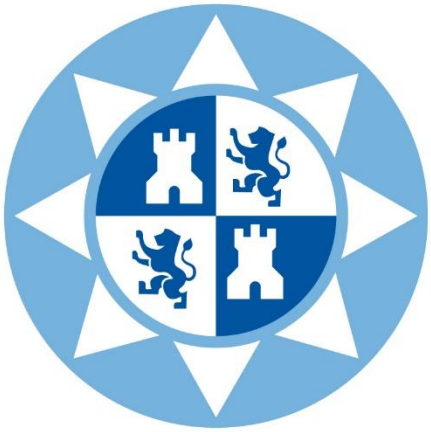

Definición y estudio de estrategias de inteligencia artificial de los personajes de un juego RPG desarrollado con Unity.

TRABAJO FINAL DE GRADO:

Octubre 2018

AUTOR:

Ángel Ruiz Rodríguez

DIRECTOR:

Juan Ángel Pastor Franco

etsit
escuela técnica superior de
ingeniería de telecomunicación

Autor	Ángel Ruiz Rodríguez
E-mail del Autor	angelruizrod@gmail.com
Director	Juan Ángel Pastor Franco
E-mail del director	juanangel.pastor@upct.es
Título del TFG	Definición y estudio de estrategias de inteligencia artificial de los personajes de un juego RPG desarrollado con Unity.
Descriptores	Unity 3D, Inteligencia Artificial, Videojuego

Resumen

El TFG se desarrolla sobre un juego RPG ya definido (juego base) y consta de tres tareas principales, que se corresponden con sus fases: (1) estudio general de las estrategias de IA que se utilizan en los juegos RPG para definir los comportamientos de los personajes, (2) selección e implementación de un conjunto de estrategias (conjuntos de estrategias asignables a los personajes), y (3) prueba de diferentes asignaciones de comportamientos hasta obtener una que sea óptima para la jugabilidad. El estudio y prueba de las estrategias tiene dos propósitos, el primero es dotar a los personajes de la mayor capacidad de decisión posible, el segundo es determinar el nivel de inteligencia que hace que el juego sea atractivo para un jugador humano, dependiendo de su grado de maestría.

Titulación	Grado en Ingeniería Telemática
Departamento	Tecnologías de la Información y las Comunicaciones.
Fecha de presentación	Octubre 2018

Índice General

Índice de figuras

1.Introducción.

1.1. Motivación.

La creación de este videojuego surge de las siguientes fuentes:

- La opción de ser parte de un proyecto grande, constando este de tres proyectos fin de grado, convirtiéndose así, en un proyecto más llamativo, innovador e interesante.

- La oportunidad de ponerme a prueba trabajando en un entorno a priori desconocido partiendo solo de mis conocimientos generales de programación.

- Finalmente, como amante de los videojuegos que soy desde pequeño, el hecho de ser parte de la creación de nuestro propio videojuego desde cero es mi mayor motivación. Poder plasmar mis gustos y los de mis compañeros en un proyecto común, y compartirlo para que la gente pueda valorar nuestro trabajo, recibir un feedback[1] para poder mejorarlo y seguir creciendo junto a él.

1.2. Objetivo.

El objetivo de este proyecto es investigar los distintos tipos de inteligencia artificial más usados en el ámbito de los videojuegos e implementarlas en un videojuego RPG[2] destinado a la plataforma móvil.

El género RPG, como sus propias siglas indican (Role-Playing Game) es la adaptación de los juegos clásicos de rol como *Dungeon & Dragons*, donde los jugadores tenían que interpretar un papel (mago, guerrero, vendedor…), en el mundo de los videojuegos.

Dentro de la plataforma móvil, el entorno seleccionado ha sido Android, debido a la gran cantidad de dispositivos móviles que lo utilizan, la facilidad para publicar el trabajo realizado en la Play Store de Google y el pequeño desembolso económico requerido para hacerlo.

En este documento, se detallará la investigación realizada junto a las herramientas e implementaciones llevadas a cabo. Para cada tipo de inteligencia artificial, primero se realizará una implementación en un entorno de prueba y luego, se añadirá al juego si ese tipo de IA es adecuada para él.

[1] FeedBack: Retroalimentación obtenida de los jugadores para mejorar la experiencia de juego.
[2] RPG (Role Playing Game): adaptación de los juegos de rol al mundo de los videojuegos.

Este proyecto se apoya en el proyecto “Desarrollo de un juego RPG para prueba de algoritmos de IA con plataforma Unity e integración con la nube” tomando de este su entorno (mapa que contiene una ciudad) para la implementación de algunas inteligencias artificiales.

1.3. Estructura de la memoria.

La memoria del proyecto sigue la siguiente estructura:

- Capítulo 2: En este capítulo se resume sobre la historia de los videojuegos y el género “RPG”.

- Capítulo 3: En este capítulo se explica qué es una IA (Inteligencia Artificial), comentando clasificaciones, diferentes metodologías para llevar a cabo su implementación, sus aplicaciones y la evolución de las inteligencias artificiales en el ámbito de los videojuegos.

- Capítulo 4: Se hablará sobre las distintas herramientas y tecnologías usadas para realizar este trabajo de fin de grado.

- Capítulo 5: Capítulo destinado a explicar la implementación de un entorno de pruebas básico.

- Capítulo 6: Capítulo destinado a la implementación de diferentes tipos de inteligencia artificial.

- Capítulo 7: Capítulo enfocado en la configuración de las herramientas utilizadas y explicación de algunas técnicas seguidas para la implementación de las IAs.

- Capítulo 8: Capítulo destinado a comentar las conclusiones obtenidas de la realización de este proyecto y el trabajo pendiente por realizar.

- Capítulo 9: Glosario donde se pueden distintas definiciones de términos utilizados en la redacción de este documento.

- Capítulo 10: Bibliografía.

1.4. Método de trabajo.

En el siguiente diagrama se define la estructura llevada a la hora de realizar el trabajo.

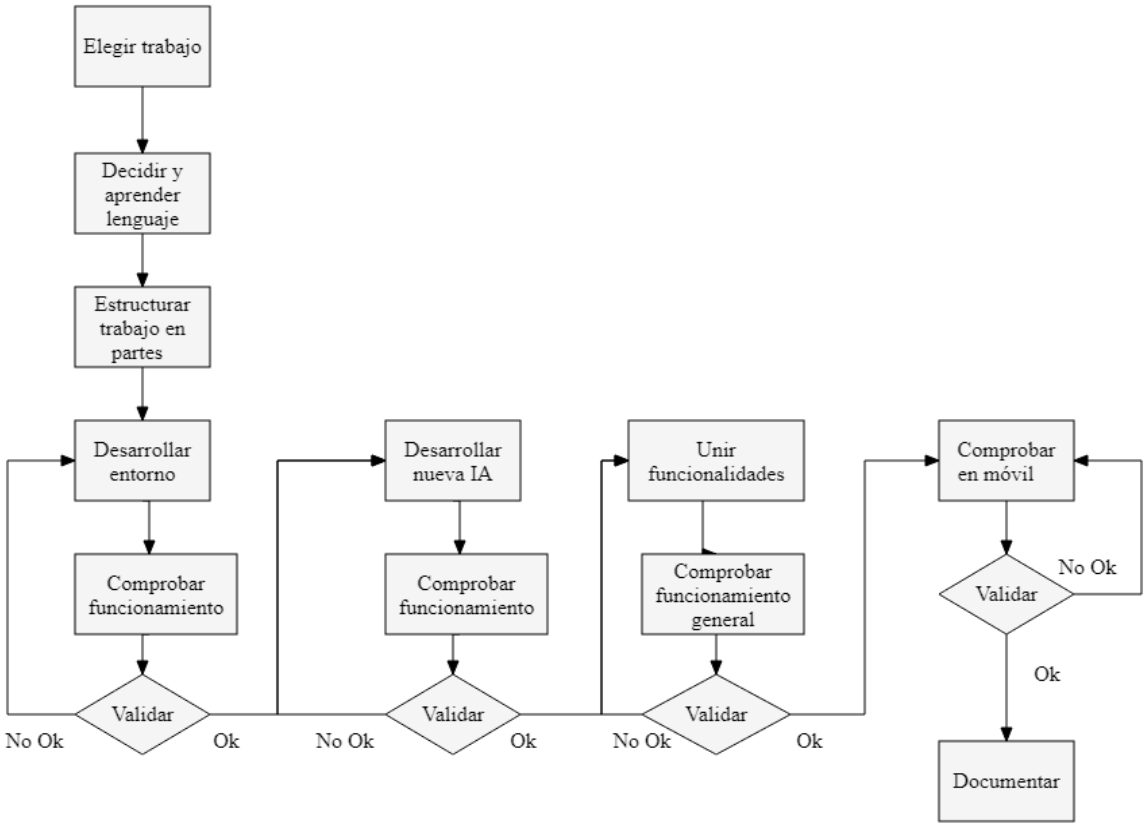

Ilustración 1: *Diagrama de metodología de trabajo.*

Como se puede apreciar en el diagrama, la metodología básica seguida en el proyecto, una vez escogido el lenguaje que íbamos a usar, ha sido, para cada tipo de IA:

1. Crear un entorno separado sencillo donde poder hacer pruebas sin mucho costo.
2. Desarrollar y entrenar la IA[3] en dicho entorno hasta un funcionamiento.
3. En caso de desear dicha IA dentro del juego, implementarla en su entorno hasta conseguir un correcto funcionamiento.
4. Comprobar la funcionalidad del juego en un dispositivo móvil.

[3] IA: Siglas pertenecientes a “Inteligencia Artifical”.

2. Historia de los videojuegos.

Si nos remontamos a los orígenes de los videojuegos, existen dos claros proyectos que fueron el punto de partida de este mundo: por un lado, el matemático Alan Turing en colaboración con D. G. Champernowne (finales de los años 40) crearon un prototipo de programa para jugar al ajedrez, el cuál sentó las bases de los juegos modernos de ajedrez, por otro lado, Alexander Douglas, había creado *OXO* (1952), también conocido como tres en raya, donde el programa permitía enfrentar a un jugador contra la máquina.

No sería hasta finales de los años 50, cuando se creó el primer juego que permitía jugar entre dos personas, enfrentándose entre sí. Este juego fue el famoso *Tennis for two* de William Higginbotham, quien, sirviéndose de un osciloscopio y un programa de cálculo de trayectos, realizo este juego para el entretenimiento del público en una feria científica. Aunque el juego causó mucha expectación, Willian no pensaba que fuera a tener mucho impactó por lo que no lo registró. Años después, Nolan Bushnell creó la compañía *Atari* cuyo juego insignia era *Pong*. En 1972, *Atari* presentó la primera máquina recreativa, donde se podía jugar a una versión mejorada del *Pong* (Véase ilustración 2).

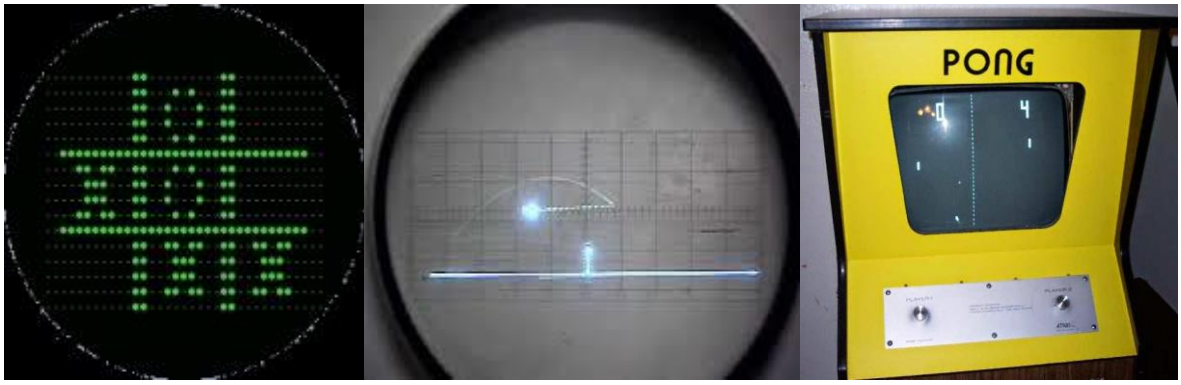

***Ilustración 2**: De izquierda a derecha: OXO, Tennis for Two y máquina recreativa con el juego Pong.*

El siguiente gran punto de inflexión en la historia de los videojuegos lo encontramos en la época de los 80, con la llamada generación de 8 bits[4]. Alentado por el gran éxito de los salones de máquinas recreativas y las primeras videoconsolas, el sector sufría un rápido crecimiento. En 1978, Japón entró en este mundo dando pasos decisivos en su avance, con la creación de juegos como *Space Invaders* por parte de *Taito* o el famoso *Pac-man*, desarrollado por *Namco* (1980), este era el primer videojuego donde se representaba un personaje con el que identificarse dentro del juego. En estos años, también se incorporó Nintendo (hasta ahora empresa de producción de cartas) en el mundo del videojuego, lanzando su famoso *Donkey Kong* (1981), y con él, los llamados juegos de plataformas, juegos basados en avanzar mientras se corre, salta y escala por una serie de plataformas (Véase ilustración 3).

[4] Generación de 8 bits: Tercera generación de consolas. Estas ya se comercializaban. Ejemplos: NES (Nintendo) o Sega Master System (SEGA).

Ilustración 3: *De izquierda a derecha: Space Invaders, Donkey Kong.*

Tras ver el éxito que estaba teniendo Japón con sus videoconsolas y juegos destinados a ellas, Estados Unidos lanzó su propia videoconsola, la NES, con un amplio catálogo de juegos, donde se podía jugar desde juegos de carreras hasta el famoso *Sonic*. Pero, no fue hasta 1985 donde se dio el siguiente punto de inflexión en el mundo de los videojuegos: *Nintendo* lanzó *Super Mario Bros* (Véase ilustración 4).

Ilustración 4: *De izquierda a derecha: Sonic, Super Mario Bros.*

Este juego supuso una revolución debido a que, hasta ese momento, todos los juegos eran “infinitos” y su objetivo era alcanzar la máxima puntuación posible, pero, en *Super Mario Bros* se introdujo por primera vez se tenía un objetivo y un fin de juego. En los años posteriores este estilo fue emulado por las demás compañías.

En los siguientes años, la industria del videojuego siguió avanzando y desarrollándose, y otros métodos de juego empezaron a cobrar fuerza. Un claro ejemplo fueron las consolas portátiles, siendo su estandarte *la Game Boy* de *Nintendo* (1989), parte de su éxito fue debido al juego ruso *Tetris* y al posterior lanzamiento de *Pokémon*, juego que innovaba con una fusión entre coleccionismo y juego.

Las videoconsolas siguieron avanzaron pasando de 8 bits a 16, y luego a 32, siendo estas últimas las que dieron un gran avance al sector, ya que introdujeron los juegos en 3D. Además, el hecho de que las consolas ya se vendían a un precio mucho más asequible supuso una cantidad de ventas mucho mayor, llegando a mucho más público y extendiendo el sector a pasos agigantados.

Con la entrada del sigo XXI nos encontramos con las “consolas de quinta generación”[5] las cuales elevaron el sector al estatus de sector multimillonario. Estas consolas se caracterizaban por el uso de CD-ROM [6] en lugar de los actuales cartuchos y fueron la principal causa de terminar de acabar con las máquinas recreativas, las cuales llevaban en decadencia desde los años 90 debido a que ya no podían ofrecer el mismo impacto visual que sus competidoras domésticas. A estas consolas había que sumarle la aparición de nuevos tipos de juegos como podían ser los juegos RTS [7](Real-Time Strategy) para PC como pudieron ser *Warcraft* o *Age of Empires*, género que fue muy bien recibido, empezando así la guerra entre PC vs consolas.

Tras estas, llegó la PlayStation 2, consolidándose como la consola más vendida de la historia con 150 millones de unidades vendidas gracias a títulos tan importantes como *Tekken*, *Final Fatasy VII*, *Resident Evil* o *Crash Bandicoot* (Véase ilustración 5). Títulos que afianzaron a sus empresas dentro del mercado y dieron lugar a grandes sagas de videojuegos, muchas de las cuáles, siguen hoy en día en distribución.

Ilustración 5: *De izquierda a derecha: Age of Empires(PC), Tekken 4(PS2).*

Actualmente, las consolas de última generación utilizan DVD[8] en lugar del anticuado CD-ROM, dando cabida a almacenar juegos más pesados de mucha más calidad como los ofrecidos en la actual PS4 (Véase ilustraciones 6 y 7). Estas consolas con capaces de dotar al jugador de una experiencia VR[9], aumentando así la inmersión de la persona en el juego.

Cabe también mencionar que con la incorporación de los smartphones [10] al mercado, dentro del sector de los videojuegos ha surgido una rama orientada a ellos. Aunque aún está por explotar, esta rama está creciendo de manera exponencial, debido a su gran popularidad entre los jugadores más jóvenes gracias a su versatilidad, la gran portabilidad

[5] Consolas de quinta generación: Primeras consolas capaces de reproducir juegos en 3D.
[6] CD-ROM (Compact Disc Read-Only Memory): Es un disco compacto con el que utilizan rayos láser para leer información en formato digital.
[7] Juego RTS (*Real-Time Strategy*): Son videojuegos de estrategia en los que no hay turnos, sino, que el tiempo transcurre de forma continua para los jugadores.
[8] DVD (Digital Versatile Disc): Es un tipo de disco óptico para almacenamiento de datos.
[9] VR (Virtual Reality): Entorno de escenas u objetos de apariencia real.
[10] Smartphone: Teléfono con capacidad de conexión a Internet.

de la consola (el propio Smartphone) y a la poca duración de las partidas, siendo estas ideales para pequeños huecos de tiempo libre como podrían ser los viajes en transporte público. Un ejemplo, de empresa dedicada íntegramente a este tipo de juegos sería *Supercell*, con una ganancia anual de superior a los 2.000 millones de euros.

Ilustración 6: *Escena del mundo "Piratas del Caribe" de Kingdom Hearts 3.*

Ilustración 7: *Representación del jugador Cristiano Ronaldo en FIFA19 (PS4).*

Como se ha podido apreciar en esta introducción, la industria del videojuego ha crecido de forma exponencial en los últimos 20 años, siendo totalmente diferente a la que era en sus inicios. Esto ha desencadenado la creación de nuevos trabajos como diseñador, programador o tester de videojuegos [11] e incluso creando carreras mayoritariamente enfocadas en ellos. Por ejemplo:

https://www.educaweb.com/curso/grado-desarrollo-videojuegos-madrid-350703/

https://www.educaweb.com/curso/grado-diseno-desarrollo-videojuegos-madrid-295693/

https://www.educaweb.com/curso/grado-desarrollo-videojuegos-madrid-350703/

[11] Tester de videojuegos: Trabajador enfocada a la prueba de videojuegos para encontrar fallos en el sistema.

https://www.educaweb.com/curso/grado-multimedia-uoc-on-line-76619/

Otra prueba del crecimiento de este sector, serían las hackathons [12] creadas exclusivamente para el sector de los videojuegos. Jornadas, donde durante unos días los participantes forman grupos compuestos por personas procedentes de diversas disciplinas para crear su propio videojuego con la finalidad de luego exponerlo en un concurso final que suele galardonar con premios en metálico.

2.1. Género RPG.

El género RPG, como sus propias siglas indican (Role-Playing Game) es la adaptación de los juegos clásicos de rol como *Dungeon & Dragons* (donde los jugadores tenían que interpretar un papel) al mundo de los videojuegos.

Estos juegos están basados generalmente en el transcurro de una historia de aventuras y fantasía épica donde los jugadores van avanzando, enfrentándose a enemigos cada vez más fuertes, con las habilidades y objetos que han ido adquiriendo durante el transcurro de la misma. Existen dos claras tendencias a la hora de desarrollar la acción en los combates:

- Combate por turnos: El combate tiene una secuencia turnos, decidimos por una estadística a elección del programador. En cada turno, el personaje actual puede realizar una acción y los demás se ven afectados por ella.

