

Universitat Politècnica De Catalunya

FACULTAD DE INFORMÁTICA DE BARCELONA

**NAVEGACIÓN DE AGENTES EN
ENTORNOS NATURALES**

TRABAJO DE FIN DE GRADO



Autor: Jesús Benítez Díaz

Director: Oscar Argudo Medrano
Codirector: Carlos Andujar Gran

Junio 2022

Resumen

La Inteligencia Artificial es un área que trata de evolucionar constantemente con el fin de poder abordar cualquier problema, consiguiendo que hoy en día prácticamente todo pueda estar automatizado. En nuestro trabajo vamos a centrarnos en el aprendizaje por refuerzo, un área que cobra cada vez más fuerza dentro de este ámbito.

Hace relativamente poco, *Unity* presentó un kit de herramientas para el aprendizaje automático llamado *ml-agents*. Esta herramienta nos permite llevar a cabo proyectos muy sencillos así como otros de mucha complejidad, y todo esto sin la necesidad de tener una gran experiencia en el ámbito.

En este proyecto vamos a diseñar e implementar un sistema complejo donde un agente aprenda a navegar por entornos naturales de la forma más realista posible. Buscaremos explotar al máximo las capacidades de esta herramienta, tratando que las decisiones que toma nuestro agente sean lógicas y coherentes en función del escenario que tenga delante.

Abstract

Artificial Intelligence is an area that is constantly evolving in order to be able to tackle any problem, so that nowadays practically everything can be automated. In our work we are going to focus on reinforcement learning, an area that is gaining more and more strength in this field.

Relatively recently, Unity introduced a machine learning toolkit called *ml-agents*. This tool allows us to carry out very simple projects as well as others of great complexity, and all this without the need to have a great deal of experience in the field.

In this project we are going to design and implement a complex system where an agent learns to navigate through natural environments in the most realistic way possible. We will seek to exploit the capabilities of these tools to the maximum, trying to ensure that the decisions made by our agent are logical and coherent depending on the scenario in front of him.

Resum

La Intel·ligència Artificial és una àrea que tracta d'evolucionar constantment per tal de poder abordar qualsevol problema, aconseguint que avui dia pràcticament tot pugui estar automatitzat. En el nostre treball ens centrem en l'aprenentatge per reforç, una àrea que cada cop cobra més força dins aquest àmbit.

Fa relativament poc temps, *Unity* va presentar un kit d'eines per a l'aprenentatge automàtic anomenat *ml-agents*. Aquesta eina ens permet dur a terme projectes molt senzills així com altres de molta complexitat, i tot això sense necessitat de tenir una gran experiència en l'àmbit.

En aquest projecte dissenyarem i implementarem un sistema complex on un agent aprengui a navegar per entorns naturals de la forma més realista possible. Buscarem explotar al màxim les capacitats d'aquestes eines, tractant que les decisions que pren el nostre agent siguin lògiques i coherents segons l'escenari que tingui al davant.

Índice general

1. Introducción y contextualización	1
1.1. Contexto	1
1.2. Conceptos	1
1.2.1. Reinforcement Learning (RL)	2
1.2.2. Agente	2
1.3. Identificación del problema	3
1.4. Personas Implicadas	3
2. Justificación	4
2.1. Estudio de soluciones existentes	4
2.2. Herramientas	5
2.2.1. Unity	5
2.2.2. ML-Agents	7
2.2.3. Tensorboard	10
3. Alcance	12
3.1. Objetivo	12
3.1.1. Versión Inicial	12
3.1.2. Versión extendida	13
3.2. Requerimientos	14
3.2.1. Requerimientos funcionales	14
3.2.2. Requerimientos no funcionales	14
3.3. Obstáculos y riesgos	15
4. Metodología	16
4.1. Herramientas	17

5. Planificación temporal inicial	18
5.1. Descripción de las tareas	18
5.2. Recursos	23
5.2.1. Recursos materiales	23
5.3. Gestión del riesgo	24
5.4. Planificación final	25
6. Gestión económica	28
6.1. Presupuesto	28
6.2. Control de gestión	33
6.3. Presupuesto final	34
7. Sostenibilidad	35
7.1. Autoevaluación	35
7.2. Dimensión económica	36
7.3. Dimensión ambiental	37
7.4. Dimensión Social	38
8. Desarrollo	39
8.1. Desarrollo del videojuego	40
8.2. Mecánicas	43
8.3. Conceptos básicos para configurar el agente	44
8.3.1. Observaciones	45
8.3.2. Acciones y movimientos	46
8.3.3. Recompensas	47
8.4. Gestión de parámetros	47
8.5. Escenarios de entrenamiento	48
8.6. Algoritmos	50
8.6.1. <i>Proximal Policy Optimization (PPO)</i>	52
8.7. Gestión de hiperparámetros	52
9. Entrenamiento	55
9.1. Movimiento del agente sin restricciones hacia al objetivo	56
9.2. Movimiento del agente en función del tipo de terreno	62
9.3. Movimiento del agente en función del tipo de terreno (2)	66
9.4. Movimiento del agente en función de la forma del terreno	71
9.5. Agente final	76

10. Leyes y regulaciones	79
11. Conclusiones	81
12. Lineas Futuras	83
A. Diagrama de Gantt Inicial	85
B. Diagrama de Gantt Final	86
Biliografía	87

Índice de figuras

1.	Flujo de trabajo de un agente inteligente (2)	2
2.	Ventana <i>Inspector</i>	6
3.	Interfaz de <i>Unity</i>	7
4.	Ejemplos de escenarios creados por <i>Unity</i> (5)	8
5.	Componentes de la librería <i>ML-Agents</i>	9
6.	Ejemplo de un entorno de aprendizaje cualquiera	10
7.	Metodología ágil (9)	17
8.	Elementos del entorno del entrenamiento	40
9.	Características del terreno a generar	40
10.	Ejemplo de terreno usado en el proyecto 1	41
11.	Ejemplo de terreno usado en el proyecto 2	41
12.	Ejemplo de la representación del agente	42
13.	Ejemplo de la representación del objetivo	43
14.	Implementación del movimiento del agente mediante teclado	44
15.	Ejemplo gráfico del flujo de aprendizaje	45
16.	Componente <i>Behavior Parameters</i>	47
17.	Ejemplo de Agente único simultáneo	49
18.	Ejemplo de un escenario Adversario	50
19.	Esquema del algoritmo actor-crítico	51
20.	Fichero de configuración	54
21.	Función <i>Heuristic()</i>	56
22.	Componente <i>Behaviour Parameters</i> experimento 1	58
23.	Ejemplo visual del componente <i>RayPerception3D</i>	59
24.	Componente <i>RayPerception3D</i>	59
25.	Ejemplo visual del problema del componente <i>RayPerception3D</i> con las alturas	60

26.	Ejemplo visual de la solución de la colisión de los rayos	60
27.	Gráficas de recompensa acumulada y duración del episodio <i>BasicAgent2</i>	61
28.	Grafica Policy/Entropy <i>BasicAgent2</i>	61
29.	Ejemplo del terreno con la textura del tipo de terreno	62
30.	Gráficas de recompensa acumulada <i>WaterOnly</i>	64
31.	Gráfica de la entropía <i>WaterOnly2</i>	65
32.	Gráfica de la recompensa acumulada <i>WaterOnly2</i>	65
33.	Gráfica de la recompensa acumulada <i>AgentWith4Classes1</i>	68
34.	Gráfica de la entropía <i>AgentWith4Classes1</i>	68
35.	Nueva zona del mapa para el experimento <i>AgentWith5Classes1</i>	69
36.	Gráfica de la recompensa acumulada <i>AgentWith5Classes1</i>	70
37.	Gráfica de la entropía <i>AgentWith5Classes1</i>	70
38.	Ejemplo de una zona peligrosa de la textura <i>exposure</i> . El color rojo indica peligro.	72
39.	Gráfica de la recompensa acumulada <i>exposure1</i>	73
40.	Gráfica de la entropía <i>exposure1</i>	74
41.	Gráfica de la entropía <i>exposureInclination1</i>	75
42.	Gráfica <i>policy loss exposureInclination1</i>	75
43.	Gráfica de la recompensa acumulada <i>agentComplete</i>	77
44.	Gráfica <i>Losses/Value Loss agentComplete</i>	78
45.	Diagrama de <i>Gantt</i> final	85
46.	Diagrama de <i>Gantt</i> final	86

Índice de tablas

1.	Tabla de tareas con la duración, dependencias y recursos necesarios.	24
2.	Tabla de tareas definitiva con la duración, dependencias y recursos necesarios.	27
3.	Costes de personal. Elaboración propia	28
4.	Tabla de partidas por tarea. Roles: JP - jefe de proyecto, I - investigador, P - programador, T - tester. Coste SS - Coste de la seguridad social. Elaboración propia	29
5.	Costes de los recursos de Software. Elaboración propia.	30
6.	Costes de los recursos de Hardware. Elaboración propia.	31
7.	Tabla de contingencia del 20%. Elaboración propia.	31
8.	Tabla del sobrecoste de cada tipo de imprevisto. Elaboración propia .	32
9.	Tabla del presupuesto final. Elaboración propia.	32
10.	Presupuesto final de los costes totales del proyecto. Elaboración propia.	34
11.	Tabla de refuerzos experimento 1	57
12.	Tabla de refuerzos experimento 2	63
13.	Tabla de refuerzos experimento 3	66
14.	Tabla de refuerzos experimento 4	73
15.	Tabla de refuerzos experimento 5	77

Capítulo 1

Introducción y contextualización

1.1. Contexto

El *machine learning* (ML) es una rama de la inteligencia artificial (IA) y la informática que se centra en el uso de datos y algoritmos para imitar la forma en la que aprenden los seres humanos, con una mejora gradual de su precisión. Aprenden dentro de un contexto específico tratando de identificar patrones complejos.

En nuestro caso nos centraremos en una rama del *machine learning* llamada *reinforcement learning* (RL), aunque también existen muchas más. En el ámbito de la IA, un agente se conoce como cualquier cosa que percibe su entorno, y en el RL éste se enfrenta a una situación similar a un juego. Para lograr que el agente haga lo que el programador quiere, recibe una respuesta en forma de recompensa o penalización por las acciones que realiza.

El ML está siendo utilizado de muchas maneras y sus aplicaciones seguirán aumentando en la medida que se entienda la importancia del uso de los datos como información valiosa en todos los sectores. Entender por tanto que la forma en la que interpretamos la información es capaz de ayudar en la toma de decisiones nos acerca a la resolución de problemas de una manera más eficiente.

1.2. Conceptos

Dentro de este ámbito hay una serie de conceptos que vamos a definir a continuación, y así facilitar la comprensión del proyecto y su planteamiento.

1.2.1. Reinforcement Learning (RL)

El RL es la ciencia de la toma de decisiones. Se trata de aprender el comportamiento óptimo en un entorno para obtener la máxima recompensa. Este comportamiento óptimo se aprende a través de las interacciones con el entorno y las observaciones de cómo responde (1). Los principales elementos de un sistema de RL son:

- Agentes
- El entorno con el que interactúa el agente.
- La política que sigue el agente para tomar acciones.
- La señal de recompensa que el agente observa al realizar acciones.

1.2.2. Agente

Tal y como se explica en (2), es una entidad capaz de aprender en un entorno cualquiera. Siempre busca responder de manera correcta y maximizar la recompensa. Perciben el medio ambiente con la ayuda de sensores y actúan a través de actuadores. Es imposible hablar de RL sin mencionar a los agentes, debido a que el Agente es el ente virtual que decide y toma decisiones.

Estos agentes siguen un flujo de acción, el cual se muestra en la figura 1. En primer lugar inspecciona el entorno y recoge una serie de datos mediante unos sensores. Una vez analizado estos datos, decide cómo actuar sobre el entorno y dependiendo de la acción recibirá una recompensa o penalización, aprendiendo así en cada una de las iteraciones.

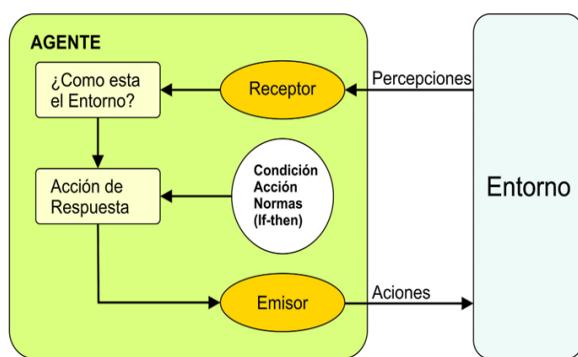


Figura 1: Flujo de trabajo de un agente inteligente (2)

1.3. Identificación del problema

Cada vez se llevan a cabo un mayor numero de estudios e investigaciones sobre el comportamiento humano. Y aunque tratar de simular todo el comportamiento humano es algo sumamente complicado, sí que podemos aislar ciertas situaciones y estudiar las diferentes respuestas que podría tener un humano.

En este trabajo de fin de grado (TFG) se va a llevar a cabo una investigación, desarrollando una aplicación sobre Unity para el entrenamiento de agentes virtuales. En concreto, los agentes deberán aprender a desplazarse por escenarios complejos de montaña, y escoger dinámicamente el itinerario hasta un objetivo en función de sus capacidades.

Una de las claves del proyecto será explotar al máximo las capacidades de aprendizaje de los agentes trabajando en un entorno lo más realista posible. Para ello usaremos entornos naturales de la vida real.

Trataremos de simular diferentes comportamientos y pensamientos que puede tener un humano ante situaciones de la vida real. El agente inteligente observará el entorno y analizará que acción es la que llevará a cumplir el objetivo, tal y como haría un humano. Un caso práctico sería decidir en el momento si subir una montaña le va resultar más costoso en cuanto a energía en vez de rodearla.

1.4. Personas Implicadas

Con el desarrollo de esta aplicación se pretende ir más allá, y que no quede en un simple TFG. La comunidad que experimenta con la IA comparte muchos de los estudios para que futuros desarrolladores utilicen esos conocimientos y que esta tecnología esté en constante desarrollo. En nuestro caso uno de los beneficiarios de este proyecto podría ser el grupo de investigación ViRVIG¹ o futuros estudiantes que decidan hacer su tesis sobre machine learning.

Por otro lado en el mundo de los videojuegos, la presencia de los NPCs² es habitual, pues este proyecto puede servir incluso para incluir el comportamiento que vamos a implementar en un videojuego.

¹Instituto de Visualización, Realidad Virtual e Interacción Gráfica (ViRVIG)

²Non-player character (NPC) (3)

Capítulo 2

Justificación

Se puede decir que el ML es una tecnología en auge, y que cada día que pasa se le encuentra una nueva aplicación. En este caso se apuesta por desarrollar algo prácticamente de cero. La idea de que un agente busque o persiga un objetivo no es algo del todo innovador, pero dependiendo de la forma en que se explore y se modele el entorno sí que pueda aportar novedades. La escasez de proyectos que se desarrolleen en un entorno similar es sorprendente, y es por esto que la aplicación a implementar se presenta como una muy buena oportunidad. Sin embargo, a continuación vamos a presentar algunas soluciones existentes, las cuales pueden servirnos de ayuda y también comentaremos las herramientas necesarias para el desarrollo del proyecto.

2.1. Estudio de soluciones existentes

Después de hacer un estudio exhaustivo sobre la existencia de proyectos similares, no hemos encontrado ninguno donde el entorno a explorar fuera un espacio tan grande como con el que vamos a lidiar. El hecho de detectar un objetivo en los proyectos existentes es medianamente sencillo debido a la cercanía en la que se movía el agente. En nuestra investigación el espacio es mucho mayor, lo cual no sabemos si será un problema.

Otro aspecto en el que nuestro proyecto se distingue del resto, es la forma en la que se escenifican los obstáculos, pues éstos se representan en forma de objetos, hablamos de muros, paredes, adversarios, etc. Como ya veremos más adelante, los obstáculos en nuestro entorno vendrán dados por la zona en la que estemos navegando en el mapa, como pueden ser precipicios, zonas de agua, etc. Ya estudiaremos como afecta al aprendizaje la forma de percibir la información.

Con todo esto queremos dejar claro la incertidumbre en cuanto a resultados, ya que a priori ciertas características que son comunes a este tipo de proyectos, en nuestro caso no contamos con ellas. Sin embargo, vamos a intentar extraer todas esas características y aplicarlas a nuestra investigación.

2.2. Herramientas

Para el desarrollo de la aplicación usaremos una serie de herramientas software que dan un gran soporte al ML. Analizaremos las tres grandes herramientas que usaremos para la implementación.

2.2.1. Unity

Una parte fundamental del proyecto, es el entorno donde los agentes aprenderán a moverse e interactuar. Por lo que habrá que elegir un motor gráfico que soporte el diseño de entornos y personajes 3D, así como la implementación de las funcionalidades de nuestros individuos.

Antes comentamos que la aplicación se iba a desarrollar en *Unity*, un software de desarrollo de videojuegos en tiempo real. Esta herramienta, creada por *Unity Technologies*, engloba motores para renderizar imágenes, motores de audio y motores de animación (4). Esta herramienta es muy usada para la creación de videojuegos, por lo que posee un amplio abanico de posibilidades para simular la física de los personajes u objetos.

Un término importante es la escena. Podemos definir la escena como el mundo donde se genera el videojuego y que está formado por *GameObjects*. En un *GameObject* podemos englobar prácticamente cualquier cosa, como puede ser un objeto 3D, cámaras, luces, etc.

Estos *GameObjects* no hacen nada por sí solos, necesitan de una pieza fundamental llamada componentes. Dependiendo de la finalidad, agregaremos una serie de componentes al objeto. Todo esto será gestionado en un apartado llamado *inspector* (figura 2). Más adelante hablaremos de diferentes componentes que van asociados a estos objetos que serán muy importantes.

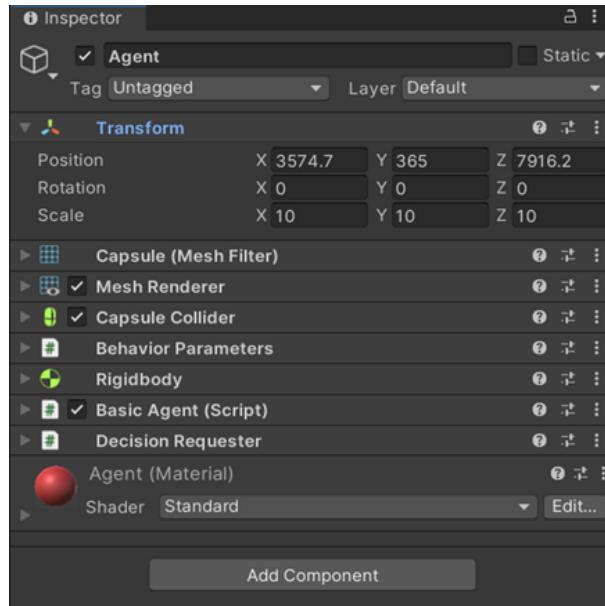
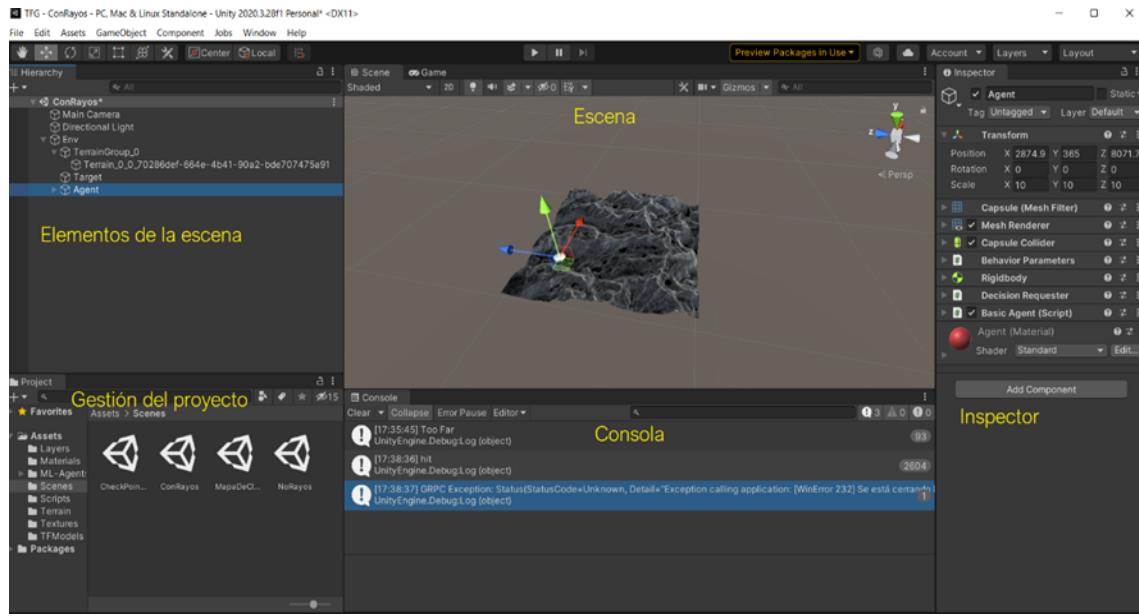


Figura 2: Ventana *Inspector*

En relación a los *GameObjects*, debemos tener en cuenta la existencia de los *scripts*. Estos archivos son totalmente independientes entre sí, pero se pueden asociar fácilmente a los *GameObjects*. En estos *scripts* vamos a definir el comportamiento o funcionalidad de los objetos, en nuestro caso serán fundamentales ya que será aquí donde programaremos todo lo relacionado con los agentes. La gestión de estos también será mediante el *inspector* donde podremos añadirlos o eliminarlos. Para terminar con los *scripts*, cabe mencionar que la programación de estos se hace mediante el lenguaje de programación C#. A continuación en la figura 3 se muestra el aspecto de la interfaz de *Unity*.

Figura 3: Interfaz de *Unity*

Vamos a comentar brevemente que función tiene cada ventana de la interfaz:

- **Elementos de la escena** – En este apartado gestionamos los elementos que hay en la escena y las dependencias entre ellos. Si queremos organizar un entorno donde hay varios elementos, pues crearemos un elemento principal donde dentro de él estarán todos los elementos que contiene.
- **Gestión del proyecto** – Aquí controlamos todos los componentes que están dentro del directorio del proyecto, ya sean *scripts*, texturas, escenas, etc.
- **Consola** – Como cualquier consola, nos informa de posibles errores de compilación y nos permite seguir el hilo de ejecución.
- **Escena** – Vista previa de los elementos que contiene la escena. Podemos modificar tamaños, posición, forma, etc.
- **Inspector** – En esta ventana controlamos que componentes tiene cada objeto. Modificando también los posibles parámetros que tenga cada componente.

