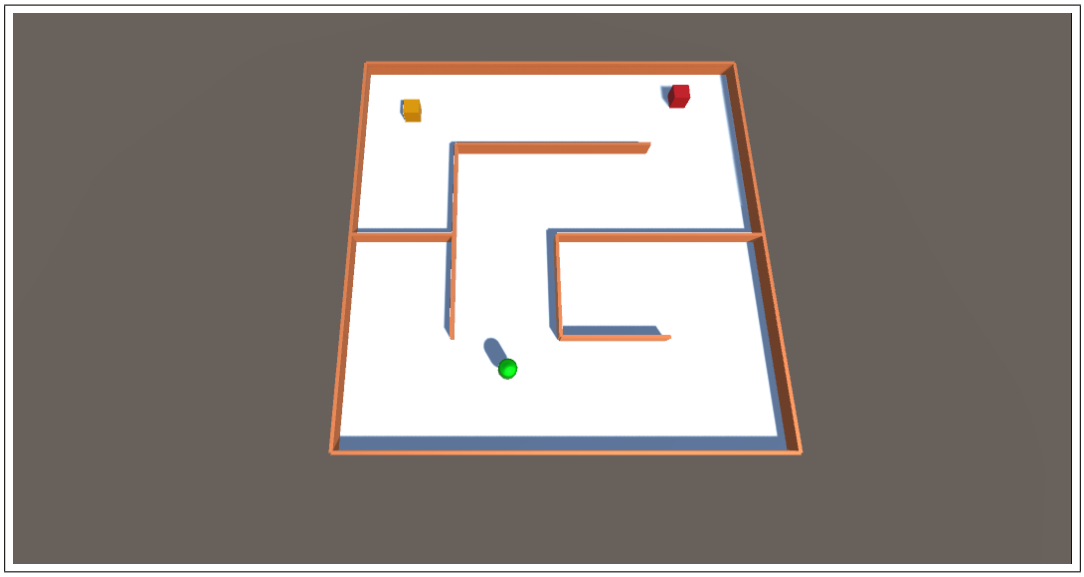


Figura 5.1: Fase 3: Escenario de dimensiones 4x4

5.2. Subfases de testeo

Para esta fase final se han planteado tres subfases. La primera de las subfases se realizará sobre un laberinto de dimensiones reducidas, concretamente 4 celdas de alto y 4 de ancho, visible en la figura 5.1. Cada una de estas celdas tiene unas dimensiones de 5 unidades de alto y 5 de ancho. Se realiza una subfase con entrenamiento sobre un laberinto pequeño para observar el comportamiento y desempeño de los agentes cuando se incluyen obstáculos y deben realizar una exploración del entorno teniendo estos obstáculos en cuenta para ello.

La segunda subfase tendrá un laberinto con mayores dimensiones, unas dimensiones de 8 celdas de alto y 8 de ancho, como el de la Figura 5.2. Con esto se pretende comprobar el desempeño de los agentes realizando un entrenamiento en un entorno con mayores dimensiones y mayor complejidad. Los agentes deberán ser capaces de realizar una mayor exploración sobre el entorno, ya que este es más grande y contiene más caminos que pueden ser explorados.

Para la tercera y última fase, se implementa un nuevo elemento en el entorno, Figura 5.3. Este elemento será referido como puerta y es un muro que cuenta con la característica de bloquear la visión de los agentes tal y como lo hacen los muros, pero al entrar en contacto con cualquiera de los agentes desaparece y permite que estos puedan ver lo que antes esta puerta no les permitía y además caminar a través del hueco que ha dejado libre en el laberinto. Estas puertas son colocadas en el laberinto de forma aleatoria sustituyendo a algunos muros en la generación de este, visible en la Figura 5.4 y se puede decidir el número de puertas con el que debe contar el laberinto. En el momento en el que el episodio termina