- RPG acción: La más utilizada actualmente, el jugador controla un personaje o varios (intercambiando el control) en tiempo real y va realizando las acciones que él crea convenientes. Este es el método de combate seleccionado para este trabajo.

Una variación de estos juegos, son los RPG en línea, también conocidos como MMORPGs[13], donde el jugador formará grupo con otros jugadores de manera opcional para enfrentarse a retos mayores. El exponente en este estilo de juegos siempre ha sido desde sus inicios *World of Warcraft* (https://worldofwarcraft.com/es-es/), juego que ha llegado a tener más de doce millones de jugadores activos jugando al mismo tiempo en sus distintos servidores (Véase Ilustración 8).

[12] Hackathon: Encuentro de programadores cuyo objetivo es el desarrollo colaborativo de software.
[13] MMORPG (Massively Multiplayer Online Role-Playing Game): Videojuegos de rol que permiten a miles de jugadores introducirse en un mundo virtual de forma simultánea a través de Internet e interactuar entre ellos.

Ilustración 8: *De izquierda a derecha: Final Fantasy VII, World of Warcraft.*

Por último, existe otra variante de los juegos RPG (inspirados en el ajedrez), conocida como TRPG[14]. En estos juegos, los controlan a un grupo de personajes los cuales deben ir moviendo por un tablero de casillas (no siempre visibles) a la vez que se enfrentan a enemigos con magias, armas y habilidades. Actualmente, el TRPG que más ha triunfado ha sido *Fire Emblem* (Véase Ilustración 9), franquicia que consta actualmente con más de quince videojuegos y millones de copias vendidas.

Ilustración 9: *Captura de juego procedente de Fire Emblem Awakening.*

[14] TRPG (Tactical Role-Playing Game): También conocidos como SRPG (Strategy Role Playing Game), es un género de videojuegos con elementos de los videojuegos de rol y de los videojuegos de estrategia.

3. Inteligencia Artificial, qué es y para qué sirve.

La inteligencia artificial (IA), también llamada inteligencia computacional, es la inteligencia exhibida por máquinas. Consideramos a una máquina inteligente, cuando es capaz de imitar el comportamiento de un agente racional flexible que percibe su entorno y puede llevar a cabo acciones basadas en él para maximizar las posibilidades de éxito de objetivo o tarea[15].

Coloquialmente se usa cuando una máquina es capaz de usar funciones cognitivas que se asocian a la mente de un ser humano, como podrían ser “aprender” y “resolver problemas”.

Otra definición posible de inteligencia artificial sería: rama de las ciencias computacionales [16] encargada de estudiar modelos de cómputo capaces de realizar actividades propias de los de los seres humanos en base a dos características primordiales: el razonamiento y la conducta.

Para Nils John Nilsson son cuatro los pilares básicos en los que se apoya la inteligencia artificial[17]:

- **Búsqueda del estado requerido** en el conjunto de los estados producidos por las acciones posibles.
- **Algoritmos genéticos** (análogo al proceso de evolución de las cadenas de ADN).
- **Redes neuronales artificiales** (análogo al funcionamiento físico del cerebro de animales y humanos).
- **Razonamiento mediante una lógica formal** análogo al pensamiento abstracto humano.

Hay diversas formas de clasificar las inteligencias artificiales, la más común de ver sería la dada por los expertos en las ciencias de la conmutación Stuart Russell y Peter Norvig:

1. **Sistemas que piensan como humanos**: automatizan actividades como la toma de decisiones, la resolución de problemas y el aprendizaje. Un ejemplo son las redes neuronales artificiales.
2. **Sistemas que actúan como humanos**: se trata de computadoras que realizan tareas de forma similar a como lo hacen las personas. Es el caso de los robots.
3. **Sistemas que piensan racionalmente**: intentan emular el pensamiento lógico racional de los humanos, es decir, se investiga cómo lograr que las máquinas puedan

[15] https://es.wikipedia.org/wiki/Inteligencia_artificial
[16] Ciencias computacionales: son aquellas que abarcan las bases teóricas de la información y la computación, así como su aplicación en sistemas computacionales.
[17] https://inteligenciaartificial170.wordpress.com/2016/09/04/pilares-basicos-segun-nilsson/

percibir, razonar y actuar en consecuencia. Los sistemas expertos se engloban en este grupo.

4. **Sistemas que actúan racionalmente**: idealmente, son aquellos que tratan de imitar de manera racional el comportamiento humano, como los agentes inteligentes.

Otra clasificación posible sería la que diferencia entre IA fuerte e IA débiles:

- La **IA débil**, también conocida como IA estrecha, es un sistema de IA que está diseñado y entrenado para una tarea en particular. Los asistentes personales virtuales, como Siri de Apple, son una forma de débil de IA.
- La **IA fuerte**, también conocida como inteligencia general artificial, es un sistema de IA con habilidades cognitivas humanas generalizadas, de modo que cuando se le presenta una tarea desconocida, tiene suficiente inteligencia para encontrar una solución.

Como curiosidad, cabe destacar, que el sector de las IA evoluciona a un ritmo tan trepidante y tiene potencial como para realizar cosas tan impresionantes que, personas como Arend Hintze, profesor asistente de biología integradora e ingeniería y ciencias de computación en la Universidad Estatal de Michigan, categorizan las IA junto a otros tipos de IA que aún no se han llegado a implementar, pero da por hecho que algún se conseguirá. Arend Hintze[18] distingue cuatro tipos de IA:

• **Tipo 1: Máquinas reactivas.** Un ejemplo es *Deep Blue*, el programa de ajedrez de IBM que venció a Garry Kasparov en los años noventa. *Deep Blue* puede identificar piezas en el tablero de ajedrez y hacer predicciones, pero no tiene memoria y no puede usar experiencias pasadas para informar a las futuras. Analiza movimientos posibles (los propios y los de su oponente) y basándose en estos, elige el movimiento más estratégico. *Deep Blue* y *AlphaGO* de *Google* fueron diseñados para propósitos estrechos y no pueden aplicarse fácilmente a otra situación.

• **Tipo 2: Memoria limitada.** Estos sistemas de AI pueden usar experiencias pasadas para informar decisiones futuras. Algunas de las funciones de toma de decisiones en vehículos autónomos han sido diseñadas de esta manera. Las observaciones son utilizadas para informar las acciones que ocurren en un futuro no tan lejano, como un coche que ha cambiado de carril. Estas observaciones no se almacenan permanentemente.

• **Tipo 3: Teoría de la mente.** Este es un término psicológico. Se refiere a la comprensión de que los demás tienen sus propias creencias, deseos e intenciones que afectan las decisiones que toman. Este tipo de IA aún no existe.

• **Tipo 4: Autoconocimiento.** En esta categoría, los sistemas de IA tienen un sentido de sí mismos, tienen conciencia. Las máquinas con conciencia de sí comprenden su estado

[18] http://theconversation.com/understanding-the-four-types-of-ai-from-reactive-robots-to-self-aware-beings-67616

actual y pueden usar la información para inferir lo que otros están sintiendo. Este tipo de AI aún no existe.

3.1. Aplicación de las inteligencias artificiales.

Una de las cualidades más destacada de la inteligencia artificial, y por la que ha demostrado ser tan importante en el desarrollo de la sociedad, es su flexibilidad, la cual le permite aplicarse a infinidad de campos. Unos ejemplos serían:

• **IA en la asistencia sanitaria:** Las mayores apuestas están en mejorar los resultados de los pacientes y reducir los costos. Las empresas están aplicando el aprendizaje de máquina para hacer diagnósticos mejores y más rápidos que los seres humanos. Una de las tecnologías sanitarias más conocidas es *IBM Watson*. Entiende el lenguaje natural y es capaz de responder a las preguntas que se le formulan. El sistema extrae datos de los pacientes y otras fuentes de datos disponibles para formar una hipótesis, que luego presenta con un esquema de puntuación de confianza. Otras aplicaciones de IA incluyen *chatbots*[19], un programa de computadora utilizado en línea para responder a preguntas y ayudar a los clientes, para ayudar a programar citas de seguimiento o ayudar a los pacientes a través del proceso de facturación, así como en asistentes virtuales de salud que proporcionan retroalimentación médica básica.

• **IA en los negocios:** La automatización de procesos robóticos se está aplicando a tareas altamente repetitivas que normalmente realizan los seres humanos. Los algoritmos de aprendizaje automático se están integrando en las plataformas de análisis y CRM[20] para descubrir información sobre cómo servir mejor a los clientes. Los *chatbots* se han incorporado en los sitios web para ofrecer un servicio inmediato a los clientes.

• **IA en la educación:** La IA puede automatizar la calificación, dando a los educadores más tiempo. Este tipo de IA puede evaluar a los estudiantes y adaptarse a sus necesidades, ayudándoles a trabajar a su propio ritmo. Los tutores basados en IA pueden proporcionar apoyo adicional a los estudiantes, asegurando que se mantengan en el buen camino. Este tipo IA podría cambiar dónde y cómo aprenden los estudiantes en un futuro.

• **IA en finanzas:** La IA aplicada a las aplicaciones de finanzas personales, como *Mint* o *Turbo Tax*, está transformando a las instituciones financieras. Aplicaciones como estas podrían recopilar datos personales y proporcionar asesoramiento financiero.

---

[19] Programa que simula mantener una conversación con una persona al proveer respuestas automáticas a entradas hechas por el usuario.

[20] CRM (Customer Relationship Management): Es una aplicación que permite centralizar en una única Base de Datos todas las interacciones entre una empresa y sus clientes.

• **IA en los procesos judiciales:** El proceso de descubrimiento, a través de la revisión de documentos, en los procesos judiciales es a menudo abrumador para los seres humanos. Automatizar este proceso proporciona un mejor uso del tiempo y en general una mayor eficiencia.

• **IA climáticas:** IA aplicadas a drones capaces de detectar la deforestación, vehículos submarinos no tripulados para detectar fugas en diferentes tipos de conducciones, edificios inteligentes adaptados para minimizar el consumo y enfrentarse a diversas catástrofes naturales, etc.

• **IA en los videojuegos:** Suponen el comportamiento del entorno y de personajes que habitan en dicho juego. Ya que este proyecto está basado en este punto, en el siguiente apartado se explicará con más detalle cómo se han ido aplicando las IA en el mundo de los videojuegos.

3.1.1. Aplicación de las inteligencias artificiales en el sector de los videojuegos.

Durante el avance de la industria del videojuego se han ido creando distintos tipos de IA, que han ido evolucionando y variando hasta tener una enorme variedad de técnicas aplicadas a esta tecnología. En este TFG se explicarán e implementarán algunas de las técnicas más comunes en el sector de los videojuegos.

Dicho esto, se procederá a hacer una breve recopilación de la historia de la IA en el mundo de los videojuegos:

Las primeras IA conocidas fueron las aplicadas en juegos de mesa como las usadas en simuladores de ajedrez y damas. Un primer planteamiento para estas máquinas era seguir dinámicas basadas en fuerza bruta, que analizarían todos los posibles movimientos usando el algoritmo *Minimax* [21] (método de decisión recursivo para *minimizar* la pérdida *máxima* esperada en juegos con adversario y con información perfecta). Pero se vio que esto no era una medida razonable, ya que, tras unos pocos movimientos, la cantidad de posibilidades era enorme lo que suponía unos tiempos de computo fuera de lo razonable. El siguiente punto de vista que surgió fue emular a los jugadores profesionales, que a través de la experiencia son capaces de analizar hasta 40-50 “buenas jugadas” y optar por la mejor de ellas. Esto derivó en un filtrado de posibilidades, donde la máquina sólo optaba por las mejores soluciones, dejando sin comprobar las “malas jugadas”. Los problemas de esta última idea son la dificultad que conllevaba depurar y crear código, y la falta de flexibilidad. Con el paso del tiempo se ha ido alternando entre estos dos movimientos: cuando surgía un nuevo hardware que mejoraba la velocidad de las máquinas se optaba por la primera configuración; por otro lado, cuando se afinaba un algoritmo dedicado a este ámbito,

[21] https://es.wikipedia.org/wiki/Minimax

destacaban las máquinas que optaban por la segunda. Actualmente, la mayoría de las máquinas de ajedrez usan el modelo aportado por el segundo enfoque.

El siguiente paso fue las IA basadas en la lógica, como la usada en el juego *Pong*, donde el programa actuaba acorde a unas ecuaciones explícitas para calcular rutas y rebotes. Poco después, entre los 80 y 90 se incorporaron diversas técnicas en este sector como podrían ser: Path-finiding [22]en el juego *Pac-Man* (donde se indicaba como encontrar un camino dentro de un laberinto), IA de batalla para enemigos usadas en *Dragon Warrior*, (juego considerado el primer RPG) o técnicas un poco más avanzadas como podrían ser: las redes neuronales, las máquinas de estados finitos o la lógica difusa.

Hoy en día prácticamente después de cada lanzamiento de un juego AAA[23], se descubre un nuevo tipo de IA que se ha empleado en él para mejorar la experiencia del usuario. Algunos ejemplos importantes serían:

- *Black & White (2001):* juego centrado en interactuar con una criatura que aprende en función de los premios y castigos recibidos, mientras tomas el rol de un dios para conquistar diferentes tribus en tiempo real.

- *FarCry (2004):* los enemigos usan tácticas militares. Además, la IA no usa “trampas” como saber la posición real del jugador, sino que almacenaba la última posición conocida.

- *Left 4 Dead (2008):* genera de forma procedural diferentes experiencias para los usuarios cada vez que juegan.

[22] Path Finding: Algoritmo destinado a la obtención del camino más corto entre dos puntos.
[23] Juego AAA: Clasificación informal para los juegos producidos y distribuidos por una distribuidora importante.

3.2. Diferentes métodos para implementar una IA.

En este apartado se describen los tipos de inteligencia artificial más comunes a la hora de realizar un videojuego, apoyados de un ejemplo teórico para facilitar su explicación y conseguir diferenciarlos de manera más precisa.

3.2.1. Máquina de estados finita.

A pesar de su sencillez, es un método muy eficaz y especialmente importante en Unity, ya que, independientemente de la tecnología que mueva tu personaje, su Animator[24] será una máquina de estados. Dependiendo de la complejidad del repertorio de acciones de nuestro personaje, se pueden encontrar: desde una máquina de estados simple con dos ó tres estados, hasta una máquina con cientos de estados (Véase ilustración 10).

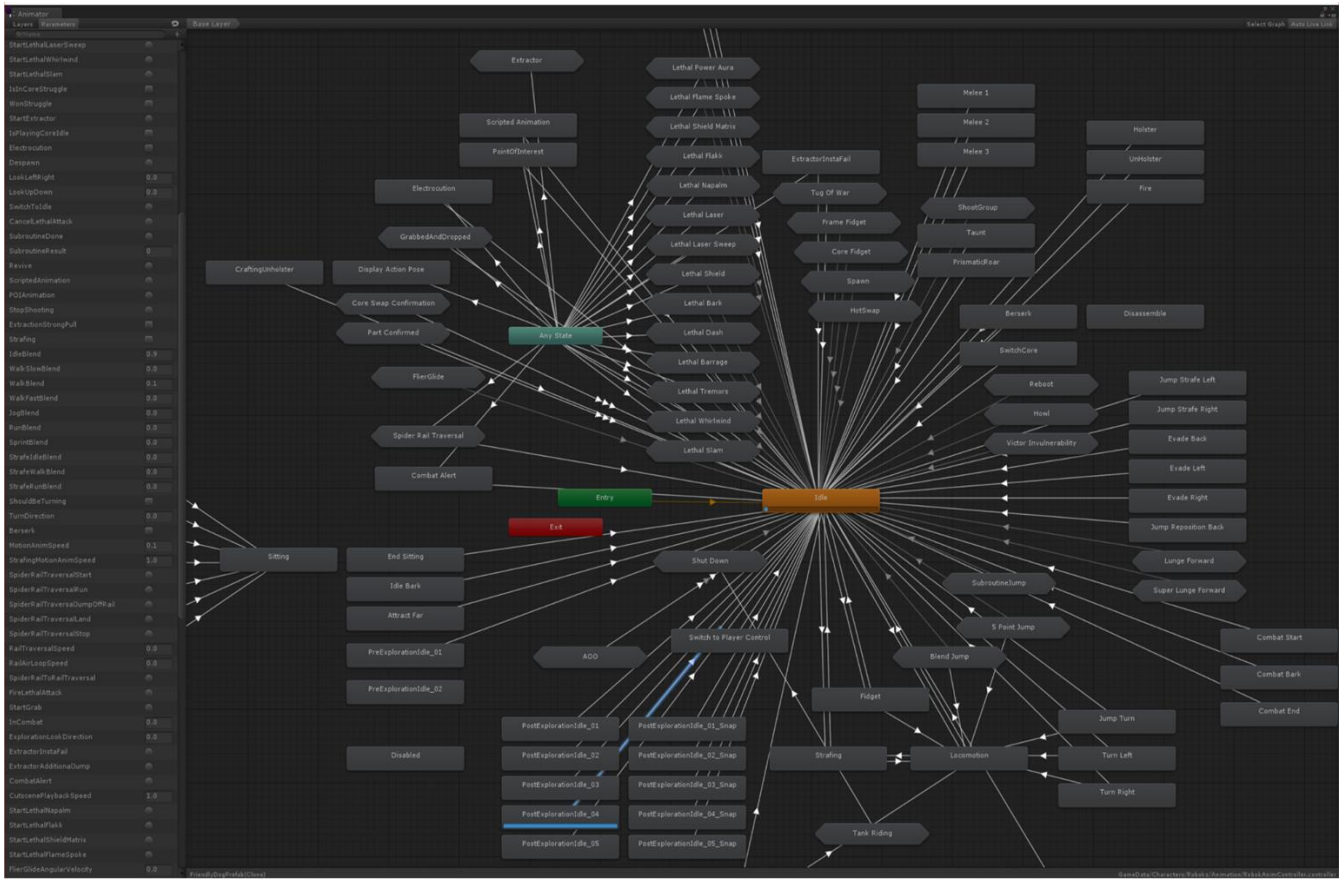

Ilustración 10: *Ejemplo de Animator complejo.*

Se denomina máquina de estados finita (FSM por Finite State Machine) a un modelo de comportamiento de un sistema con entradas y salidas, donde las salidas no dependen únicamente de las entradas actuales, sino también de las anteriores.

---

[24] Animator: Máquina de estados que contiene las distintas animaciones relacionadas con un personaje, así como la definición de las transiciones unidas a ellas.

Estas máquinas se definen como un conjunto de estados que sirven de intermediarios entre la entrada y la salida, haciendo que el historial de señales de entrada determine, para cada instante, un estado para la máquina, de forma tal que la salida depende únicamente del estado y las entradas actuales.

Las FSM también conocidas como autómatas finitos se caracterizan por tener un estado inicial, desde el cual, al recibir una cadena de símbolos, por cada uno de ellos, puede cambiar de estado o seguir en el mismo. También constan de un conjunto de estados finales que nos indican si una cadena pertenece al lenguaje final de una lectura.

Estos autómatas se pueden clasificar en 2 tipos: determinista y no determinista.

- **Autómatas finitos deterministas**: Para cada entrada en cada estado, solo hay un estado siguiente. Estos son los tipos de autómatas con los que se ha trabajado en este proyecto.

- **Autómatas finitos no deterministas**: Para cada estado, pueden existir varios estados siguientes diferentes ante la misma entrada. También se puede dar la posibilidad de cambiar de estado sin recibir ninguna entrada.