2.2.2. ML-Agents

ML-Agents es un *plugin* desarrollado por *Unity* de código abierto que permite que los juegos y las simulaciones sirvan como entornos para entrenar agentes inte-

ligentes (5). En la Figura 4 podemos ver algunos escenarios que han sido creados por *Unity* y que podemos usar libremente.

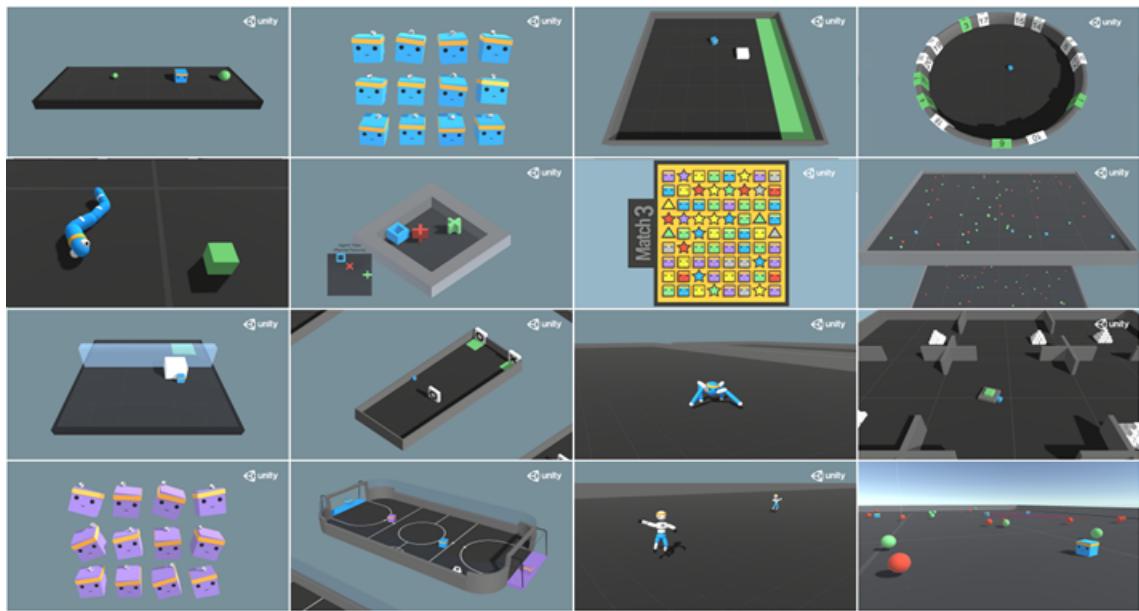


Figura 4: Ejemplos de escenarios creados por *Unity* (5)

La librería cuenta con tres componentes muy importantes:

- **Entorno de Aprendizaje** – Engloba todo lo relacionado con la escena creada en *Unity*, los objetos, etc.
- **API de Python** – Permite controlar todo el flujo de información para el entrenamiento de los agentes con diferentes algoritmos, lo cual facilita aún más la gestión de la aplicación. Más adelante veremos el algoritmo que usaremos.
- **Comunicador externo** – Se encarga de conectar la API con el entorno de aprendizaje.

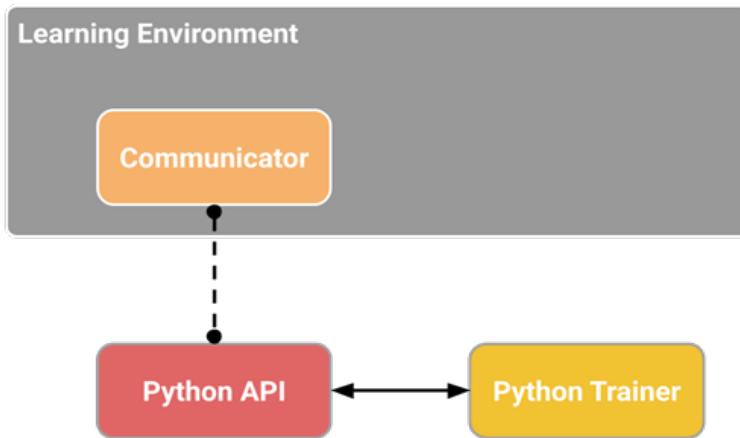


Figura 5: Componentes de la librería *ML-Agents*

Entorno de aprendizaje

Dentro del entorno de aprendizaje hay tres componentes muy importantes. El primero de ellos es el agente, del cual ya hemos hablado un poco y del que profundizaremos más adelante. Los otros dos componentes son el cerebro y la academia.

- **Cerebro** – Cada cerebro contiene la lógica que utiliza el agente para la toma de decisiones, teniendo un estado específico y su propio estado de acciones. Podemos distinguir 4 tipos de cerebro:
 - *External* – Las decisiones vienen determinadas por la API de python. Con todas las observaciones y recompensas recopiladas, la API devuelve la acción que debe realizar el agente.
 - *Internal* – Las decisiones son tomadas por un modelo ya entrenado previamente
 - *Player* – Este tipo de cerebro es usado para realizar pruebas, ya que el comportamiento es controlado por un humano como si fuera un jugador. Aquí las recompensas y observaciones no se tienen en cuenta.
 - *Heuristic* – Las decisiones y acciones se toman mediante un comportamiento predefinido.
- **Academia** – La academia es un objeto encargado de controlar todos los cerebros del entorno de aprendizaje. Este objeto organiza el proceso de observación y toma de decisiones. Además es responsable de los atributos del entorno, como son la velocidad de entrenamiento, cuánto durará cada episodio, etc.

En la figura 6 vemos un ejemplo de cómo se organizaría un entorno de aprendizaje. Podemos ver que en este ejemplo varios agentes comparten el mismo cerebro, y a su vez todos estos están conectados a la academia. Los agentes que comparten el mismo cerebro deben tener las mismas características, pero no significa que tengan en todo momento los mismos valores de observaciones y acciones. Más adelante explicaremos diferentes formas de organizar los agentes y los cerebros.

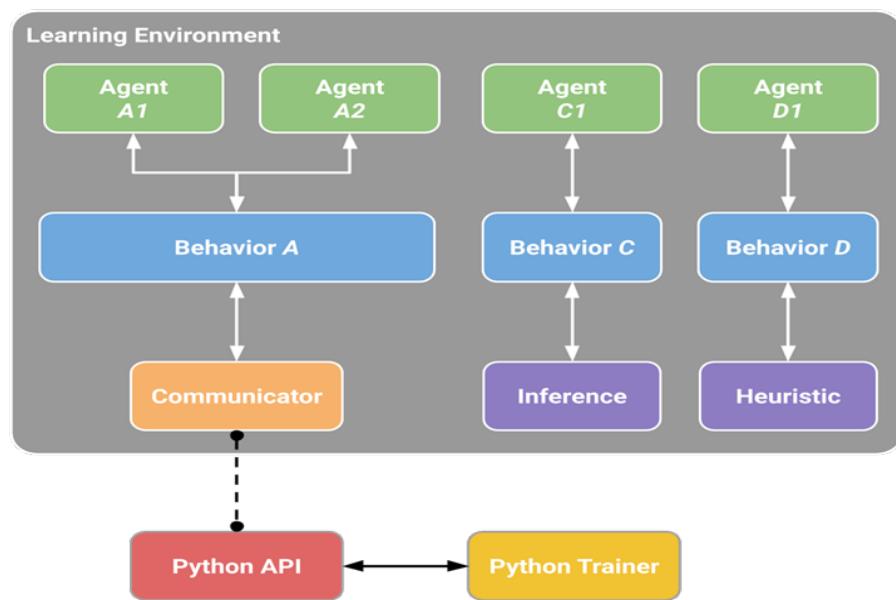


Figura 6: Ejemplo de un entorno de aprendizaje cualquiera

La librería que permite todos los cálculos complejos que conllevan estos procesos se llama *TensorFlow*. Esta es una gran plataforma para construir y entrenar redes neuronales, que permiten detectar y descifrar patrones, análogos al aprendizaje y razonamiento usados por los humanos (6).

2.2.3. *Tensorboard*

Durante la sesión de aprendizaje, la librería *Ml-Agents* guarda las estadísticas sobre el rendimiento del agente. Estas métricas podemos verlas mediante una herramienta de la librería *TensorFlow* llamada *TensorBoard* (7).

Las estadísticas son proporcionadas en forma de gráficas, permitiendo ver su evolución incluso durante el entrenamiento. Vamos a comentar algunas gráficas que nos servirán en el futuro a la hora de evaluar los resultados.

- ***Environment/Cumulative Reward*** – La recompensa acumulada media del episodio sobre todos los agentes. Debería aumentar durante una sesión de entrenamiento exitosa.
- ***Environment/Episode Length*** – La duración media de cada episodio en el entorno para todos los agentes. Dependiendo del objetivo del problema, interesará que el episodio tenga más o menos duración.
- ***Losses/Policy Loss*** – La magnitud media de la función de pérdida de la política. Se correlaciona con cuánto está cambiando la política (proceso para decidir acciones). La magnitud de esto debería disminuir durante una sesión de entrenamiento exitosa.
- ***Losses/Value Loss*** – La pérdida media de la actualización de la función de valor. Se correlaciona con qué tan bien el modelo puede predecir el valor de cada estado. Esto debería aumentar mientras el agente está aprendiendo y luego disminuir una vez que la recompensa se estabilice.

Capítulo 3

Alcance

Una parte fundamental del proyecto es definir el alcance del mismo. Hay que ser realista respecto al tiempo y realizar una buena planificación. Vamos a definir los objetivos, requerimientos, obstáculos y riesgos.

3.1. Objetivo

Como ya sabemos, el objetivo principal del proyecto es el desarrollo de un aplicación para el entrenamiento de agentes virtuales en espacios naturales. Este planteamiento es muy abierto y da pie a desarrollarlo de formas muy diferentes. Por ese motivo vamos definir una serie de objetivos básicos que trataremos de cumplir a lo que podemos llamar versión estándar, y otra serie de modificaciones que se irán haciendo en función del tiempo disponible.

Algo que también debemos tener en cuenta, son los aspectos que no vamos a tratar. Una de las cosas que vamos dejar de lado, es la representación y presencia del agente a nivel gráfico. El nivel de detalle visual no es lo primordial en este proyecto, y es por esto que una de las decisiones ha sido que el agente se represente de la forma más simple posible. El único requisito que buscamos que cumpla, es que permita el seguimiento de los diferentes movimientos que haga el agente.

3.1.1. Versión Inicial

Objetivo

El objetivo del agente es algo muy importante, y que dependiendo del tiempo se podrá ver alterado. En un primer momento el objetivo principal del agente será

llegar de un punto A a un punto B. Las únicas condiciones serán que lo haga de la forma más óptima posible y sin morirse.

Número de agentes y comportamientos

En esta versión inicial vamos a centrarnos en trabajar con un único agente, y obviar la existencia de otros individuos en el entorno. No obstante, algo muy interesante será los diferentes tipos de comportamientos que pueda tener el agente.

Supongamos que tenemos un agente para el cual definimos que pueda escalar, pues la idea es que este agente aprenda y decida que le sale más rentable, si rodear la montaña o escalarla. Habrá otros agentes que no puedan escalar, y por lo tanto debe rodear la montaña sí o sí. Aquí entran en juego muchos factores, como puede ser el de la velocidad de movimiento o la energía consumida. El agente no se moverá de la misma forma si va por un camino asfaltado que por un campo a través, o si sube una pendiente con mucha inclinación gastará mucha más energía que si camina unos kilómetros de más, pero en llano.

La idea es definir diferentes tipos de comportamientos y ver si realmente usan sus habilidades para llegar al objetivo de la forma más inteligente posible, optimizando la energía, el tiempo y la distancia recorrida.

3.1.2. Versión extendida

En función del tiempo y de cómo se vaya desarrollando la aplicación, veo conveniente definir una serie de subobjetivos opcionales que pueden ser implementados para mejorarla.

En primer lugar, una de las cosas a mejorar puede ser el objetivo del agente. Hemos definido que su función es llegar de un punto a otro. Como mejora podríamos añadir la existencia de unos *checkpoints*, simulando que son una serie de puntos de interés que el agente debe visitar antes de llegar a su objetivo final.

Como segunda mejora podemos incluir en nuestro planteamiento la existencia de más agentes en el espacio. Esto implicaría unos estímulos extras y una serie de nuevos comportamientos que podríamos implementar. Por ejemplo un agente podría tener un comportamiento donde evite a toda costa el encontrarse con otro individuos, lo cual haría que deba tener muchas cosas en cuenta.

3.2. Requerimientos

Vamos definir una serie de requisitos que son necesarios para el correcto funcionamiento de nuestra aplicación:

3.2.1. Requerimientos funcionales

1. **Entorno 3D** - Cargar correctamente un entorno donde los agentes entrenen es un requisito básico, así como la visualización del mismo para poder seguir el desarrollo.
2. **Configuración** - Establecer una configuración dinámica y de fácil modificación para las diferentes versiones de la aplicación. Vamos a querer entrenar diferentes tipos de agentes tal y como hemos visto en el punto 3.1.1, pues para ello es conveniente modelar el código de manera que sea fácil elegir la configuración del modelo a entrenar. En este punto se engloba todo lo relacionado a la configuración del entorno, ya sea la posición inicial, posición final, distancia entre ellos, etc.

3.2.2. Requerimientos no funcionales

Otro aspecto importante son los requisitos no funcionales del sistema. Estos requisitos son aquellos impuestos por el desarrollador al programa y que son necesarios para su correcto funcionamiento (8).

1. **Reusabilidad** - Una cosa a tener en cuenta en nuestro proyecto es la reusabilidad. En el punto 1.4 hemos hablado de la necesidad de compartir el proyecto con futuros investigadores, pues si facilitamos que el sistema pueda ser integrado en otros entornos habremos cumplido con nuestro objetivo.
2. **Eficiencia** - Todo lo que tiene que ver con el entrenamiento de agentes lleva un tiempo elevado dependiendo de qué casos. Habrá que optimizar en la medida de lo posible el código y disminuir el tiempo de ejecución.

3.3. Obstáculos y riesgos

Como cualquier proyecto, nos podemos encontrar con diferentes obstáculos durante el desarrollo. Estos temas deben ser analizados e intentar prevenirlos para minimizar los daños. Vamos a comentar cuales podrían ser los problemas que dificulten el desarrollo de la aplicación:

1. El entrenamiento de los distintos agentes puede suponer un tiempo de cómputo considerable, por lo que el tiempo será nuestro mayor enemigo. Esto recae directamente sobre las prestaciones de nuestro PC. Las simulaciones se harán con un PC que cuenta con una *Nvidia GEFORCE GTX 1050*.
2. Otro aspecto que a priori puede preocupar, es la familiarización con el entorno de Unity y con la librería *ML-Agents*. Esto en un principio podría suponer un problema, pero que no tiene un impacto tan grande como podría ser el tiempo.
3. En cuanto a la parte de visualización podemos encontrarnos con ciertas limitaciones. Unity tiene un buen motor gráfico y hace muchas optimizaciones, pero siempre existe el riesgo de que alguna escena que queramos simular sea demasiado costosa. En ese caso se podría reducir el tamaño del "mundo"(escena).
4. Una situación que se puede dar es la de perder archivos del proyecto. Estamos trabajando con muchos archivos y es posible que se pueda extraviar en algún momento uno de ellos. Para evitar esto, trataremos de hacer copias de seguridad periódicamente.

Capítulo 4

Metodología

La implementación de la aplicación lleva un tiempo de experimentación y de desarrollo donde seguramente surjan complicaciones. Elegir una buena metodología que permita cambios y que se adapte a cambios en la planificación es muy importante. Por ello creo que la metodología ágil es ideal para nuestro trabajo.

Tal y como se indica en la metodología ágil (9), se harán reuniones semanales con el director y codirector para hacer un seguimiento del estado actual del proyecto. Con estas reuniones se pretende tener un *feedback* continuo, y así prevenir futuros problemas.

El método de trabajo será el siguiente: diseñaremos una nueva funcionalidad y esta será testeada y comentada con el profesorado. Si la funcionalidad es válida se pasará a la siguiente. Esto se repetirá hasta alcanzar el objetivo, teniendo siempre de respaldo una versión del proyecto en funcionamiento.

Como se ha comentado en el punto 3.3 unos de los posibles obstáculos es el tiempo. De cara a la validación de las funciones implementadas se intentará tener una versión bastante sólida antes de proceder al entrenamiento de los agentes. Con este método evitamos pérdidas de tiempo innecesarias entrenando un modelo el cual está bastante lejos de su versión final.



Figura 7: Metodología ágil (9)

4.1. Herramientas

Para facilitar el control de las versiones y de las fases del proyecto se usará *GitLab* (10). Esta herramienta facilitará la comunicación entre el profesorado y el alumno cuando se traten temas de código. Además tener un repositorio remoto nos asegura tener siempre una copia de seguridad de todo nuestro trabajo, y así evitar uno de los riesgos comentados en el apartado anterior.

En cuanto a la planificación usaremos uno de los software más populares, Gant-Pro (11). Esta herramienta nos permite mostrar gráficamente el tiempo que le dedicamos a cada etapa durante el proyecto. Para llevar un control exacto del tiempo empleado, se irá actualizando semanalmente el tiempo que conlleve cada tarea.

Para realizar la memoria se va a usar *Google Docs* y *Microsoft Word*. *Google Docs* al ser una herramienta *online* nos permitirá mostrar en todo momento el progreso al profesorado, ya que se puedan hacer comentarios sobre el mismo documento. Por otro lado se mantendrá actualizado en *Word*, y será donde se harán todo el desarrollo, ya que ofrece unas mejores prestaciones.

Capítulo 5

Planificación temporal inicial

La organización y planteamiento del proyecto es una parte fundamental si queremos cumplir con los objetivos propuestos. Debemos realizar una planificación temporal realista y dividir el proyecto en tareas.

Se empezó a trabajar en el proyecto el 9 de Febrero de 2022, y se prevé que la presentación sea el día 27 de Junio de 2022. El desarrollo del trabajo llevará 138 días aproximadamente y se estima que se inviertan unas 425 horas. La dedicación diaria será de 3 horas aproximadamente, pudiendo elegir el horario de trabajo con total libertad.

5.1. Descripción de las tareas

En este apartado vamos a detallar las tareas y fases en las que se va a dividir el proyecto. Vamos a diferenciar varios bloques en los que se van a agrupar las diferentes tareas, y hacer más fácil la organización. En el apéndice A se muestra el diagrama de Gantt con la planificación inicial del proyecto.

La agrupación de las tareas en bloques no significa que se deban completar todas las tareas sobre uno los bloques para pasar al siguiente. Esta agrupación será una especie de indicativo sobre el tipo de trabajo que requiere.

GP - Gestión del proyecto

En este bloque se tratarán las tareas que tengan que ver con la planificación, documentación, reuniones de seguimiento, etc. El conjunto de las tareas que engloba este bloque, se prevé que tenga una duración de 140 horas.

GP.1 - Reuniones

Las reuniones de seguimiento son algo fundamental en el desarrollo del proyecto. Debemos tener un feedback continuo con el profesorado que dirige nuestro trabajo, y así tener la posibilidad de hacer cambios en la planificación si fuera necesario. Se harán reuniones semanales con el director y el codirector de 1 hora. En total, serán 20 semanas hasta la presentación, por lo que se estima una dedicación de 20 horas.

GP.2 - Alcance

Antes de empezar con el desarrollo, es necesario definir y discutir con el profesorado hasta que punto va a llegar el trabajo. Es muy importante hacer una pequeña investigación sobre el tema a tratar, así como las capacidades del alumno para decidir el alcance del proyecto. Una vez hecho esto podemos decidir las funcionalidades que se desarrollarán. La duración estimada de esta tarea será de 20 horas.

GP.3 - Planificación

Si queremos llegar a los objetivos definidos en el alcance del proyecto, debemos realizar una buena planificación temporal. Esta planificación conlleva definir qué recursos y herramientas necesita cada tarea. Además, como ya hemos comentado en otras ocasiones, hay que tener en cuenta diferentes soluciones para los problemas y riesgos que puedan surgir. Esta tarea ha llevado 15 horas de trabajo.

GP.4 - Presupuesto

Debemos tener en cuenta el coste total del proyecto, diferenciando costes personales y equipo. Tendremos en cuenta costes de partida y costes imprevistos. En este trabajo, se cuenta con la suerte en cierta medida que el material necesario para poder llevar a cabo el trabajo no es más que un PC. No obstante, pueden surgir complicaciones y es importante hacer una previsión realista del coste. Se estima una dedicación de 10 horas.

GP.5 - Informe de sostenibilidad

Se hará un análisis del impacto económico, medioambiental y social que tendrá el proyecto durante las fases de planificación y desarrollo. Se estima que este estudio conlleve 5 horas.

GP.6 - Documentación

Una de las partes más importantes de un TFG es la memoria final. Esta tarea se irá desarrollando de forma paralela al resto, de esta forma será más fácil comentar las diferentes fases del proyecto. Al ser de las tareas más importantes, también llevará más tiempo de trabajo, se estima una duración de 60 horas.

GP.7 - Lectura

La última fase de este bloque será preparar la lectura final del TFG. Esta fase conlleva la preparación de material de soporte para la presentación y ensayos. En total tendrá una duración de 10 horas.

TP - Trabajo Previo

Hay una serie de tareas que son de preparación y estudio previo, las cuales son básicas antes de meterse de lleno con el desarrollo del trabajo. Estas se realizarán en paralelo a las tareas de gestión, lo cual incluso puede ayudar a complementar tareas de ese bloque. Se prevé una duración de 25 horas.

TP.1 - Estudio del estado del arte

Como ya hemos comentado previamente, este es un trabajo que lida con una tecnología en auge, por lo que se debe investigar bien qué tipo de aplicaciones ya desarrolladas se pueden asemejar a nuestra idea. Analizaremos proyectos existentes, así como sus capacidades y características, y poder decidir qué funcionalidades pueden ser interesante desarrollar o mejorar. La duración aproximada será de 10 horas.