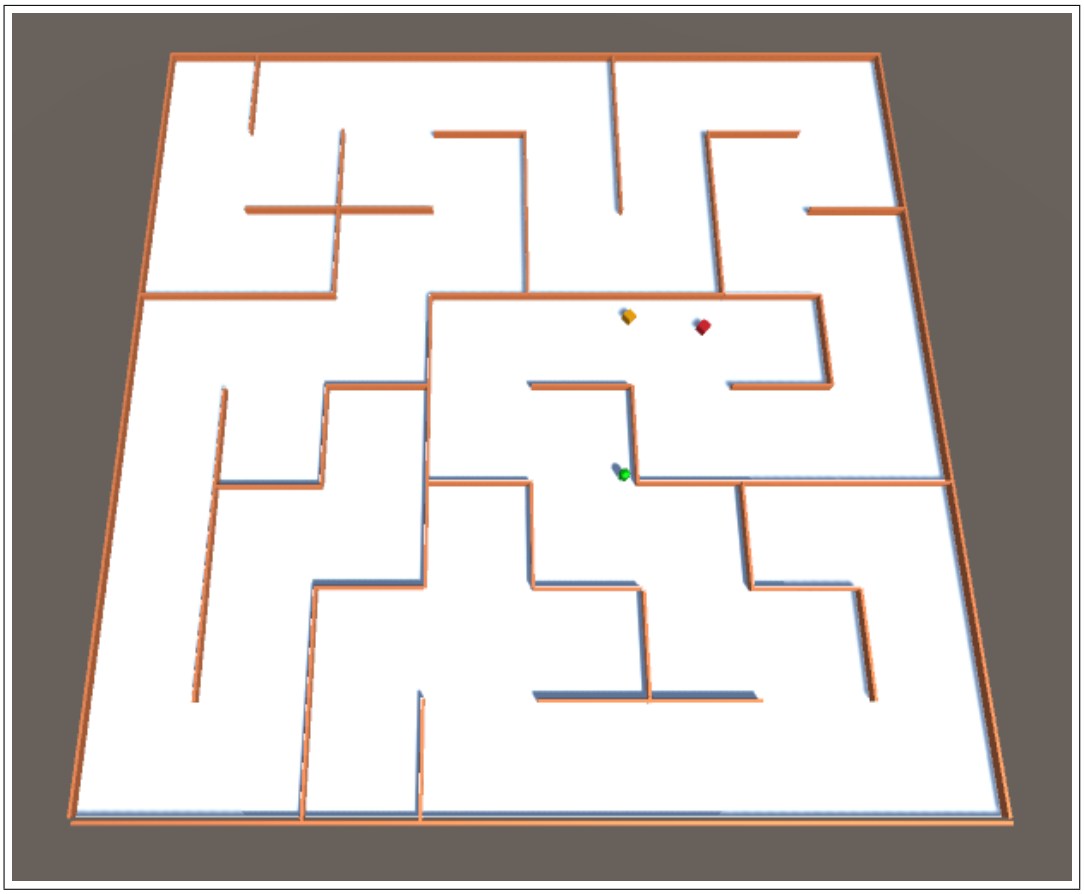


Figura 5.2: Fase 3: Escenario de 8x8

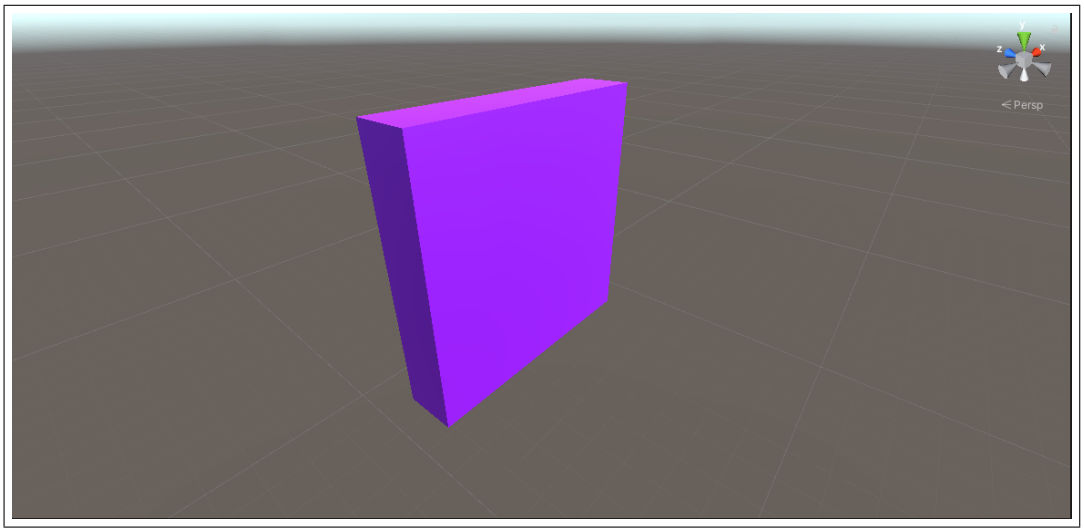


Figura 5.3: Puerta creada para la Fase 3

estas vuelven a aparecer en caso de haber sido desactivadas. El objetivo de estas puertas en el laberinto es facilitar, en cierta medida, que el agente pueda explorar el laberinto por zonas a las que no podría acceder de forma directa sin dar un rodeo por los pasillos del laberinto. Además de ganar visión sobre otros puntos del laberinto, puede ayudar al agente

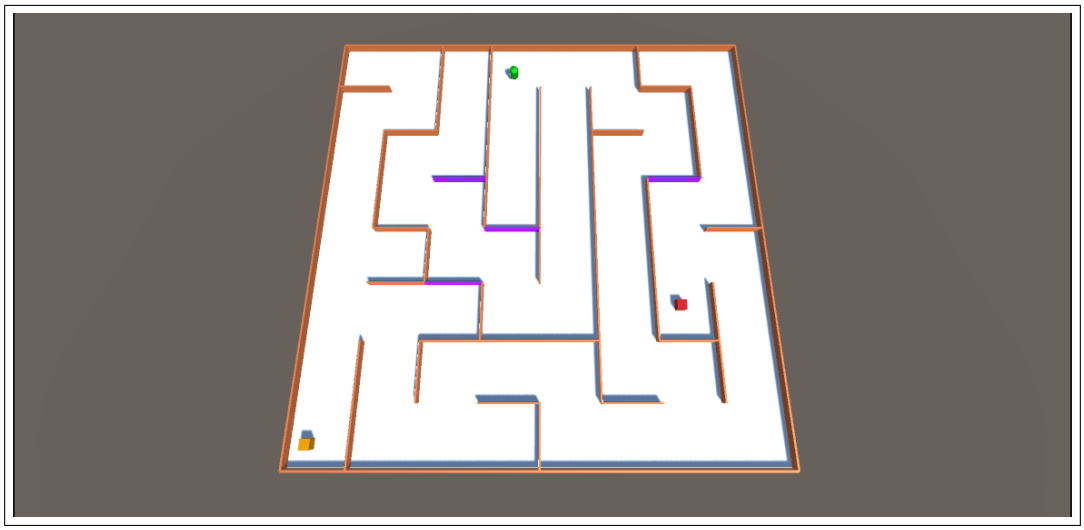


Figura 5.4: Fase 3: Escenario de 8x8 con varias puertas

a acortar distancias con respecto al objetivo de forma sencilla y eficaz.