Veamos qué quiere decir esto con un ejemplo: la máquina de estados utilizada en un fantasma del popular juego: *Pac-Man* (Véase la Ilustración 11).

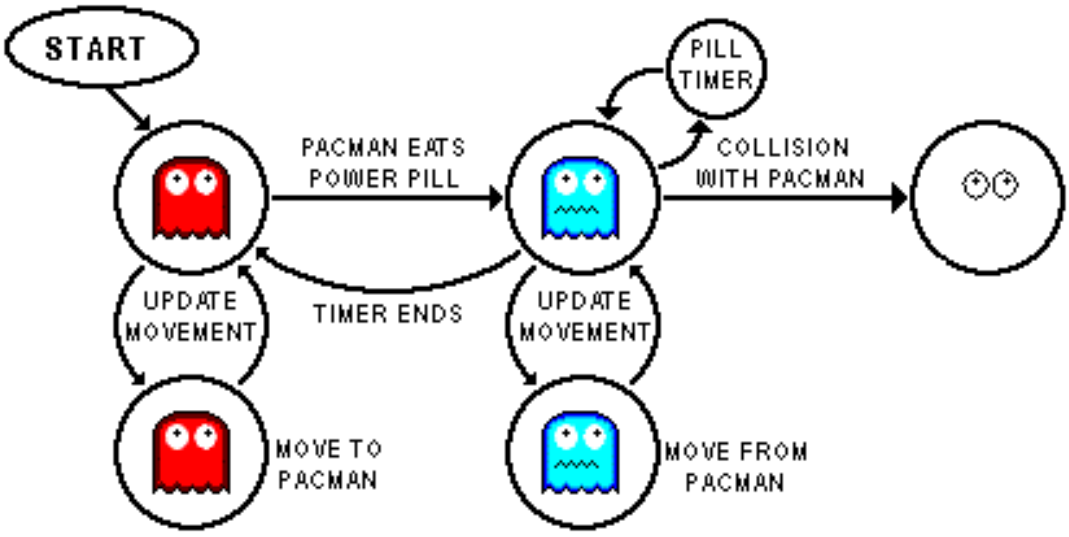

Ilustración 11: *Máquina de estados finita de un fantasma del juego Pac-Man.*

Como se puede observar, la máquina consta de un estado inicial, el denominado START, con el que comenzará el comportamiento de nuestro personaje. Nada más empezar la partida, los fantasmas se encontrarán en estado normal (rojo), por lo que perseguirán a Pacman para comérselo de manera indefinida, pero, si en algún momento de la partida,

Pacman se come la píldora de poder, estos pasarán a su estado de debilidad (azul) hasta que un timer expire, en este estado, los fantasmas en lugar de perseguir a Pacman, huirán de él, ya que, si sufren una colisión con Pacman, morirán, siendo este el estado final de nuestra máquina de estados finita.

3.2.2. Aprendizaje por refuerzo.

El aprendizaje por refuerzo es un área que en los últimos años ha atraído a una gran cantidad de investigadores y que ha contribuido a lograr enormes avances en otras áreas como son el desarrollo de buscadores web, robótica cognitiva o la minería de datos.

El aprendizaje por refuerzo consiste en aprender qué acciones realizar, dado el estado actual de su entorno, con el objetivo de maximizar una señal de recompensa numérica, lo cual requiere de un mapeo de situaciones a acciones (Véase Ilustración 12).

El aprendizaje se realiza en la mayor parte de los algoritmos mediante la interacción entre el agente y su "medio ambiente". El gran dilema que se presenta a la hora de desarrollar una inteligencia artificial basada en el aprendizaje por refuerzo es el dado entre exploración y explotación. Por un lado, para obtener mejores recompensas, el agente debe realizar las operaciones que en el pasado han servido para obtener una recompensa mayor (explotación), pero, por otro lado, el agente también tiene que explorar para intentar llegar a una ejecución con una recompensa mayor (exploración).

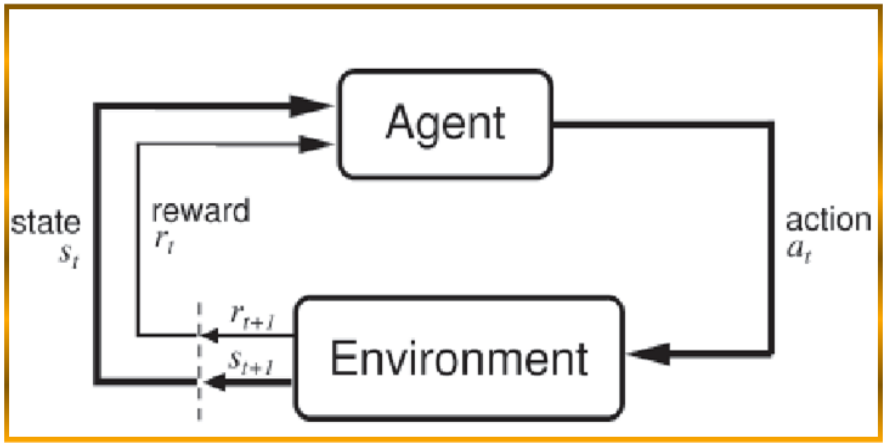

Ilustración 12: *Diagrama del comportamiento general del aprendizaje por refuerzo.*

Para explicar esto, podemos suponer que los aprendices siguen en su decisión un proceso de decisión de Markov[25], es decir:

[25] Proceso de Márkov: Es un fenómeno aleatorio dependiente del tiempo para el cual se cumple una propiedad específica.

- El agente percibe un **conjunto finito**, *S*, de estados distintos en su entorno, y dispone de un conjunto finito, *A*, de acciones para interactuar con él.

- El **tiempo avanza de forma discreta**, y en cada instante de tiempo, *t*, percibe un estado concreto s_t, una acción posible, a_t, y la ejecución de esta le lleva a un estado nuevo, $s_{t+1} = a_t * s_t$.

- El entorno responde a esta acción del agente por medio de una **recompensa/castigo** $r(a_t, s_t)$. Esta recompensa/castigo será un número, y cuanto más alto sea, mayor es el beneficio.

- Tanto la recompensa como el estado siguiente no tienen por qué ser conocidos por el agente a priori. Es decir, antes de aprender, cuando el agente lleve a cabo una acción desde un estado en particular, no sabe que pasará. Gracias a esto, el agente puede aprender a evitar realizar acciones con castigo y realizar las acciones que mayor recompensa le aportan.

En resumen, se busca que el agente aprenda lo que se denomina una política, lo que podría verse como una aplicación que le indica en cada estado que acción tomar.

3.2.3. Curriculum learning.

Se define como curriculum learning la forma de entrenar comportamientos complejos mediante una serie de lecciones de dificultad creciente. Las lecciones deben estar separadas de forma equidistante en la que sería una escala de dificultad. Un ejemplo teórico sería (Véase Ilustración 13):

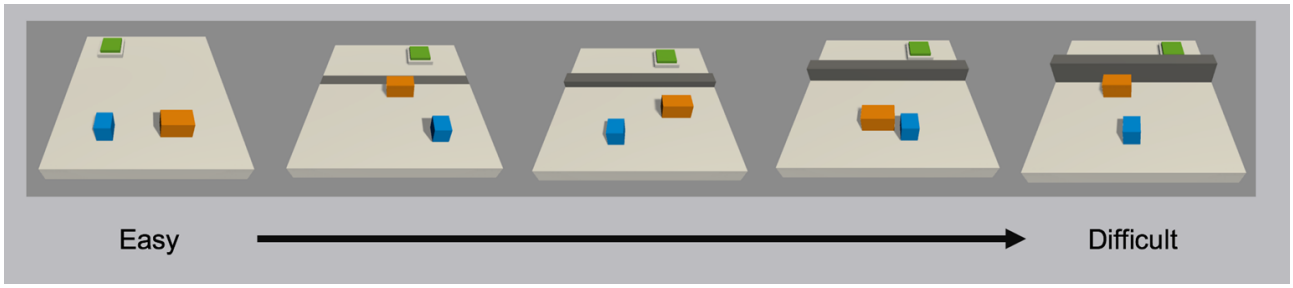

Ilustración 13: *Representación de distintas lecciones en un proceso de curriculum learning.*

Nos gustaría que el agente (cubo azul), pueda llegar a la meta (cubo verde) en un entorno donde hay obstáculos (cubos naranjas) y un muro (elevación gris oscuro) de la manera más óptima posible. El conjunto de lecciones optada para desarrollar el comportamiento de nuestro agente podría ser:

- Primero, conseguir que el cubo llegue hasta la meta en un camino llano.

- Una vez conseguido esto, se insertaría un muro, cada vez más alto, pero siempre de una altura que el agente sea capaz de saltar por sí solo.

- Finalmente, en las últimas lecciones, el muro alcanzará una altura que el agente no pueda saltar por sí solo, por lo que deberá utilizar el obstáculo para superarlo.

Una vez definidas las lecciones, se debe valorar que parámetros definir para poder llevarlas a cabo, como podrían ser en este caso: el número de obstáculos, la altura del muro o la distancia inicial hasta la meta. Para ello se creará un archivo JSON [26] que describirá nuestro curriculum (conjunto de lecciones junto a los parámetros necesarios para definirlas y los valores límite necesitados para "aprobar" una lección y pasar a la siguiente). El archivo JSON tendría una apariencia similar al siguiente (Véase Ilustración 14):

```
{
    "measure" : "progress",
    "thresholds" : [0.1, 0.3, 0.5],
    "min_lesson_length" : 2,
    "signal_smoothing" : true,
    "parameters" :
    {
        "big_wall_min_height" : [0.0, 4.0, 6.0, 8.0],
        "big_wall_max_height" : [4.0, 7.0, 8.0, 8.0],
        "small_wall_height" : [1.5, 2.0, 2.5, 4.0]
    }
}
```

Ilustración 14: *Ejemplo de archivo JSON para declarar un curriculum.*

En el archivo nos encontramos con los siguientes campos a rellenar:

- **Measure**: Indica que medida se utilizará para medir el progreso del aprendizaje. Existen dos tipos de media:

  - **Reward**: Indica que para pasar a la siguiente lección hay que obtener una recompensa obtenida igual o mayor a la indica en "**thresholds**[27]". En la figura ejemplo, si el agente se encuentra en la primera lección, una vez obtenida una recompensa de 0.1 o mayor, pasará a la siguiente lección.
  - **Progress**: Los valores indicados en "**thresholds**" serían el porcentaje del entrenamiento total que llevamos hecho. Ejemplo: si tomáramos los valores de la imagen como thresholds y nuestro entrenamiento estuviera configurado con 1000 pasos máximo, tras dar el 10% de ellos, es decir, 100 pasos, se pasaría a la lección 2, independientemente de los resultados obtenidos hasta el momento.

[26] JSON (JavaScript Object Notation): Es un formato de texto ligero para el intercambio de datos.
[27] Threshold: Umbral que delimita el comportamiento de una aplicación.

- **Min_lesson_length**: Indica el número de veces que se debe alcanzar el treshold establecido para avanzar a la siguiente lección. Ejemplo: siguiendo con la foto anterior, el agente estando en la primera lección, deberá obtener como mínimo dos veces, 0.1 de recompensa obtenida para pasar a la siguiente lección.

- **Signal_smoothing**: Suavizado de la señal. Cuando está activado, suaviza la señal tomando el valor actual con un peso del 75% y el valor actual con un peso del 25%. Esto es interesante para evitar picos como los que podrían producirse debidos al ruido.

- **Parameters**: Indica los valores que tienes los parámetros definidos para cada una de las lecciones. Ejemplo: para la lección 2, la altura máxima del muro será 7.0.

Detalles a tener en cuenta:

- En el campo “Measure” es recomendable usar Reward, ya que es la opción que indica de manera más exacta si el agente está realizando bien el trabajo. Con la opción Progress, se podría estar al 90% del entrenamiento y el agente no haber aprendido nada, ya que en las lecciones “fáciles”, no se le dio el tiempo suficiente para aprender.

- El valor treshold solo se comprueba cuando la academia ha terminado.

- En el apartado 7.2 se explica cómo utilizar este JSON.

Para terminar en esta gráfica (Véase Ilustración 15) se puede observar la diferencia entra la velocidad de aprendizaje que tiene un agente dependiendo de si usa curriculum learning o no.

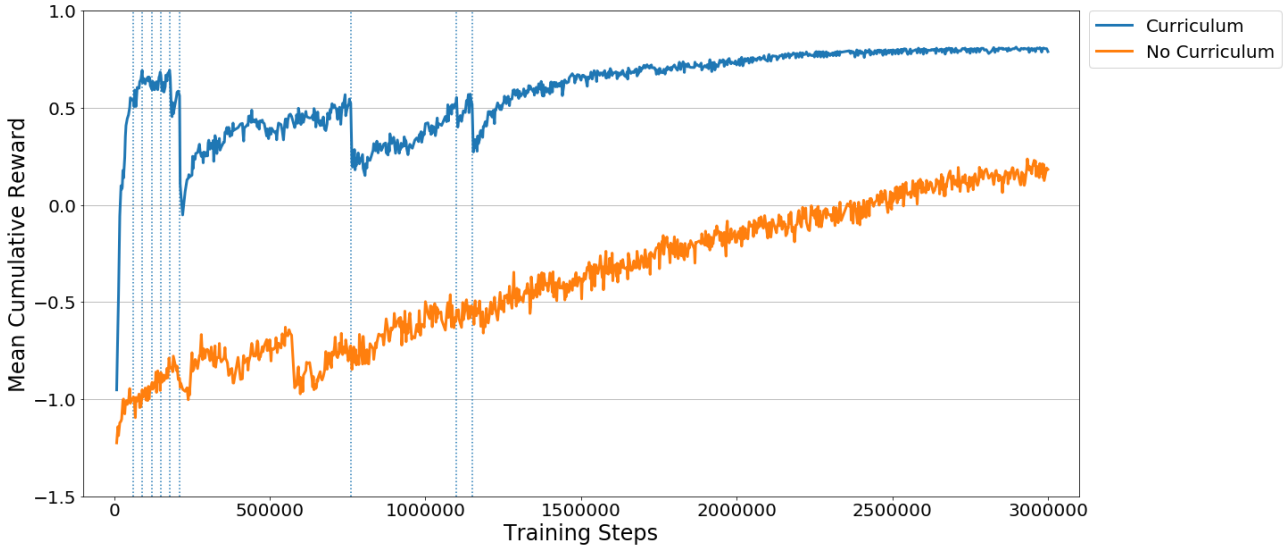

Ilustración 15: *Gráfica comparativa entre usar curriculum learnig o no*

Como se puede apreciar, el rendimiento es mucho mayor si usamos curriculum learning. Las grandes caídas representan el cambio entre dos lecciones con un salto de dificultad importante, como podría ser el que existe entre la última lección donde nuestro agente puede superar el muro solo y la siguiente, donde nuestro agente necesita la ayuda de un obstáculo para superarlo.

4.Herramientas utilizadas.

En este apartado se hablará sobre las herramientas utilizadas para la creación del proyecto, en que parte se han utilizado y por qué se ha decidido usar estar y no sus competidores.

4.1. Unity 3D.

Unity es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible para Windows, OS X y Linux, y permite crear juegos para Windows, OS, Linux, varias generaciones de XBOX y PlayStation, WII, WII U, iPad, iPhone, Android y Windows Phone.

Unity ha sido la plataforma donde se han creado los escenarios y mayoría de elementos del entorno del proyecto y donde se han integrado los diferentes scripts utilizados.

Este software tiene dos versiones, la versión Professional (pro) y la personal (versión gratuita y la utilizada en este proyecto).

El script se basa en Mono, la implementación de código abierto de .NET Framework para Linux. Los programadores pueden utilizar UnityScript (un lenguaje personalizado inspirado en la sintaxis ECMAScript), C# (el utilizado en este proyecto) o Boo (que tiene una sintaxis inspirada en Python).

4.1.1. Unity 3D vs Unreal engine.

A continuación, se verán los pros y contras entre los dos pilares del software dedicados a la creación de videojuegos:

- **Ventajas de Unity:** Gran cantidad de información (ya sea en forma de tutorial o de manual por Internet), una comunidad muy activa y una curva de aprendizaje fácil debido a su interfaz y a su lenguaje de programación (C#).

- **Desventajas de Unity:** Mala gestión de la memoria (las librerías .Net pueden resultar lentas a veces), poca calidad de creación de terrenos de base, bugs que necesitan un parche posterior para solucionarse.

- **Ventajas de Unreal:** Código abierto, por lo cual se puede mejorar desde la parte de usuario. Gran variedad de en el ámbito de iluminación, shadders y materiales disponibles.

- **Desventajas de Unreal:** Tiempo de aprendizaje debido al lenguaje utilizado (C++) y conceptos de programación del entorno como los Actors. Poca optimización de cara a proyectos móviles por la mala gestión de Draw Calls que el engine utiliza.

Teniendo estos puntos en cuenta, nuestra elección fue Unity, tanto por utilizar un lenguaje con el que estamos más familiarizados, como por ser mejor en proyectos destinados a la plataforma móvil como es nuestro caso.

4.2. Anaconda.

Anaconda es un software que nos permite gestionar entornos separados para diferentes distribuciones de Phyton. Este software se ha utilizado para crear un entorno enfocado en machine-learning.

Además, se utilizará para instalar dentro del entorno ml-agents la librería de código abierto TensorFlow dedicada a machine-learning y redes neuronales liberada por Google.

En términos generales el funcionamiento de Anaconda en este proyecto es el siguiente: tenemos un entorno de Unity, en el que se definen GameObjects[28] que contienen scripts encargados de modelar su funcionamiento. A través de estos scripts, le enviaremos a nuestro agente (el personaje en este contexto) los valores contenidos en su vector de observación [29] (previamente configurado en Unity).

Anaconda recibe el proyecto Unity compilado y para cada personaje usa su vector de observación (información del personaje y de su entorno) y ejecuta las acciones (vector de acciones definido en Unity) del personaje de forma en principio aleatoria. Cada acción tiene asociados un castigo y una recompensa. A través de la ejecución sucesiva de acciones se van seccionando aquellas que llevan a la recompensa y evitando las que originan un castigo. De esta manera Anaconda sirve para entena de forma automática a los personajes.

A través de una serie de comandos en el terminal Anaconda Prompt (ya que carece de interfaz gráfica), estos datos serán recogidos y utilizados para el entrenamiento de nuestro agente.

En el capítulo 7 se explicará con detalle la configuración.

[28] GameObjects: Son contenedores. Estos pueden guardar las diferentes piezas (componentes) que son requeridas para crear un elemento en nuestro entorno.
[29] Vector de observación: Vector que contiene el conjunto de parámetros que nos interesa analizar en cada observación del entorno.

5. Creación de un entorno de pruebas básico.

Antes de dar paso a ver en profundidad el modelado de los distintos tipos de inteligencia artificial implementados se debe tener en cuenta algo: en los casos en los que la inteligencia artificial que se desea desarrollar esté basada en machine-learning, la inteligencia inicial de nuestro agente será nula, y esto no cambiará hasta que complete varios entrenamientos. A esto hay que sumarle, que el número de entrenamientos necesarios para que nuestro agente se comporte de la manera deseada escalará potencialmente con la dificultad de las tareas que se desea que desempeñe y con el tipo de tecnología que se esté implementando. Dicho esto, una mala práctica sería implementar directamente nuestra inteligencia artificial basada en machine-learning en el entorno final, ya que, realizar los entrenamientos consumiría muchos más recursos, y eso puede derivar a un entorpecimiento del desarrollo. Para solucionar esto, se suelen realizar diferentes baterías de pruebas, dependiendo del tipo de IA:

- Para inteligencias muy complejas, se podría dividir la inteligencia artificial en pequeñas inteligencias artificiales. Por ejemplo, si se desea programar la inteligencia artificial que controlará un bot de asistencia médica, se podrían dividir sus tareas en: comunicación con el paciente, consular respuesta en base de datos, ponerse en contacto con el hospital y enviar resultado.

