eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ Trabajo Fin de Grado ▪

Computación

El laberinto de Teseo y el Minotauro en un entorno virtual: Desarrollo de dos agentes usando aprendizaje por refuerzo en Unity

---

Xabier Ugarte Amundarain

Junio 2020

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ Trabajo Fin de Grado ▪

Computación

El laberinto de Teseo y el Minotauro en un entorno virtual: Desarrollo de dos agentes usando aprendizaje por refuerzo en Unity

---

Xabier Ugarte Amundarain

Junio 2020

Director:

Imanol Usandizaga Lombana

Agradecimientos

En primer lugar agradecer a mi director, Imanol Usandizaga, por toda la ayuda ofrecida y la orientación que me ha dado en todos estos meses, sin su ayuda hubiera sido imposible la realización de este proyecto.

Agradecer a mi familia, mi padre, mi madre y mi hermana, por toda la ayuda y apoyo que me han otorgado. A la familia de mi novia, por ser mi segunda familia, a mis amigos por animarme a no rendirme y por supuesto y no menos importante, a mi novia Maddi, que no solo es responsable de que haya conseguido terminar la carrera, también lo es de este trabajo. Gracias Maddi.

A Laika, DEP, a la cual dedico este proyecto por ser la mejor amiga y compañera que jamás pude pedir.

Resumen

En 2017 Unity presentó al mundo un kit de herramientas de aprendizaje automático. Así, la herramienta más popular para la creación de videojuegos se sumergió de lleno en una de los estudios más populares actualmente en el ámbito de las ciencias de la computación y una de las ramas de la inteligencia artificial más desarrolladas en los últimos años.

Son muchos los progresos logrados en estos últimos 3 años, la cantidad de mejoras han abierto la puerta a que muchos desarrolladores investiguen y desarrollen productos usando una herramienta que hace años solo parecía accesible a una pequeña porción de los desarrolladores de computación. Con su fácil accesibilidad, además, Unity proporciona una herramienta sencilla y fácil de usar, desde la cual se pueden aprender desde las partes más sencillas del mundo del aprendizaje automático hasta los objetivos avanzados que solo experimentados en el campo pueden lograr.

Uno de los retos más difíciles para los programadores en el campo de la computación de la informática no es simplemente aprender las nuevas herramientas que pueden servir para desarrollar nuevos proyectos, también la enseñanza de estos a futuros trabajadores en el mundo de la computación. Lo que el aprendizaje automático de Unity puede ofrecer es la creencia de que aún hay herramientas que pueden acercar a desarrolladores a este mundo sin que resulte demasiado complicado ni demasiado difícil y permite obtener resultados satisfactorios sin necesidad de haber tenido una experiencia muy amplia en este ámbito.

Una de las propuestas más fáciles para introducirse en este mundo que nos ofrece la propia página de Unity es el desarrollo de distintas pruebas y ejemplos, pero realmente no enseñan el potencial completo que puede ofrecernos esta herramienta. Así que, ¿Cómo sería el desarrollo de un gran proyecto basado en el aprendizaje automático en Unity?.

Índice de contenido

Índice de figuras

Índice de tablas

1

1 Introducción y definiciones

En esta sección introductoria se realiza un vistazo a la perspectiva general de las ideas que han llevado a cabo el proyecto.

1.1 Introducción

Uno de los problemas más antiguos del mundo de la computación relacionado con el mundo de los videojuegos es el de que un programa pueda encontrar el mejor camino en un escenario con obstáculos y además su decisión de camino se sienta natural, el simple hecho de que la inteligencia artificial no constituya una manera de caminar humanizada cuestiona al cerebro de la humanidad de dicho robot.

El problema proviene desde el funcionamiento de la propia inteligencia artificial, el programa que lo ejecuta siempre va a saber cuál es el mejor resultado. Se intenta lograr que simule una elección de distintos caminos dependiendo de un algoritmo aleatorio que determine o aparente pensamientos que tendría un humano, pero por dentro y si se quisiera, no sería difícil hacer que el personaje programado siempre vaya al mejor camino, mostrando jugadas inhumanas que alertan al cerebro de que no se está jugando con un humano, si no con un robot[1].