5.3. Observaciones

Para esta fase se mantienen las mismas entradas de observación que en la primera fase y en la segunda fase. El oponente cuenta con un componente que otorga los mismos rayos con la misma longitud y mismos ángulos que los del agente, es decir, contará con los 15 rayos con los que cuenta el agente. El agente es capaz de detectar las paredes, las puertas, el objetivo y al oponente y de la misma manera, el oponente es capaz de detectar las paredes, las puertas, el objetivo y al agente. Puesto que ahora también son capaces de detectar las paredes, se añade una nueva entrada para cada rayo en la red neuronal, pasando a ser el número de entradas de ambas redes neuronales a 90 (15 rayos x 6 datos de entorno) y las entradas que proporciona cada rayo así:

- Entrada 1: 1.0 si el rayo colisiona con una pared, 0.0 si no lo hace.
- Entrada 2: 1.0 si el rayo colisiona con el objetivo, 0.0 si no lo hace.
- Entrada 3: 1.0 si el rayo colisiona con el oponente, 0.0 si no lo hace.
- Entrada 4: 1.0 si el rayo colisiona con una puerta, 0.0 si no lo hace.
- Entrada 5: 1.0 si el rayo no ha colisionado con nada, 0.0 si no lo hace.
- Entrada 6: 1.0 La distancia a la que está el objeto con el que ha colisionado (1.0 si no colisiona con nada).

En el caso del oponente, en vez de reconocer la colisión con el oponente, reconoce la colisión con el agente.

En esta fase sigue siendo relevante que los dos agentes puedan reconocerse entre ellos ya que se ha visto que en ocasiones estos pueden interactuar entre ellos a pesar de no obtener recompensa por hacerlo.

5.4. Movimiento

De la misma forma que en la fase anterior, ambos agentes contarán con los mismos movimientos, planteados desde la primera fase. Se seguirá el mismo proceso de obtención de movimientos que se tuvo en la segunda fase, es decir, los movimientos y la decisión de estos sigue el mismo método que seguía en la segunda fase.

Para la primera fase se siguen manteniendo los pasos máximos por agente de las anteriores fases, 50000. En las segunda y tercera subfase estos son ampliados a 250000 ya que el escenario es claramente más grande y así se les brinda la oportunidad de tener el tiempo necesario para realizar la exploración.

5.5. Recompensas

La aparición de un entorno más complejo puede requerir de sistemas de recompensas más depurados y sofisticados, que ayuden a los agentes a obtener mejores resultados guiándose en la recompensa que obtienen tras realizar sus acciones. Es por esto que para esta fase se han modificado las recompensas que se han utilizado hasta el momento. El sistema de recompensas se aplica para ambos agentes, igual que en las dos fases anteriores.

Se han mantenido varias recompensas de fases anteriores. Se mantiene la recompensa positiva de +1 cuando el agente llega a la meta. También se mantiene la recompensa negativa de -1 que se aplica cuando el agente toca o colisiona con una de las paredes del entorno. La otra recompensa negativa mantenida, la cual se aplica cada vez que el agente realiza una acción,es la que tiene un valor de -10^{-4}.

Para esta fase se han añadido varias recompensas. La primera recompensa tiene un valor de 0.001 y se aplica cada vez que cualquiera de los agentes reduzca su distancia con respecto al objetivo en cada episodio, es decir, siempre que el agente acorte la distancia a la que ha estado del objetivo durante ese episodio ganará una recompensa de 0.001. Esta recompensa se aplica con el objetivo de que los agentes tengan una noción mínima de hacia donde está el objetivo, a pesar de ser un laberinto, ya que muchas veces podría resultar de ayuda en la toma de decisiones. También se aplica una recompensa positiva de 0.0001 cada vez que los agentes se mueven hacia delante. Esta recompensa está pensada para ayudar a los agentes en su labor de exploración del laberinto.

La última recompensa añadida se otorga al activar una puerta, aplicando una recompensa positiva de 0.001 por cada puerta activada. La activación de las puertas ayuda a que los agentes puedan explorar zonas del laberinto que no son accesibles sin recorrer largas distancias, creando así pequeños atajos que podrán usar para así lograr una mejor recompensa.

5.6. Conclusiones

En la ejecución de la primera subfase se observó que ambos agentes aprendieron a solucionar el laberinto teniendo unas dimensiones reducidas. Estos fueron capaces de realizar una exploración a través de los pasillos y llegar al objetivo. Las recompensas de ambos agentes en las gráficas obtenidas son bastante similares, siendo la Figura 5.5 la del

agente y la Figura 5.6 la del oponente. Ambas tienen una mejoría visible en la recompensa obtenida a lo largo de la ejecución.

En la gráfica del agente puede verse que tiene un desempeño bastante malo hasta el millón de pasos ejecutados, más o menos, por las recompensas negativas tan bajas obtenidas, cercanas a -0.6 (a excepción de algunos picos hacia arriba por encontrar el objetivo con movimientos aleatorios, muy probablemente). A partir del *step* 1 millón comienza a aumentar la recompensa y se puede ver que se mantiene entre los valores 0.4 y -0.4, por lo que se puede intuir que está aprendiendo a llegar al objetivo siempre que le resulte relativamente sencillo acceder a él. A partir del millón y medio de pasos las recompensas bajan durante 250000 pasos aproximadamente, fruto de una labor de aprendizaje en cuanto a cómo realizar la exploración por el laberinto. Tras esta bajada las recompensas suben de nuevo y se mantienen, en general, en valores positivos. Esto indica que el agente ha sido capaz de aprender a explorar el laberinto y que es capaz de buscarlo por el laberinto sin problemas en la mayoría de los episodios.