TP.2 - Preparación de entorno de trabajo

Una de las ventajas de investigar esta tecnología es que el material de trabajo está al alcance de casi cualquier usuario. Básicamente necesitaremos un PC con una GPU³ de gama media-alta, en el cual instalaremos Unity (4) y una serie de librerías. Una vez instalado el software necesario, es importante hacer una serie de pruebas para familiarizarnos con el entorno y evitar hacer este trabajo a la hora de empezar a desarrollar. Esta tarea ha llevado una duración de 15 horas.

³Graphic Processing Unit (GPU) en inglés, es un dispositivo dedicado a la generación de gráficos para PC.

DA - Desarrollo de la aplicación

En este bloque vamos a desarrollar lo que será la base de nuestro proyecto. Vamos a tener que integrar el entorno natural en Unity, modelar la interacción de los agentes con el entorno e implementar los diferentes comportamientos que puedan tener. El desarrollo llevará aproximadamente 160 horas.

DA.1 - Integración del entorno natural

Lo primero que debemos hacer antes de programar nada, es incorporar el entorno donde van a realizar la exploración los agentes. Este entorno como ya hemos comentado, será un espacio natural y real, por lo que nos vamos a ahorrar la parte de diseñar y modelar el entorno. Sin embargo, esta tarea nos llevará trabajo, sobre todo entender cómo tratar y acceder a la información que nos proporciona el modelo. En total, familiarizarse con el entorno y realizar pequeñas pruebas llevará 25 horas de trabajo aproximadamente.

DA.1 - Integración del entorno natural

Podríamos aventurarnos a decir que esta tarea es la que más tiempo nos va a llevar. El tiempo que dediquemos a ella y cómo se desarrolle determinará el alcance que tendrá nuestro trabajo. En esta tarea vamos a trabajar en la implementación del comportamiento de los diferentes tipos de agentes. Aquí englobamos lo relacionado a las recompensas, objetivos, tipos de movimiento, etc. Se prevé una duración de 125 horas, las cuales se dividirán en varias subtareas.

1. **DA.2.1 - Observaciones del agente** - Este es el primer estado en el que se encuentra un agente cuando inicia una iteración. Debemos implementar qué es lo que verá y tomará como observaciones para poder tomar una decisión. Duración: 25 horas.
2. **DA.2.2 - Toma de decisiones** - Una vez haya observado el escenario, el agente debe tomar una decisión en función de lo que sepa. Aquí definiremos los tipos de acciones que podrá realizar el agente, como por ejemplo, desplazarse horizontalmente, verticalmente, rotación, saltar, etc. Todas estas acciones deben ser valoradas y testeadas. Duración: 45 horas

3. **DA.2.3 - Refuerzos positivos/negativos** - Después de decidir qué acción realizar, debemos decirle cómo de bien o mal lo ha hecho. Estableceremos unos criterios que dictan si la acción merece una recompensa o una penalización. Duración: 30 horas.
4. **DA.2.4 - Configuración de parámetros** - Hay una serie de parámetros que vienen definidos por defecto, y que se suelen obviar a la hora de entrenar agentes. Si queremos que nuestra aplicación tenga mejor rendimiento debemos prestar atención y tomarnos molestias para explorar las diferentes opciones que nos ofrece. Duración: 25 horas.

DA.3 - Representación de los agentes

Esta tarea se realizará en paralelo a la DA.2. Se basará en la representación del agente, utilizando formas geométricas sencillas como cilindros o esferas, y de su presencia en la escena. Se hará una primera representación bastante básica tal y como se comentó en el alcance del proyecto, y en función de los avances se podrá dedicar un poco más tiempo a esta tarea, sumándole realismo a la escena. En el diagrama de Gantt se mostrará como una tarea que se llevará a cabo durante todo el proceso de desarrollo, aunque haya días que no se trabaje en ella. Se estima que se invierta un tiempo máximo de 10 horas.

EA - Entrenamiento y análisis

EA.1 - Entrenamiento

Possiblemente la parte más incierta de todo el proyecto. Esta es una tarea donde el número de horas que invertimos es muy difícil de controlar. En esta fase posterior a la programación de los agentes, debemos dejar que entrenen. Teniendo de referencia otro proyectos los cuales hemos probado a entrenar, se prevé unas 20 horas para cada entrenamiento. Sabiendo que possiblemente se necesiten varios entrenamientos y que se pueden ajustar ciertos parámetros para que el tiempo disminuya, podemos hacer una estimación de 70 horas aproximadamente.

EA.2 - Análisis resultados

Una vez entrenado el modelo, o incluso durante la propia ejecución tendremos acceso a una serie de datos sobre el desarrollo del entrenamiento y de las capacidades de los agentes entrenados. Nuestra tarea ahora será analizar si el comportamiento de los agentes es el esperado. Si no fuera así debemos replantearnos la configuración que hemos establecido. Esta tarea puede que se repita cada vez que se haga un entrenamiento, por lo que al igual que con la tarea DA.3 se mantendrá abierta durante todo el entrenamiento. La duración estimada es de 30 horas.

5.2. Recursos

Vamos a definir cuatro roles que estarán presentes en el desarrollo del proyecto: jefe de proyecto, investigador, programador y tester. Como el trabajo se realiza por una única persona, este deberá asumir el papel que sea necesario dependiendo de la etapa.

1. **Jefe de proyecto** - Llevará el peso del proyecto en cuanto a la organización. Debe organizar las reuniones de seguimiento y documentar todo.
2. **Investigador** - Se encarga de investigar sobre las diferentes técnicas a implementar. Debe comparar diferentes alternativas y analizar los resultados obtenidos.
3. **Programador** - Lleva a cabo las ideas planteadas por el investigador. Se encarga de montar el entorno de trabajo e implementar el sistema.
4. **Tester** - Su función será validar el funcionamiento del sistema. Ejecutará diferentes pruebas, recogerá los resultados y hará un informe para que el investigador haga una valoración.

5.2.1. Recursos materiales

Presentaremos a continuación los recursos materiales indispensables para desarrollar la aplicación.

1. PC - Será nuestra principal herramienta de trabajo, donde realizaremos tanto el desarrollo e investigación como la gestión del proyecto.

2. Unity - Motor gráfico donde modelamos y simularemos todo el entorno de la aplicación.
3. GanttPRO - Programa de gestión de proyectos para crear diagramas de Gantt.

En la Tabla 1 podemos ver un resumen de cada una de las tareas y lo que conlleva cada una de ellas.

Id Tarea	Nombre de tarea / Título	Roles	Tiempo	Recursos	Predecesor
1	GP - Gestión del proyecto		140		
1.1	GP.1 - Reuniones	JP, I, P, T	20	PC	
1.2	GP.6 - Documentación	JP, I	60	PC	
1.3	GP.2 - Alcance	JP	20	PC	2.1
1.4	GP.3 - Planificación	JP	15	PC	1.3
1.5	GP.4 - Presupuesto	JP	10	PC	1.4
1.6	GP.5 - Informe de sostenibilidad	JP	5	PC	1.5
1.7	GP.7 - Lectura	JP	10	PC	1.2
2	TP - Trabajo Previo		25		
2.1	TP.1 - Estudio del estado del arte	I	10	PC	
2.2	TP.2 - Preparación de entorno de trabajo	P	15	PC,Unity	1.4
3	DA - Desarrollo de la aplicación		160		
3.1	DA.1 - Integración del entorno natural	P	25	PC,Unity	2.2
3.2	DA.2 - Programación de agentes		125	PC,Unity	3.1
3.2.1	DA.2.1 - Observaciones del agente	I,P	25	PC,Unity	3.2
3.2.2	DA.2.2 - Toma de decisiones	I,P	45	PC,Unity	3.2.1
3.2.3	DA.2.3 - Refuerzos positivos/negativos	I,P	30	PC,Unity	3.2.2
3.2.4	DA.2.4 - Configuración de parámetros	I,P	25	PC,Unity	3.2.1
3.3	DA.3 - Representación de los agentes	I,P	10	PC,Unity	3.1
4	EA - Entrenamiento y análisis		100		
4.1	Entrenamiento	P,T	70	PC,Unity	3.0
4.2	Análisis	I,T	30	PC,Unity	4.1
	Total	-	425		

Tabla 1: Tabla de tareas con la duración, dependencias y recursos necesarios.

5.3. Gestión del riesgo

Es muy importante tener en cuenta los riesgos y complicaciones que puedan surgir. En este trabajo vamos a tener que definir una serie de alternativas a nuestros objetivos iniciales, y prever posibles obstáculos que aparezcan.

1. **Dificultades imprevistas** - El *reinforcement learning* es una técnica compleja y que requiere una gran comprensión. Puede que no todas las funcionalidades puedan ser implementadas, ya sea por dificultad o por tiempo. Por esta

razón se hará una valoración realista unas semanas antes sobre si se van a poder cumplir estos objetivos, y si no es así eliminar el desarrollo de alguna funcionalidad. Esto puede suponer un aumento en las horas de trabajo si se decide seguir adelante con las funcionalidades que dan problemas. Por el contrario, si somos críticos y desde un primer momento descartamos la funcionalidad, supondría una ganancia de tiempo que se puede invertir en otro aspecto y minimizar daños.

2. **Rendimiento** - Como ya hemos comentado antes, el tiempo que lleve entrenar a los agentes va a depender en gran medida de las capacidades de nuestro PC. Esto puede llevar a que al igual que en el punto anterior, tengamos que sacrificar algunas funcionalidades. Un plan alternativo para este problema será tener un PC de respaldo con el cual hacer entrenamientos en paralelo. Que el PC con el que se va a trabajar no rinda como se espere, puede añadir unas 20 horas de duración extra.

Ambos problemas cuentan con una alta probabilidad de que sucedan, no obstante contamos con las alternativas para solventar este tipo de imprevistos. Es por eso que podemos catalogarlos dentro de un riego medio.

5.4. Planificación final

Durante la etapa de desarrollo y entrenamiento que veremos en el punto 9, nos hemos visto condicionado por varios factores. Se podría decir que todos ellos giran en torno a la complejidad que conlleva el *machine learning*, una tecnología que requiere de muchos recursos para que los resultados sean factibles y rentables. Y que hace que la cantidad de horas extras que ha requerido este proyecto, nos lleve a replantearnos ciertos objetivos. Estos problemas surgen en cierta medida del desconocimiento de esta tecnología. Es un proyecto de investigación en el cual me he aventurado con unos conocimientos básicos y quizás con unos objetivos demasiado ambiciosos para las herramientas con las que se contaban. Vamos a repasar ciertos aspectos importantes que han hecho cambiar nuestra planificación inicial.

En un primer momento hablamos de una versión inicial en la cual definimos unos objetivos básicos que queríamos cumplir. Estos consistían en incorporar poco a poco nuevas funcionalidades al comportamiento del agente, buscando condensar

todo lo aprendido en una versión final. Por otro lado teníamos una serie de objetivos extras a lo que llamamos versión extendida, la cual hemos dejado de lado a mitad del proyecto e intentar tener una versión sólida.

En lo que respecta a la estructura de la planificación no cambia mucho, pero sí que cambia la cantidad de horas dedicadas a las diferentes tareas, así como los objetivos que nos planteamos al principio.

La preparación del entorno fue una de las tareas que más tiempo llevaron, sobre todo por todo el desconocimiento previo a empezar el proyecto sobre *Unity*. Esta parte se demoró bastante, pasando de unas 15 horas previstas a 40.

En un primer momento englobamos la parte de implementación en un gran bloque llamado programación del agente. En este bloque diferenciamos cuatro partes, haciendo referencia a las cuatro aspectos que hay que tener en cuenta para programar el comportamiento del agente. Después de varias horas de trabajo, hemos comprendido que lo mejor es separar en funcionalidades que se vayan a implementar, y condensar las cuatro partes existentes en una funcionalidad. Como hemos dicho, la versión extendida se ha dejado de lado por lo que la versión básica contará con cuatro funcionalidades. Cuando hablamos de funcionalidades nos referimos a distintos comportamientos que puede adoptar el agente.

En resumen, contaremos con 4 bloques de trabajo, cada uno de ellos con los puntos que contaba antes el punto 3. Después de completar el primer bloque de trabajo, se empezará con los entrenamientos y se irá planificando e implementando las siguientes funcionalidades. La nueva tabla con la distribución de horas final la vemos a continuación (tabla 2) y el nuevo diagrama de Gantt se muestra en el apéndice B. El tiempo de entrenamiento también aumenta debido a la reestructuración que se ha llevado a cabo.

Id Tarea	Nombre de tarea / Título	Roles	Tiempo	Recursos	Predecesor
1	GP - Gestión del proyecto		145		
1.1	GP.1 - Reuniones	JP, I, P, T	20	PC	
1.2	GP.6 - Documentación	JP, I	60	PC	
1.3	GP.2 - Alcance	JP	20	PC	2.1
1.4	GP.3 - Planificación	JP	18	PC	1.3
1.5	GP.4 - Presupuesto	JP	12	PC	1.4
1.6	GP.5 - Informe de sostenibilidad	JP	5	PC	1.5
1.7	GP.7 - Lectura	JP	10	PC	1.2
2	TP - Trabajo Previo		50		
2.1	TP.1 - Estudio del estado del arte	I	10	PC	
2.2	TP.2 - Preparación de entorno de trabajo	P	40	PC,Unity	1.4
3	DA - Desarrollo de la aplicación		175		
3.1	DA.1 - Integración del entorno natural	P	25	PC,Unity	2.2
3.2	DA.2 - Programación de agentes		150	PC,Unity	3.1
3.2.1	DA.2.1 - Funcionalidad 1	I,P	25	PC,Unity	3.2
3.2.2	DA.2.2 - Funcionalidad 2	I,P	35	PC,Unity	3.2.1
3.2.3	DA.2.3 - Funcionalidad 3	I,P	40	PC,Unity	3.2.2
3.2.4	DA.2.4 - Funcionalidad 4	I,P	50	PC,Unity	3.2.2-(3.2.3)
3.3	DA.3 - Representación de los agentes	I,P	10	PC,Unity	3.1
4	EA - Entrenamiento y análisis		120		
4.1	Entrenamiento	P,T	85	PC,Unity	3.0
4.2	Análisis	I,T	30	PC,Unity	4.1
Total			490		

Tabla 2: Tabla de tareas definitiva con la duración, dependencias y recursos necesarios.

En el diagrama de Gantt las dos primeras funcionalidades se han realizado de forma secuencial ya que el desconocimiento previo hizo que dependiésemos de los resultados del primer experimento. Una vez acabada con la segunda y visto el éxito, decidimos iniciar en paralelo la implementación de lo que sería nuestra versión final. Esta la hicimos a la vez que la funcionalidad 3, que a pesar de que estuviera pendiente de testear, por cuestiones de tiempo nos adelantamos y decidimos arriesgar.

Por último, hemos hecho un cambio en cuanto a las herramientas usadas para realizar la memoria. Al obviar la implementación de funcionalidades extras, hemos decidido emplear ese tiempo en pasar la memoria a *LaTeX*, un sistema de composición de textos, orientado a la creación de documentos escritos que presentan una alta calidad tipográfica (12).

Capítulo 6

Gestión económica

En este apartado vamos a hacer una estimación de todos los costes necesarios para llevar a cabo el proyecto. Haremos una distinción dependiendo del tipo de coste, como puede ser el coste de personal, espacio de trabajo y herramientas necesarias.

Por otro lado, no solo tendremos en cuenta los costes programados. Hay una serie de obstáculos y riesgos ya comentados, para los cuales trazaremos un plan de contingencia y una partida de imprevistos, y así poder controlar el presupuesto final.

6.1. Presupuesto

Coste del personal

En el apartado de planificación dividimos el proyecto en tareas y definimos diferentes roles. Pues teniendo en cuenta esto calcularemos el coste del personal, diferenciando entre jefe de proyecto, investigador, programador y tester. En la tabla 3 se hace un desglose del precio por hora de cada tipo de trabajador:

Rol	Coste/hora
Jefe de proyecto	30€/h
Investigador	20€/h
Programador	16€/h
Tester	16€/h

Tabla 3: Costes de personal. Elaboración propia

En la tabla 4 haremos un desglose del coste de cada tarea en función de quien se ocupe de ella y de cuantas horas se empleen. Calcularemos el coste teniendo en cuenta la seguridad social multiplicando por 1.3. El coste final que supone el personal del proyecto es de 20.098€.

Nombre de tarea / Título	Roles	Tiempo	Coste	Coste SS
GP - Gestión del proyecto		140	6.440 €	8.372 €
GP.1 - Reuniones	JP, I, P, T	20	1.640 €	2.132 €
GP.6 - Documentación	JP, I	60	3.000 €	3.900 €
GP.2 - Alcance	JP	20	600 €	780 €
GP.3 - Planificación	JP	15	450 €	585 €
GP.4 - Presupuesto	JP	10	300 €	390 €
GP.5 - Informe de sostenibilidad	JP	5	150 €	195 €
GP.7 - Lectura	JP	10	300 €	390 €
TP - Trabajo Previo		25	440 €	572 €
TP.1 - Estudio del estado del arte	I	10	200 €	260 €
TP.2 - Preparación de entorno de trabajo	P	15	240 €	312 €
DA - Desarrollo de la aplicación		160	5.260 €	6.838 €
DA.1 - Integración del entorno natural	P	25	400 €	520 €
DA.2 - Programación de agentes		125		
DA.2.1 - Observaciones del agente	I,P	25	900 €	1.170 €
DA.2.2 - Toma de decisiones	I,P	45	1.620 €	2.106 €
DA.2.3 - Refuerzos positivos/negativos	I,P	30	1.080 €	1.404 €
DA.2.4 - Configuración de parámetros	I,P	25	900 €	1.170 €
DA.3 - Representación de los agentes	I,P	10	360 €	468 €
EA - Entrenamiento y análisis		100	3.320 €	4.316 €
Entrenamiento	P,T	70	2.240 €	2.912 €
Análisis	I,T	30	1.080 €	1.404 €
Total		425	15.460 €	20.098 €

Tabla 4: Tabla de partidas por tarea. Roles: JP - jefe de proyecto, I - investigador, P - programador, T - tester. Coste SS - Coste de la seguridad social. Elaboración propia

Costes genéricos

En el apartado de planificación se estableció como lugar de trabajo el propio domicilio, en el cual se trabajará durante toda la semana. Vamos a estimar el precio del alquiler de este espacio en función de la tarifa de un espacio *coworking*⁴. Tomaremos como referencia otros ejemplos de alquiler en Barcelona que consten de todos los servicios necesarios para trabajar en el proyecto, los cuales son: Internet,

⁴Oficinas compartidas en las que profesionales autónomos y empresarios se dan cita para trabajar

agua, electricidad, acceso todos los días de la semana y mesa individual. El coste de un espacio con esas prestaciones es de 250 euros al mes (13), suponiendo un coste total de 1250 euros a lo largo de los 5 meses.

Por otro lado tenemos los gastos en relación al software usado en el proyecto. Vamos a mencionar a continuación el coste de estas herramientas y en la tabla 5 se ofrece un desglose más detallado:

1. *Google Docs* - Al ser una herramienta de código abierto no supone ningún tipo de coste.
2. *Microsoft Word* - Al contar con la licencia de la universidad no supone ningún tipo de coste.
3. *GanttPro* - Supone un gasto de 15 euros por usuario al mes. Solamente se contratará una licencia para el jefe de proyecto.
4. *Unity* - Vamos a usar una versión gratuita llamada *Unity Hub*, con lo cual no supondrá ningún tipo de coste.

Software	Coste/mes	Meses	Total
Google Docs	0 €	5	0 €
Microsoft Word	0 €	5	0 €
GanttPro	15 €	5	75 €
Unity	0	5	0 €
Total			75 €

Tabla 5: Costes de los recursos de Software. Elaboración propia.

Una vez calculado los costes de software, nos quedaría calcular los costes de los dispositivos de Hardware, en nuestro sólo contaremos con un ordenador portátil. Para calcular la amortización que tendrá este dispositivo hemos calculado el coste que tendría por hora. Para ello hemos tenido en cuenta que un año tiene 220 días hábiles y 8 horas laborales por día, por lo que el coste por hora es :

$$\text{CostePortatil}/(\text{VidaUtil} * 220 * 8)$$

$$\text{CostePorHoraPortatil} = 1400/(4 * 220 * 8) = 0,19$$

Hardware	Precio	Unidades	Vida útil	Horas	Amortización
Portatil	1400	1	4	425	80,75 €

Tabla 6: Costes de los recursos de Hardware. Elaboración propia.

Contingencia

Es muy importante contar con un posible sobrecoste para cubrir imprevistos. Como ya hemos comentado, es un proyecto con mucha incertidumbre respecto a los plazos debido a que es una tecnología bastante compleja. Contando que vamos a encontrarnos con problemas durante el desarrollo, hemos fijado un 20 % de sobrecoste. En la siguiente tabla (7) mostramos el sobrecoste de cada una de los tipos de gastos que hemos comentado anteriormente.

Tipo	Coste	Coste Contingencia
Espacio	1.250 €	250 €
Personal	20.098 €	4.019,60 €
Software	75 €	15 €
Hardware	80,75 €	16,15 €
Total	21.503,75 €	4.300,75 €

Tabla 7: Tabla de contingencia del 20 %. Elaboración propia.

Imprevistos

Para terminar con la parte de presupuesto, vamos a calcular el coste que supondría lidiar con los diferentes obstáculos ya comentados en la planificación. Vamos a analizar la probabilidad que tendrían estos imprevistos y el coste que pueden suponer.

1. **Problemas en el desarrollo** - Como ya comentamos, puede que se produzca un aumento en el tiempo de desarrollo. Esto supondría un mayor número de horas por parte del programador y del *tester*. Estimamos que en este caso el número de horas de trabajo se incrementan hasta 40, de las cuales 30 corresponden al programador y las otras 10 al *tester*, suponiendo un gasto adicional de 640€. La probabilidad de que esto suceda es bastante elevada, debido a la complejidad de esta nueva tecnología, así que estimamos un 20%.