- Para inteligencias artificiales no tan complejas, una buena acción podría ser probar el funcionamiento de estas en un entorno simplificado. Es decir, crear un entorno, lo más básico posible donde poner a prueba nuestro agente, para poder diagnosticar si realiza sus funciones de manera correcta para, en caso de que así sea, llevar a cabo su implementación en el proyecto final.

Como las inteligencias artificiales basadas en machine-learning de este proyecto han sido realizadas en un entorno de prueba básico, a continuación, se procederá a explicar cómo crear este tipo de entornos desde cero.

Como en casi cualquier software de hoy en día, al iniciar Unity nos encontraremos con una pestaña principal, en caso de haberse trabajado anteriormente en proyectos, nos encontraríamos una lista recopilatoria de dichos proyectos, para tener un acceso rápido a ellos. Como nuestra intención es crear un entorno de pruebas desde cero, se deberá pulsar el botón “New” (Véase Ilustración 16) situado arriba a la derecha, y se rellenará una serie de parámetros estándar: Nombre del proyecto, Localización del proyecto y tipo de template. Este último puede ser el único campo que genere alguna duda. En esta pestaña se deberá elegir el tipo de template más preciso para nuestro proyecto, en nuestro caso, como nuestro proyecto contiene movimiento y elementos en 3D, se marcará 3D. Finalmente, se pulsará el botón “Create proyect”.

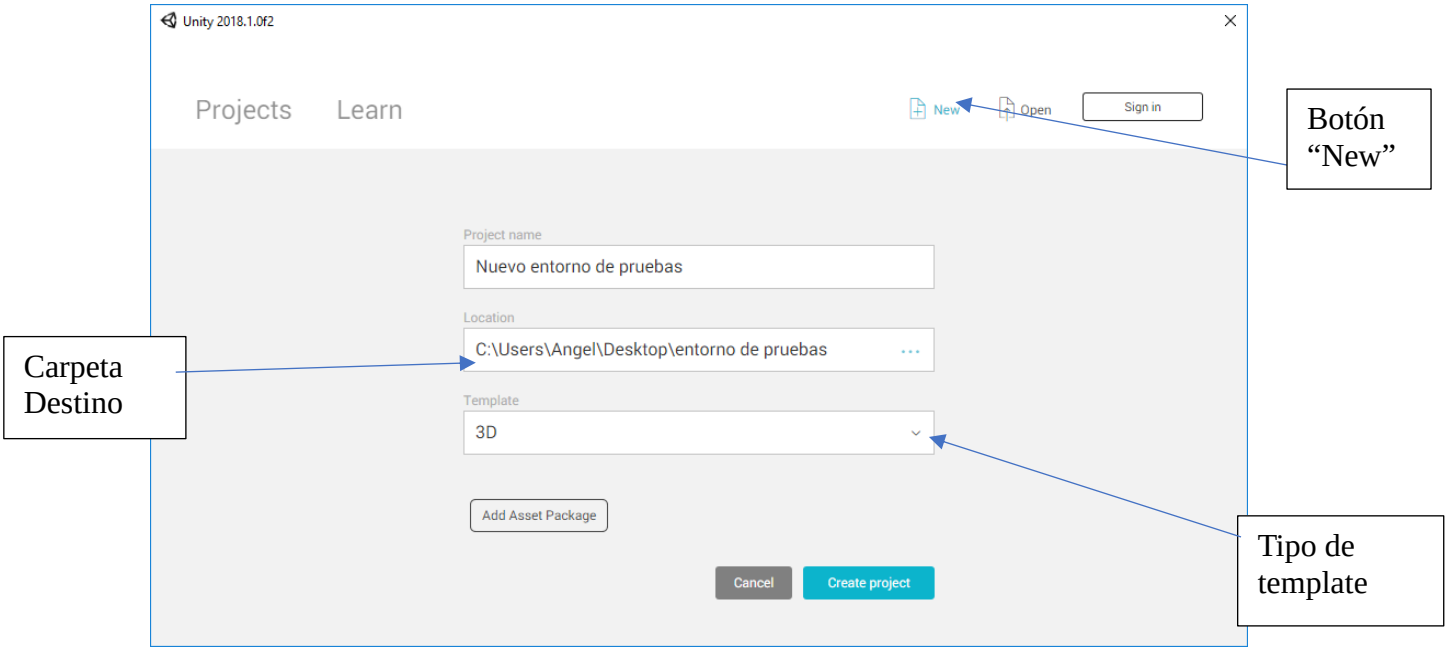

Ilustración 16: *Primeros pasos para crear un entorno de pruebas.*

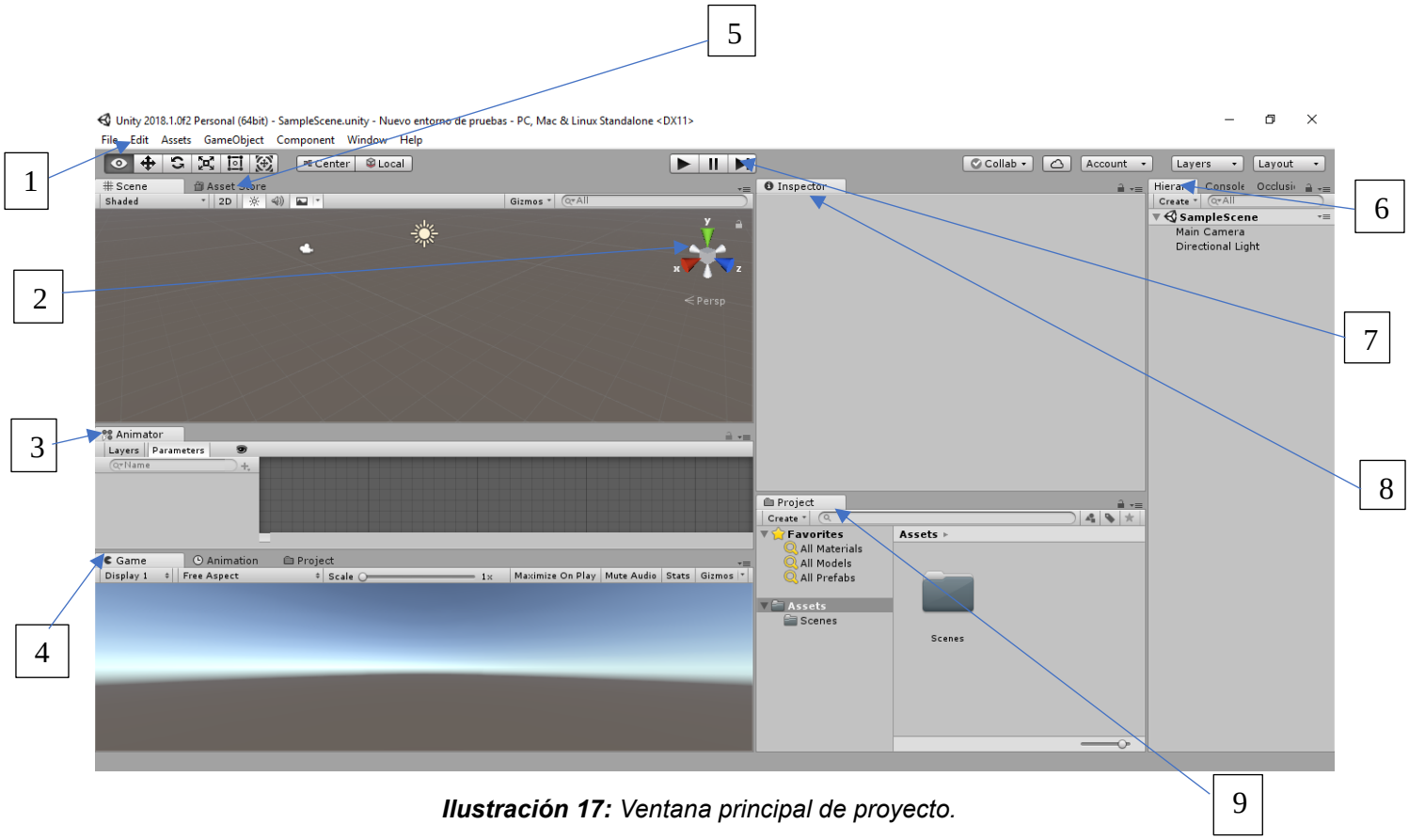

Ilustración 17: *Ventana principal de proyecto.*

Una vez creado el proyecto, encontraremos la ventana principal de Unity, ventana en la que se trabajará durante el 90% del tiempo destinado a trabajos que difieran de programar código. Esta ventana es configurable, por lo que según el ordenador desde donde se cree el proyecto, puede tener un aspecto u otro, aunque las funcionalidades básicas mostradas por defecto serán las mismas (siempre y cuando no lo modifique el programador). En la Ilustración número 17 se puede apreciar mi ventana principal, en la cual me apoyaré para explicar los puntos más importantes de esta:

1. **Menú de navegación del programa:** Permite desde crear/cargar un proyecto hasta crear/cargar cualquier tipo de elemento dentro de este. También tiene funciones como agregar o quitar pantallas a la ventana principal, compilar el proyecto, manual de ayuda…etc.
2. **Acceso rápido a cámara:** Permite configurar la perspectiva que se tiene de nuestro proyecto de manera rápida e intuitiva. Los tres conos coloreados indican los tres ejes de coordenadas.
3. **Animator:** permite observar y modificar el animator (conjunto de animaciones disponibles para un personaje) del elemento seleccionado, así como implementar el momento en el que se quiere disparar un action event (explicado en el anexo 7.8).
4. **Game**: Nos proporciona una vista WYSIWYG **(**acrónimo de “What You See Is What You Get”, que en español significa “lo que ves, es lo que tienes”). Se refiere a la vista final de una escena, es decir, la que se vería en tiempo de ejecución.
5. **Asset Store:** Permite descargar assets (elementos) desde la tienda de Unity. Estos elementos varían desde humanos o animales mitológicos, hasta elementos destinados a la creación de escenarios, como podrían ser cajas, mesas, sillas, etc., pasando por elementos que se pueden usar de complementos como podrían ser las armas. Algunos de estos assets pueden venir con animaciones ya hechas, para que, una vez el programador implemente una acción, “morir” por ejemplo, se tenga una animación que se pueda utilizar para complementar dicha acción, dando así más realismo al juego y favoreciendo la inmersión del jugador. El precio de los assets varía desde assets totalmente gratuitos a paquetes de assets que superan los 1000€.
6. **Hierarchy/Console:** La pestaña hierarchy muestra la jerarquía de nuestro proyecto dentro de Unity, es decir, como se tienen estructurados los diferentes elementos físicos que lo componen. La pestaña Console está destinada a la depuración de código, cuando se ejecuta/compila un proyecto, en ella se puede observar información sobre errores, warning o líneas de código que el programador desee mostrar por pantalla.
7. **Play/Pause/Step:** Estos botones sirven para controlar el tiempo de ejecución de nuestro proyecto. Una vez pulsado Play, se podrá probar nuestro juego (en la pestaña game antes descrita) tal y como se vería en la plataforma para la que está destinado.
8. **Inspector:** Una de las pestañas más importantes en esta ventana. Gracias a esta ventana se pueden añadir y configurar componentes al elemento seleccionado de manera rápida y sencilla. Un componente es una cualidad que tienen en común varios elementos, por lo tanto, está encapsulada en formato “component” para su

sencilla reutilización. Para añadir un componente lo único que hay que hacer es pulsar en el botón “Add component” como se verá más adelante. Los componentes más comunes en la creación de videojuegos son:

- **Animaciones:** Contienen una secuencia finita de cambios de lugar y rotaciones que representan la realización de una acción,
- **RigiBody:** Permite que nuestro elemento pueda actuar bajo los efectos de la física. Gracias a este componente, el personaje puede responder a distintas fuerzas como la gravedad o las diseñadas por el programador vía scripting.
- **Collider:** Proporciona al elemento la capacidad de poder chocar con otros elementos. Este elemento es invisible en tiempo de ejecución y no necesariamente tiene que tener exactamente la misma forma que el elemento en sí. El collider no se deforma debido a los choques con objetos. Este componente también se puede configurar como “sensor” marcando la propiedad “Is Trigger”. De esta manera, el collider no chocará físicamente con otros colliders, pero a cambio, servirá de ayuda al programador ya que gracias a él se podrá trabajar con numerosas funciones trigger [30] , como OnTriggerEnter, la cual se dispara cuando un elemento entra en contacto con el trigger.
- **Scripts:** Gracias a la funcionalidad “Drag and drop” (arrastrar y soltar), se puede arrastrar un Script, previamente creado por un programador, dentro de un elemento para proporcionarle “vida”.
- **Material:** Proporciona color a nuestro elemento. Este color no tiene por qué ser un color plano, puede ser una textura (como las que se suelen utilizar para los terrenos) o un diseño, como el utilizado para objetos que no están pintados únicamente de un color (un humano compuesto por diferentes partes, ojos, boca, camisa, piel, pantalones… cada parte tiene un color distinto).
- **Transform:** Componente que determina la posición, escalado y rotación de cada objeto de la escena. Cada GameObject tiene su transform.

9. **Proyect:** En esta pestaña se puede observar cómo se tiene organizado nuestro proyecto fuera de Unity, es decir, a nivel de Scripts, carpetas y objetos. En la subpestaña assets se puede ver el contenido de la carpeta seleccionada**.**

Una vez vistos los puntos más importantes de la ventana principal, se procederá a ver los pasos que se han llevado a cabo para la creación del entorno de pruebas básico.

En primer lugar, ya que se quiere trabajar en un entorno que va a utilizar machine-learning, lo primero se debe realizar es crear un *GameObject Academia* con su respectivo *GameObject Brain*. Para ello, nos situaremos en la pestaña Hierarchy, se hará click derecho y crearemos un nuevo GameObject vacío al que llamaremos academia. Tras esto, se seleccionará dicho objeto, y se creará otro GameObject hijo

[30] Funciones que se ejecutan automáticamente al cumplirse su condición de diseño.

del anterior al que denominaremos Brain (Véase Ilustración 18). La configuración de estos objetos se encuentra detallada en el anexo 7.4 (para el objeto academia) y anexo 7.5 (para el objeto Brain).

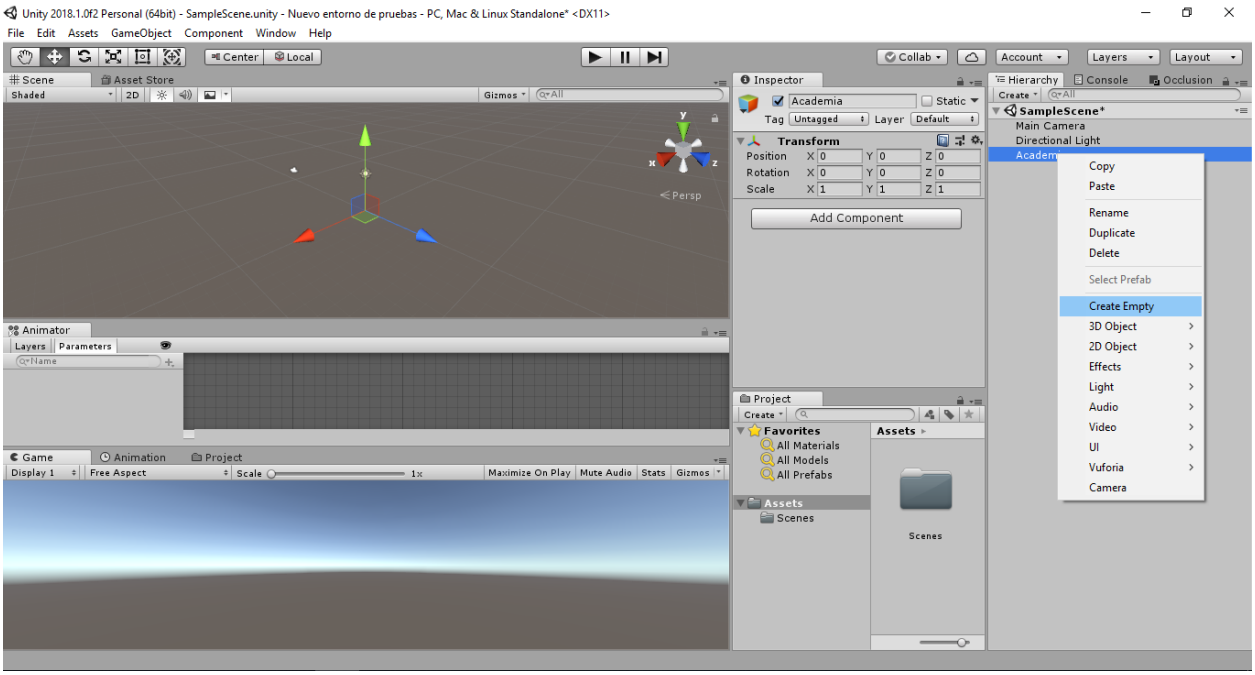

Ilustración 18: *Creación del objeto Brain.*

Tras esto, se creará la física de nuestro escenario. El primer paso es crear un objeto padre donde agrupar todos los elementos físicos que componen nuestro proyecto. Para ello creamos otro GameObject vacío, asegurándose de que dentro de su transform se indicada la posición (0,0,0) (Véase Ilustración 19) y lo nombraremos, por ejemplo: “escenario”.

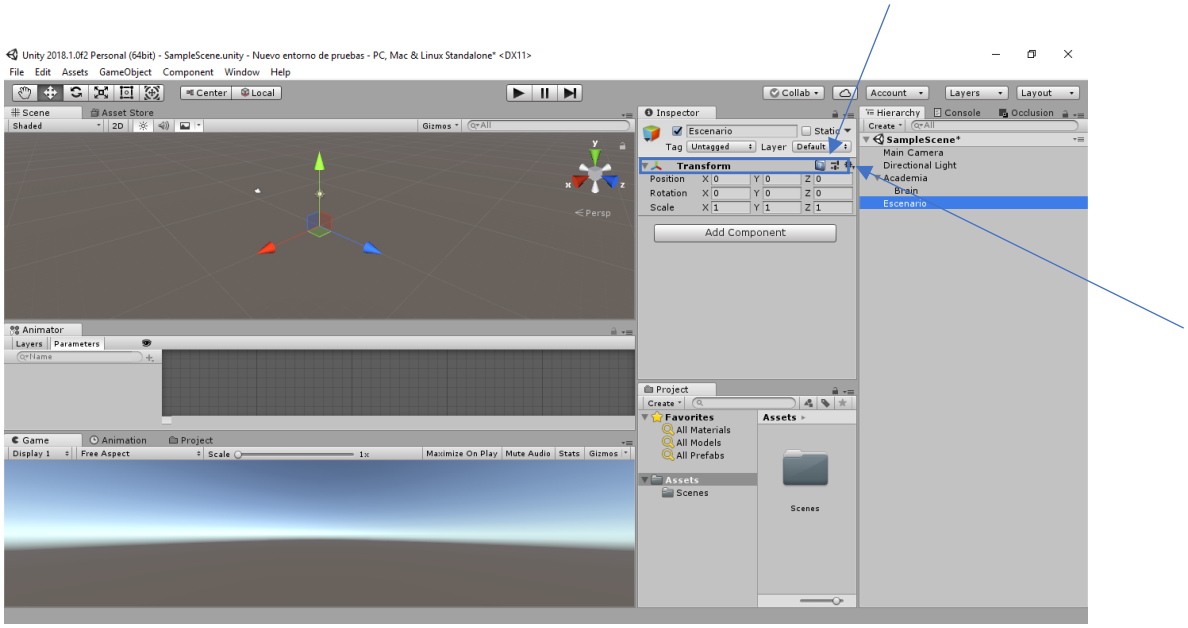

Ilustración 19: *Creación del objeto escenario.*

A continuación, se procederá a crear la superficie donde se situarán los distintos objetos de nuestro proyecto. En este caso, como se desea un entorno básico, nuestro suelo será una superficie totalmente lisa. Para crear dicha superficie, dentro del objeto escenario, se creará un Objeto3D del tipo Cubo (Véase Ilustración 20), se le proporcionará la escala que se le considere apropiada y se desplazará en el eje “y” a la posición -1 (para que no interfiera con los demás objetos, ya que, en el momento de su creación, aparecen por defecto en la posición 0 del eje “y”) (Véase Ilustración 21). Tras esto, se le proporcionará la escala que se considere apropiada y nos aseguramos de que contenga el componente Box collider (un collider rectangular), en caso de no tenerlo, se lo tendremos que añadir. Como para nuestros proyectos no se utilizarán funciones trigger basadas en la interacción con el suelo, no se marcará la casilla *“Is Trigger”*.