Por otro lado, en el caso del oponente, hasta los dos millones de pasos no tiene recompensas altas y suelen ser por lo general una sucesión de altibajos en la zona de negativa de recompensas, teniendo algunos claros en los que se deduce que el agente no era capaz de llegar al objetivo. A partir de los dos millones la recompensa del oponente sube y comienza a oscilar entre los valores 0.4 y -0.4 hasta el final de la ejecución. No son recompensas especialmente altas pero se aprecia que el oponente ha aprendido cuál es su objetivo y como llegar hasta él en algunas ocasiones, seguramente sin ser capaz de realizar una exploración tan buena como la del otro agente.

En el anexo 6 del documento se ha incluido un enlace a un video en el que puede notarse como a pesar de que ambos agentes son capaces de llegar al objetivo, el agente rojo es mas habil que el amarillo. Esto deja plasmado de forma más visual la información obtenida de estas gráficas.

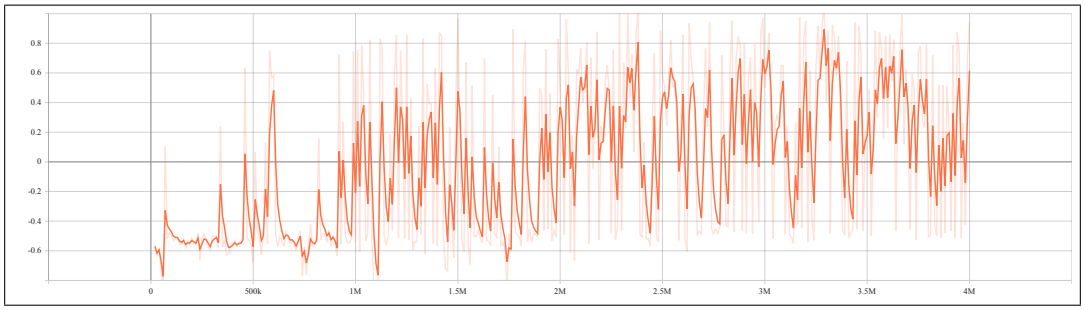


Figura 5.5: Gráfica de recompensa del agente en la en las primeras dos subfases

En la segunda subfase, con un escenario más grande y más complicado, se obtuvieron unos resultados bastante peores, tal y como puede verse en las gráficas 5.7 y 5.8. Ambos agentes obtienen unas recompensas negativas durante la gran mayoría de la ejecución y no consiguen aprender a solucionar el problema. Ambas gráficas se mueven durante gran parte de la ejecución por valores cercanos al -1, teniendo pequeños picos hacia arriba en los que consiguen hacer que la recompensa supere el umbral de 0, tras llegar al objetivo, pero que no son para nada representativos, ya que a lo largo de la gráfica puede verse que sus recompensas son siempre bajas. Principalmente se puede deber a la aparición cercana del objetivo con respecto a cualquiera de los dos agentes.

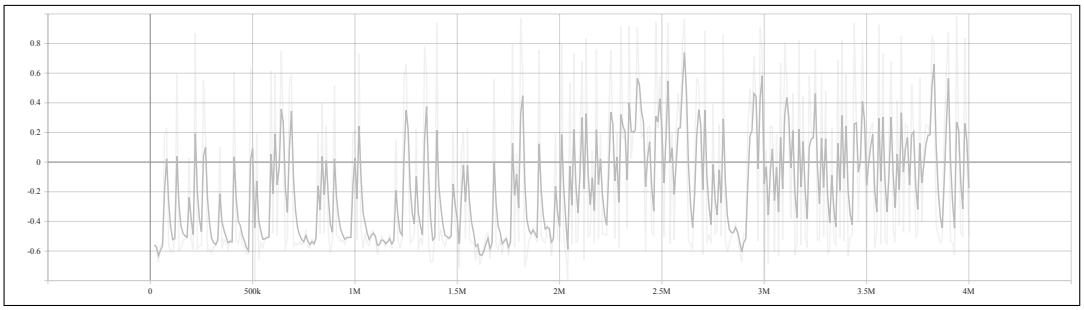


Figura 5.6: Gráfica de recompensa del oponente en las primeras dos subfases

La gráfica muestra unos valores que indican que el problema no es resuelto por los agentes y que la complejidad de este ha sido demasiado alta para su aprendizaje. Puesto que la cantidad de veces que se ha superado el umbral de 0 ha sido bastante baja, podría achacarse a que el sistema de recompensas no estaba lo suficientemente depurado como para que los agente fueran capaces de aprender su objetivo. Además, como el entorno no se recorre por cuadrículas sino por pasos de una distancia determinada en un espacio, a los agentes les puede resultar muy difícil realizar la exploración por lo que podría deberse a una cuestión de capacidad de exploración.