- 2. Rendimiento** - Este aspecto será posiblemente nuestro mayor enemigo durante el proyecto. Pues es un tiempo de espera en el que no podemos hacer prácticamente nada. En la planificación estimamos que esto nos puede retrasar unas 20 horas. Esas horas de más, suponen un coste adicional en la fase de entrenamiento y análisis. Vamos a suponer que dedicaremos 15 horas a entrenar y otras 5 al análisis, el gasto sería de 680€. La probabilidad de encontrarnos ante esta situación es de un 15 %.
- 3. Fallo de los dispositivos** - Es una posibilidad que se debe tener en cuenta. Nuestro ordenador portátil es nuestra principal herramienta, y si esta se rompe tendremos que comprar uno nuevo. Es una posibilidad bastante remota, y por eso se estima un riesgo de un 5 %.

A continuación, en la tabla 8 se muestra un desglose de los gastos imprevistos:

Imprevisto	Coste	Probabilidad	Coste Total
Problemas en el desarrollo	640 €	20 %	128 €
Rendimiento	680 €	15 %	102 €
Fallo de los dispositivos	1.400 €	5 %	70 €
Total		2.720 €	300 €

Tabla 8: Tabla del sobrecoste de cada tipo de imprevisto. Elaboración propia

Coste total

Ahora que ya tenemos desglosado cada uno de los gastos del proyecto, vamos a recoger cada uno de esos gastos y calcular el coste total que tendrá el proyecto. En la tabla 9 se presenta el presupuesto final, el cual asciende a 26.105€.

Tipo	Coste
Personal	20.098 €
Espacio	1.250 €
Software	75 €
Hardware	80,75 €
Contingencia	4.300,75 €
Imprevisto	300 €
Total	26.105 €

Tabla 9: Tabla del presupuesto final. Elaboración propia.

6.2. Control de gestión

Vamos a definir ahora los mecanismos para controlar las posibles desviaciones que podamos tener en nuestro proyecto. También tendremos unos indicadores numéricos que nos den una mayor seguridad en el control, e iremos actualizando el presupuesto en base a estos indicadores.

Los indicadores numéricos serán los siguientes:

- Desviación coste personal por tarea: $(coste_{Estimado} - coste_{Real}) \times horas_{Reales}$
- Desviación realización tareas: $(horas_{Estimadas} - horas_{Reales}) \times coste_{Real}$
- Desviación total en la realización de tarea: $(coste_{EstimadoTotal} - coste_{RealTotal})$
- Desviación total de recursos: $(coste_{EstimadoTotal} - coste_{RealTotal})$
- Desviación total coste de imprevistos : $(coste_{EstimadoImprevistos} - coste_{RealImprevistos})$
- Desviación total de horas : $(horas_{Estimadas} - coste_{horasReales})$

6.3. Presupuesto final

Tal y como hemos comentado en la planificación final en el punto 5.4, el proyecto ha sufrido ciertas modificaciones que recaen directamente sobre el presupuesto. Vamos a actualizar los nuevos costes mediante los mecanismos de control de gestión que acabamos de establecer. Mostramos a continuación una tabla que indica el coste estimado y el real de aquellos aspectos que han sufrido modificaciones.

Tipo de coste	Coste Estimado	Coste Real	Desviación
Costes por actividad			
GP.1 - Reuniones	2.132 €	2.132 €	0 €
GP.6 - Documentación	3.900 €	3.900 €	0 €
GP.2 - Alcance	780 €	780 €	0 €
GP.3 - Planificación	585 €	702 €	-117 €
GP.4 - Presupuesto	390 €	468 €	-78 €
GP.5 - Informe de sostenibilidad	195 €	195 €	0 €
GP.7 - Lectura	390 €	390 €	0 €
TP.1 - Estudio del estado del arte	260 €	260 €	0 €
TP.2 - Preparación de entorno de trabajo	312 €	832 €	-520 €
DA.1 - Integración del entorno natural	520 €	520 €	0 €
DA.2 - Programación de agentes	5.850 €	7.020 €	-1.170 €
DA.3 - Representación de los agentes	468 €	468 €	0 €
EA.1 Entrenamiento	2.912 €	3.536 €	-624 €
EA.2 Análisis	1.404 €	1.404 €	0 €
Total coste por actividad	20.098 €	22.607 €	-2.509 €
Costes genéricos			
Hardware	80,75 €	93,10 €	-12,35 €
Total desviación			-2.521,35 €

Tabla 10: Presupuesto final de los costes totales del proyecto. Elaboración propia.

El aumento de horas en el personal ha supuesto un gasto extra de 2509€, que sumado al gasto extra del uso del PC hacen un total de 2521,35€. Esto significa que la estimación del presupuesto fue correcta, ya que el presupuesto era de 26.105€, de los cuales 4300€ iban destinados a contingencia y 300€ a imprevistos, por lo que queda un balance positivo de 2078,65€. De los 26.105€ disponibles se han usado 24.026,35 €.

Capítulo 7

Sostenibilidad

Como en todo proyecto, es fundamental analizar en profundidad la sostenibilidad. Tendremos en cuenta las tres dimensiones: económica, ambiental y social. Haremos una evaluación tal y como se nos pide sobre el dominio de la competencia de la sostenibilidad, así como un análisis de las tres dimensiones dentro del marco de nuestro trabajo.

7.1. Autoevaluación

El tema de la sostenibilidad es algo que está presente en nuestro día a día, y al cual no le prestamos la atención que merece. En el ámbito de la informática hacer un buen análisis de la sostenibilidad puede condicionar mucho el resultado que tendrá, y es haciendo un trabajo de este tipo cuando nos damos cuenta la importancia que tiene.

De las tres dimensiones, a la que más importancia se le da como normal general es al aspecto económico, pues siempre se intenta maximizar los beneficios que tendrá el proyecto y sacarle la mayor rentabilidad posible. Hoy en día, nos estamos dando cuenta que el aspecto económico ya no es lo único que importa. El medio ambiente está condicionando cada vez más el desarrollo de la tecnología, pues la alta demanda de dispositivos electrónicos de alta generación tiene un impacto directo sobre la sociedad y el medio ambiente.

En los últimos años, se está frenando el desarrollo tecnológico por culpa del daño que está teniendo sobre el medio ambiente. Entre otros problemas podemos destacar: el cambio climático, contaminación de aguas, vertederos de desechos, etc. Es por esto que dependiendo del proyecto considero que se debe dar prioridad a

un desarrollo sostenible en cuanto al medioambiente se refiere, ya que la cantidad de recursos que requiere es excesivo.

Para terminar, siempre debemos tener en cuenta y hacer una reflexión sobre la utilidad que va a tener nuestro proyecto. Todos los trabajos siempre deben tener un trasfondo social, en cual se busque cubrir las necesidades que pueda tener la sociedad. Una de las motivaciones que tiene este proyecto es iniciar un posible desarrollo profesional hacia el *machine learning*, por lo que esta motivación extra espero que sirva para desarrollar un buen proyecto y que pueda contribuir con el desarrollo de esta tecnología.

7.2. Dimensión económica

En cuanto al coste económico que supone llevar a cabo el proyecto, si tenemos en cuenta las futuras aplicaciones que puede tener en el ámbito de los videojuegos, y siendo esta una de las industrias más rentables y potentes de la actualidad, considero adecuado el presupuesto establecido inicialmente.

En el apartado anterior hemos hecho un desglose de cada uno de los costes que supone el proyecto, todos ellos en una situación ideal. Después de varios meses de desarrollo nos encontramos ante un gasto final de 24.026,35 €, que supone un 92 % del presupuesto inicial. El resto se ha dedicado a imprevistos y contingencia. Hemos hecho una buena estimación, pero que podría ser mejor. Sin embargo, al tratarse de una tecnología que genera tanta incertidumbre en ciertas fases del proyecto, no se deberían hacer recortes en relación a los costes de imprevistos y contingencias.

En próximas ocasiones cuando hagamos proyectos de RL, una de las mejoras podría ser invertir en un buen PC. Esto aunque suponga un gasto extra lo compensaremos en tiempo de entrenamiento, reduciendo así costes de personal.

Si prestamos atención a la vida útil del proyecto nos encontramos ante una aplicación funcional, pero que podría tener muchas mejoras que la hagan mucho más potente. Si se decidiera en un futuro hacer una ampliación del mismo, obviamente supondría un gasto extra.

Por último, durante el desarrollo de la investigación nos hemos visto en situaciones donde parecía muy difícil sostener la viabilidad del trabajo. Una vez hemos

terminado, hemos visto que hay decisiones que afectan directamente en el funcionamiento y en unos buenos resultados, y que obviamente pueden poner en peligro el resultado final. Tras muchas pruebas fallidas nos hemos dado cuenta que no hay una única forma correcta de llegar a unos buenos resultados, y que también hay que saber lidiar con estas situaciones.

7.3. Dimensión ambiental

Uno de los puntos a favor del proyecto, es el mínimo impacto que va a tener el medioambiente. El único dispositivo que afecta negativamente es el PC con el que se realizará el desarrollo. Si quisiéramos reducir el impacto, nos sería prácticamente imposible, ya que estamos utilizando los recursos mínimos para que el proyecto se pueda llevar a cabo. Si usamos varios ordenadores, o uno más potente, el impacto medioambiental sería mucho mayor.

Sin embargo, por mínimo que sea el impacto que tenga un proyecto de este tipo, el constante desarrollo de esta tecnología, conlleva la fabricación de CPUs⁵ y GPUs, ya que sin estos componentes no podríamos llevar a cabo un proyecto de este tipo. Podríamos decir que por mucho que intentemos reducir el impacto en nuestro trabajo, a nivel global de aquí a un tiempo será insostenible.

A pesar de ser muy complicado reducir el impacto medioambiental, hemos realizado algunas tareas en paralelo con el fin de reducir gasto energético. Debido a que el entrenamiento del agente conlleva muchas horas, mientras este entrena siempre se ha tratado de avanzar en otras funcionalidades o incluso en la memoria. En el caso de volver a realizar un proyecto de este tipo, los recursos necesarios serán los mismo, pero si que reduciríamos el tiempo para llevarlo acabo. Al ser una tecnología que desconocía, hay una cantidad de horas perdidas por desconocimiento que en próximas ocasiones se pueden invertir en otras cosas.

En cuanto a la vida útil, podemos enlazarlo con lo comentado anteriormente. Si quisiéramos ampliar el alcance de la investigación obviamente vamos a emplear más recursos, pero optimizándolos y aprovechándolos al máximo. Una ventaja que ofrece la librería *ML-Agents* es que podemos empezar un entrenamiento a partir del conocimiento de un modelo ya entrenado, cosa que se puede explotar mucho.

⁵Central Processing Unit (CPU) en inglés, es coloquialmente el cerebro del ordenador.

Por último, ya nos hemos visto en situaciones donde no conseguíamos avanzar, y que tras varias pruebas y entrenamientos perdidos seguíamos obteniendo malos resultados. Esto se ve reflejado directamente en la huella ecológica, por lo que es sumamente importante la toma de decisiones antes de realizar un entrenamiento.

7.4. Dimensión Social

La aplicación que se va a desarrollar no es más que una mera simplicidad de lo que se puede llegar a hacer con esa tecnología. Sin embargo, nuestro proyecto pretende ser la base de futuras implementaciones que mejoren nuestro diseño.

Como ya comentamos en la dimensión económica, la industria de los videojuegos es de las más potentes, y que más demanda tiene en la actualidad. Nuestro proyecto pretende dar soporte a muchas de las necesidades que presenta el desarrollo de videojuegos. Ya comentamos que en prácticamente cualquier juego están presenten los NPCs (3), y en este trabajo se va a desarrollar un comportamiento similar.

Durante estos meses de investigación y desarrollo he podido experimentar desde dentro lo que supone enfrentarse a diversas dificultades dentro del campo de la investigación. Puedo decir que ahora tengo una percepción distinta y he aprendido a apreciar el gran trabajo que conlleva. A partir de ahora tendré una visión mas completa y podré decidir con criterio en el futuro si quiero o no dedicarme al campo de la investigación.

Teniendo en cuenta todo esto, y sumándole el gran interés que tengo en el ámbito del *machine learning*, el resultado final de este TFG puede ser de gran utilidad en el ámbito social. La comunidad que engloba a los desarrolladores de *Unity* y *ML-Agents* es muy extensa y puede aprovecharse de este trabajo. Por otro lado, no considero que pueda perjudicar a nadie, de hecho todo lo contrario.

Por último, hemos de decir que uno de los temores eran encontrarnos ante limitaciones por parte de *Unity* o *ML-Agents*, y que ciertas funcionalidades no pudieran implementarse por fallos o simplemente falta de contenido. Por suerte hemos podido desarrollar sin problema cada una de las funcionalidades previstas.

Capítulo 8

Desarrollo

En este apartado vamos a comentar de manera cronológica cuales han sido las diferentes tareas y aspectos que se han abordado para entender y llevar a cabo la parte más técnica del proyecto.

Una de las principales razones por las que se escogió Unity para representar gráficamente el comportamiento de los agentes, es la facilidad para crear mecánicas de movimiento y así poder hacer un mejor seguimiento del aprendizaje. Para esto se propuso la creación de un videojuego que simplemente simula el movimiento de una persona dentro de un entorno natural hasta llegar a un objetivo, pudiendo así testear gran parte de las funcionalidades antes de adentrarnos en el entrenamiento.

Vamos a diferenciar dos grandes bloques de trabajo, el primero de ellos está relacionado con lo comentado anteriormente, el desarrollo y adaptación de un entorno realista y óptimo para representar a modo de videojuego un comportamiento lo más humano posible. Recordemos que el objetivo principal de los agentes será llegar a un punto aleatorio dentro del entorno, por esta razón la manera en la que estos se mueven e interactúan con él, influirá directamente en los resultados del problema.

Después de desarrollar el videojuego, ya podremos pasar a un segundo bloque más extenso, donde se programará el comportamiento de los agentes, entrenándolos y analizando los resultados. Obviamente esta distinción entre etapas no es estricta ni secuencial, ya que hay una constante readaptación de ciertas características del entorno para que sean compatibles y permitan el entrenamiento del agente. Es por esto que en este apartado también explicaremos diversos conceptos que nos serán de gran ayuda para entender todo el proceso.

8.1. Desarrollo del videojuego

El entorno de entrenamiento consta de tres elementos que están englobados dentro de un elemento principal llamado “Entorno”, tal y como vemos en la figura 8. Vamos a diferenciar entre el terreno (*Terrain*), el agente (*Agent*) y el objetivo (*Target*) que tratará de alcanzar el agente.



Figura 8: Elementos del entorno del entrenamiento

Terreno

El primer objetivo dentro de esta fase es incorporar el terreno por donde el agente se moverá. Como uno de los objetivos del trabajo es la navegación por entornos naturales, hemos optado por usar un *heightmap* (mapa de altura). Un *heightmap* debe ser una imagen en escala de grises, con áreas blancas que representan las áreas altas de su textura y negro que representa las áreas bajas. Con esto conseguimos dar un aspecto bastante realista de las protuberancias que hay en un terreno de la vida real. En (14) están todas las indicaciones que hemos seguido trabajar con un *heightmap*.

En *Unity* hay una herramienta/paquete que dada una imagen en escala de grises devuelve el terreno que representa. Para generar el terreno, debemos proporcionar a la herramienta la imagen por supuesto, pero a parte necesita saber una serie de características sobre el terreno a generar. En concreto necesita saber las proporciones del terreno, así como la diferencia entre su punto más bajo y más alto. En la figura 9 tenemos un ejemplo de las características de un terreno usado en el proyecto. Con esto y el mapa de alturas conseguimos un terreno como el de las figuras 10 y 11.

General	
Total Terrain Width(m)	9820
Total Terrain Length(m)	10020
Terrain Height(m)	1352

Figura 9: Características del terreno a generar

En este caso, el director ha proporcionado todos los datos sobre dos terrenos de la vida real. Los datos son del Institut Cartogràfic i Geològic de Catalunya (ICGC) (15). Entre otros datos, contamos con mapas de elevaciones, imágenes aéreas, y mapas en formato imagen.

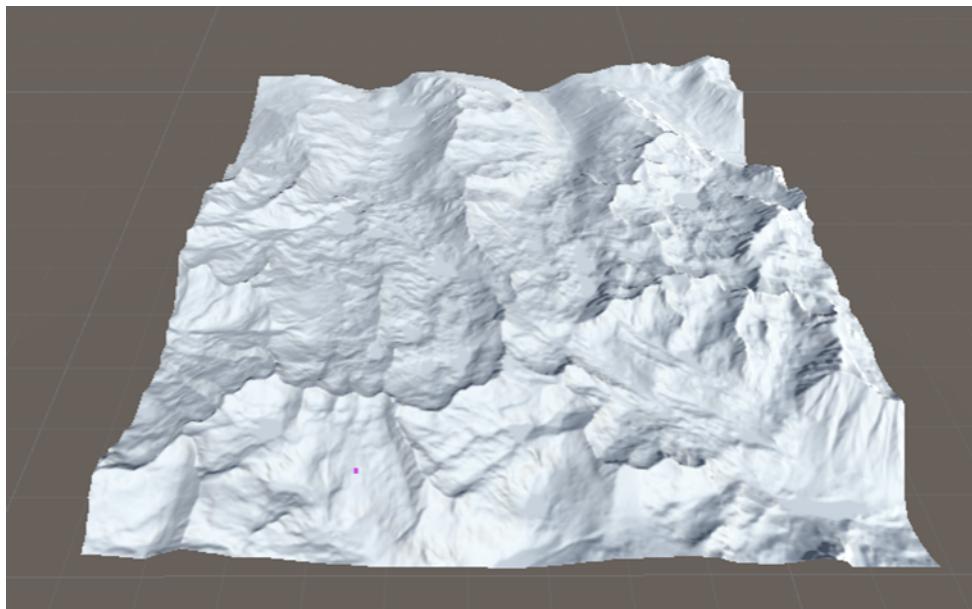


Figura 10: Ejemplo de terreno usado en el proyecto 1

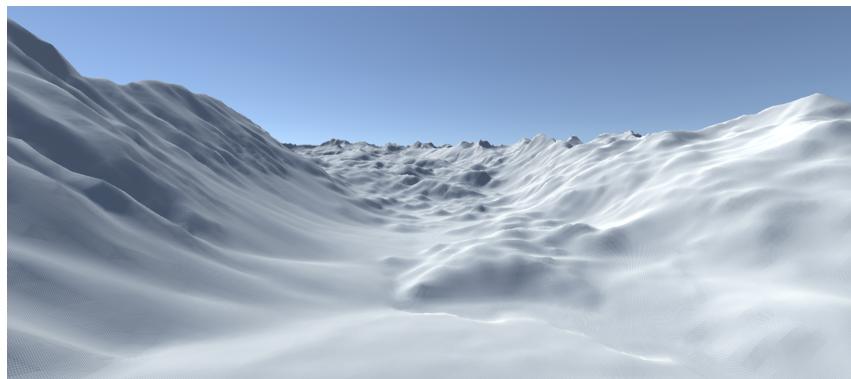


Figura 11: Ejemplo de terreno usado en el proyecto 2

Para acabar con la modelización del terreno, tenemos que hablar de un componente al cual haremos referencia también cuando hablamos del agente y el objetivo. El terreno debe tener activado un componente llamado *Collider*. Los componentes *Collider* definen la forma de un objeto y permiten detectar colisiones físicas. Tener activo esto en el terreno permitirá que otros objetos que colisionen con él, y que también tengan activo este componente, no atraviesen el terreno.

Agente

Cuando hablamos sobre la representación del agente, establecimos que íbamos a llevar a cabo una representación gráfica sencilla. Después de hacer varias pruebas con distintos objetos, hemos elegido la cápsula para simular a la persona. Como no tiene otra finalidad más allá de simular movimientos básicos, es bastante irrelevante. En la figura 12 tenemos un ejemplo de como se vería la cápsula sobre un terreno ya texturizado.



Figura 12: Ejemplo de la representación del agente

Objetivo

El objetivo será representado mediante un cubo, el cual su única finalidad es marcar una referencia gráfica para controlar que el agente está llegando hasta él. El componente *Collider* de este objeto es importante, ya que cuando el agente colisiona con él significa que lo ha encontrado. La cápsula de colisión del objeto se puede modificar para que sea más grande, y así añadir una ayuda extra. En este caso hemos ampliado esta cápsula a un tamaño de 2x2x2, tal y como en la figura 13.

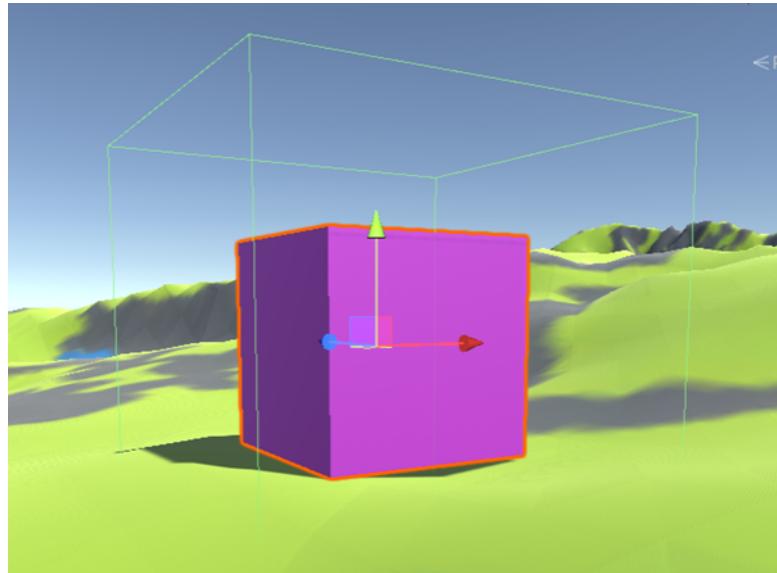


Figura 13: Ejemplo de la representación del objetivo

8.2. Mecánicas

El tipo de movimiento que seguirá el agente es de las primeras cosas que debemos implementar. En un primer momento se decidió implementar este movimiento de forma que en cada iteración pudiera escoger una de las direcciones. Este movimiento se testeó mediante un script sencillo asociado al objeto del agente. El movimiento aparentemente funcionaba bien, pero quisimos tener una alternativa por si surgiera alguna complicación con esta elección.