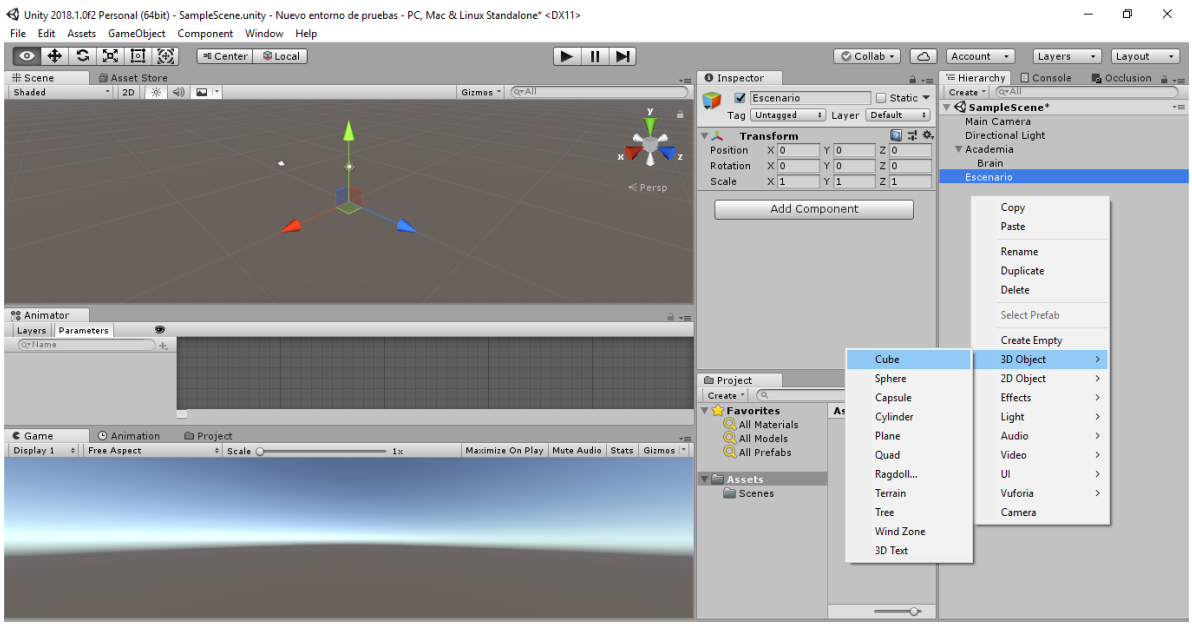

Ilustración 20: *Creación del objeto suelo.*

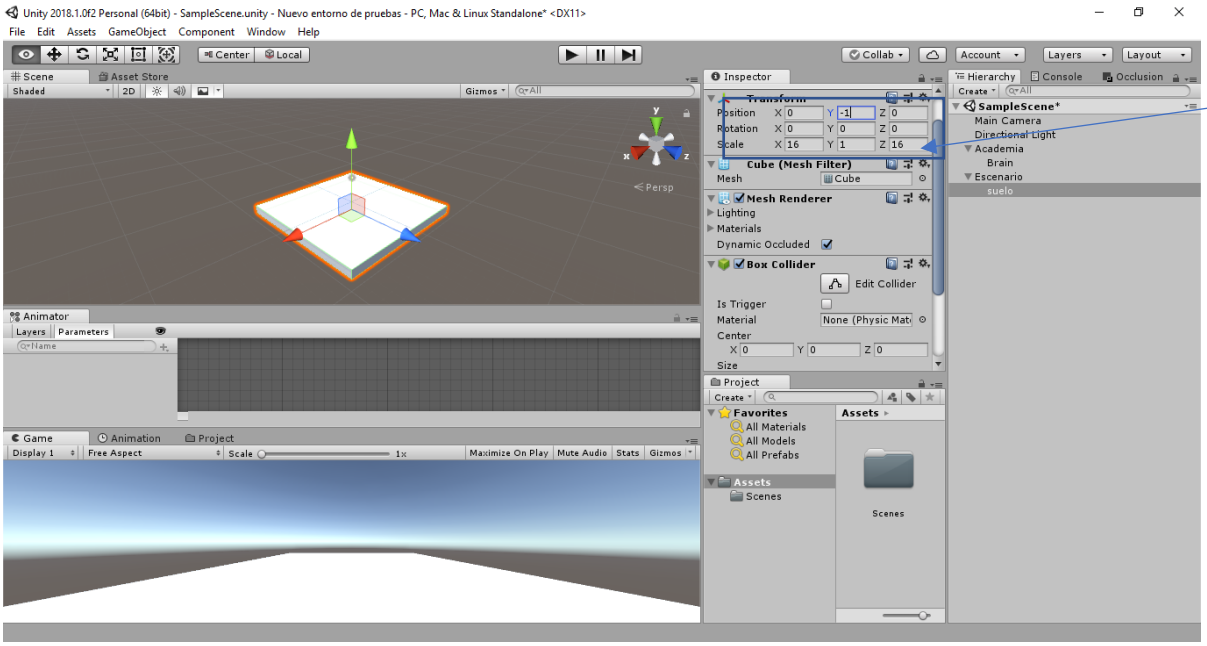

Ilustración 21: *Configuración del objeto suelo.*

Una vez hecho esto, se procederán a crear los agentes que van a participar en nuestro desarrollo de la inteligencia artificial. Como nuestro objetivo es crear un entorno de prueba lo más básico posible para no malgastar recursos en la carga de elementos costosos, tanto nuestro agente como nuestro objetivo serán dos cubos lisos. A estos cubos, además de añadirles el componente "box collider", se les marcará la casilla *"Is Trigger"* (ya que será necesario saber cuándo el cubo agente ha colisionado con el cubo objetivo) y se les añadirá el componente RigiBody (es importante marcar la casilla "*Use Gravity"* para que la gravedad afecte a nuestro objeto) (Véase Ilustración 22).

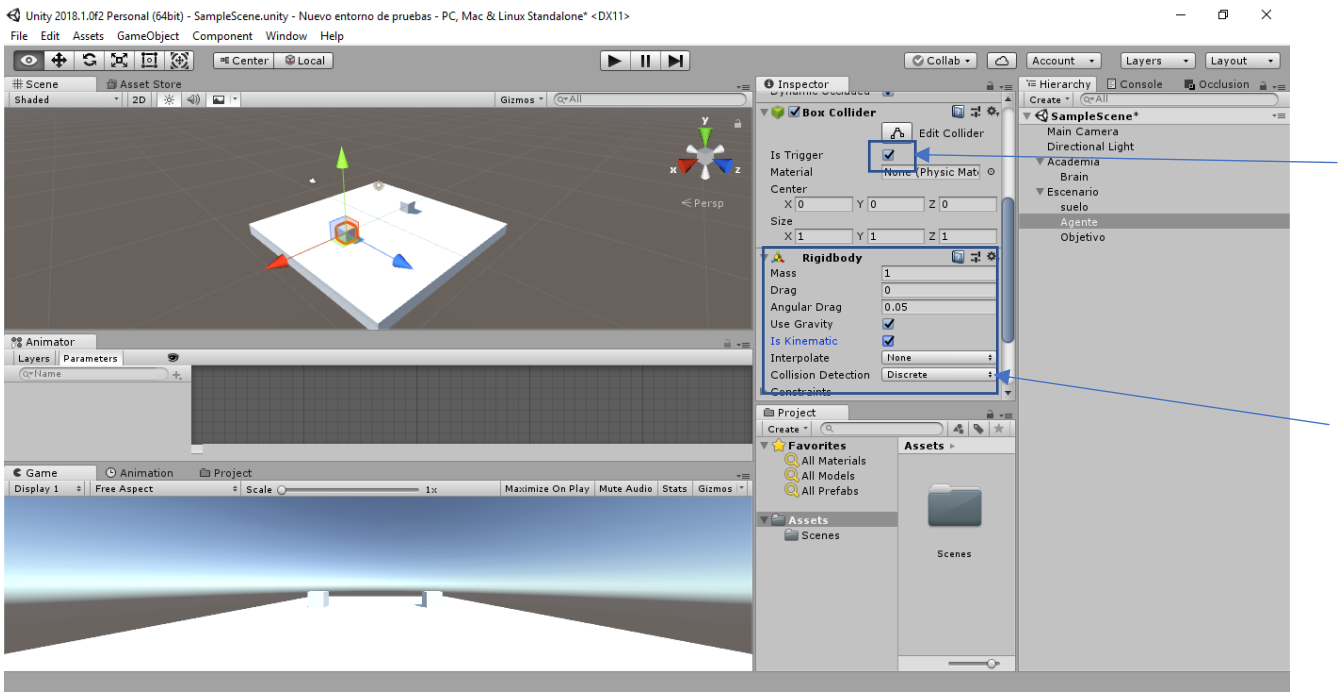

Ilustración 22: *Creación del objeto Agente.*

Como se ha comentado anteriormente, se requerirá que estos últimos cubos tengan un trigger para saber cuándo conectan entre sí, pero para poder llevar a cabo esto, se necesitará "algo" que nos indique quién es quién, para que, en caso de existir múltiples objetos, saber que el objeto alcanzado es el deseado. Para ello, Unity cuenta con una mecánica de "tags"[31], que se pueden configurar para luego utilizarse a la hora de programar scripts. Para añadirle el tag "objetivo" a uno de los dos cubos, se seleccionará el cubo deseado y pulsará en *"tag"* y después en el tag deseado (Véase Ilustración 23). En caso de querer un tag que no está creado por defecto, como en este caso, pulse *"Add tag"* y en la lista de tags pulse el botón *"+"* (Véase Ilustración 24), escriba el nombre deseado para su tag y pulse *"Save"*. Tras la creación, al pulsar en cualquier elemento del proyecto ya debería aparecer el nuevo tag disponible (Véase Ilustración 25).

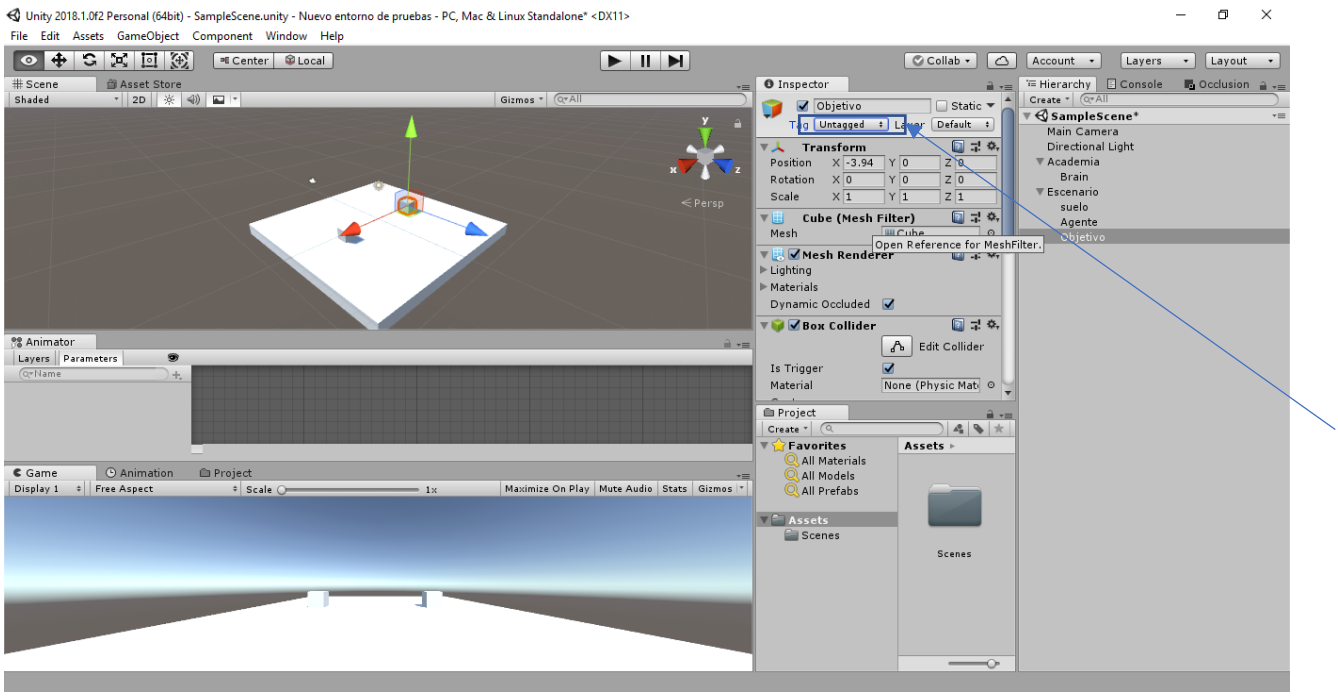

Ilustración 23: Selección de tag.

[31] Tag: Etiqueta que dictamina la "especie" de nuestro objeto. Ej: enemigo, aliado, objetivo, entorno, etc

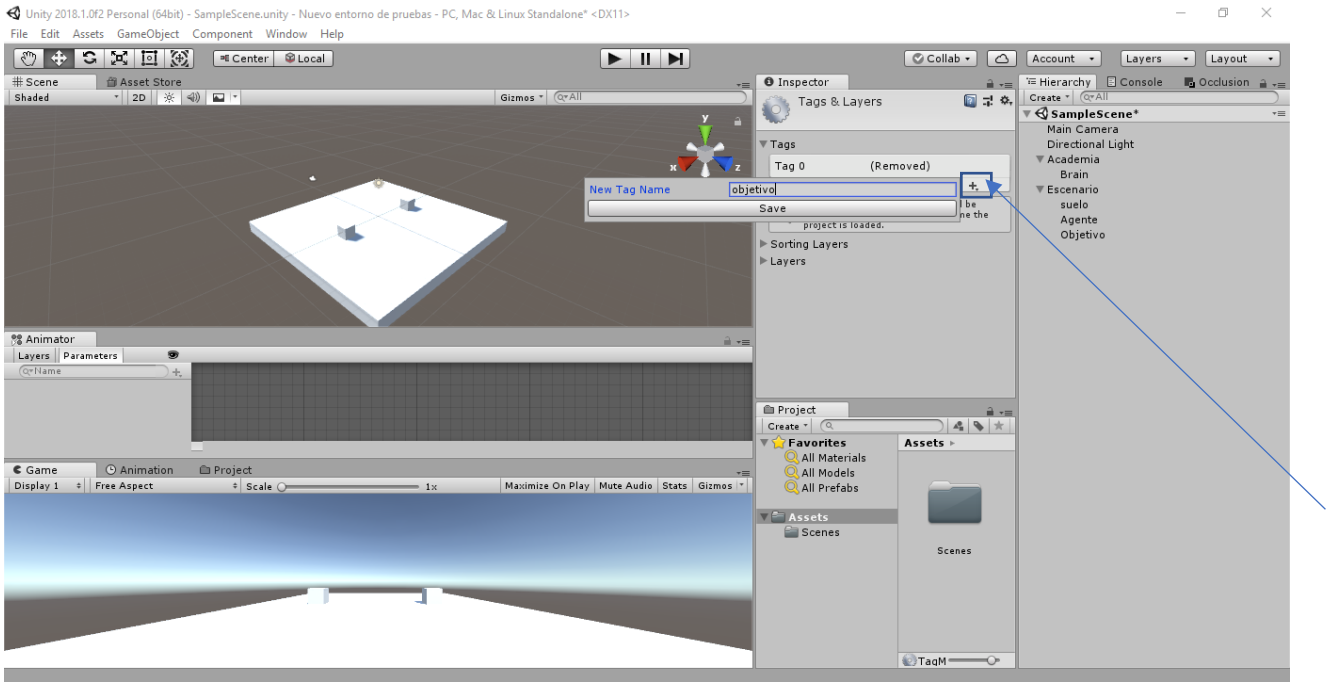

Ilustración 24: *Creación de tag.*

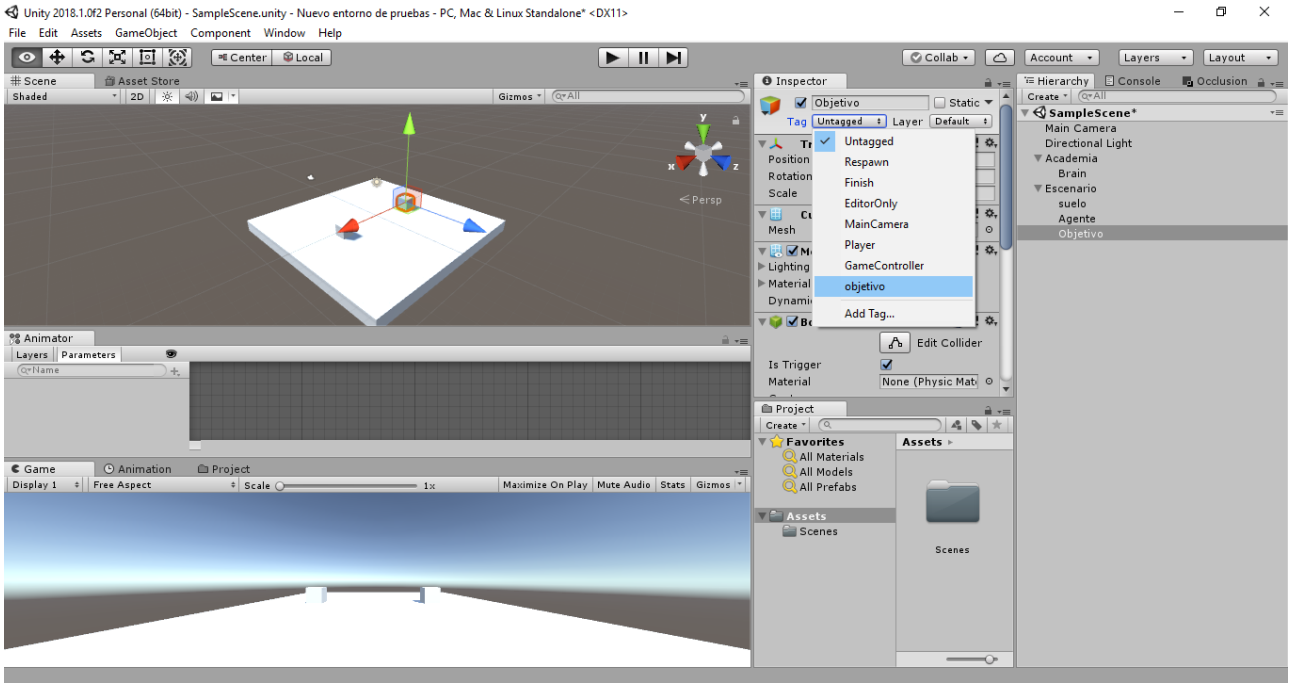

Ilustración 25: *Nuevo tag creado.*

Una vez hecho esto, para terminar la parte no controlada por código, solo faltaría dar color a los distintos objetos de nuestro entorno para poder diferenciarlos a simple vista. Para esto, nos hará falta un material. Para crearlo, en la pestaña *"Assets"*, hará click derecho en *"Create"* y luego en *"Material"* (Véase Ilustración 26).