¿Por qué entonces la inteligencia artificial usada en los videojuegos no usa aprendizaje automático para crear personajes con comportamientos competentes?

La razón es sencilla: según Laura E. Shummon Maass el propósito de una IA (inteligencia artificial) no es, en principio, la de intentar ganar al jugador, si no ayudarle a que sea mejor y disfrute del enfrentamiento y de la experiencia (2019). Además, el aprendizaje automático no es sencillo de aplicar, aún con todas las herramientas y mejoras accesibles de las cuales se hablará en este documento, todavía queda mucho para encontrar algoritmos sencillos y fáciles de adaptar que compitan con el costo menor que tienen las inteligencias artificiales tradicionales.

Los ejemplos y aplicaciones futuras son sencillas, un ejemplo podría ser el de tener un soldado inexperto en el combate que directamente vaya a eliminar al jugador frente a otro con más experiencia que sepa que la mejor táctica es una buena emboscada, los dos en el mismo entorno y en el mismo videojuego, teniendo los dos el mismo código base pero con distintas experiencias.

Por esto, siempre es interesante humanizar lo máximo posible a las inteligencias artificiales, ya que aunque haya videojuegos en donde que los personajes no sean humanos sea un punto a favor (como por ejemplo en videojuegos donde se eliminen personas), tener la opción de humanizarlos abre la posibilidad de vivir mejor la experiencia y no sacar al jugador del videojuego por un personaje atascado en una esquina. Esto no solo podría servir en el mundo de los videojuegos, también en la de la simulación, como por ejemplo un camionero recorriendo una autopista o incluso un granjero intentando educar a distintos animales. Como en el cine o como en los libros, un personaje debería ser único no solo por la historia que pueda contener a

1 Esta idea se podría relacionar con el "experimento de La habitación china" realizado por John Searle: dialogando con una mujer china, un programa de ordenador es capaz de superar la prueba de Turing. Además, cambiando al ordenador con una persona que no sabe chino pero que posee un organigrama del programa, la mujer no se da cuenta de que su interlocutor no sabe chino. Por tanto, el experimento cuestiona si el ordenador entiende la conversación que ha mantenido con la mujer. Véase Alfonseca, M. (2014). "¿Basta la prueba de Turing para definir la "inteligencia artificial"?". *Scientia et Fides*, 2(2), 129–134. DOI: http://dx.doi.org/10.12775/SetF.2014.018, consultado en junio de 2020.

sus espaldas, si no también por cómo piensa debido a esta, y, al final, si dos personajes son iguales, ¿Qué relevancia tiene que uno de ellos desaparezca?.

1.2 Contenido

Esta memoria contiene 8 capítulos. En el primer y segundo capítulo se explicarán y se introducirán las ideas que han hecho posible este proyecto, junto con los objetivos propuestos que son comparados más adelante con los logrados. En el tercer capítulo se expresará la organización y la planificación que han llevado a la posibilidad de este proyecto en el periodo de tiempo acordado.

Como para entender el proyecto se deben de tener unas nociones básicas, se explicarán primero los conceptos que contextualizan el entorno del proyecto con referencias a las respectivas páginas que explican con mayor detenimiento el producto mencionado.

Primero, se expondrán las nociones básicas de aprendizaje automático en el capítulo 4.1 y, concretamente, de los conceptos que más han servido en la realización del proyecto. Segundo, se mencionará la aplicación usada para el desarrollo, Unity, en el capítulo 5.1 y cómo los conceptos de aprendizaje automático pueden desarrollarse usando esta herramienta.

Dentro de la explicación del desarrollo propio del proyecto, capítulo 6, se explanará la adecuación del escenario para hacer posible el desarrollo y se pondrán en práctica las nociones explicadas en el primer y segundo punto.