Tras probar varias alternativas, encontramos otra que se adapta bastante bien, y que además es más realista que la anterior. Esta mecánica, así como otros con conocimientos que me han servido para llevar a cabo este proyecto ha sido inspirada de un proyecto ya existente visto en (16). El movimiento se basa en dos acciones, una donde se decide si moverse o no, y en el caso de moverse elige si continuar recto, girar a la izquierda o a la derecha. Podemos verlo implementado en la figura 14.

```
public override void Heuristic(in ActionBuffers actionsOut)
{
    var discreteActionsOut = actionsOut.DiscreteActions;
    discreteActionsOut.Clear();
    int lForward = 0;
    int lTurn = 0;
    if (Input.GetKey(KeyCode.UpArrow))
    {
        lForward = 1;
    }
    if (Input.GetKey(KeyCode.LeftArrow))
    {
        lTurn = 1;
    }
    else if (Input.GetKey(KeyCode.RightArrow))
    {
        lTurn = 2;
    }

    float dist = lForward * _speed * Time.fixedDeltaTime;
    transform.position += transform.forward * dist;
    transform.Rotate(transform.up * lTurn * _turnSpeed * Time.fixedDeltaTime);
```

Figura 14: Implementación del movimiento del agente mediante teclado

8.3. Conceptos básicos para configurar el agente

Una de las partes más importante del proyecto, y que más repercusión tendrá en los resultados obtenidos, es la configuración del agente. En un primer momento, el agente partirá desde un entorno que no conoce sin saber cual es su objetivo. A partir de unas condiciones preestablecidas y unas respuestas por parte del programador, éste tratará de completar su tarea.

En todo problema relacionado con el aprendizaje por refuerzo, el hilo de ejecución del agente en cada iteración del aprendizaje es el mismo, y consta de cuatro partes que ya comentamos por encima, pero que ahora vamos a profundizar más:

1. **Observación** - El agente recoge la información que necesita del entorno.
2. **Decisión** - A partir de lo aprendido en ejecuciones anteriores, el agente debe decidir qué acción realizar.
3. **Acción** - Una vez tomada la decisión sobre qué hacer, a partir de la política de movimiento establecida por el programador, el agente realiza la acción.

4. Recompensas - Por último, una vez llevada a cabo la acción, el programador debe dar un *feedback* positivo o negativo si la acción lo requiere, y así el agente saber qué hacer en próximas ejecuciones.

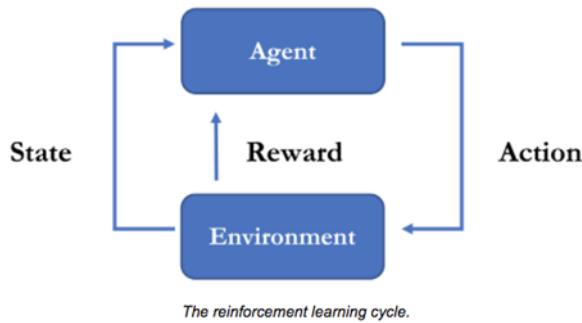


Figura 15: Ejemplo gráfico del flujo de aprendizaje

El entorno de entrenamiento de *Unity* el cual usaremos, viene definido por una clase de la librería *ML-Agents*, esta clase se llama *Agent*. En esta clase hay una serie de funciones predefinidas, que pueden ser sobreescritas para modelar el hilo de ejecución previamente comentado. Todo esto será programado en diferentes *scripts* según la versión, que serán asignados al objeto que simula al agente. Este *script* heredará de la clase previamente comentada ajustando así las funciones a nuestras necesidades.

Entre otras funciones, el *script* será el cargado de inicializar y finalizar todo lo que tiene que ver con el entorno, como puede ser la posición inicial del agente y del objetivo. La clase *Agent* cuenta con dos funciones que gestionan este flujo, una que termina el episodio, y otra que lo empieza. Llamaremos episodio a cada intento que realizan los agentes, es decir el número de veces que han tratado de llegar hasta el objetivo.

8.3.1. Observaciones

La manera en la que el agente recoge la información del entorno puede variar, y dependiendo del tipo de problema conviene usar una u otra. En nuestro caso vamos a usar un vector de observaciones, donde vamos a poder almacenar tanto información numérica o física.

Para llenar este vector de observaciones, se utiliza la función *CollectObservations* (*VectorSensor sensor*) de la clase *Agent*. A cada iteración, añadiremos la información al vector mediante el método *AddObservation* del “sensor” que recibe como parámetro. Es un aspecto sumamente importante, ya que la mala elección de observaciones influyen mucho sobre los resultados. El hecho de pasar observaciones de más no hará que el agente aprenda más rápido, si no que confundirá lo confundirá en su tarea. Esta función permite recibir los siguientes tipos de datos:

- *AddObservation(Int32)*
- *AddObservation(Single)*
- *AddObservation(Vector3)*
- *AddObservation(Vector2)*
- *AddObservation(Quaternion)*
- *AddObservation(Boolean)*
- *AddObservation(IEnumerable<Single>)*
- *AddOneHotObservation(Int32, Int32)*

8.3.2. Acciones y movimientos

Al igual que con las observaciones, a la hora de realizar una acción el agente llama a la función *OnActionReceived* (*float [] vectorAction*). Este parámetro que recibe indica la acción a realizar en cada paso, que puede ser de dos tipos, continuas o discretas.

1. **Continuas** - Cuando establecemos que el tipo de acción sea continua, el parámetro *vectorAction* es un *array* de números en coma flotante. Este tipo de acciones se usa cuando se quiere tener un control más exacto sobre la acción a realizar, como puede ser el lanzamiento de una pelota, pudiendo ajustar la velocidad con la que se lanza la pelota.
2. **Discretas** - Usando este tipo de acciones, los números que se reciben en el vector son enteros, representando una acción discreta específica. Al contrario que las acciones continuas, en las discretas simplemente indicamos con un

índice la acción a realizar, por ejemplo si solo queremos decidir en qué dirección se moverá un jugador y no nos interesa cuanto se mueve, es una buena opción usar este tipo de acción.

8.3.3. Recompensas

Este es un aspecto, que como ya hemos comentado en varias ocasiones, determina en cierta manera el potencial de aprendizaje del agente. Hay dos funciones que se usan para dar el refuerzo positivo o negativo al personaje. Una es *AddReward()*, la cual usaremos cuando queramos añadir un valor a la recompensa ya dada durante ese episodio. Por otro lado tenemos *SetReward()*, que sobreescribe la recompensa dada durante ese episodio. Es interesante usar una u otra según el caso.

8.4. Gestión de parámetros

Entender las diferentes fases del hilo de ejecución es vital, pero para terminar de diseñar el entorno de entrenamiento debemos especificar de alguna manera al agente cuantas observaciones recibirá, qué tipo de acciones podrá hacer, etc. Todo esto nos lo permite un componente llamado de *Behavior Parameters* que asociaremos al agente desde el *inspector* (figura 16).

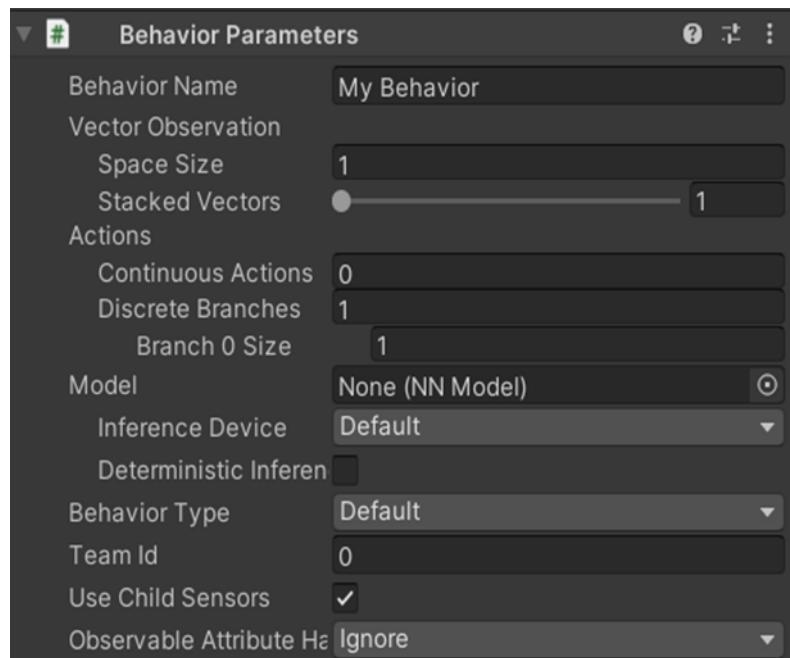


Figura 16: Componente *Behavior Parameters*

Vector Observation

- **Space Type** - Aquí estableceremos el tipo de acción que se usará. Como ya hemos visto, pueden ser continuas o discretas.
- **Space Size** - Si vamos a usar acciones continuas, aquí especificamos el tamaño del vector de acción.
- **Branches** - Este apartado lo tendremos en cuenta cuando usemos un tipo de acción discreta. Especificaremos el número de ramas y el número de acciones que contiene cada rama.

Behavior type

- **Default** - Seleccionamos este tipo de comportamiento cuando queremos entrenar al agente.
- **Heuristic Only** - En este modo indicamos al agente que siga la órdenes de movimiento previamente programadas en la función *Heuristic* de la clase *Agent*. Este método también será sobreescrito para adaptarlo a nuestro movimiento.
- **Inference Only** - Una vez tengamos al agente entrenado, podremos seleccionar este modo, el cual requiere el modelo entrenado previamente. De este modo podremos probar que tan inteligente es nuestro agente.

8.5. Escenarios de entrenamiento

En el punto 2.2.2 explicamos la existencia de un componente muy importante como era el cerebro, viendo que se podrían organizar de diferentes maneras. Pues la manera en la que asociamos agentes y cerebros, nos permite crear múltiples tipos de escenarios. La organización que se vaya a usar depende mucho del tipo de problema, e influye también en el rendimiento del aprendizaje.

- **Agente único** - La forma más básica de entrenar a un agente, donde tenemos un único agente conectado a un cerebro. Este tendrá sus señales de observaciones y recompensas independientemente.

- **Agente único simultáneo** - Un sistema de recompensa independiente vinculado a un solo cerebro. Consta de varios agentes que comparten la misma tarea, cada uno en su entorno, acelerando así el aprendizaje. Tenemos un ejemplo en la figura 17.

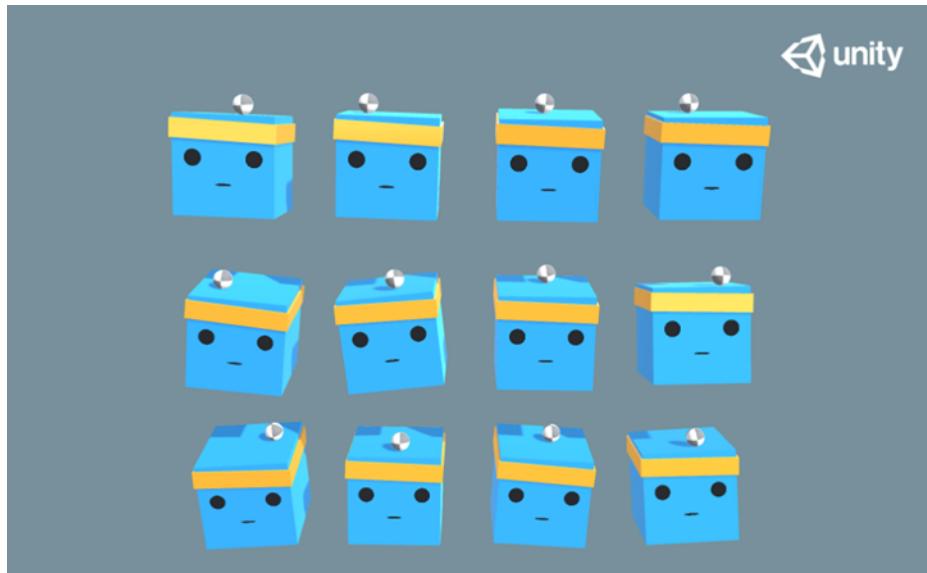


Figura 17: Ejemplo de Agente único simultáneo

- **Adversario** - En este escenario nos encontramos ante dos o más agentes que tienen señales de recompensa opuestas, pero conectados a un mismo cerebro. La recompensa de uno es la penalización del otro. Dos agentes que juegan al fútbol es un escenario perfecto para esta distribución. Tenemos un ejemplo en la figura 18.
- **Multiagente cooperativo** - Varios agentes que trabajan para resolver una tarea que no puede completar un único agente. La señal de recompensa es compartida, aunque pueden estar vinculados a uno o varios cerebros, dependiendo de las acciones que pueda hacer cada uno. Un ejemplo sería un escenario donde haya que recolectar ciertos elementos, y cada uno tenga unas habilidades de movimiento distintas, haciendo así que cada agente sea capaz de alcanzar ciertos elementos.

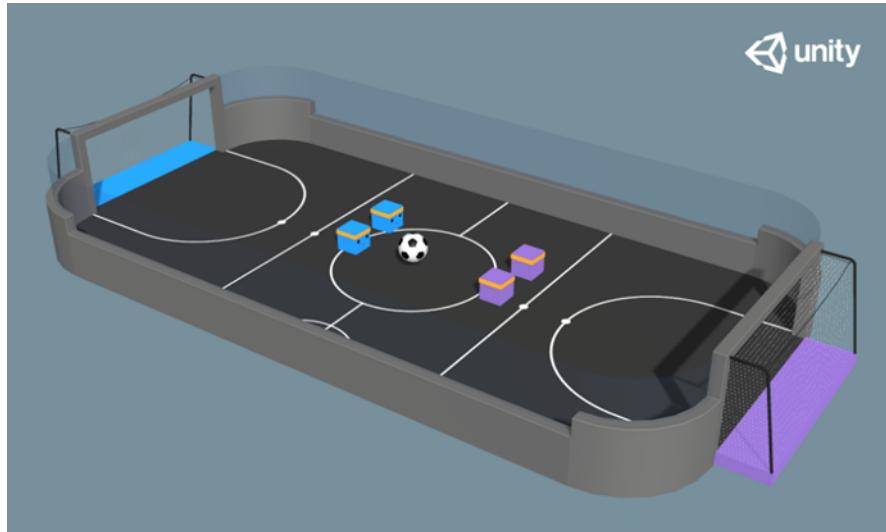


Figura 18: Ejemplo de un escenario Adversario

La elección del tipo de escenario es muy importante para acelerar el entrenamiento. Lo ideal para nuestro proyecto sería un agente único simultáneo, replicando el entorno tantas veces como sea necesario hasta encontrar el punto óptimo. Esto conlleva también un costo importante en cuanto a recursos en el PC, pero es que además en nuestro proyecto contamos con un terreno muy grande y muy costoso de replicar. Si tenemos en cuenta también las limitaciones del PC con el que se va a experimentar, estamos sometidos a una desventaja considerable en términos de eficiencia.

Una de las alternativas propuestas fue replicar el agente y objetivo varias veces, pero dentro en diferentes zonas del mapa. Esto aún así seguía siendo bastante costoso para el PC, consiguiendo que el entrenamiento fuera fluido solo con dos agentes y dos objetivos.

8.6. Algoritmos

Vamos a ver diferentes formas de clasificar los algoritmos de aprendizaje por refuerzo. En primer lugar vamos a comentar los algoritmos basados en el valor (*Value-Based*) y los algoritmos basados en la política (*Policy-Based*).

Los algoritmos que utilizan únicamente una función de valor o de acción-valor sin implementar una política de forma explícita, entran en el grupo de los algoritmos basados en el valor. Estos algoritmos no te dicen qué acción debes tomar

explícitamente, sino que te indican cuánta recompensa recibirás desde cada estado o estado-acción. Por tanto, el programador debe decidir qué acción tomar tras ver estos valores. Un ejemplo puede ser tomar siempre la acción con el valor-Q más alto (19).

Por otro lado tenemos los algoritmos que implementan una política, y que es esta la que decide qué acción realizar en cada momento. Estos algoritmos obviamente están dentro de los algoritmos basados en la política, y están implementados de tal forma que deciden la probabilidad de tomar cada acción en cada momento. Tal y como se explica en (17), esta función no nos dice cuánta recompensa recibirá el agente desde cada estado. No aprende ni una función de valor $V(s)$ ni una de acción-valor $Q(s,a)$. Estos algoritmos definen solamente una función de política (*Policy Function*) ($a|s$) que estima la probabilidad de tomar cada posible acción desde cada estado.

Por último vamos a comentar la existencia de un tercer grupo, que básicamente se basa en lo bueno de los dos algoritmos comentados anteriormente. Se aprovecha de una función de valor y de la política de aprendizaje. Estos algoritmos se llaman actor-crítico (*actor-critic*).

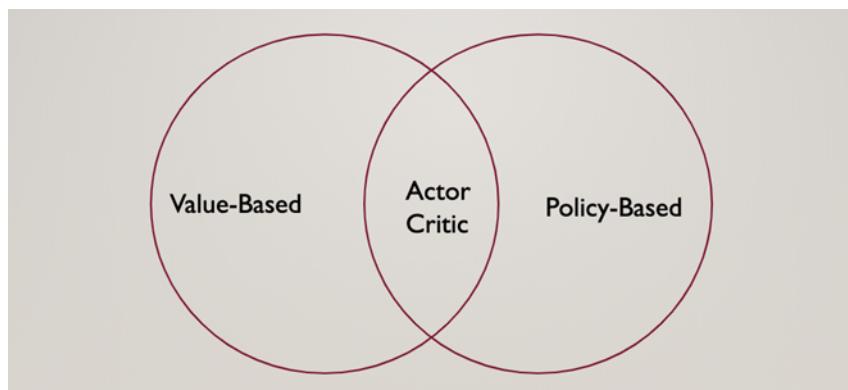


Figura 19: Esquema del algoritmo actor-crítico

La librería *ML-Agents* implementa dos algoritmos de *reinforcement learning* distintos, uno basado en la política y otro de actor-crítico:

- ***Proximal Policy Optimization (PPO)*** - En este escenario nos encontramos ante dos o más agentes que tienen señales de recompensa opuestas, pero conectados a un mismo cerebro. La recompensa de uno es la penalización del otro. Dos agentes que juegan al fútbol es un escenario perfecto para esta distribución.

- **Soft Actor-Critic (SAC)** - Es un algoritmo fuera de la política basado en el marco de aprendizaje por refuerzo de máxima entropía. En este marco, el agente busca maximizar la recompensa esperada al tiempo que maximiza la entropía. Es decir, tener éxito en la tarea actuando de la forma más aleatoria posible (18). Se puede decir que es una ventaja en cuanto a eficiencia respecto a otros algoritmos, pero también supone muchos más cambios en el modelo.

8.6.1. *Proximal Policy Optimization (PPO)*

En el aprendizaje por refuerzo se utilizan métodos de gradiente de políticas los cuales predicen qué camino deben seguir las políticas para mejorar sus resultados aplicando el descenso del gradiente en las redes neuronales. Pero debido a la inestabilidad comentada anteriormente es muy fácil que esta progresión de la mano de los métodos de gradiente de políticas rápidamente se desvíe y produzca resultados ineficientes o erróneos, deformando la política del agente si se aplica el descenso de gradiente en repetidas ocasiones.

Este algoritmo tal y como se explica en (19) surge de proponer una nueva familia de métodos de gradiente para el aprendizaje por refuerzo, que alternan entre el muestreo de datos a través de la interacción con el entorno y la optimización de una función objetivo "sustituta" mediante el ascenso de gradiente estocástico. Tratando así de solucionar el problema comentado.

El método gradiente aplicado a los métodos de política buscan mover los parámetros en los que se basa esta política buscando maximizar los resultados, en nuestro caso la recompensa. PPO está basado en otros algoritmos como TRPO, GAE y A2C/A3C que muestran un rendimiento de vanguardia sobre métodos tradicionales como *Q-learning* (20).

8.7. Gestión de hiperparámetros

Aparte de los parámetros de configuración ya comentados y del algoritmo que se va a usar, hay que tener en cuenta algo llamado fichero de configuración. No es un fichero que sea exclusivo de *Unity*, se trata de un fichero de configuración del motor de *machine learning* Anaconda, que es el que usa *Unity* para sus *Ml-Agents*.

Es un apartado más importante de lo que parece, ya que este fichero cuenta con muchos parámetros y que el simple hecho de ajustar mal un valor puede cambiar

por completo el aprendizaje. Vamos a explicar un poco los hiperparámetros que más influyen:

- ***trainer Type*** - Aquí escogemos qué algoritmo se usará, como ya hemos explicado usaremos PPO.
- ***tax Steps*** - Indica el número de pasos que se podrán ejecutar hasta finalizar el entrenamiento. Cuando estamos frente a problemas complejos conviene que sean valores altos.
- ***time horizon*** - Corresponde a la cantidad de pasos que podrán pasar para asociar una recompensa presente con una acción pasada. Cuando hay recompensas o penalizaciones con frecuencia, conviene que el valor de este parámetro no sea muy alto. Un rango de valores apropiado es de 32 a 2048.
- ***batch size*** - Número de experiencias que tendrá en cuenta en cada iteración para el descenso de gradiente.
- ***buffer size*** - Número de experiencias que recolectar antes de actualizar la política.
- ***learning rate*** - Corresponde al valor inicial para el ratio de aprendizaje del descenso de gradiente.
- ***beta*** - El valor de esta variable indica lo aleatorio que será la política. Aumentando este valor conseguiremos que el agente tome más decisiones al azar. En un entrenamiento exitoso el valor de la entropía debe bajar lentamente, pues si decrece demasiado rápido o demasiado lento, es debido al valor de beta.
- ***num layers*** - Número de capas ocultas de la red neuronal.
- ***hidden nits*** - Número de unidades que hay en cada capa oculta de la red neuronal.