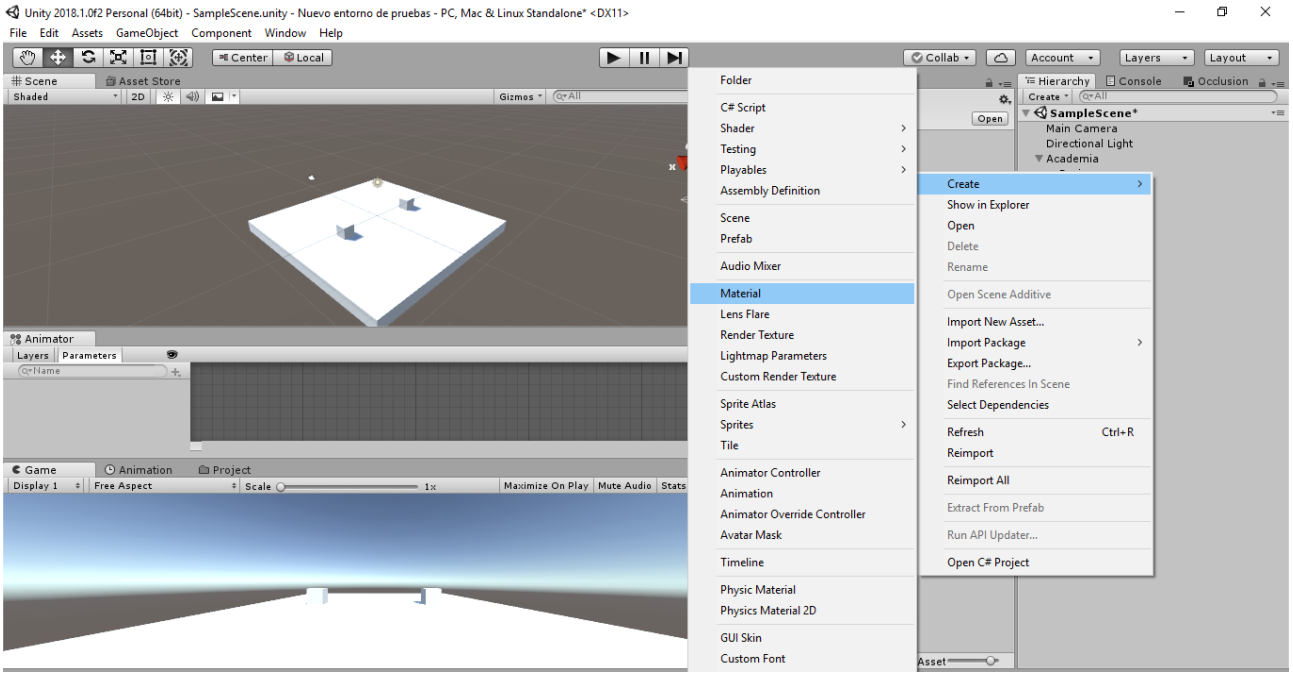

Ilustración 26: *Creación de material.*

Una vez creado el material, haga doble click en él y seleccione el color deseado para nuestro elemento. Finalmente, arrastre el material creado a nuestro elemento para darle color (Véase Ilustración 27).

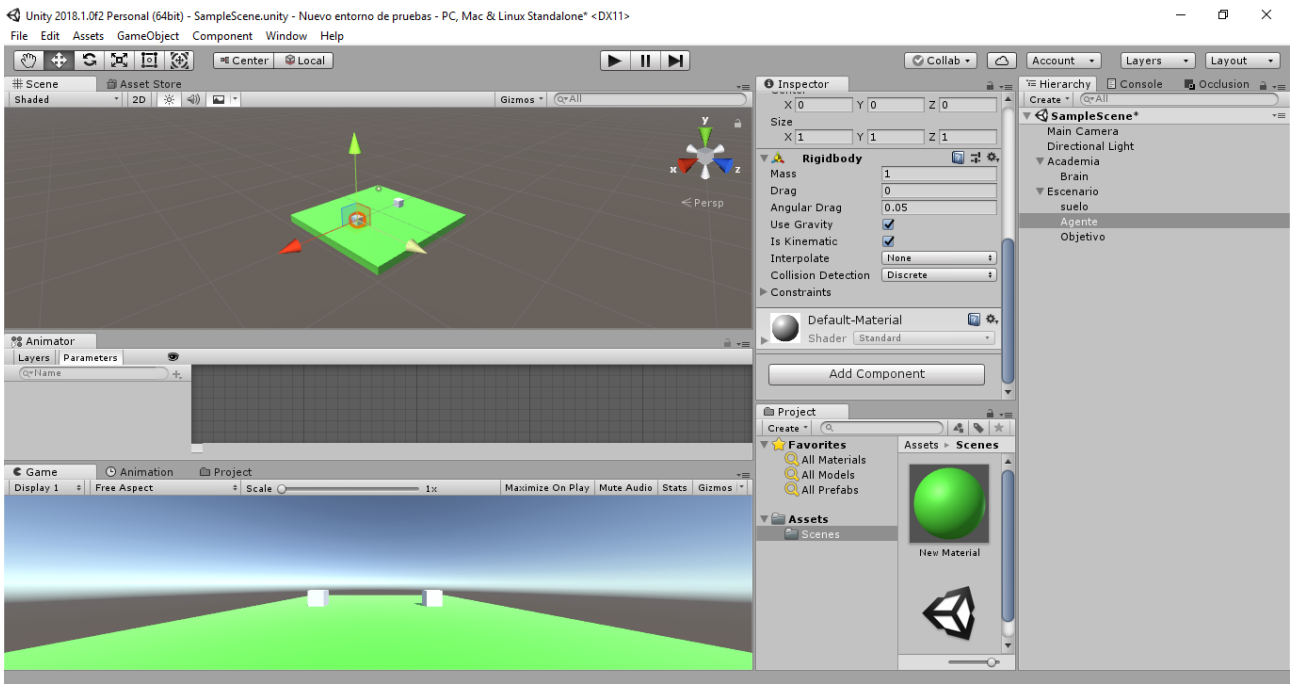

Ilustración 27: *Color cambiado con éxito.*

6. Modelado de las inteligencias artificiales implementadas.

En este apartado se describirán las distintas inteligencias artificiales implementadas en entornos de prueba, y su implementación dentro del juego. Para llevar esto a cabo, se explicará en profundidad como se han realizado, y se mostrarán porciones del código implementado en los scripts que modelan las IAs.

6.1 Máquina de estados finita.

En nuestro juego se han utilizado máquinas de estados formar todos los Animators de los personajes que forman la escena del juego y para modelar parte del comportamiento de algunas criaturas como el Boss [32] Dragón Púrpura (Véase Ilustración 28). El comportamiento de éste viene dado por una máquina de estados finita que se explicará a continuación, junto a una serie de pasivas que aumentarán la dificultad del encuentro.

Ilustración 28: *Enemigo Dragón púrpura*.

El comportamiento general de este enemigo viene dado, por una máquina de estados (Véase Ilustraciones 29 y 30) con los siguientes estados:

- **Idle:** estado inicial. El personaje se mantiene en reposo. Si durante este estado, entrara un enemigo en su campo de visión, el dragón pasará al estado Perseguir. En caso contrario, tras un breve Time Out, dejará de estar estático y pasará al estado Patrullar.

- **Patrullar:** El personaje patrulla bajo una ruta establecida. Si durante este estado, entrara un enemigo en su campo de visión, pasará al estado Perseguir. En caso

[32] Boss: enemigo final que suele encontrarse al final de los niveles de videojuego.

contrario, tras un breve Time Out, dejará de patrullar para volver al estado Idle. Para que el personaje esté siempre la zona del mapa destinada a su encuentro, nos ayudamos de un boolean auxiliar que indica si debe patrullar en sentido horario o antihorario. Esta variable cambia su valor cada vez que se entra al estado Patrullar.

- **Perseguir:** El personaje persigue al enemigo más cercano dentro de su campo de visión[33]. Si el personaje logra alcanzar a su objetivo, es decir, logra que su objetivo esté en su campo de acción, se pasará al estado Atacar. En caso de que no se logre alcanzar al objetivo, el personaje se mantendrá persiguiéndolo hasta que este salga de su campo de visión, donde volvería al estado Idle.

- **Atacar:** El personaje ataca a su objetivo. Si el objetivo muere, el personaje atacará al siguiente objetivo más cercano dentro de su campo de acción y en caso de no existir más enemigos, pasará al estado Idle. En caso de que el objetivo salga del campo de acción[34], el personaje cambiará su objetivo y atacará al enemigo más cercano que se encuentre dentro de dicho campo. En caso de que el objetivo salga del campo de acción y no se encuentren más enemigos en él, el personaje pasará al estado Perseguir.

- **Muerte:** estado final. Estado accesible desde cualquier estado, se accede a él una vez la vida del personaje llega es igual o inferior a 0. En este estado el personaje muere y desaparece de la escena.

Las siglas RDV y RDA que aparecen en las transiciones del gráfico significan Rango De Visión y Rango De Ataque respectivamente.

[33]

[34] La forma en la que se han calculado los campos de visión y de acción se explican la sección Anexo.

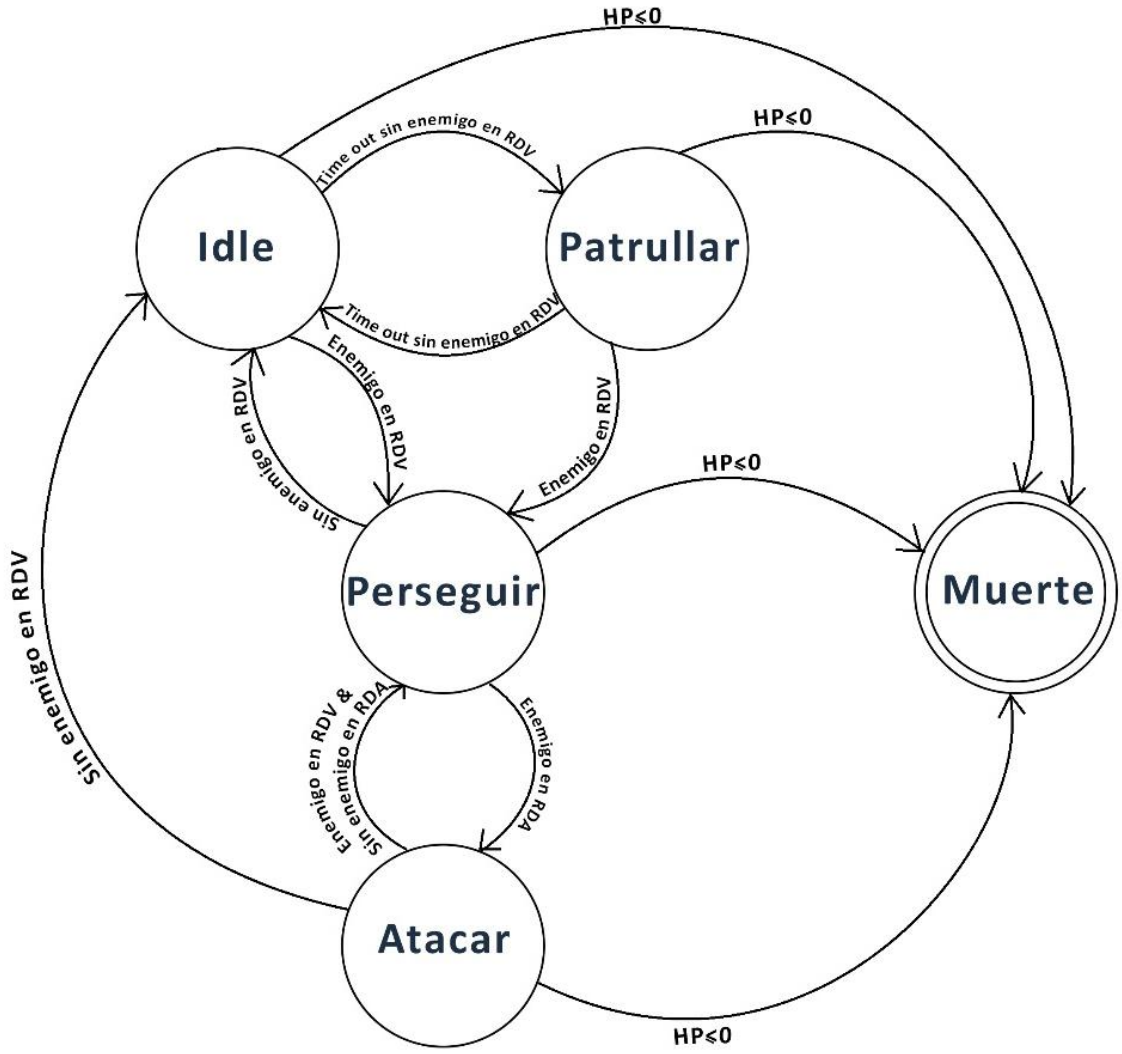

Ilustración 29: *Máquina de estados que modela el comportamiento del enemigo dragón púrpura.*

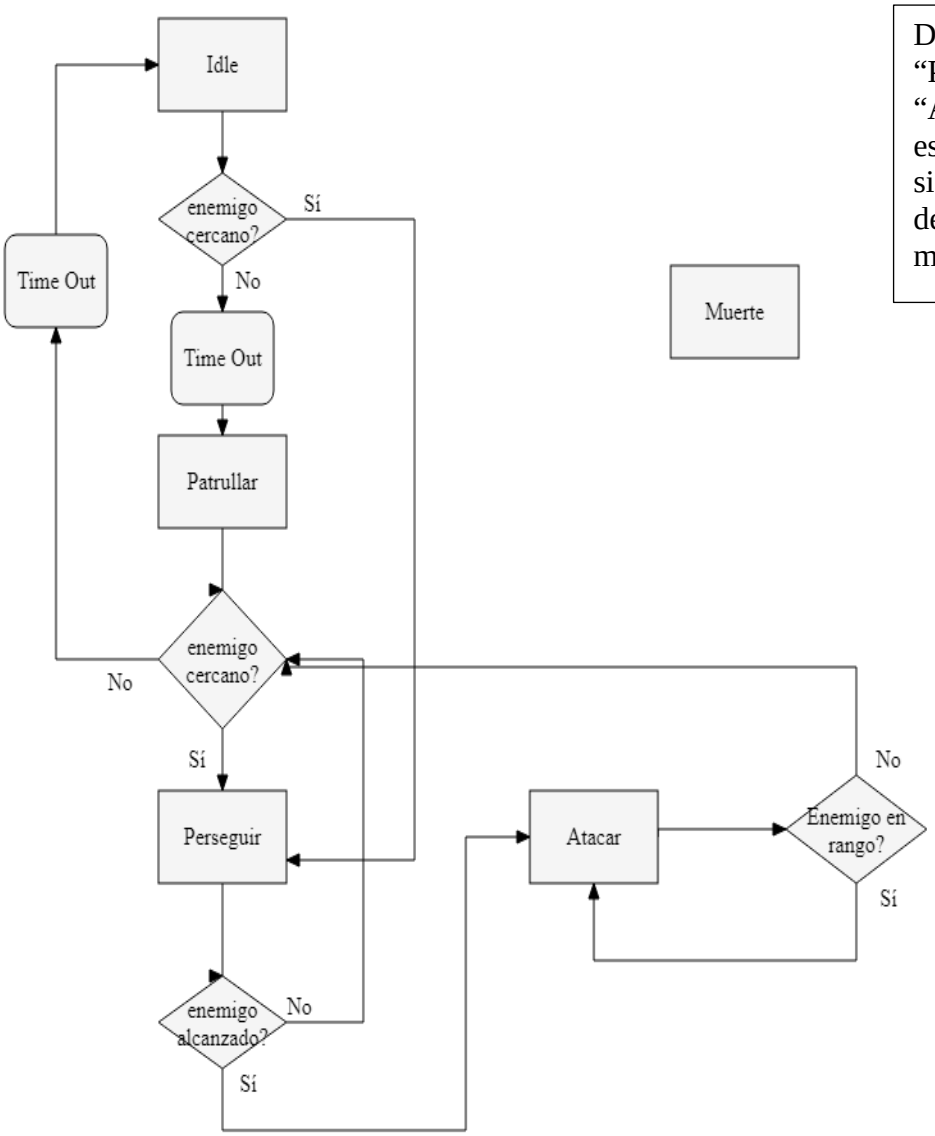

Ilustración 30: *Flujograma equivalente a la máquina de estados finita encargada de controlar a Dragón Púrpura.*

Como se explicó en dicho estado, el estado final Muerte, es accesible desde cualquier estado de la máquina de estados siempre y cuando la vida del personaje sea igual o menor a cero, pero se ha decidido ocultar estas transiciones en la representación dada por el flujograma para tener una mayor claridad.

A continuación, se verán fragmentos de código de la implementación:

```
switch (state)
{

    case 0://Idle
        Idle();
        break;
    case 1://Patrol
        Patrol();
        break;
    case 2://Perseguir
        Perseguir();
        break;
    case 3://Attack
        Attack();
        break;
    case 4://Attack2
        Attack2();
        break;
    case 5://Die
        Die();
        break;
}
```

Ilustración 31: *Switch que modula la máquina de estados.*

La máquina de estados ha sido implementada mediante un Switch (Véase Ilustración 31) donde dependiendo de una variable pública *"state"* se indica en qué estados se encuentra nuestro personaje. A continuación, se explicará el proceso de cómo el dragón realiza un ataque:

- Tras cumplirse una serie de condiciones, la variable *"state"* valdrá 3, entrando en la entrada tres del Switch. Está entrada llamará a la función *Attack()* (Véase Ilustración 32), la cual fuerza la animación del dragón a la asignada a dicha acción. Esto es importante porque esa animación tiene asociado un action event[35](más información en el apartado 7.8) que llamará al script *HurtPlayer* que a su vez hará uso de la función *HurtPlayer()* alojada en el Script *PlayerHealthManager*. Como puede ver, las funciones se han implementado de una manera modular, similar a la utilizada en los modelos vista-controlador[36], para poder reutilizar la máxima cantidad de scripts posibles.

[35] Action Event: Evento que se ejecuta cada vez que se ejecuta la animación.
[36] Modelo-vista-controlador (MVC): Es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

```
void Attack()
{
    GetComponent<Animator>().SetTrigger(Animator.StringToHash("Basic Attack"));
    transform.Translate(new Vector3(0, 0, 1) * Time.deltaTime);


}
```

Ilustración 32: *Función Attack().*

Una cosa interesante es saber cómo detecta el personaje, Dragón Púrpura en este caso, a su enemigo más cercano. Para llevar a cabo esta acción, el Dragón analiza todos los enemigos existentes en el mapa, y compara su posición con la de cada uno de ellos, quedándose con la que resulte menor (Véase Ilustración 33).

```
Transform enemigoMasCercano(GameObject[] enemigo)
{
    Transform target = null;
    float menorDistancia = Mathf.Infinity;
    Vector3 posicionActual = transform.position;
    foreach (GameObject objetivoPotencial in enemigo)
    {
        Transform prueba = objetivoPotencial.transform;
        Vector3 direccionAlObjetivo = prueba.position - posicionActual;
        float SqrtAlObjetivo = direccionAlObjetivo.sqrMagnitude;
        if (SqrtAlObjetivo < menorDistancia)
        {
            menorDistancia = SqrtAlObjetivo;
            target = prueba;

        }
    }
    return target;
}
```

Ilustración 33: *Búsqueda del enemigo más cercano.*

6.2. Aprendizaje por refuerzo.

A continuación, se mostrará la implementación de esta metodología en un entorno separado del juego:

El problema propuesto consiste en: dado un espacio, donde se generan con una posición aleatoria un personaje (representado por un cubo azul) y un objetivo (representado por un cubo blanco), el personaje debe ser capaz de encontrar al objetivo, llegar a él de la manera más eficiente posible y tocarlo. Al agente se le otorgará una recompensa grande

cada vez que toque al objetivo, una recompensa leve cada vez que reduzca su distancia con el objetivo y un castigo cada vez que aumente su distancia con el objetivo.