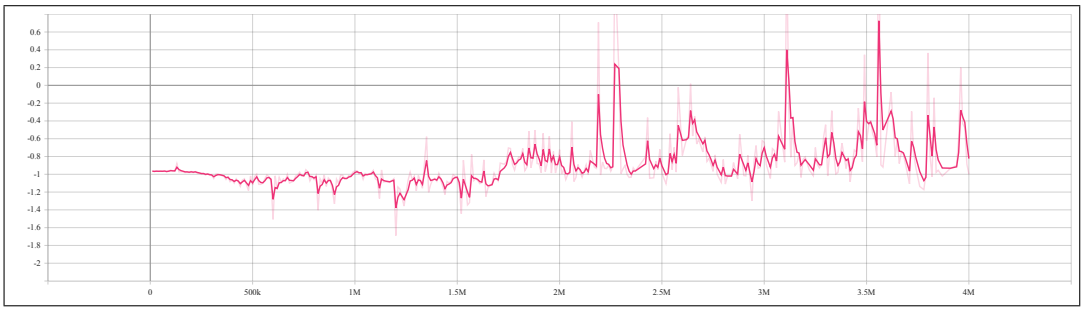


Figura 5.7: Gráfica de recompensa del oponente en la primera subfase

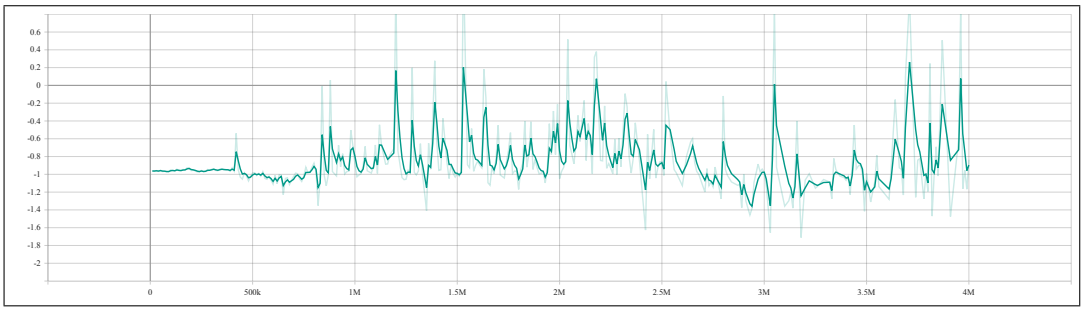


Figura 5.8: Gráfica de recompensa del agente en la primera subfase

Para la última subfase se implementa el nuevo elemento comentado anteriormente, las puertas. Con esto se espera que puedan ser capaces de realizar una exploración con mayor facilidad en un entorno que presenta complejidad para el aprendizaje de los agentes y en el que no se han obtenido unos resultados favorables.

Las gráficas visibles tanto en la Figura 5.9, referente a la gráfica de recompensas del agente, y la Figura 5.10 dejan ver que los resultados son muy similares a los vistos en la anterior subfase, en gran parte de la ejecución se mantienen negativos y se pueden observar

picos en los que la recompensa de los agentes pasa a ser superior a 0 de forma puntual. En estas gráficas obtenidas se observa que las recompensas negativas tienen como eje central un valor de -0.8, aunque en una ocasión el agente obtiene una recompensa positiva de 0.6, siendo este su máximo, y una recompensa negativa de -1.6, convirtiéndose este en su mínimo. El oponente también tiene unos impactos parecidos en su gráfica, siendo los valores de su máximo 0.4 y de su mínimo -1.8.

Haciendo una comparativa global de los resultados obtenidos en las gráficas de la anterior subfase y está, la aparición de las puertas ha aumentado ligeramente la recompensa que han obtenido los agentes a lo largo de la ejecución, muy probablemente por la obtención de recompensas al activar una puerta. Puesto que gracias a las puertas los agentes han podido obtener una mayor capacidad de exploración en el entorno, el problema apunta a que el sistema de recompensas no es el adecuado para un problema complejo como el que se está tratando y debe corregirse.

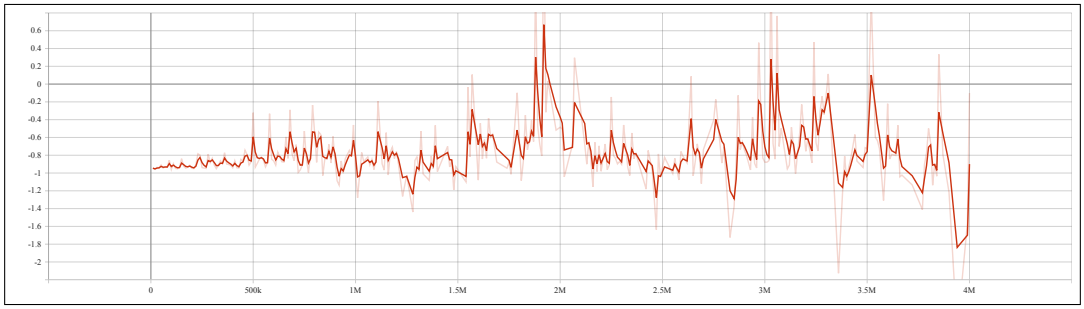


Figura 5.9: Gráfica de recompensa del agente en la ejecución independiente

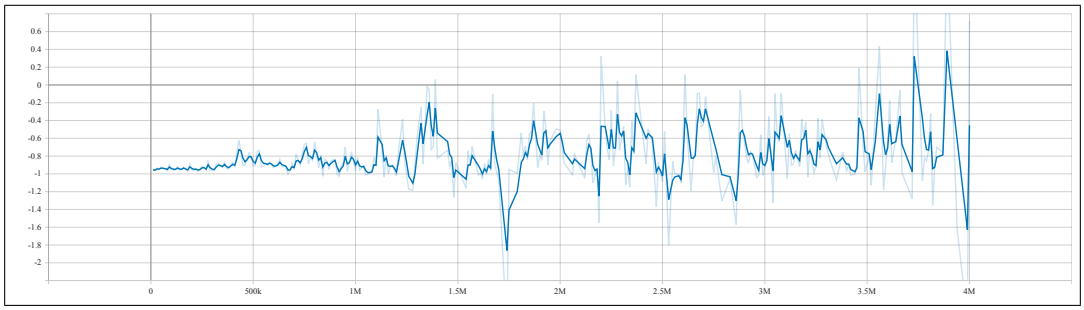


Figura 5.10: Gráfica de recompensa del oponente en la ejecución independiente

Este sistema ha resultado efectivo para un entorno más pequeño y asequible para los agentes en los que la cantidad de obstáculos era más baja, pero a medida que la dificultad de este entorno aumenta, los resultados comienzan a ser peores, ya que se incrementa la cantidad de caminos posibles y su recorrido. La estrategia que deben desarrollar es mucho más compleja e improbable de obtener.