La figura 20 muestra el fichero de configuración que hemos usado para la experimentación. Este fichero proviene de un proyecto llamado *Hallway*, el cual es proporcionado por la propia librería, y que está basado en un objetivo parecido a nuestro proyecto. La librería *Ml-Agents* tiene a disposición de los usuarios múltiples ejemplos, y consideramos que este se adapta bastante bien a nuestro problema.

```
default_settings: null
behaviors:
  My Behavior:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 1024
      learning_rate: 0.0003
      beta: 0.03
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    memory:
      sequence_length: 64
      memory_size: 128
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
  keep_checkpoints: 5
max_steps: 10000000
time_horizon: 64
summary_freq: 10000
```

Figura 20: Fichero de configuración

Capítulo 9

Entrenamiento

Hemos llegado a la parte más importante del trabajo, y por lo tanto la que más tiempo nos ha llevado. Vamos a profundizar en los aspectos comentados en el punto 8.3, relatando paso a paso qué cambios hemos ido introduciendo con el fin de mejorar y optimizar el aprendizaje. Explicaremos en primer lugar ciertas decisiones y configuraciones que serán idénticas en los diferentes experimentos.

Una decisión que hemos tomado y que no variará a lo largo de la experimentación es el criterio para establecer las posiciones iniciales del agente y del objetivo. En cuanto al agente, cada vez que empiece una nueva búsqueda partirá de una posición distinta. Esta vendrá determinada aleatoriamente, y además con un cierto ángulo de rotación, consiguiendo así una gran variedad de puntos de partida. Para la objetivo, lo mantendremos en un posición fija hasta que el agente consiga llegar hasta el, una vez lo alcance se moverá a una nueva posición aleatoria.

Una función que no variará durante la experimentación es *Heuristic()*. Dentro los diferentes tipos de comportamiento que explicamos, vimos la existencia del heurístico, y que gracias a esto podemos usar el movimiento programado en el método *Heuristic()*, y así probar si la lógica implementada funciona. En nuestro caso hemos sobreescrito esta función con uno los movimientos ya explicado en el punto 8.2, pero que podemos cambiar en cualquier momento. En la figura 21 vemos la función implementada.

```

public override void Heuristic(in ActionBuffers actionsOut)
{
    var discreteActionsOut = actionsOut.DiscreteActions;
    discreteActionsOut.Clear();
    int lForward = 0;
    int lTurn = 0;
    if (Input.GetKey(KeyCode.UpArrow))
    {
        lForward = 1;
    }
    if (Input.GetKey(KeyCode.LeftArrow))
    {
        lTurn = 1;
    }
    else if (Input.GetKey(KeyCode.RightArrow))
    {
        lTurn = 2;
    }
    discreteActionsOut[0] = lForward;
    discreteActionsOut[1] = lTurn;
}

```

Figura 21: Función *Heuristic()*

A partir de aquí vamos a hacer un avance progresivo de experimentos en lo que a dificultad se refiere. Empezaremos con implementaciones a priori sencillas, e iremos incorporando restricciones y viendo si conseguimos llegar a la versión deseada, donde el agente se mueva en un entorno complejo. Una ventaja en cierta manera de tener un mapa tan grande es que nos permite escoger en función del entrenamiento, una zona del mapa que se adapte a los estímulos que necesita el agente.

9.1. Movimiento del agente sin restricciones hacia al objetivo

En esta primera versión, vamos a simplificar al máximo la tarea del agente, reduciéndola nada más a que su única misión sea llegar al objetivo de la forma más rápida posible sin tener en cuenta nada más. El *script* donde se programará este comportamiento se llamará *BasicAgent*.

Si recordamos el flujo de aprendizaje del agente, primero debe recolectar información del entorno, esto lo hacía mediante la función *CollectObservations* (*Vector-Sensor sensor*). En este experimento el agente sólo tendrá en cuenta información relacionada con el objetivo, en concreto vamos a tomar la distancia al objetivo, la

dirección en 2D para llegar a él y la dirección actual del agente en 2D. La distancia al objetivo estará normalizada respecto a una distancia máxima que se puede alejar el agente, ya que *ML-Agents* recomienda que las observaciones estén normalizadas.

En cuanto a las recompensas y castigos se muestra una tabla a continuación con las acciones diferentes acciones que se deben tener en cuenta y que decisión tomar.

Acción	Recompensa/Castigo
Encontrar el objetivo	+1f
Salir del mapa	-1f
Alejarse más de la distancia permitida	-1f
Dar más pasos de los permitidos	$-distanceToTarget/SEARCH_RADIUS$
Dar un paso	$-1,0/MAX_WALK_STEPS$

Tabla 11: Tabla de refuerzos experimento 1

Para conseguir que el agente tarde lo menos posible hemos establecido un número máximo de pasos que puede dar que definimos como *MAX_WALK_STEPS*, y toma valor 15000. A cada iteración sumamos un paso, y cuando supere esa variable le penalizamos proporcionalmente a lo lejos que se haya quedado del objetivo, siempre controlando que lo máximo que se le pueda restar sea -1.

Para resumir gran parte de lo dicho anteriormente vamos a echar un vistazo al componente *Behavior Parameters* en la Figura 22. En cuanto a las observaciones, tenemos la distancia al objetivo (*float*), la dirección para llegar a él (*Vector2*) y la dirección actual del agente (*Vector2*), que en total suman 5. En cuanto al movimiento, tenemos dos ramas, una que decide si nos movemos o no (dos acciones), y otra para decidir si seguimos hacia delante, hacia la izquierda o hacia la derecha (tres acciones). También hemos establecido *Stacker Vectors* a 3, consiguiendo así que el agente recuerde las observaciones de las últimas tres iteraciones.

El movimiento usado para este experimento es el de las 8 direcciones, tratando de simplificar al máximo la tarea del agente. Sabiendo la dirección en 2D del objetivo y 8 direcciones en las que moverse debería ser casi trivial llegar hasta el objetivo.

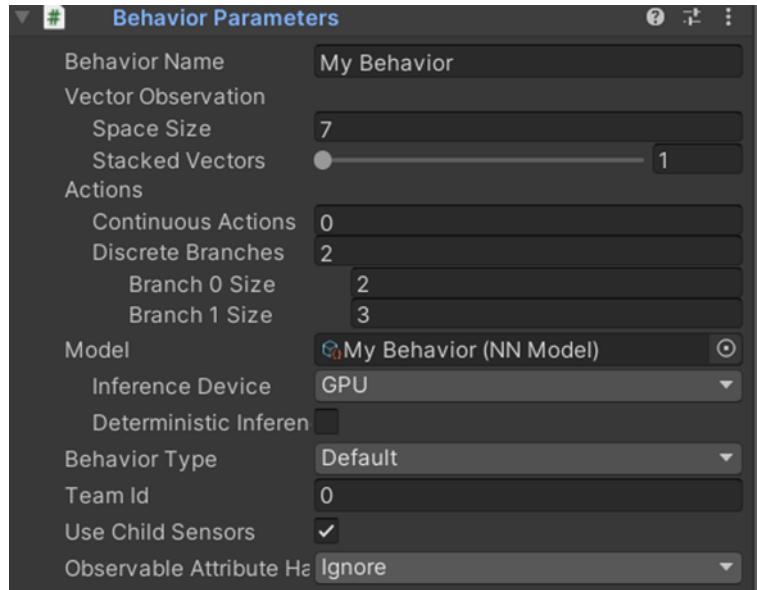


Figura 22: Componente *Behaviour Parameters* experimento 1

El primer test de este experimento lo llamaremos *BasicAgent1*. Una vez que pasamos a entrenar al agente nos damos cuenta de que algo está fallando, ya que después de más de 600 mil pasos sigue teniendo un comportamiento muy aleatorio. Los resultados de este experimento son muy malos, pero rápidamente rectificamos y pasamos a incluir un componente que nos dará mucha más información del entorno, y al ser un terreno tan amplio, el agente se ve escaso de información. Vamos a introducir en el siguiente punto el componente *RayPerception3D*.

El componente que hemos añadido mejora considerablemente la observaciones del agente. Estas nuevas observaciones son obtenidas mediante rayos, detectando a su paso los objetos que se le han indicado que puede detectar. Al detectar estos objetos, el agente obtendrá información que refuerza las que ya tiene. En la figura 23, vemos de manera visual cómo sería este componente cuando uno o más rayos colisionan con el objetivo.

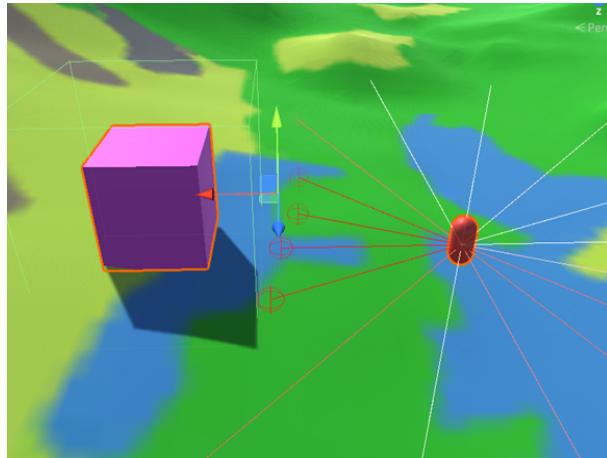


Figura 23: Ejemplo visual del componente *RayPerception3D*

Como primera característica de este componente, en la figura 24 vemos lo que acabamos de comentar, permite indicarle los *tags* de los objetos que queremos detectar, en nuestro caso advertimos del objetivo y de todo aquello que no tenga etiqueta. Del resto de parámetros vamos a explicar los tres más relevantes. El primero *Rays Per Direction* que dirá cuántos rayos generará el agente, en nuestro caso le indicamos que sean 8 rayos por dirección, en total serán 16. El segundo *Max Ray Degrees* que establecemos en 180, consiguiendo así una vista de 360 grados. Y por último *Ray Length*, que indica la longitud de cada rayo, y que después de probar varias longitudes consideramos que 20 m es adecuado, ya que el agente se moverá bastante lejos del objetivo en cierta ocasiones.

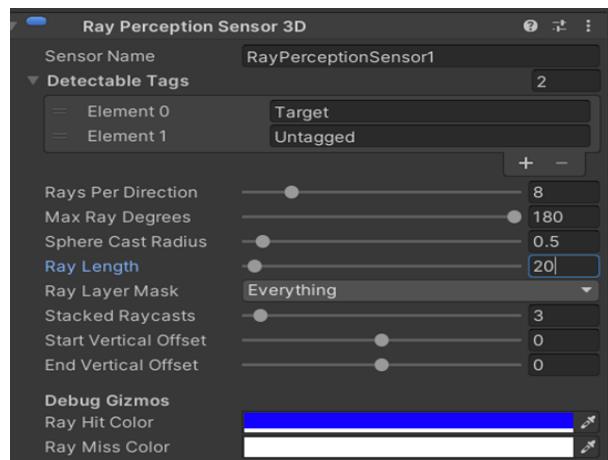


Figura 24: Componente *RayPerception3D*

Al hacer las primeras pruebas con los rayos ya incorporados al agente, nos dimos cuenta de otro problema debido a la irregularidad del terreno. Pongámonos

en la situación donde el objetivo está en una pendiente y el agente va subiendo esa pendiente, tal y como están situados los rayos, simulan que están en el pecho del humano, pues de esta manera estarían colisionando contra el terreno hasta que lo tenga justo delante, siendo esto un comportamiento poco realista. Una persona podría ver el objetivo si está a corta distancia, aunque fuese subiendo. De forma análoga pasaría si estuviera bajando, los rayos irían al aire hasta que lo tenga enfrente. En la figura 25 tenemos un claro ejemplo de lo que hemos descrito anteriormente.

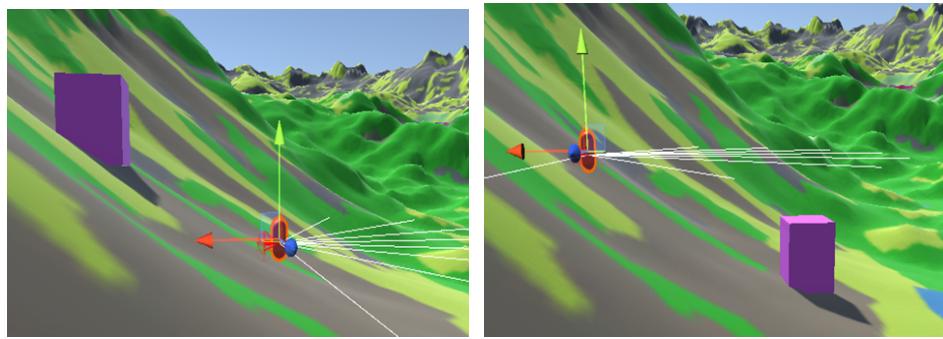


Figura 25: Ejemplo visual del problema del componente *RayPerception3D* con las alturas

Para solucionar este problema, hemos pensado en añadir dos componentes *RayPerception3D* más, pero situados en otra zona del agente y así tener un rango de visión más realista. Hemos decidido situar uno por encima de donde sería la cabeza del agente y otro por debajo de los pies. El mismo ejemplo que teníamos en la figura 25 lo vemos ahora en la figura 26, pero solucionando el problema de la visibilidad. Los rayos verdes son situados encima de la cabeza y detectan perfectamente cuando el objeto está subiendo la pendiente. De igual forma los rayos azules, situados por debajo de los pies, detectan el objetivo mientras baja la pendiente

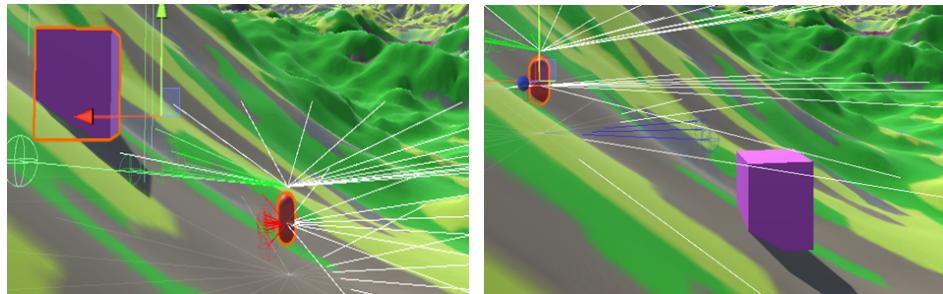


Figura 26: Ejemplo visual de la solución de la colisión de los rayos

Una vez bien configurados los rayos, procedemos con el segundo test, *BasicAgent2* obteniendo unos resultados muy buenos. A continuación se muestran las gráficas de la recompensa acumulada, la duración del episodio y la política/entropía.

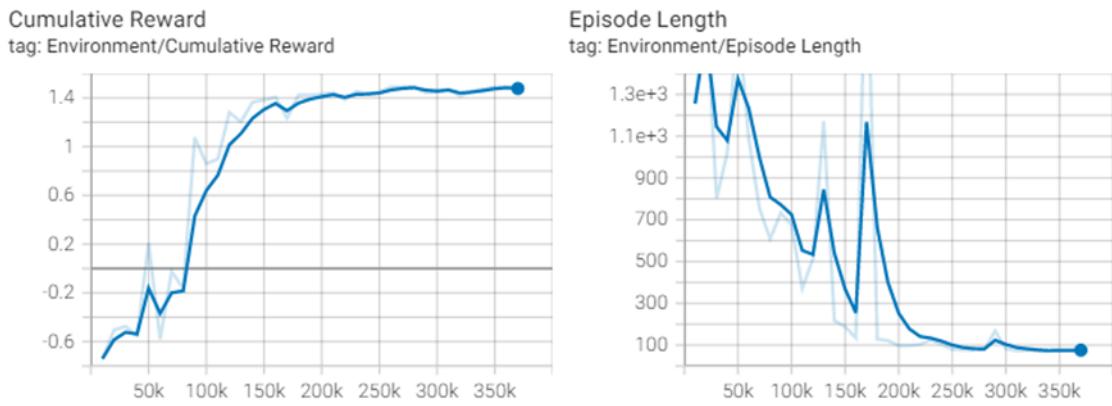


Figura 27: Gráficas de recompensa acumulada y duración del episodio *BasicAgent2*

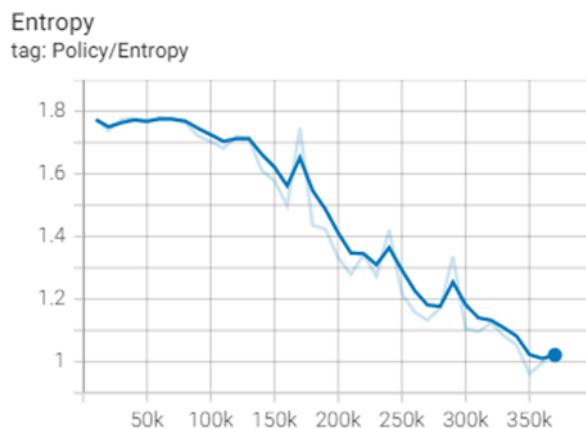


Figura 28: Grafica Policy/Entropy *BasicAgent2*

De la primera gráfica podemos abstraer el comportamiento aleatorio que tiene durante los primeros 70 mil pasos. A partir de este punto empieza a encontrar el objetivo cada vez más rápido, tal y como vemos en la gráfica que muestra la duración del episodio. En cuanto a la figura 28, vemos que la entropía disminuye, indicando que las decisiones del agente cada vez son menos aleatorios. Antes de pasar al siguiente experimento hemos decidido probar con la otra mecánica de movimiento implementada y ver si funcionaba igual de bien. Recorremos que en esta versión, el agente escoge una de las 8 direcciones para realizar

su movimiento. Tras casi 200 mil pasos al igual que antes consigue resultados igual de buenos. Decidimos continuar con esta mecánica ya que ofrece un movimiento un poco más realista y pasamos al siguiente experimento.

9.2. Movimiento del agente en función del tipo de terreno

El siguiente objetivo será que tenga en cuenta el tipo de terreno por el que camina. Contamos con una textura que indica por cada píxel (resolución 1m/píxel) el tipo de terreno que hay. En la figura 29 vemos el aspecto que tendrá el terreno. El *script* que usaremos para esta prueba se llamará *AgentWithClasses*.

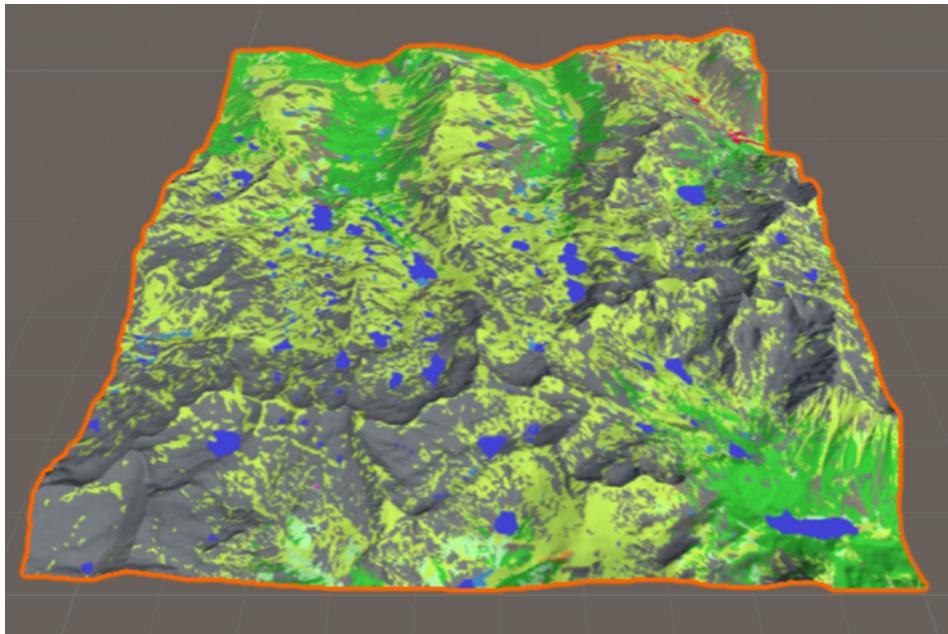


Figura 29: Ejemplo del terreno con la textura del tipo de terreno

Hay muchos tipos de terreno en esta textura, pero para simplificar un poco las cosas en esta primera prueba solo tendremos en cuenta el agua. Tendremos una matriz del tamaño del terreno que dirá para cada punto si hay agua o no.

En cuenta a las observaciones, mantendremos las mismas que teníamos en el experimento anterior más unas nuevas que van relacionadas con el tipo de terreno. Para tomar las muestras del terreno la idea será la siguiente, tendremos 8 direcciones, las cuales dividiremos en 4 intervalos. Para cada intervalo buscaremos si hay agua o no, en el caso de que haya la observación que se la pasará al agente será

un 1, por el contrario recibirá un 0. Con un total de 8 direcciones y 4 intervalos tenemos 32 nuevas observaciones, que sumadas a las 5 de antes contamos con 37 en total. Los cuatro intervalos serán 15m, 60m, 150m y 300m desde la posición del agente. Las muestras no serán tomadas igual en cada intervalo, para el primero tomaremos muestras cada píxel, para el segundo cada 2, para el tercero cada 4 y para el ultimo cada 8, dándole mayor importancia a lo que tiene más cerca.

Ya que hemos añadido el agua como nuevo elemento a tener en cuenta, también tendremos que penalizar al agente cuando pase por ella. A la tabla de refuerzos del experimento 1 le sumamos una nueva fila.

Acción	Recompensa/Castigo
Encontrar el objetivo	+1f
Salir del mapa	-1f
Alejarse más de la distancia permitida	-1f
Dar más pasos de los permitidos	$-distanceToTarget/SEARCH_RADIUS$
Dar un paso	$-1,0/MAX_WALK_STEPS$
Tocar el agua	-1f

Tabla 12: Tabla de refuerzos experimento 2

Esta primera prueba que haremos se llamará *WaterOnly* y no sale como esperamos, tenemos los resultados tras dos pruebas en la figura 30. Tras varios intentos fallidos por encontrar el error, nos dimos cuenta que el problema estaba en el fichero de configuración. Al haber escogido un fichero de otro problema y aunque este fuera bastante similar, había ciertos parámetros que no estaban del todo ajustados a nuestro problema. El más relevante es la beta, este controla como se reduce la entropía durante el entrenamiento. Este valor estaba más alto, lo que hacía que perdiera aleatoriedad muy rápido y no llegara a explorar muchas soluciones, quedándose estancado en una mala política de acciones. Al reducir beta, la entropía decrece más lentamente, y le permitimos durante más ciclos de entrenamiento ir tomando acciones aleatorias, por eso acaba descubriendo la buena estrategia.