Para agilizar el proceso de aprendizaje, se ha optado por duplicar varias veces el escenario de pruebas, para aumentar el número de datos recogidos de forma simultánea. En este caso, se crearon 9 escenarios de prueba, para aumentar por 9 la velocidad de aprendizaje (Véase Ilustración 34).

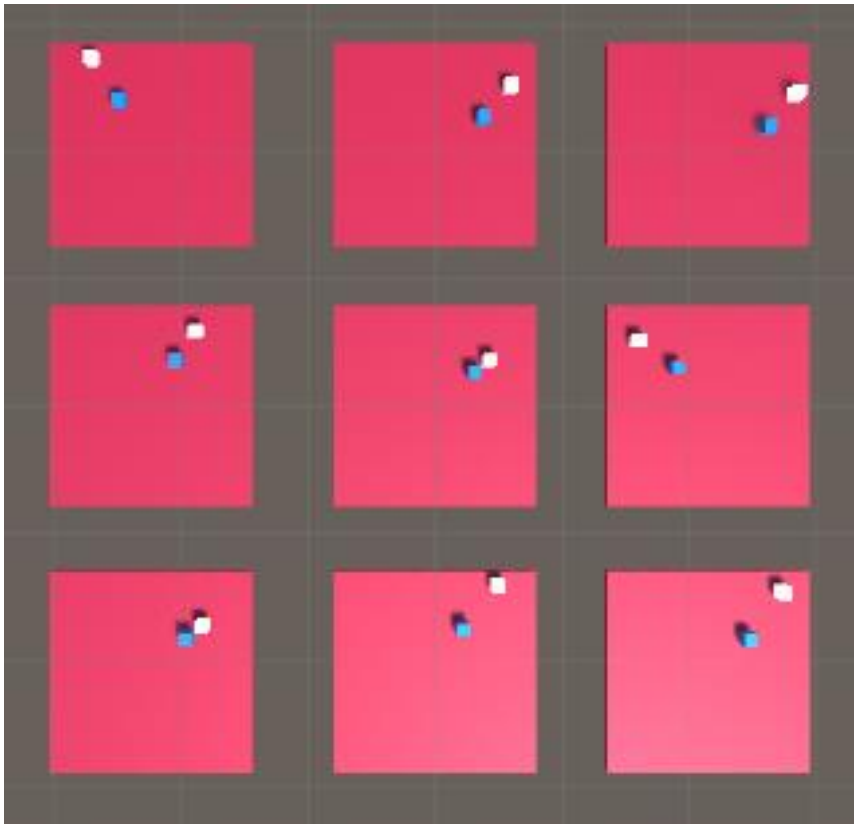

Ilustración 34: *Captura aérea de los escenarios de prueba para aprendizaje por refuerzo.*

Tras configurar nuestro entorno tal como se explica en el anexo, programar los scripts y poner en marcha el entrenamiento, se obtendrá algo semejante lo mostrado en la ilustración 35:

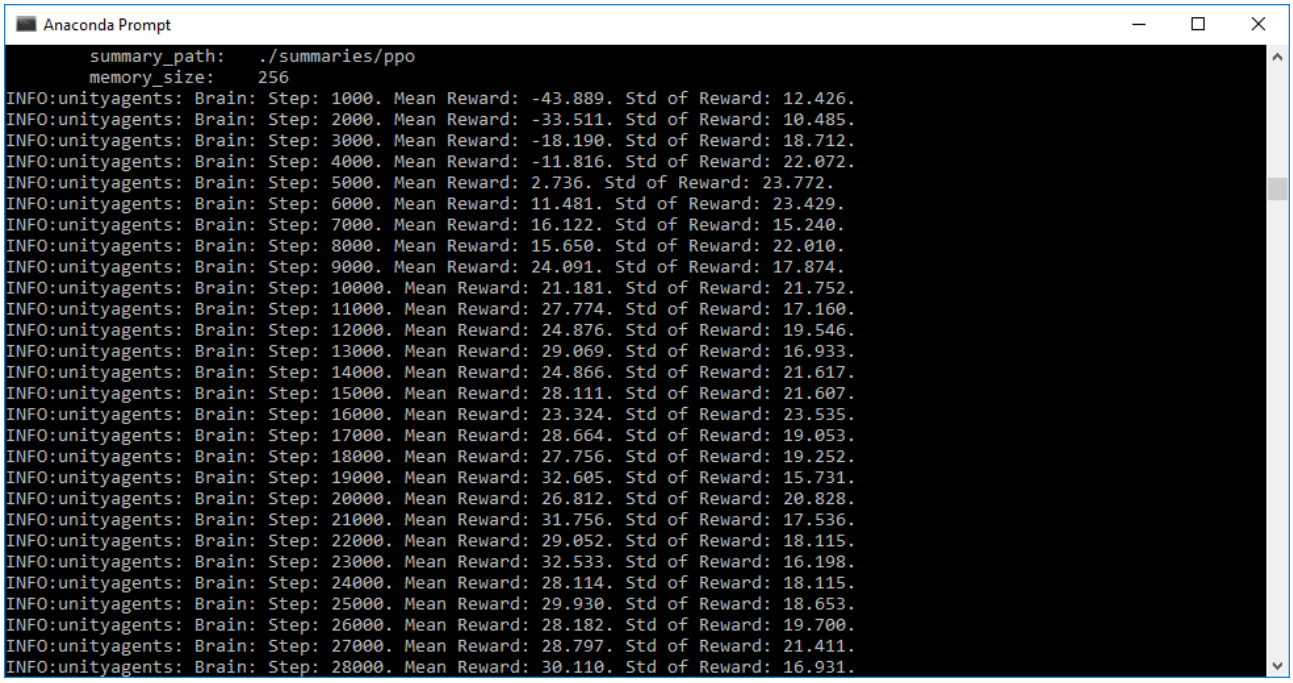

Ilustración 35: *Salida de Anaconda Prompt tras un tiempo entrenando bajo aprendizaje por refuerzo.*

Como se puede observar, conforme avanza el tiempo, y el agente ha entrado más tiempo, la recompensa media va aumentado, aunque a veces decrece un poco.

¿Qué significa este decrecimiento?

Estos decrecimientos intercalados se deben a que cada vez que el entrenamiento llega a los pasos máximos indicados, se resetea el entrenamiento. Entonces, todos los agentes que se han quedado con su entrenamiento a mitad se quedan con la recompensa que habían obtenido hasta el momento. Estos valores se incluyen a la hora de hacer la media de recompensas, lo que determina una recompensa media menor al final de cada entrenamiento. Este problema se puede minimizar incrementando el número máximo de pasos. La cantidad de pasos elegida debe ser lo suficientemente grande como para que el agente pueda encontrar a su objetivo, y a su vez, lo suficientemente pequeño como para que el agente no pueda quedarse mucho tiempo perdido.

Por último, procedemos a comentar una porción de código (Véase Ilustración 36):

```
public override void CollectObservations()
{
    AddVectorObs(transform.localPosition.x);
    AddVectorObs(transform.localPosition.z);
    AddVectorObs(Objetivo.transform.localPosition.x);
    AddVectorObs(Objetivo.transform.localPosition.z);
}

public override void AgentAction(float[] vectorAction, string textAction)
{
    float distanciaInicialAlObjetivo = Vector3.Distance(Objetivo.transform.localPosition, transform.localPosition);
    float horizontal = 0, vertical = 0;

   if(brain.brainParameters.vectorActionSpaceType == SpaceType.continuous)
    {

        horizontal = Mathf.RoundToInt(Mathf.Clamp(vectorAction[0], -1, 1));
        vertical = Mathf.RoundToInt(Mathf.Clamp(vectorAction[1], -1, 1));

    }else if (brain.brainParameters.vectorActionSpaceType == SpaceType.discrete)
    {
        switch ((int)vectorAction[0])
        {
            case 0:
                horizontal = 1;
                break;
            case 1:
                horizontal = -1;
                break;
            case 2:
                vertical = 1;
                break;
            case 3:
                vertical = -1;
                break;
        }
    }
```

Ilustración 36: *Porción de código del proyecto destinado al aprendizaje por refuerzo.*

En la foto podemos observar el método CollectObservations, gracias al cual se le indicará a la red neuronal la posición del agente y del objetivo, para que pueda saber hacia qué dirección debe moverse el agente.

Tras esto, tenemos el método AgentAction (método llamado en cada paso de la simulación), que es el causante de guiar a nuestro agente en función de los datos recibidos por el vector de acciones. Antes de empezar a analizar el comportamiento y ver si se debe dar una recompensa o un castigo, hay que analizar en el tipo de espacio en el que nos encontramos. En Unity se puede trabajar en dos tipos de espacio, discreto y continuo. En nuestro caso, como se desea que se comporte igual, sea cual sea el tipo de espacio, se redondeará los valores del espacio continuo a ‘1’ y ‘-1’, para que se comporte igual que un espacio discreto.

```
        if (distanciaFinalAlObjetivo < distanciaInicialAlObjetivo)
        {
            AddReward(0.1f);
        }
        else
        {
            AddReward(-0.3f);
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Objetivo"))
        {
            AddReward(30f);
            Done();
        }
    }
```

Ilustración 37: *Segundo fragmento de código de aprendizaje por refuerzo.*

En la ilustración 37 se puede observar cómo se han repartido las recompensas, por un lado, cuando el agente se desplaza en dirección al objetivo o llega a tocarlo, gana una recompensa positiva, por otro lado, si el agente se aleja, gana una recompensa negativa. Para que el aprendizaje sea un éxito, el castigo que se entrega por moverse debe ser varias veces mayor que la recompensa obtenida al realizar dicha opción, para que el agente descarte realizar ese tipo de movimiento. Por otra parte, si el agente consigue alcanzar al objetivo, la recompensa debe ser lo suficientemente grande para marcar un punto de inflexión en el entrenamiento, y que, a partir de este momento, las rutas probadas empiecen a alcanzar siempre al objetivo.

6.3. Curriculum learning.

En nuestro ejemplo práctico, el agente, representado como un cubo azul, deberá llevar el objetivo, representado como un cubo naranja, a la meta, representada con la textura de un tablero de ajedrez (Véase Ilustración 38):

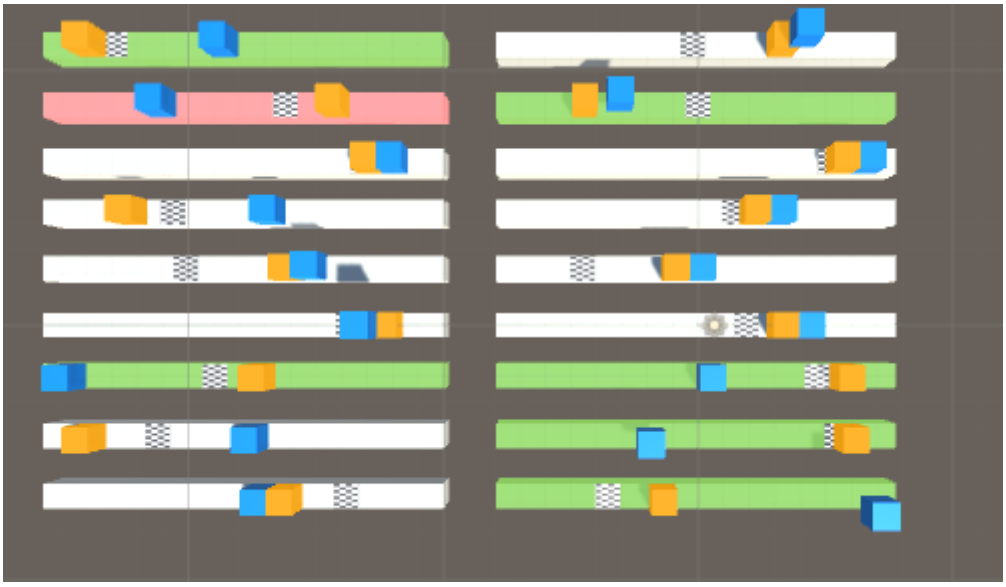

Ilustración 38: *Captura aérea de los escenarios de prueba para curriculum learning.*

A continuación, se verá y comentará una sección de código de este proyecto:

- Con fin de ser más eficiente, y agilizar el entrenamiento, en lugar de pasar la posición exacta del agente y el objetivo al vector de observación, se le envía la posición de manera relativa. Esto agiliza el rendimiento por dos motivos: por un lado, disminuye el tamaño del vector de observación, y, por otro lado, cuando se envían posiciones exactas, esos datos solo pueden ser utilizados para ese caso. En cambio, al trabajar con posiciones relativas, los datos pueden ser reutilizados en caso posteriores similares (Ver ilustración 39). Por ejemplo, si nuestro agente se encuentra en la posición (1,0,0), nuestro objetivo en la posición (3,0,0) y nos movemos una unidad positiva en el eje X, es decir, nos desplazamos a la posición (2,0,0), obtendríamos una recompensa ya que se ha reducido la distancia al objetivo. En caso de trabajar con posiciones absolutas, esos datos recogidos solo se utilizarán cuando los elementos estén en esas posiciones exactas. En cambio, si trabajamos con posiciones relativas, en este caso la posición objetivo sería (+2, 0, 0) respecto a la posición de nuestro agente y los datos recogidos podrían reutilizados por ejemplo cuando el agente estuviera en la posición (3, 0, 0) y el objetivo en la (5, 0, 0).

```
public override void CollectObservations()
{
    //Enviamos los paramatros que va a necesitar nuestro vector de observaciones
    //Posicion relativa del cubo frente al agente
    AddVectorObs(Cubo.transform.localPosition.x - transform.localPosition.x);
    //Posicion en y, ya que puede estar saltando
    AddVectorObs(transform.localPosition.y);
    AddVectorObs(Objetivo.transform.localPosition.x > Cubo.transform.localPosition.x ? 1f : 0f);
    AddVectorObs(enSuelo ? 1f : 0f);
}
```

***Ilustración 39**: Envío de posición del objetivo de manera relativa.*

- Se ha utilizado una corrutina para volver el suelo durante un segundo de color verde en caso de que el entrenamiento haya tenido éxito y rojo en el caso de tener error fatal como podría ser caerse de la plataforma o tirar el cubo fuera de ella. Gracias a eso se puede saber a simple vista si el entrenamiento ha ido bien o no aun teniendo múltiples escenarios (Ver Ilustración 40).

```
private IEnumerator CambiarColor(Material material)
{
    mr.material = material;
    yield return espera;
    mr.material = Normal;
}

public void TareaCompletadaExito()
{
    StopAllCoroutines();
    StartCoroutine(CambiarColor(Correcto));
}

public void TareaCompletadaFracaso()
{
    StopAllCoroutines();
    StartCoroutine(CambiarColor(Incorrecto));
}
```

***Ilustración 40:** Fracción del código de la corrutina.*

- Las lecciones que se han implementado han consistido en incrementar la distancia del objetivo a la meta de manera progresiva hasta que el objetivo pudiera aparecer en cualquier parte de la plataforma (Ver ilustración 41).

```
{
    "measure" : "reward",
    "thresholds" : [26, 26, 26, 25, 24, 24],
    "min_lesson_length" : 1,
    "signal_smoothing" : true,
    "parameters" :
    {
        "separacion_max" : [0.21, 0.5, 1, 3, 6.5, 10, 13.5]
    }
}
```

***Ilustración 41:** Reglas establecidas para el entrenamiento por curriculum learning*

Tras llevar a cabo la creación y configuración del entorno y la creación del curriculum y scripts, este es resultado que obtenemos por Anaconda Prompt (véase las ilustraciones 42 y 43):

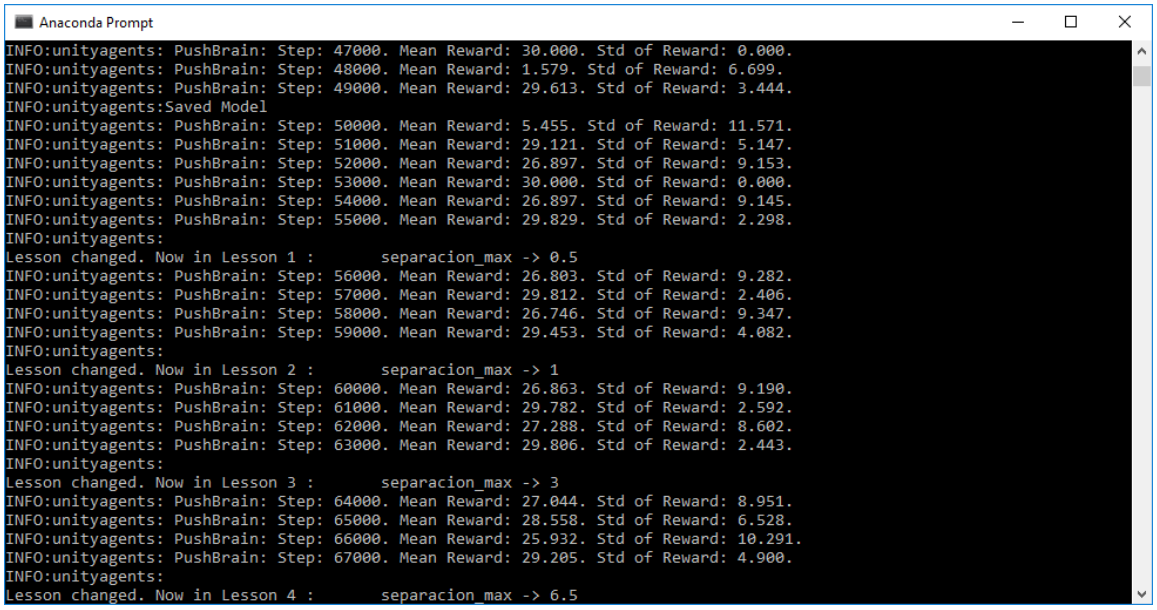

Ilustración 42: *Salida de Anaconda Prompt tras las primeras lecciones bajo curriculum learning.*

.