CAPÍTULO 6

Conclusiones y trabajo futuro

Este proyecto comenzó con una comparación entre algoritmos para poder obtener el comportamiento de estos en diferentes escenarios, siendo estos cada vez más complejos. Esta comparación se desarrolló en tres fases distintas, de complejidad ascendente, que tras avanzar descartó algún algoritmo bien por complicaciones con las tecnologías a usar o bien porque no era adecuado para el entrenamiento.

En la primera fase se realizó la comparación entre los algoritmos SAC y PPO, ya que para SARSA y Q-Learning no fue posible realizar la ejecución debido al comunicador. Para esta comparación se usó un agente en un entorno pequeño y sin obstáculos. Este agente debía llegar a un objetivo situado en este entorno tras realizar ambos una aparición aleatoria en este y colisionar con el objetivo. Para este problema concreto PPO demostró ser más rápido que SAC y obtener mejores resultados que este por el entrenamiento realizado con el uso de réplicas. Puesto que el entrenamiento se seguiría realizando con réplicas, seguir usando el algoritmo SAC no iba a aportar ningún tipo de beneficio e iba a suponer una gran sobrecarga computacional, por lo que se continuó comprobando el desempeño de PPO.

En la segunda fase se comprobó con dos agentes compartiendo un entorno pero con el mismo objetivo que en la fase anterior. Los éxitos de un agente tenían una influencia negativa directa sobre las recompensas del otro agente y viceversa, llegando así a la conclusión de que esta no favorecía el aprendizaje de estos, sino que lo entorpecería. Se decidió realizar el entrenamiento de los agentes de forma independiente entre ambos para observar sus resultados y los de ambos fueron positivos, llegando a interactuar entre ellos en ocasiones, lo cual no estaba previsto y fue sorprendente.

En la tercera y última fase se realizó la prueba sobre distintos escenarios, siendo el primero un laberinto simple con unas dimensiones de 4 celdas de ancho y 4 celdas de alto. En este laberinto los agentes fueron capaces de resolver el problema planteado sin demasiada complicación y obteniendo unos resultados aceptables. Para la segunda subfase se amplió el área del laberinto, convirtiéndolo en un laberinto con dimensiones de 8 celdas de alto y 8 celdas de ancho y por tanto aumentando la complejidad. Con una complejidad mayor, los agentes dejan de obtener recompensas positivas y queda expuesto un sistema de recompensas que no funciona para el problema. Estos no consiguen obtener un buen entrenamiento y por tanto no saben llegar al objetivo explorando el laberinto. Tras in-

corporar puertas de forma aleatoria a este laberinto, los agentes mejoran ligeramente la recompensa pero no se soluciona el problema, siendo necesario un nuevo planteamiento para este sistema de recompensas.

Dentro del proyecto como tal existen varias líneas de desarrollo. Como primera línea lógica a seguir, se podrían comparar los algoritmos que no se han podido comprobar e intentar encontrar formas de mejorar los resultados obtenidos en los ya ejecutados. Otra línea de desarrollo podría ser la implementación de un sistema de recompensas que sea realmente válido para un entorno complejo como un laberinto grande o un método que facilite en entrenamiento competitivo entre los agentes sin que esto dificulte el entrenamiento. También se podrían seguir implementando más agentes y simular una competición de varios jugadores.

Una línea de trabajo futuro extraíble de este proyecto sería la aplicación de aprendizaje incremental en estos agentes a lo largo de las fases que van atravesando, es decir, que los agentes inicien el aprendizaje de la segunda fase con la política aprendida en la primera, y que una vez son capaces de dominar esta segunda fase, pasen a la tercera. Con esta línea de trabajo se podría buscar aumentar la eficacia del entrenamiento que realizan, puesto que no tienen que aprender una política desde 0, sino que pueden adaptarse.

Otro punto a tratar para próximos trabajos sería la aparición de todos los agentes a la misma distancia del objetivo, no siendo distancia de lejanía sino de recorrido a realizar para llegar a este. Este punto está orientado a los laberintos, ya que en el resto de entornos del proyecto se ha aplicado. Este punto está en gran medida orientado a los trabajos que estén orientados a entornos competitivos.

Puesto que es una tecnología que está en constante desarrollo y cada vez obtiene funcionalidades distintas, el análisis e implementación de entrenamientos competitivos o cooperativos haciendo uso de curriculums y metacurriculums daría pie a una profundización mayor sobre cómo incrementar la eficiencia de entrenamiento en entornos multiagente de Unity.

Apéndice

Videos de las ejecuciones

- Resultado primera fase. En este primer video se puede ver que el agente es capaz de llegar al objetivo sin ningún problema haciendo una exploración del entorno para localizar el objetivo.
- Resultado negativo para ambos agentes en la primera y segunda subfase de la segunda fase. En este video se observa que ninguno de los dos agentes es capaz de realizar movimientos de exploración y mucho menos llega al objetivo. No han obtenido buenos resultados tras el entrenamiento.
- Resultado negativo para uno de los agentes en la primera y segunda subfase de la segunda fase. En este caso, se puede ver como uno de los agentes es capaz de realizar una exploración completa del entorno buscando el objetivo mientras el otro no muestra una estrategia clara para encontrar el objetivo y apenas se mueve del punto de origen.
- Resultado positivo para agentes en la primera y segunda subfase de la segunda fase. En este video puede verse que ambos agentes son capaces de llegar al objetivo, por lo que realizan una exploración del entorno y tratan de llegar al objetivo antes que su oponente.
- Resultado positivo para agentes en la primera y segunda subfase de la segunda fase con interacciones. En este video se muestra como los agentes se obstaculizan en múltiples ocasiones un entorno reducido.
- Resultado de la primera subfase de la tercera fase. En este video se muestra como los agentes son capaces de solucionar el laberinto planteado en la primera subfase de la fase final tras realizar el entrenamiento.