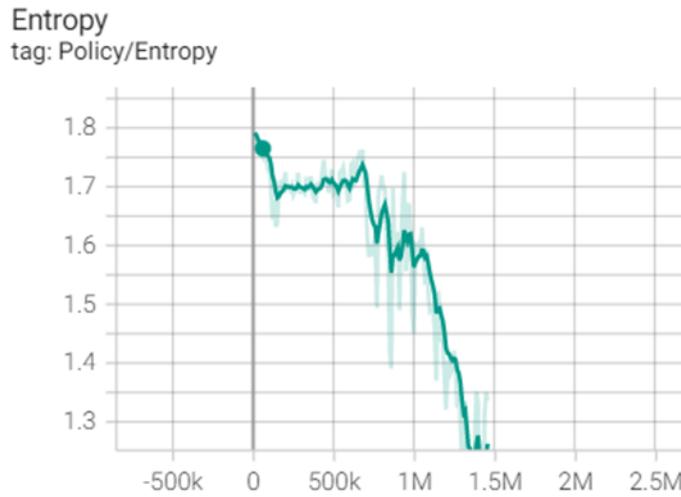
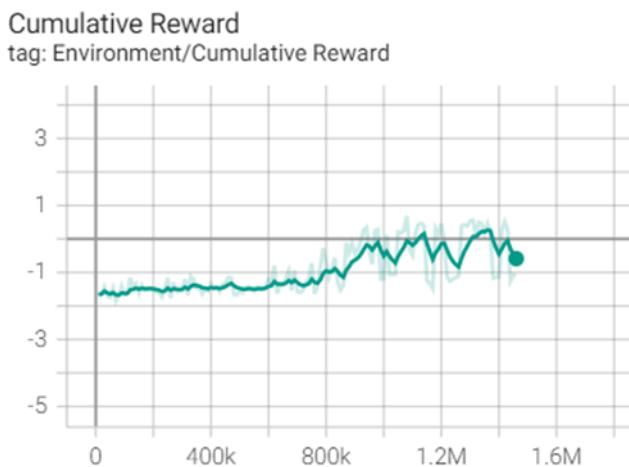
La entropía es una medida de cómo de aleatorias son las acciones del agente. La hemos ajustado a 0.004 de forma que fuera menos aleatorio que antes. Tras estos ligeros ajustes repetimos la prueba que se llamará *WaterOnly2*.



Figura 30: Gráficas de recompensa acumulada *WaterOnly*

Volvemos a repetir la prueba y la llamamos *WaterOnly2*. La figura 31 muestran como la entropía disminuye considerablemente, lo que indica que las decisiones que toma el agente son cada vez con más criterio. Y en la figura 32 observamos que la recompensa a pesar de no ser perfecta mejora mucho. El comportamiento es totalmente distinto, ya que las veces que es penalizado es por tardar demasiado y no por tocar el agua. Con más tiempo de entrenamiento la recompensa acumulada seguiría subiendo.

Hay una serie de mejoras que se podrían implementar en los siguientes experimentos. Vamos a explicar algunas situaciones que requieren una solución. Hay ocasiones donde el agente tarda mucho o directamente no puede llegar al objetivo. Esto pasa cuando está muy al borde del agua o cuando está al otro lado de un lago muy amplio. En este ultimo caso morirá siempre ya que supera el radio máximo que se puede alejar del objetivo o la distancia máxima que puede caminar, lo cual se soluciona fácilmente aumentando el radio máximo y el numero de pasos máximo que puede dar. El otro caso para cuando esta muy cerca del agua, se podría aumentar el numero de direcciones cuando toma las muestras. Hay situaciones donde la porción de terreno que no es agua es tan pequeña que muy difícilmente acierta a tomar muestras en esa dirección, por lo que siempre detecta que no puede pasar.

Figura 31: Gráfica de la entropía *WaterOnly2*Figura 32: Gráfica de la recompensa acumulada *WaterOnly2*

Para terminar, hemos pensado que quizás el hecho de tener mal ajustado el fichero de configuración fuera la razón por la que el entrenamiento sin rayos no funcionara, ya que al ser una tarea tan sencilla nos extrañó que no funcionara sin ellos. Para salir de dudas eliminamos los rayos y volvimos a entrenar el modelo consiguiendo unos resultados igual de buenos. Los rayos se ven que son una buena opción y que consigue acelerar el aprendizaje, y en nuestro caso compensar que nos equivocáramos en el fichero de configuración. Pero realmente los rayos tienen sentido cuando hay múltiples objetos que identificar, en nuestro caso con un único objeto en la escena aparte del agente no tenía mucho sentido. Por eso cuando la tarea se complicó con observaciones que no podían ser reforzadas por los rayos no funcionaba. A partir de ahora continuaremos sin los rayos.

9.3. Movimiento del agente en función del tipo de terreno (2)

Una vez que sabemos que el agente puede llegar a un objetivo de posición variable evitando el agua, decidimos aumentar la complejidad. Ahora no solo debe evitar el agua, si no que su velocidad se verá afectada en función del tipo de terreno que camine. Además, si el tipo de terreno es rocoso, habrá una probabilidad de que se tuerza el tobillo o incluso morirse por desprendimiento.

Sabiendo esto, las observaciones aumentarán bastante. Mantendremos igualmente la idea de las 8 direcciones y de los 4 intervalos, pero en este caso por cada intervalo tendremos 4 observaciones. Vamos a tener en cuenta el agua, roca, bosque denso u otra clase. Por cada intervalo mediremos con qué frecuencia aparece cada clase en esa porción de terreno, y por supuesto estas observaciones estarán normalizadas. Con todo esto ascendemos a tener $4^8 \cdot 4$ observaciones más las 5 iniciales, que son un total de 133.

Vamos a mostrar a continuación lo que será la nueva tabla de recompensas y penalizaciones y partir de ella explicaremos estas decisiones.

Acción	Recompensa/Castigo
Encontrar el objetivo	+1f
Salir del mapa	-1f
Alejarse más de la distancia permitida	-1f
Dar más pasos de los permitidos	$-distanceToTarget / SEARCH_RADIUS$
Dar un paso	$-1.0 / MAX_WALK_STEPS$
Tocar el agua	-1f
Torcedura de tobillo	-0.2f
Desprendimiento de una roca	-1f

Tabla 13: Tabla de refuerzos experimento 3

Estas dos nuevas situaciones implicarán un comportamiento diferente por parte del agente. En primer lugar vamos a tener una variable llamada *twistedAnkle*, que indicará si nuestro agente se ha torcido el tobillo. Esta posibilidad se habilitará cuando este caminando por terreno rocoso y tendrá una probabilidad de 0.0005, si es así sufrirá una penalización de 0.2 y estará con el tobillo torcido hasta que acabe el episodio. Esta torcedura también implicará moverse más lento, por lo que tardará más en llegar al objetivo. Ya de por sí, si circula por roca la velocidad se

reduce por 0.3, si le sumamos la torcedura la velocidad se reduce por 0.5. La penalización que se le da a cada paso es independiente de la distancia, por eso si avanza más lento tendrá que dar más pasos y obtendrá una recompensa menor al final del episodio. Así, indirectamente le estamos enseñando a evitar torcerse el tobillo.

Por otro lado, la posibilidad de desprendimiento surge de la misma manera, hay una cierta probabilidad, y en este caso de 0.00001. Es una probabilidad bastante baja debido a la cantidad de roca que hay en el mapa y si no terminaría por no entrar nunca, y no es eso lo queremos. Tratamos que circule por la roca, pero que sepa que no siempre es conveniente, queremos que haya un equilibrio entre saber cuando es necesario arriesgarse y cuando no.

Al igual que con la roca, circular por bosque denso supone una modificación en la velocidad de movimiento. Estas modificaciones afectan directamente sobre la recompensa, ya que irá más lento. Con todo esto trataremos de conseguir un comportamiento bastante completo, ya que habrá muchas cosas en juego. Un ejemplo claro de lo que buscamos puede ser decidir si le vale la pena cruzar un terreno rocoso y arriesgar a morir o torcerse el pie, con tal de no superar el número de pasos máximo o alejarse demasiado del objetivo. La primera prueba se llamará *AgentWith4Classes1*.

Tras casi 4 millones de pasos el agente ha adoptado un comportamiento muy interesante. Por un lado la lección de no pisar el agua la cumple a la perfección, lo cual indica que las observaciones las está interpretando bien. Por otro lado, la posibilidad de torcerse el pie o morir cuando esta caminando por roca también ha tenido efecto, pero quizás demasiado. El agente evita en la mayor parte de las veces adentrarse en terreno rocoso y esto hace que no haya aprendido a llegar al objetivo. No consideramos que haya sido un fracaso porque tiene un comportamiento totalmente lógico, ha aprendido a evitar zonas de roca que lo matan, al igual que el bosque denso porque lo retrasa. Para aprovechar esto y no perder el progreso, vamos a disminuir la distancia de partida del agente, y una vez haya conseguido una recompensa estable volveremos a la distancia original.

Después de reducir la distancia hemos retomado el entrenamiento desde donde lo habíamos dejado y ahora si que ha aprendido a llegar al objetivo. Ahora da igual la distancia desde la que parta que siempre llegará hasta el. A continuación (figura 33) vemos como la recompensa acumulada es bastante baja hasta que aprende a llegar al objetivo que es cuando hemos reducido la distancia. De todas formas el

agente está constantemente aprendiendo, lo podemos ver en la gráfica de la figura 34 que muestra como disminuye la entropía a lo largo del entrenamiento. La bajada tan brusca en la figura 34 a los 2 millones de pasos no es más que un error durante el entrenamiento, hemos parado el entrenamiento un momento y lo hemos vuelto a reanudar con error en el código, pero rápidamente ha sido corregido y la gráfica volvió a la normalidad.



Figura 33: Gráfica de la recompensa acumulada *AgentWith4Classes1*

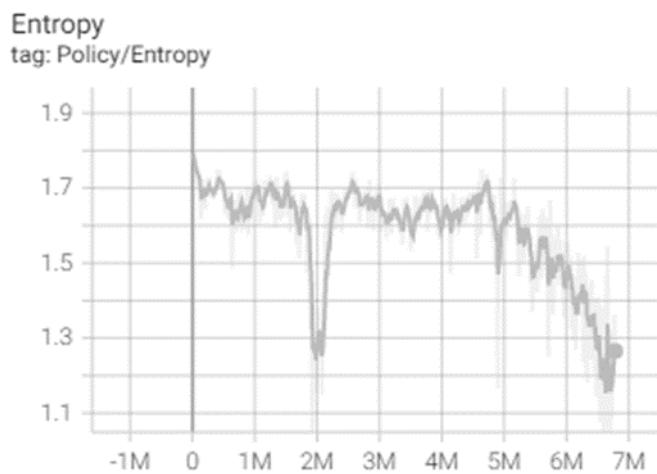


Figura 34: Gráfica de la entropía *AgentWith4Classes1*

Hemos demostrado que el agente es capaz de adoptar un comportamiento u otro en función de las respuestas que le demos. En este caso ha aprendido que la roca puede perjudicarle mucho y por lo tanto no la frecuenta mucho, al igual que

el agua. En este experimento no tuvimos en cuenta cada una de las zonas que hay en el mapa, y simplemente establecimos que todas las zonas que no fueran agua, roca o bosque denso eran las “seguras”. Para enriquecer un poco más esta versión, vamos a incorporar una nueva clase de terreno. Haremos que las zonas urbanas y de agricultura, que son zonas asfaltadas o caminos, en ellas el agente pueda aumentar su velocidad, simulando que puede correr. De esta manera intentamos que el agente aprenda que estas zonas son ventajosas y seguras. Para forzar aún más este comportamiento nos trasladaremos a una zona del mapa donde existen cantidad de caminos de este tipo, y por si fuera poco colocaremos al objetivo sobre esos caminos. El objetivo es que el agente aprenda a seguir un camino. Esta prueba la llamaremos *AgentWith5Classes*. La nueva zona del mapa la vemos en la figura 35, y la superficies rojas y naranjas son las nuevas zonas que tendremos en cuenta.

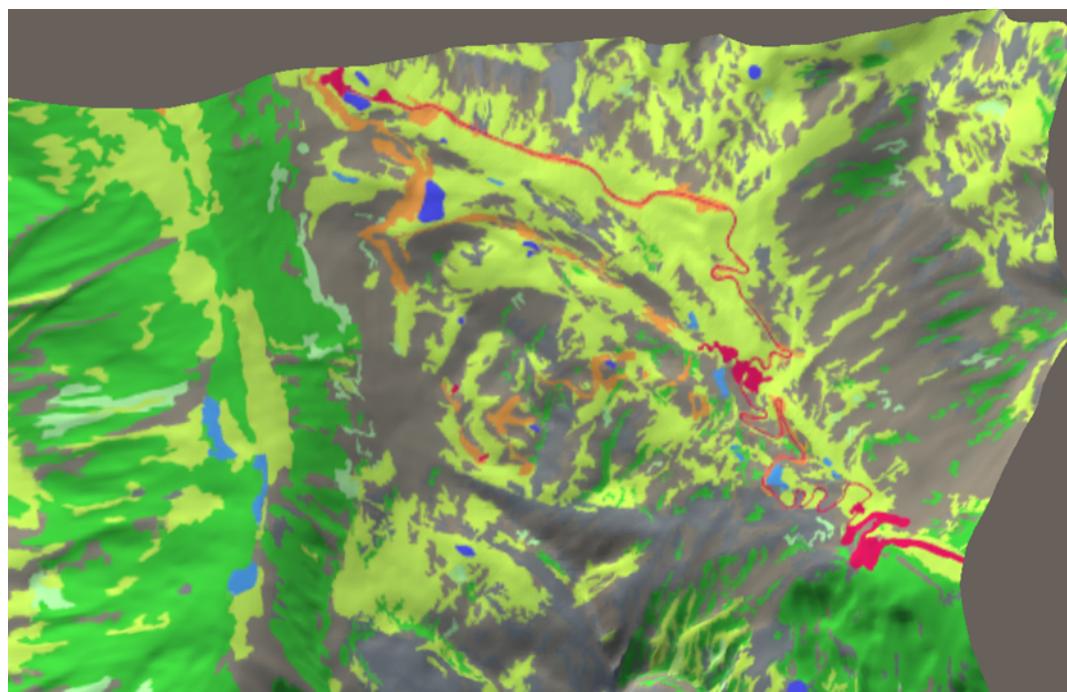


Figura 35: Nueva zona del mapa para el experimento *AgentWith5Classes1*



Figura 36: Gráfica de la recompensa acumulada *AgentWith5Classes1*

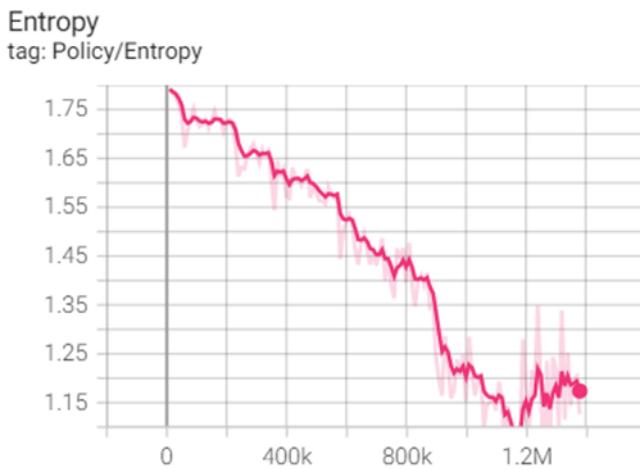


Figura 37: Gráfica de la entropía *AgentWith5Classes1*

Las figuras 36 y 37 muestran los buenos resultados nuevamente. El agente ha aprendido que los caminos son la mejor opción para llegar hasta el objetivo. Después del millón de pasos hemos parado el entrenamiento para cambiar la posición inicial del objetivo y que no solo estuviera en zonas asfaltadas, y ver que hacía cuando tenía que salir de los caminos para llegar al objetivo. Al hacer esto la recompensa bajó, tal y como se ve en la gráfica de la figura 36. Al principio cuando el objetivo estaba en otras zonas que no fuera las urbanas le costaba llegar, ya que se quedaba recorriendo las zonas rojas a ver si encontraba una conexión. Después de un tiempo empezaba a explorar y aprendía a salir de esas zonas encontrando el objetivo, pero siempre primero buscando una ruta por zona urbana. Vemos como después de un tiempo vuelve a subir la recompensa, y lo mismo pasa con la

entropía, hay un momento que no sabe muy bien que está pasando, y es lo que vemos reflejado en la figura 37 cuando la entropía baja y luego vuelve subir.

Con más tiempo de entrenamiento, esta última versión en la que hemos incluido las zonas urbanas mejoraría bastante. Vamos a ver a continuación como podemos mejorarla teniendo en cuenta otros aspectos del terreno.

9.4. Movimiento del agente en función de la forma del terreno

Nuestro próximo objetivo será incorporar dos nuevos criterios que debe tener en cuenta el agente en su navegación. Por un lado vamos a tener en cuenta la posibilidad de caerse en zonas estrechas con mucha caída. Y por otro lado también buscaremos que el agente optimice sus movimientos en base a la inclinación. Para facilitar el testeo de esta idea vamos dejar de lado el tipo de terreno y centrarnos en estos dos nuevos criterios. El *script* para este experimento se llamará *AgentInclination*.

Para modelar esta idea contamos en primer lugar con una textura llamada *slope* en formato .png de 8 bits proporcionada por el tutor, indicando un valor de 255 una pendiente de 90 grados y 0 una superficie completamente llana. De la misma forma tenemos otra textura llamada *exposure* que indica de la misma forma que *slope* la peligrosidad de la zona. Corresponde para cada *píxel* la caída de mayor altura en una ventana cuadrada de 20x20, teniendo una caída menor de 2m valor cero y una de 20m o más valor 255. Zonas estrechas y altas tendrán valores muy altos.

En primer lugar vamos a trabajar con los datos de la textura *exposure*, y si el entrenamiento es exitoso pasaremos a incorporar los datos sobre la inclinación. Así podremos debugar los errores con mayor facilidad. Esta primera prueba la llamaremos *exposure1*.

Para empezar hemos normalizado los datos entre 0 y 1 para mantener la estabilidad en las observaciones. La forma de tomar las observaciones no varía y seguimos manteniendo las 8 direcciones y los 4 intervalos, pero en este caso en cada intervalo vamos a tomar el valor de máxima exposición. A partir de ahora cuando hablemos de zona con mucha exposición estaremos refiriéndonos a zonas

altas y peligrosas. Por tanto tendremos 37 observaciones al igual que en el segundo experimento donde solo teníamos en cuenta el agua.

Para que el agente detecte cuales son las zonas peligrosas, vamos a establecer dos umbrales a partir de los cuales tendrá consecuencias o no cuando navegue por esas zonas. El primer umbral lo hemos fijado en 0.2 y para valores menores a 0.2 no habrá posibilidad de morir por caída. El otro umbral ha sido fijado en 0.8, donde inevitablemente morirá cuando pase por zonas con valores iguales o mayores a 0.8. Para valores intermedios hemos calculado la probabilidad de morir de la siguiente manera:

$$prob = (actualExposure - 0,2f)/0,8f;$$

$$prob = 0,001f * prob^2;$$

Para hacernos una idea de lo que consideramos zonas peligrosas, tenemos a continuación en la figura 38 un ejemplo de la textura *exposure* donde el color rojo indica una zona con mucho peligro. Obviamente tanto cuando pase por zonas con valor igual o mayor a 0.8 y tanto cuando muera con esa cierta probabilidad, penalizaremos al agente por morir. La tabla con las recompensas y castigos la tenemos a continuación (tabla 14).

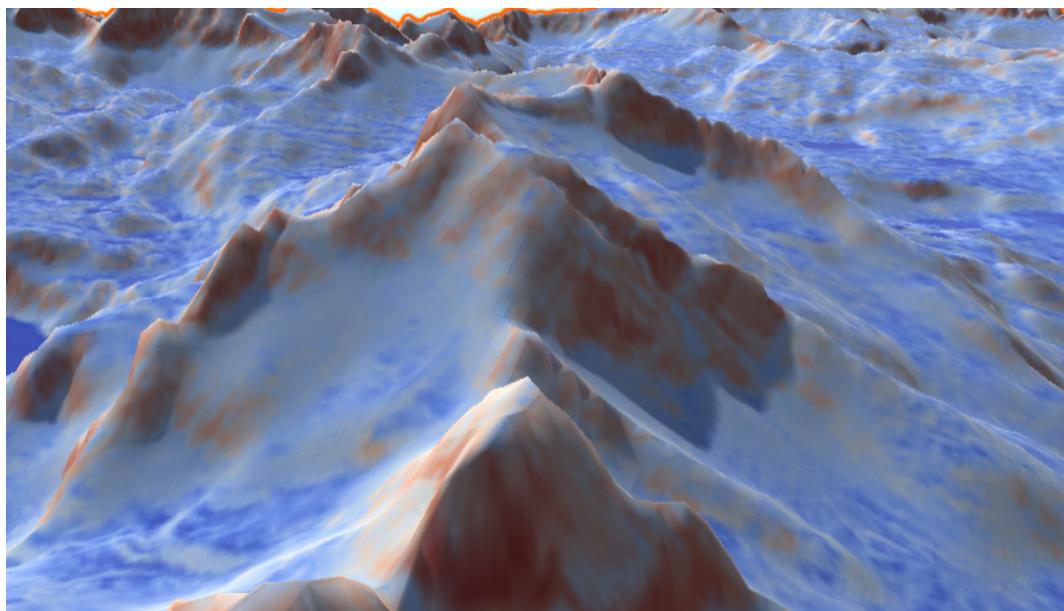


Figura 38: Ejemplo de una zona peligrosa de la textura *exposure*. El color rojo indica peligro.

Acción	Recompensa/Castigo
Encontrar el objetivo	+1f
Salir del mapa	-1f
Alejarse más de la distancia permitida	-1f
Dar más pasos de los permitidos	$-distanceToTarget/SEARCH_RADIUS$
Dar un paso	$-1.0/MAX_WALK_STEPS$
Muerte por caída	-1f

Tabla 14: Tabla de refuerzos experimento 4

Los resultados de esta prueba son muy buenos, cumpliendo a la perfección con los objetivos. Hemos seguido la estrategia seguida en experimentos anteriores en cuanto acercar al agente al objetivo durante los primeros episodios y una vez ha aprendido esa tarea lo alejamos y que siga mejorando sus aprendizaje. De esta forma aceleramos el aprendizaje de manera considerable. Los buenos resultados los vemos reflejados en la figura 39 y 40.