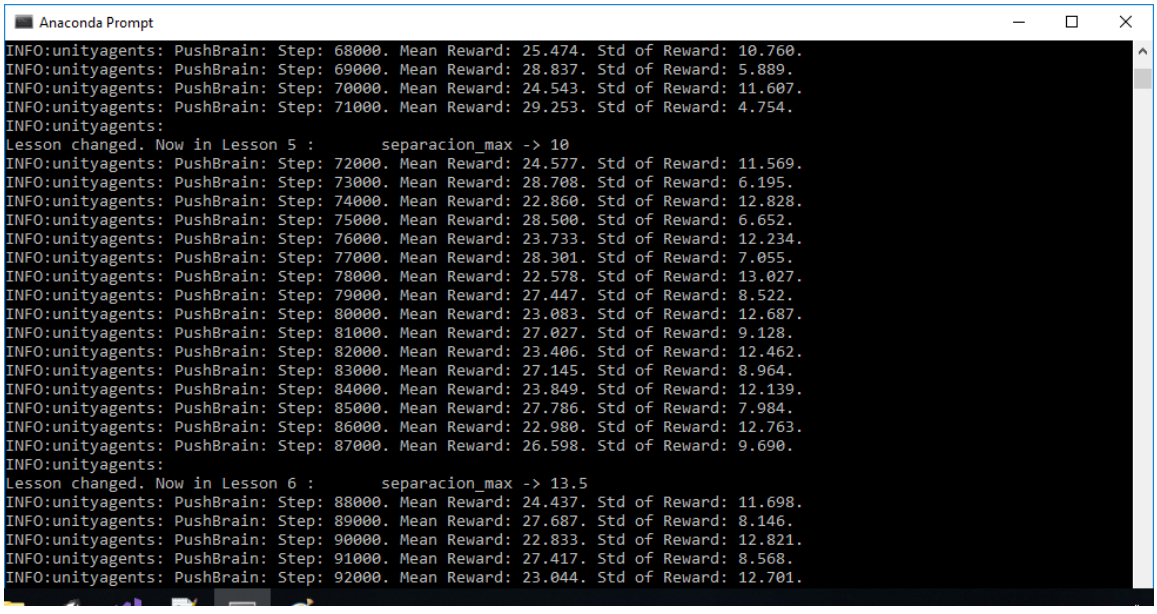

Ilustración 43: *Salida de Anaconda Prompt tras las últimas lecciones bajo curriculum learning.*

Como detalle curioso mencionar que el agente siempre aprenderá primero a introducir el objetivo en la meta empujando hacia un mismo lado, y seguirá intentando tener éxito en su misión empujando hacia ese lado hasta que, sin querer, aprenda que puede entrar en la meta empujando en la otra dirección. A partir de ese momento, el proceso se

agilizará bastante. Esta es una de las razones por la que la primera lección es la que más ha tardado en completarse aun siendo la más sencilla.

7. Configuración de las herramientas.

Capítulo destinado a la aclaración de puntos de interés tenidos en cuenta a la hora de realizar el proyecto.

7.1. Instalación de Anaconda Prompt.

En este apartado se aprenderá como configurar un entorno ml-agents para Unity con Anaconda Prompt.

Una vez instalado Anaconda, incie Anaconda Navigator. Una vez inciado, si pidiera actualizar, actualice, y cierre el programa.

Tras esto, localice las variables de entorno y compruebe que se hayan añadido a la variable del sistema Path las siguientes rutas (en caso negativo, añádalas usted):

```
%UserProfile%\Anaconda3\Scripts
%UserProfile%\Anaconda3\Scripts\conda.exe
%UserProfile%\Anaconda3\
%UserProfile%\Anaconda3\phyton.exe
```

Una vez hecho esto, se procederá a configurar y activar el nuevo entorno. Para ello inicie la aplicación Anaconda Prompt. El primer comenda que deberá escribir será:

```
conda create -n ml-agents python=3.6
```

Con este comando, crearemos un entorno python 3.6 llamado ml-agents. Tras esto, se mostrarán unos paquetes que deberá descargar de Internet y su peso, para continuar pulse *“Y”* y la tecla *Enter*. Una vez que el proceso termine, deberá escribir el siguiente comando para activar el entorno:

```
activate ml-agents
```

Cada vez que quiera hacer algo con los ml-agents, deberá escribir este comando primero para activar el entorno. Una vez ejecutado, si se ha hecho todo bien, le aparecerá entre paréntesis el nombre del entorno donde se encuentra. Ahora vamos a instalar la librería tensorflow, para ello escriba:

```
pip install tensorflow==1.7.1
```

Cuando termine de descargarse e instalarse tensorflow, deberá descargar e instalar los paquetes que le faltan, para ello utilice:

```
git clone https://github.com/Unity-Technologies/ml-agents.git
```

Una vez descargado, sitúese en la carpeta Python que contiene, si el archivo se encontrara en la carpeta Downloads el comando sería:

```
cd C:\Downloads\ml-agents\python
```

*Si en los nombres que forman la ruta hay algún espacio, la ruta debe ir entre comillas

Finalmente, para terminar, escriba:

```
pip install .
```

7.2. Entrenamiento con Anaconda Prompt.

Para entrenar un agente con Anaconda Prompt, lo primero es activar el entorno en el que estamos y luego ir a la carpeta donde se encuentra nuestro entorno:

```
conda activate ml-agents
cd C:\Downloads\ml-agents
```

Tras esto, el comando que se deberá introducir para entrenar nuestro agente dependerá del tipo de entrenamiento que vayamos a utilizar:

-Para aprendizaje por refuerzo:

```
python python/learn.py “ruta del ejecutable creado con al compilar con unity” --train
```

-Para curriculum learning:

```
python python/learn.py “ruta del ejecutable creado con al compilar con unity” --curriculum=”archivoJson” --run –id=”nombre” --train
```

*En ninguno de los dos comandos las comillas son necesarias, están puestas en esta descripción para indicar que eso es un campo que variará dependiendo de donde alojemos nuestro proyecto. El archivo JSON deberá estar dentro de la carpeta ml-agents.

7.3. Conceptos de academia y brain.

Durante el transcurso de este documento y en los proyectos realizados se han utilizado los términos academia y brain, a continuación, se procederá a explicar estos objetos:

- El objeto **academia**, se encargará de gestionar todo el proceso de observación y toma de decisiones, haciendo de intermediario entre nuestro proyecto y Python.

- El objeto **Brain**, es un hijo de academia que se encargará de tomar las decisiones en base a las observaciones que se le envíen. Hay que tener en cuenta que, aunque varios agentes tengan el mismo cerebro, no realizarán las mismas acciones, ya que estas dependen de las observaciones que se le estén enviando al cerebro en ese momento.

7.4. Configuración del objeto academia.

A la hora de configurar nuestra academia en Unity se deberá tener en cuenta varios parámetros:

- **Max steps:** Número de pasos máximo que tendrá nuestro entrenamiento.

Los demás valores son configuraciones como la velocidad a la que se mueve el entrenamiento o el tamaño de la pantalla que se genera al entrenar un agente, es decir, son valores que no interfieren con el resultado.

7.5. Configuración del objeto Brain.

Lo primero que debemos configurar es el vector de observaciones, para esto se deberá decidir si el tipo de espacio será continuo o discreto y el tamaño de dicho vector. Si no respetamos estos parámetros a la hora de programar nuestros scripts, el programa no compilará.

Lo mismo habría que hacer con el vector de acción.

Tras esto, nos encontramos con el campo Brain Type, los valores utilizados para este proyecto han sido (Véase figura 44):

- **Player:** En este modo, se le asignan teclas a las acciones que puede hacer un agente y es el programados quien va moviendo el personaje a través de estas teclas. Este siempre ha sido el primer paso para comprobar que el script funcionaba de manera correcta.

- **External:** Utilizado para entrenar la escena a través de Anaconda Prompt.

- **Internal**: Una vez entrenada una escena, se puede utilizar el fichero de extensión ppo generado para darle vida a nuestro agente con la IA construida.

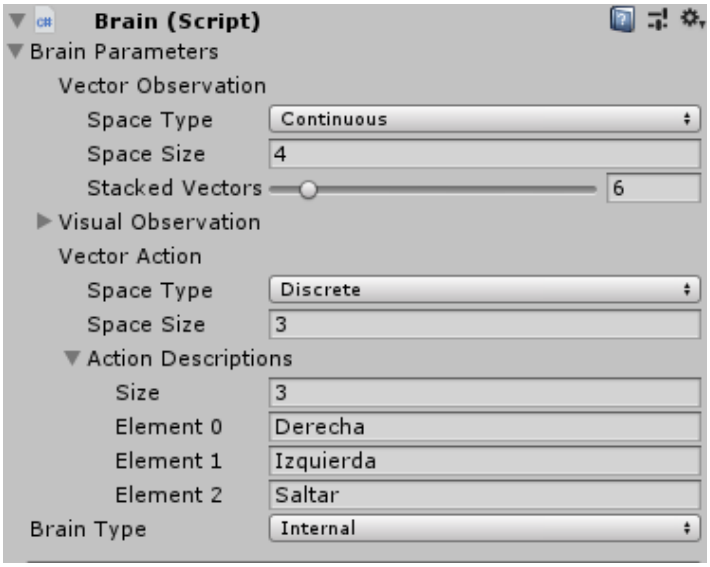

Ilustración 44: *Campos del objeto Brain a configurar a través de Unity.*

7.6. Archivo trainer_config.yaml.

Dentro de la carpeta Python, encontrará el archivo trainer_config.yaml. En este archivo podrá modificar variables que intervienen en el entrenamiento. Entre otras cosas, podrá indicar cosas como el tipo de entrenamiento o el número de pasos que desea que tenga el entrenamiento. Si no indica nada, el proyecto tomará los valores por defecto, si desea que un Brain tome unos valores específicos deberá crear su propio apartado (Véase Ilustración 45).

```
default:
    trainer: ppo
    batch_size: 1024
    beta: 5.0e-3
    buffer_size: 10240
    epsilon: 0.2
    gamma: 0.99
    hidden_units: 128
    lambd: 0.95
    learning_rate: 3.0e-4
    max_steps: 100000
    memory_size: 256
    normalize: false
    num_epoch: 3
    num_layers: 2
    time_horizon: 64
    sequence_length: 64
    summary_freq: 1000
    use_recurrent: false

PushBrain:
    max_steps: 5.0e6
    batch_size: 128
    buffer_size: 2048
    hidden_units: 256
```

Ilustración 45: *Sección del archivo trainer_config donde se aprecian los valores por defectos y los valores que se usaron para el proyecto de curriculum learning.*

7.7. Rango de visión y rango de ataque.

Dependiendo del personaje estas cualidades se han implementado de una de estas dos maneras:

- Calculando matemáticamente la distancia entre el personaje y el objetivo y comparándola con una serie de límites establecidos.

- Usando Colliders triggers: los personajes tienen un collider invisible que los alertará si otro personaje del tipo querido entra o sale de él (Véase figura 46).

Ilustración 46: *Collider trigger que delimita el campo de visión del enemigo Dragón Púrpura.*

7.8. Action Events.

Uno de los problemas que se tuvo al principio, fue que no conseguíamos que solo se realizará un ataque por animación, esto se ha podido solucionar gracias a los actionEvents. Este tipo de programación reactiva permite comunicación desacoplada entre scripts. En nuestro caso, está programado que justo al final de la animación de atacar, se invoque la función de dañar que reside en los scripts.

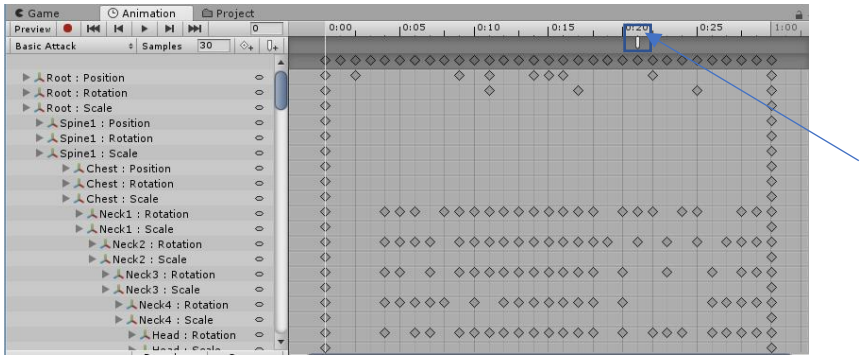

Ilustración 47: *Action Event situado en la animación de ataque del enemigo Dragón Púrpura.*

7.9. Corrutina.

Cuando se llama a una función, esta se ejecuta antes de en su totalidad antes de retornar. Es decir, que cualquier acción que conlleva se tiene que llevar a cabo en un mismo frame. Si por ejemplo ejecuta el siguiente código:

```
void Fade() {
	for (float f = 1f; f >= 0; f -= 0.1f) {
		Color c = renderer.material.color;
		c.a = f;
		renderer.material.color = c;
	}
}
```

Si nuestra idea era desvanecer poco a poco el color de nuestro material, el resultado no será ese, sino que desaparecerá de golpe. Para este tipo de ejemplos existen las corrutinas, estas son unas funciones que tienen la habilidad de pausar su ejecución y devolverle el control a Unity, para luego continuar por donde lo habían dejado en el siguiente frame. El ejemplo anterior en corrutina sería de la siguiente manera:

```
function Fade() {
	for (var f = 1.0; f >= 0; f -= 0.1) {
		var c = renderer.material.color;
		c.a = f;
		renderer.material.color = c;
		yield;
	}
}
```

En Unity cualquier función que contenga la instrucción yield se entiende como corrutina y no es necesario indicar el tipo de entorno IEnumerator. Esta instrucción es el punto donde la función se pausará hasta el siguiente frame.

8. Conclusiones y trabajo futuro.

A lo largo de este proyecto se ha dado una visión de la historia del sector del videojuego, profundizando más en el género RPG, al que pertenece el juego desarrollado.

Tras esto, se ha ofrecido una breve panorámica de la inteligencia artificial, las distintas formas de implementarla y el gran impacto que tienen en la sociedad hoy en día debido a su sin fin de aplicaciones Después, se ha descrito brevemente su aplicación en el sector del videojuego. También se ha visto cómo llevar a cabo la implementación de varias técnicas de desarrollo de inteligencias artificiales como son las máquinas de estados finitas, el aprendizaje por refuerzo o el curriculum learning.

Cada implementación tiene sus pros y sus contras. Si nos centramos en las técnicas basadas en machine-learning, obtenemos un proceso mucho más automático, donde la máquina aprende a realizar la función para la que está diseñada de la manera más óptima posible, pero, en contra parte, su implementación es mucho más costosa. Esto es debido a la dificultad de la tecnología y al gran gasto de recursos y tiempo destinados a entrenar nuestras máquinas.

Por otro lado, tendríamos las máquinas de estados, que no son tan potentes como los otros métodos, y en proyectos grandes podrían darnos bastante problemas de escalado de este método, pero que en desarrollos pequeños proporcionan flexibilidad y son mucho más sencillas de implementar. Además, en caso de querer modificar algo a posteriori, las máquinas de estado permiten estas modificaciones al instante. En cambio, la tecnología basada en machine-learning, te obliga a entrenar de nuevo al agente tras el más leve de los cambios.

Por estas razones, y debido a la magnitud de nuestro proyecto, aconsejo utilizar máquinas de estados o técnicas similares a la hora de realizar proyectos similares.

Finalmente, comentar las funcionalidades futuras que se han quedado pendientes de implementar a la hora de la entrega de este documento pero que se tienen previstas antes del lanzamiento oficial del videojuego:

- Una IA aliada. Aunque el juego esté destinado a ser un juego multijugador, queremos ofrecer al usuario la posibilidad de poder jugar en modo jugador único si así lo desea. Para esto, se ha pensado diseñar una serié de IAs que imiten el comportamiento que podrían tener los compañeros de equipo que normalmente están controlados por otros usuarios humanos.
- El escalado de habilidades de los enemigos en función de la dificultad. Nuestra intención es tener diferentes dificultades de juego, y que, para cada una de ella, los enemigos se hagan más difíciles de enfrentar. Esta dificultad añadida se tiene pensado implementar programando un comportamiento más

agresivo del personaje y un aumento de estadísticas, así como nuevas acciones solo disponibles para los niveles de dificultad más elevados.

- Una IA de entorno que controle el clima, afectando este a la jugabilidad directamente. Por ej: en un entorno donde sea de noche se reduce la visión del personaje.

9. Glosario.

A

- **Animator:** Máquina de estados que contiene las distintas animaciones relacionadas con un personaje, así como la definición de las transiciones unidas a ellas.

B

- **Boss:** enemigo final que suele encontrarse al final de los niveles de videojuego.

C

- **CD-ROM (Compact Disc Read-Only Memory):** Es un disco compacto con el que utilizan rayos láser para leer información en formato digital.
- **Chatbots:** Programa de computadora utilizado en línea para responder a preguntas.
- **Ciencias computacionales:** son aquellas que abarcan las bases teóricas de la información y la computación, así como su aplicación en sistemas computacionales.
- **Consolas de quinta generación:** Primeras consolas capaces de reproducir juegos en 3D.
- **CRM (Customer Relationship Management):** Es una aplicación que permite centralizar en una única Base de Datos todas las interacciones entre una empresa y sus clientes.

D

- **DVD (Digital Versatile Disc):** Es un tipo de disco óptico para almacenamiento de datos.

E

- **Escena de juego:** Conjunto de elementos que forman una instancia de juego.

G

- **GameObjects:** Son contenedores. Estos pueden guardar las diferentes piezas (componentes) que son requeridas para crear un elemento en nuestro entorno.
- **Generación de 8 bits:** Tercera generación de consolas. Estas ya se comercializaban. Ejemplos: NES (Nintendo) o Sega Master System (SEGA).

H

- **Hackhaton:** Encuentro de programadores cuyo objetivo es el desarrollo colaborativo de software.

I

- **IA:** Siglas de Inteligencia Artificial. También puede ser visto como AI (Artificial Intelligence).

J

- **JSON (JavaScript Object Notation):** Es un formato de texto ligero para el intercambio de datos.
- **Juego AAA:** Clasificación informal para los juegos producidos y distribuidos por una distribuidora importante.
- **Juego RTS (*real-time strategy*):** Son videojuegos de estrategia en los que no hay turnos, sino, que el tiempo transcurre de forma continua para los jugadores.

M

- **Machine learning:** Disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente.
- **Modelo-vista-controlador (MVC):** Es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

P

- **Path Finding:** Algoritmo destinado a la obtención del camino más corto entre dos puntos.
- **Prefab:** Duplicado de un objeto que conserva todas sus características y componentes. Actúa como plantilla para poder crear nuevas instancias de dicho objeto en la escena.
- **Proceso de Márkov:** Es un fenómeno aleatorio dependiente del tiempo para el cual se cumple una propiedad específica.

T

- **Tester de videojuegos:** Trabajador enfocada a la prueba de videojuegos para encontrar fallos en el sistema.
- **Threshold:** Umbral que delimita el comportamiento de una aplicación.
- **TRPG (Tactical Role-Playing Game):** También conocidos como SRPG (Strategy Role Playing Game), es un género de videojuegos con elementos de los videojuegos de rol y de los videojuegos de estrategia.

V

- **Vector de observación:** Vector que contiene el conjunto de parámetros que nos interesa analizar en cada observación del entorno.
- **Vista WYSIWYG:** acrónimo de “What You See Is What You Get”, que en español significa “lo que ves, es lo que tienes”. Se refiere a la vista final de una escena, es decir, la que veríamos en tiempo de ejecución.
- **VR (Virtual Reality):** Entorno de escenas u objetos de apariencia real.

10. Bibliografía.

10.1. Bibliografía estándar.

Okita, A. (2014). *Learning C# programming with Unity 3D*. AK Peters/CRC Press.

Poole, D. L., Mackworth, A. K., & Goebel, R. (1998). *Computational intelligence: a logical approach* (Vol. 1). New York: Oxford University Press.

Fernández, D. V., & Angelina, C. M. Desarrollo de Videojuegos: Arquitectura del Motor. *Triangle*, *5*, 10.

Thorn, A., & Marc Sch arer. (2014). *Pro Unity Game Development with C#*. Apress.

Bennett, C., & Sagmiller, D. V. (2014). *Unity AI Programming Essentials*. Packt Publishing Ltd.

10.2. Bibliografía digital.

https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-ML-Agents.md
https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-PPO.md
https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Curriculum-Learning.md
https://docs.unity3d.com/es/current/Manual/Coroutines.html
http://www.victoruc.com/RL_intro_Victor_Uc.pdf
http://aigamedev.com/open/highlights/top-ai-games/
http://www.cs.us.es/~fsancho/?e=109
https://docs.unity3d.com/es/current/Manual/Coroutines.html
https://searchdatacenter.techtarget.com/es/definicion/Inteligencia-artificial-o-AI
https://www.deustoformacion.com/blog/diseno-produccion-audiovisual/pros-contras-programar-unity-vs-unreal-engine
http://theconversation.com/understanding-the-four-types-of-ai-from-reactive-robots-to-self-aware-beings-67616