ML-Agents Entorno de aprendizaje

En el capítulo introductorio de este documento se hacía referencia al entorno de aprendizaje que se tiene en ML-Agents sin profundizar en el, pero es conveniente conocer los diferentes elementos que lo componen y así tener una mayor comprensión sobre la conexión que se realiza entre los diferentes elementos del proyecto. El entorno de aprendizaje cuenta con tres elementos:

- Agent: El agente es explicado en la sección introductoria del documento junto al resto de elementos que actúan sobre él.

- Brain: Se encarga de definir un estado y decide lo que va a hacer el agente. Existen diferentes tipos de *brains* pero en este proyecto se usa el de tipo externo, siendo Tensorflow el que decida las acciones que se deben tomar y transmitiendolas a través de un *socket*.

- Academy: Este objeto contiene los cerebros del entorno. Solo se tiene un Academy por entorno y guarda información como la duración global del episodio.

En la Figura 1 se puede ver un pequeño esquema con los componentes comentados. Aunque solo se vea un cerebro, existe la posibilidad de tener más de uno y que cada uno de estos cerebros esté conectado a un agente o a más de uno.

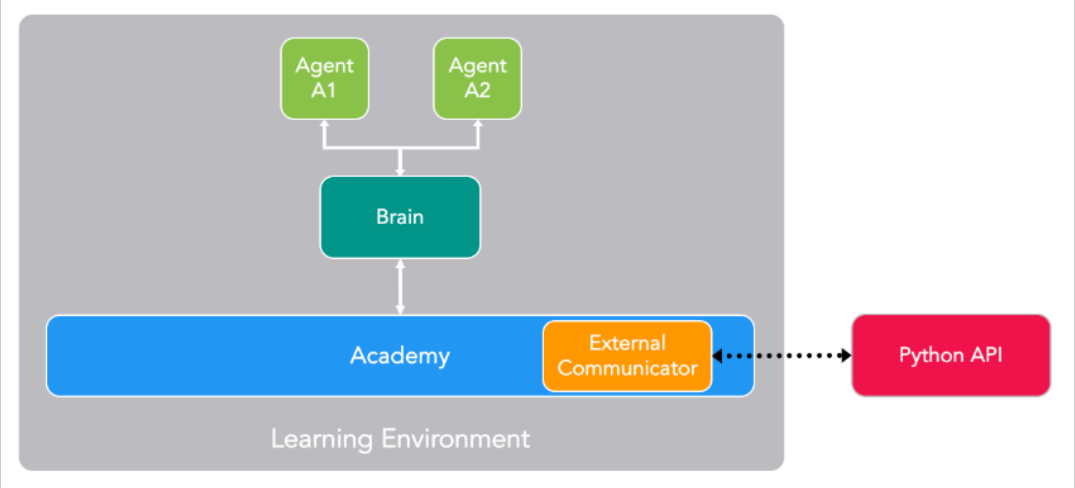


Figura 1: Diagrama de componentes del entorno de aprendizaje

Entrenamiento de los agentes en Unity

En esta sección se comentará de forma breve cómo se realiza el entrenamiento de los agentes.

Para ejecutar el entrenamiento de estos agentes, primero se debe abrir una ventana de comandos, después cargar el environment en el que se están cargando todos los programas de este proyecto, en caso de tenerlo creado. Es aconsejable crear un environment para que las versiones de los programas empleados en el proyecto no interfieran con los instalados en el equipo para uso cotidiano y viceversa.

Una vez se tiene el environment creado y activado, se ejecuta el siguiente comando: mlagents-learn config/PPODobleAgente.yml –run-id=Agente, el cual indica que se debe entrenar un cerebro que se llamará Agente usando como base los parámetros y las configuraciones indicadas en el archivo PPODobleAgente.yml

Una vez se realice el entrenamiento, se generarán siempre 3 elementos distintos.

- Resúmenes. Son las métricas que obtenemos con la información que se ha ido recogiendo durante el entrenamiento. Esta información se puede visualizar a través de TensorBoard.

- Modelos. Contiene los diferentes puntos de guardado que se generan durante el entrenamiento y el modelo final en formato ONNX (Open Neural Network Exchange).

- Timers. Los timers son archivos JSON generados de forma automática en el entrenamiento, los cuales contienen métricas agregadas sobre el proceso de aprendizaje que se ha realizado.

config

Para determinar los parámetros de entrenamiento de los agentes, se usa un archivo de configuración en el que se determina el tamaño de la red neuronal para el entrenamiento del agente, como los steps máximos que puede dar, los valores de beta, epsilon...

Este archivo es un archivo con formato YAML y se pueden crear tantos como quieras, ya que a la hora de entrenar se debe especificar cual es el que quieres emplear.

Para la primera fase se emplearon dos archivos de configuración para el único agente de la escena, uno con la configuración del algoritmo SAC y otro con la configuración del algoritmo PPO.