Figura 39: Gráfica de la recompensa acumulada *exposure1*

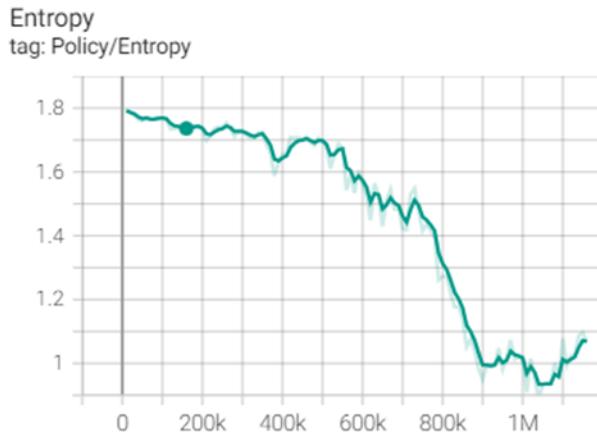


Figura 40: Gráfica de la entropía *exposure1*

Después de los buenos resultados vamos a incorporar ahora el factor de la inclinación. Esta será un multiplicador de lo que ya restamos a cada paso que da. Si recordamos en la tabla de recompensas y penalizaciones del experimento 1 restábamos $-1,0/MAX_WALK_STEPS$, ahora añadiremos un multiplicador a esta penalización en función de la inclinación del terreno de la siguiente forma:

$$t = \max(0, (actualInclination - 20f)/(90f - 20f));$$

$$mult = 1 + t^2;$$

$$AddReward((-1,0f/MAX_WALK_STEPS) * (mult));$$

Para las observaciones, además de coger el punto de máxima exposición por cada dirección e intervalo, cogeremos el punto de máxima inclinación, teniendo así 69 observaciones. Esta prueba se llamará *exposureInclination1*.

Tras varias pruebas y muchas horas de entrenamiento el resultado no es el esperado. Las observaciones han funcionado bien, ya que el agente sigue evitando esas zonas de muchas exposición al igual que en la prueba anterior. De la misma forma hace con las subidas y bajadas con mucha pendiente tratando siempre de evitarlas. Si nos fijamos en la gráfica de la figura 41, la cual muestra la entropía, hay una cosa muy curiosa y es que hasta en tres ocasiones la entropía baja considerablemente. Esto significa que el agente encontraba una estabilidad en su aprendizaje

y sus decisiones, pero algo pasa que hace que vuelva a subir y pierda gran parte del progreso. Esto también nos lo confirma la gráfica de la figura 42 que muestra la *policy loss*. Esta gráfica indica cuánto está cambiando la política (proceso para decidir acciones), y debería disminuir durante una sesión de entrenamiento exitosa. Si nos fijamos, en los puntos donde la entropía disminuye, la *policy loss* también disminuye.

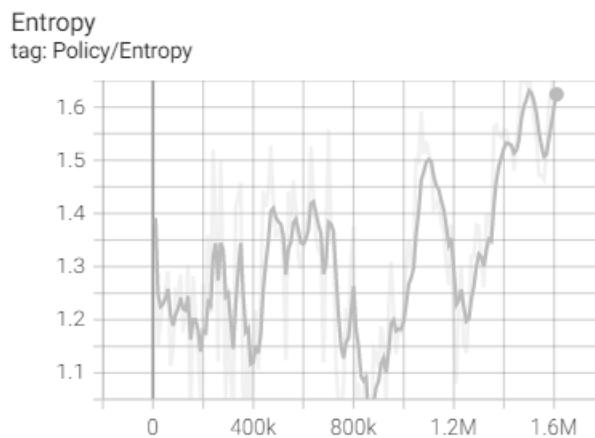


Figura 41: Gráfica de la entropía *exposureInclination1*



Figura 42: Gráfica *policy loss exposureInclination1*

Estos cambios continuos en la política se deben a un mal ajuste de las recompensas y penalizaciones. Como ya hemos visto en otros experimentos, lo que más le cuesta al agente es aprender a llegar al objetivo. Y en ese tiempo que navega por el terreno sin saber qué tiene que hacer, aprende a otras cosas, como es evitar las zonas peligrosas y las pendientes inclinadas en este caso. Pero el problema viene cuando aprende que su misión es llegar al objetivo, pues esa recompensa que

obtiene al llegar a él es muy superior a lo que perdería evitando pendientes, por lo que empieza darle igual la inclinación. Sin embargo, las zonas de mucha exposición las sigue evitando, ya que en esas zonas ha aprendido que puede morir y recibir una penalización considerable.

Por cuestiones de tiempo vamos a quedarnos con la primera prueba y seguir adelante. El siguiente y ultimo experimento será recopilar todo lo visto hasta ahora y condensarlo en un único agente.

9.5. Agente final

Después de implementar varias funcionalidades por separado, es hora de unificarlas todas en un único agente tal y como habíamos previsto. De todo lo que hemos implementado lo único que no vamos a incorporar en esta ultima versión serán las observaciones relacionadas con la inclinación. Tal y como explicamos en el experimento anterior, por cuestiones de tiempo no nos ha dado tiempo de intentar identificar y solucionar los problemas que impedían el aprendizaje en función de la inclinación.

Para esta última versión no hay nada nuevo que no hayamos explicado ya. Lo único que cambia es el número de observaciones, que como es lógica aumenta. Vamos a realizar un desglose de todas las observaciones para este último entrenamiento (para las observaciones relacionadas con el tipo de terreno, tendremos en cuenta las 5 clases usadas para el último experimento visto en el punto 9.3 llamado *AgentWith5Classes1*) :

- Observaciones relacionadas con el objetivo: $2+2+1 = 5$ observaciones
- Observaciones relacionadas con el tipo de terreno: $4 * 8 * NUM_CLASES = 165$ observaciones
- Observaciones relacionadas con la peligrosidad del terreno: $4 * 8 = 32$ observaciones

En total tenemos 197 observaciones para el ultimo experimento. Para los recompensas y penalizaciones haremos lo mismo, juntaremos la lógica de cada una de las pruebas. La tabla 15 muestra todo esto.

Acción	Recompensa/Castigo
Encontrar el objetivo	+1f
Salir del mapa	-1f
Alejarse más de la distancia permitida	-1f
Dar más pasos de los permitidos	$-distanceToTarget/SEARCH_RADIUS$
Dar un paso	$-1.0/MAX_WALK_STEPS$
Tocar el agua	-1f
Torcedura de tobillo	-0.2f
Desprendimiento de una roca	-1f
Muerte por caída	-1f

Tabla 15: Tabla de refuerzos experimento 5

Después de 3 millones de pasos, el entrenamiento ha sido exitoso. Temíamos que al juntar todo en un único agente con tantas observaciones, pudiera confundirse y no aprender nada. No ha sido así, y pesar que la gráfica que muestra la recompensa acumulada (figura 43) no sea del todo buena, con más tiempo de entrenamiento la recompensa se estabilizaría. Aún así se observa claramente como en los 2,5 millones de pasos, el agente empieza a acumular cada vez más recompensa. Esto lo confirma la gráfica de la figura 44 *Losses/Value Loss*, que si recordamos lo explicado en el punto 2.2.3 esta gráfica muestra qué tan bien el modelo puede predecir el valor de cada estado y debe aumentar mientras el agente está aprendiendo y luego disminuir una vez que la recompensa se estabilice. Pues poco después de los 2,5 millones de pasos, empieza a bajar considerablemente.

Figura 43: Gráfica de la recompensa acumulada *agentComplete*



Figura 44: Gráfica *Losses/Value Loss agentComplete*

Capítulo 10

Leyes y regulaciones

Este proyecto se rige por la Normativa del trabajo final de grado del grado en Ingeniería Informática de la Facultad de Informática de Barcelona (21) aprobada por la Comisión Permanente en fecha 19/09/2012.

La propiedad industrial e intelectual de este Trabajo de Fin de Grado queda regulada por esta normativa, por la que se aprueba la confidencialidad, responsabilidad y propiedad industrial e intelectual en la Universidad Politécnica de Cataluña. También se hará público en a través del servicio de bibliotecas.

El proyecto se basa en gran parte en *Unity3D*, un software que dicho uso queda bajo la condicionadas firmadas al registrarnos como usuarios del mismo, la cuales podemos encontrar en (22). *Unity* otorga una licencia limitada, no exclusiva, intransferible, sin derecho de conceder su licencia a terceros, para acceder, ver, descargar e imprimir cualquier contenido solamente para su personal y sin fines de lucro. Por otro lado se reconoce y acepta que toda la información, datos, texto u otros materiales, ya sea que se publiquen o se transmitan de forma privada, son responsabilidad exclusiva de la persona de quien se originó el contenido del usuario. Al poner a disposición cualquier contenido a través del sitio y los servicios, otorgamos a *Unity* una licencia de alcance mundial, irrevocable

Por otro lado, hemos hecho uso de muchos datos proporcionados por el Institut Cartogràfic i Geològic de Catalunya (ICGC). Este establece unas condiciones de uso en (23) que permiten fomentar la reutilización de la información geográfica, geológica y geotemática, especialmente la reproducción y divulgación por cualquier medio y la creación de productos o servicios de información con valor añadido basados en estos datos. Tanto para modelar el terreno como para hacer uso de los mapas de cubiertas y que hemos usado como texturas, hemos seguido los

términos establecidos para la reutilización de la información basados en la licencia conocida como CC BY 4.0 (24). Al hacer uso de ella somos libres de compartir y adaptar todo este material, pero bajo el término de reconocimiento. Se debe reconocer la autoría de forma apropiada, proporcionar un enlace a la licencia e indicar si ha realizado algún cambio.

Capítulo 11

Conclusiones

Como broche final a este proyecto debemos hacer autocrítica de los resultados y de lo aprendido a lo largo de la investigación. Haremos énfasis en las dificultades que han hecho que los resultados no sean del todo satisfactorios, buscando cuales pueden ser las claves para que esta investigación continúe.

En cuanto a lo personal, desde un principio afrontar este proyecto suponía un gran reto debido al desconocimiento de esta tecnología. Pero el gran interés por innovar y explorar en lo desconocido ha hecho que el trabajo se haya llevado a cabo con mucha dedicación y esmero.

En primer lugar algo que ya teníamos previsto en cierta manera son las limitaciones de tiempo y de recursos. Las investigaciones de RL requieren de mucho tiempo y de muchos recursos tratando de buscar la configuración adecuada para la tarea a completar por el agente. Obviamente no es comparable la investigación realizada con una a gran escala, pero si que influye y mucho la efectividad y la buena toma decisiones si tenemos que cumplir unos plazos. Una prueba fallida implica casi un día perdido, y mucho tiempo de formación en mi caso para identificar cual es el problema. Todo esto sumado a las limitaciones de nuestro PC hacen que cada mala decisión influya mucho. Como ya se comentó en la investigación, el mal ajuste de los parámetros en el fichero de configuración supuso mucho tiempo de entrenamiento y de análisis perdido, quizás si hubiéramos detectado previamente este fallo hubiéramos podido implementar más funcionalidades y cumplir con la programación prevista.

Sin embargo, se ha llegado a crear una versión con mucho potencial y que puede ser una herramienta complementaria a muchos proyectos. No se considera que

haya sido un fracaso ni mucho menos, pero si que hay aspectos que mejorar de cara a próximas investigaciones.

Todos los *scripts*, texturas y modelos usados en este proyecto se encuentran en el siguiente repositorio: <https://github.com/JesusBD11/TFG>.

Capítulo 12

Lineas Futuras

Una de las principales ideas que había alrededor de este proyecto era introducir elementos para diferenciarlo de algo que pudiera resolverse con un algoritmo de *pathfinding*. En un primer momento se puede pensar que resolverlo de esta manera puede simplificar mucho, pero la existencia de eventos aleatorios es lo que marca la diferencia. Por eso consideramos que una combinación de ambos puede ser una buena idea de cara al futuro. Teniendo un punto de partida y otro de llegada, con un *pathfinding* tendríamos una ruta ideal, pero sin tener en cuenta ciertos aspectos que puedan mejorar este camino. Pues la idea es subdividir esa ruta en secciones, y que en cada sección apliquemos el conocimiento del agente para conseguir una navegación más realista. De esta forma, quizás el agente encuentre una alternativa que le interese más a la persona que va a realizar esa ruta.

Obviamente una serie de mejoras en cuanto a eventos aleatorios es clave para que la aplicación cobre más valor. Por falta de tiempo no hemos podido probar otro tipo de mapas donde hubieran más tipos de terreno. Teniendo otros modelos con más variedad algunas de las mejoras podrían ser:

- Probabilidad de quedarse atrapado en terrenos pantanosos.
- Los cambios de rasante convexos son más peligrosos que los cóncavos (a misma pendiente) debido a que en los convexos la nieve tiende a estirarse y es más inestable.
- Probabilidad de resbalón en barro, roca mojada, hierba mojada, nieve.
- Probabilidad de avalancha.

Teniendo en cuenta aspectos de este tipo, nuestra aplicación sería capaz de encontrar senderos que se adapten a cada persona. Una persona puede interesarle rutas a través de rocas, bosques y así ahorrar tiempo. Sabiendo esto, podríamos entrenar al agente de forma distinta a la que hemos hecho, simplemente cambiando el balance de prioridades podemos conseguir que priorice estas superficies.

Otro aspecto a mejorar sería la simulación gráfica. En un principio dejamos claro que este aspecto no era lo primordial, pero una vez conseguidos ciertos objetivos podríamos meternos de lleno a mejorar cuestiones visuales.

Apéndice A

Diagrama de Gantt Inicial

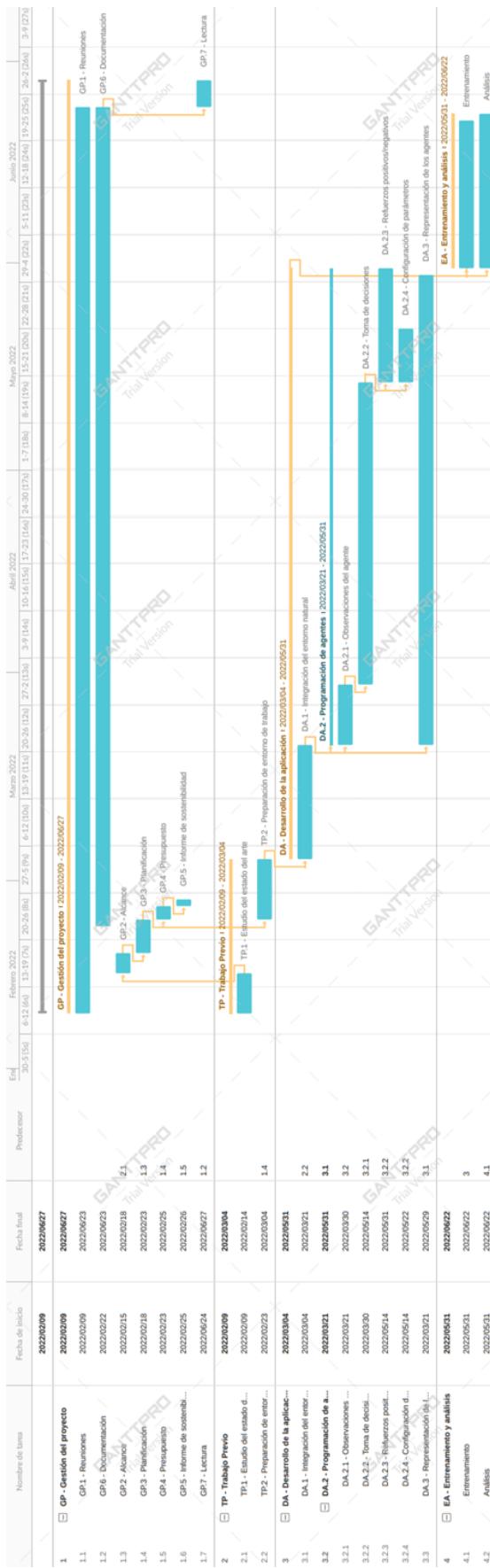


Figura 45: Diagrama de Gantt final

Apéndice B

Diagrama de Gantt Final

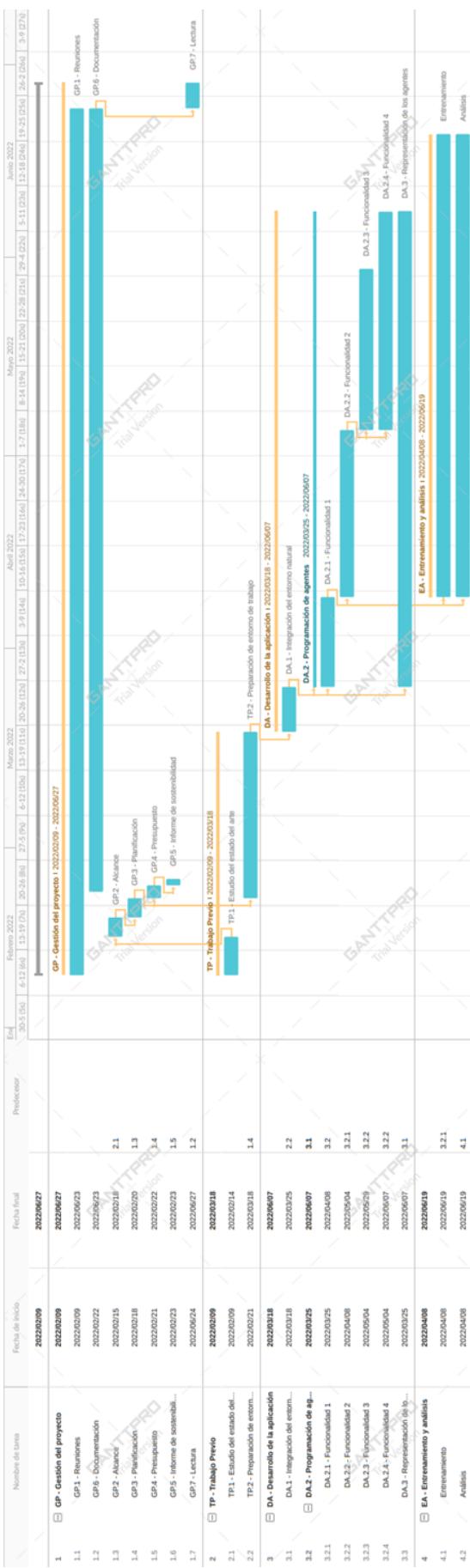


Figura 46: Diagrama de Gantt final

Bibliografía

- [1] VERMA, P y DIAMANTIDIS, S. *What is Reinforcement Learning? – Overview of How it Works*, 2021, April 27. <http://www.ics.uci.edu>.
- [2] *¿Qué es y qué tareas cumple un agente inteligente?* Wikipedia [https://es.wikipedia.org/wiki/Agente_inteligente_\(inteligencia_artificial\)](https://es.wikipedia.org/wiki/Agente_inteligente_(inteligencia_artificial))
- [3] *NPC* Wikipedia https://es.wikipedia.org/wiki/Personaje_no_jugador
- [4] IFP. 2022, MAY 25 *¿Qué es Unity y para qué puedo utilizarlo?* <https://www.ifp.es/blog/que-es-unity-y-para-que-puedo-utilizarlo>
- [5] *¿Qué es ML-Agents?* <https://github.com/Unity-Technologies/ml-agents>
- [6] *Todo lo que necesitas saber sobre TensorFlow* <https://acortar.link/XUW4h3>
- [7] *TensorBoard* <https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Using-Tensorboard.md>
- [8] *Requisito no funcional* Wikipedia https://es.wikipedia.org/wiki/Requisito_no_funcional
- [9] TRÍAS, C. *¿Qué es la metodología Agile y por qué está de moda?* Progressa Lean, 2020, May 7. <https://www.progressalean.com/metodologia-agile/>.
- [10] *¿Qué es GitLab?* Wikipedia <https://ca.wikipedia.org/wiki/GitLab>
- [11] *GanttPRO* <https://ganttpro.com/es/>
- [12] *LaTeX* <https://es.wikipedia.org/wiki/LaTeX>

- [13] *Precio espacio coworking* <https://coworkidea.com/ca/preus/>
- [14] FUENTES, M y GRAUSCHOPF, S. *Using the Terrain Tools in Unity.* 2021, October 13. <https://www.raywenderlich.com/21911888-using-the-terrain-tools-in-unity>.
- [15] *Institut Cartogràfic i Geològic de Catalunya (ICGC)* <https://www.icgc.cat/es/Descargas>
- [16] *Curso avanzado de ML-Agents de UNITY* <https://cursos.uadla.com/curso/curso-intermedio-de-ml-agents/>
- [17] SANZ, M. *Aprendizaje por refuerzo: políticas de gradiente.* 2020, November 24. <https://markelsanz14.medium.com/introducci%C3%A9n-al-aprendizaje-por-refuerzo-parte-5-pol%C3%ADticas-de-gradiente-8e92725e9c8f>.
- [18] *SAC - Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.* 2018, January 4. <https://arxiv.org/abs/1801.01290>.
- [19] *Proximal Policy Optimization Algorithms.* 2017, July 20. <https://arxiv.org/abs/1707.06347>.
- [20] KIM, T. *Understanding Proximal Policy Optimization.* 2021, May 5. <https://acortar.link/r7efri>.
- [21] *Normativa del Treball de Fi de Grau —Facultad d'Informàtica de Barcelona* <https://www.fib.upc.edu/sites/fib/files/documents/estudis/normativa-tfg-mencio-addicional-gei-br.pdf>
- [22] *Política de privacidad Unity3D* <https://unity3d.com/es/legal/terms-of-use>
- [23] *Condicions d'ús de la geoinformació ICGC* <https://www.icgc.cat/ca/L-ICGC/Informacio-publica/Transparencia/Reutilitzacio-de-la-informacio>
- [24] *Reconeixement 4.0 Internacional (CC BY 4.0)* <https://creativecommons.org/licenses/by/4.0/deed.ca>