Para la configuración de PPO se utilizo un archivo con el siguiente registro:

```
behaviors:
  MoverAObjetivo:
    trainer_type: ppo
    hyperparameters:
      batch_size: 32
      buffer_size: 2048
      learning_rate: 1.0e-5
      beta: 1.0e-2
      epsilon: 0.3
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 4
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 1000000
    time_horizon: 64
```

```
    summary_freq: 10000
```

Para el caso del algoritmo SAC se ha utilizado este otra configuración, ya que los algoritmos no usan exactamente los mismos parámetros:

```
behaviors:
    MoverAObjetivo:
    trainer_type: sac
      hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 1.0e-3
      learning_rate_schedule: constant
      buffer_init_steps: 0
      tau: 0.01
      steps_per_update: 10.0
      save_replay_buffer: false
      init_entcoef: 0.5
      reward_signal_steps_per_update: 10.0
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.95
       strength: 1.0
    max_steps: 1000000
    time_horizon: 64
    summary_freq: 10000
```

Para la segunda fase, se utilizó un único archivo que recogía la configuración de ambos agentes, siendo estas independientes entre sí. Para entrenar los dos agentes a la vez, se han configurado las redes neuronales en el mismo archivo de config, llamado PPODobleAgente. Para ambos agentes se emplea la misma configuración y se entrenan con el mismo algoritmo. El archivo en cuestion recogía esta configuración:

```
      MoverAObjetivoAgente:
    trainer_type: ppo
    hyperparameters:
      batch_size: 32
      buffer_size: 2048
      learning_rate: 1.0e-5
      beta: 1.0e-2
      epsilon: 0.3
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
```

```
  network_settings:
    normalize: false
    hidden_units: 256
    num_layers: 2
    vis_encode_type: simple
  reward_signals:
    extrinsic:
      gamma: 0.99
      strength: 1.0
  keep_checkpoints: 5
  max_steps: 5000000
  time_horizon: 64
  summary_freq: 10000
MoverAObjetivoOponente:
  trainer_type: ppo
  hyperparameters:
    batch_size: 32
    buffer_size: 2048
    learning_rate: 1.0e-5
    beta: 1.0e-2
    epsilon: 0.3
    lambd: 0.99
    num_epoch: 3
    learning_rate_schedule: linear
  network_settings:
    normalize: false
    hidden_units: 256
    num_layers: 2
    vis_encode_type: simple
  reward_signals:
    extrinsic:
      gamma: 0.99
      strength: 1.0
  keep_checkpoints: 5
  max_steps: 5000000
  time_horizon: 64
  summary_freq: 10000
```

Tabla de resultados de las ejecuciones de PPO de la primera fase

Algoritmo PPO				
PPO Episodes	Epsilon	Beta	Cumulative Reward	Episode Length
1	0,1	0,0001	0,974	0 : 29 : 06
2	0,1	0,001	0,971	0 : 43 : 47
3	0,1	0,01	0,969	0 : 28 : 12
4	0,2	0,0001	0,961	0 : 19 : 58
5	0,2	0,001	0,976	0 : 20 : 27
6	0,2	0,01	0,975	0 : 20 : 26
7	0,3	0,0001	0,946	0 : 21 : 32
8	0,3	0,001	0,920	0 : 20 : 33
9	0,3	0,01	0,976	0 : 19 : 17

Tabla 1: Valores obtenidos en las ejecuciones del algoritmo PPO

Tabla de resultados de las ejecuciones de SAC de la primera fase

Algoritmo SAC				
SAC Episodes	TAU	Learning Rate	Cumulative Reward	Episode Length
1	0,005	0,00001	−0,499	1 : 21 : 44
2	0,005	0,0001	0,854	1 : 22 : 14
3	0,005	0,001	−0,424	1 : 21 : 27
4	0,05	0,00001	−0,499	1 : 21 : 27
5	0,05	0,0001	0,880	1 : 22 : 12
6	0,05	0,001	−0,5	1 : 20 : 20
7	0,01	0,00001	−0,335	1 : 21 : 55
8	0,01	0,0001	0,463	1 : 21 : 53
9	0,01	0,001	−0,404	1 : 20 : 57

Tabla 2: Valores obtenidos en las ejecuciones del algoritmo SAC

Bibliografía

[1] AEVI-Asociación Española de Videojuegos. La industria del videojuego en españa en 2021. Technical report, Asociación Española de Videojuegos, Calle María de Molina, 54, planta 2, 28006 Madrid, Abril 2022. Ver página 1.

[2] Timothy Revell. Ai takes on top poker players. *New Scientist*, 233(3109):8, 2017. Ver página 1.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2018. Ver páginas 1, 8.

[4] Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Combining policy gradient and q-learning. 2016. Ver páginas 1, 8.

[5] Jim X. Chen. The evolution of computing: Alphago. *Computing in Science Engineering*, 18(4):4–7, 2016. Ver página 2.

[6] Torres J.;(2021). *Introduccion al aprendizaje por refuerzo profundo. Teoría y practica en Python.* Book Series. Kindle Direct Publishing. Ver páginas 2, 9.

[7] Jeronimo Hernández-González, Inaki Inza, and Jose A. Lozano. Weak supervision and other non-standard classification problems: a taxonomy. *Pattern Recognition Letters (2015).* Ver página 2.

[8] Thorsten Wuest, Daniel Weimer, Christopher Irgens, and Klaus-Dieter Thoben. Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1):23–45, 2016. Ver página 2.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. July 2017. Ver página 6.

[10] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. 2015. Ver página 6.

[11] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. 2018. Ver página 7.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2017. Ver página 8.

[13] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. 2018. Ver página 8.

[14] Christopher J C H Watkins and Peter Dayan. Q-learning. *Mach. Learn.*, 8(3-4):279–292, 1992. Ver página 8.

[15] Richard S Sutton. Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3(1):9–44, August 1988. Ver páginas 8, 9.

[16] G. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 11 1994. Ver página 9.

[17] Alexander Yong and David Yong. An estimation method for game complexity. 2019. Ver página 9.

[18] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. 2018. Ver página 12.

[19] Aditya Gudimella, Ross Story, Matineh Shaker, Ruofan Kong, Matthew Brown, Victor Shnayder, and Marcos Campos. Deep reinforcement learning for dexterous manipulation with concept networks, 2017. Ver página 29.