Finalmente, se desarrollarán las conclusiones y la visión del futuro que la construcción de este trabajo puede aportar, cerrando en la posibilidad de continuar con este proyecto en el futuro.

1.3 Contexto

El mundo de los videojuegos agranda cada día: “The video game industry has a great potential, which has been growing ever faster since 2005.” (Tan y Li, 2008: 1), y junto a él, las herramientas para la creación de estos.

Unity es, en gran medida, uno de los responsables de la creación de muchos artistas que no pueden permitirse un motor gráfico pagado y aprenden con este programa, pero lejos de la confusión, las capacidades que ofrece este motor gráfico de videojuegos no son ninguna broma, y por ello se considera el rey del momento para la creación de videojuegos “in the fourteen years since Unity’s engine launched, the size of the global gaming market has exploded from $27 billion to $135 billion” (Peckham, 2019).

La demanda exige trabajo, y el trabajo, formación. Aunque no sea algo indispensable por el momento, es siempre favorable tener estudios en el campo. Es por ello que la razón de este trabajo de fin de grado es la de introducirse en el mundo de la creación de videojuegos con esta herramienta, puesto que los másteres y demás formaciones que se ofrecen sobre el ámbito la usan, y además, las herramientas que ofrece para el campo de la computación son novedosas y prometedoras, así que es una buena manera de compaginar el futuro con lo aprendido durante la realización de la carrera de ingeniería informática.

1.4 Asignaturas

El trabajo de fin de grado es, en gran medida, la demostración de la experiencia y conocimientos adquiridos a lo largo de la carrera.

Debido a esto se ha decidido que la mejor manera para que futuros estudiantes sepan cómo se puede llegar a último año con los conocimientos necesarios para realizar este trabajo es explicando qué asignaturas se han realizado y para qué parte del proyecto son necesarias de cursar. Así, si en un futuro un nuevo estudiante está pensando en realizar un trabajo de fin de grado parecido a este, tendrá constancia de las mejores elecciones de materias, debido a que muchas asignaturas que han ayudado eran optativas y no eran necesarias de realizar.

La rama escogida fue la de Computación puesto que es la rama que más se acerca al campo de inteligencia artificial.

- Gestión de proyectos (Asignatura obligatoria): La asignatura de gestión de proyectos, tal y como su nombre indica, enseña a gestionar trabajos o planes en corto o largo plazo. Sirve para planificar un trabajo tan grande como de esta índole y aprovechar el tiempo al máximo.
- Gráficos por computador y Visión en entornos virtuales (Asignaturas obligatorias en la rama de Computación): Son las asignaturas que introducen el mundo de las escenas virtuales, aplicando las matemáticas aprendidas a lo largo de los años cursados a un entorno virtual y se enseña todo lo necesario para manejar cámaras, luces, incluso renderizado que pueden servir para manejarse en la aplicación Unity.
- Diseño de algoritmos (Asignatura obligatoria en la rama de Computación): Todo tipo de videojuego contiene trazas de diseño de algoritmos; Desde la renderización óptima de la escena, el conocimiento de cómo funciona un buen algoritmo de ordenación de vectores o incluso, como para este proyecto, el aprendizaje de un algoritmo capaz de encontrar el camino más corto de un punto a otro, todo ello nos ofrece la enseñanza de esta asignatura. Se puede decir que es de las asignaturas más útiles para el futuro de cualquier graduado.
- Inteligencia artificial y técnicas avanzadas de inteligencia artificial (Asignatura obligatoria en la rama de Computación y Asignatura optativa, respectivamente): Todos los conceptos de aprendizaje automático, de inteligencia artificial o de agentes lo introducen estas dos asignaturas.

2

2 Objetivos y recursos

En este capítulo se abordan los objetivos propuestos en un principio y se habla sobre las herramientas que sea han usado para la realización de este proyecto.

2.1 Objetivos iniciales

Los objetivos iniciales fueron estos:

- Demostrar que dos agentes construidos de la misma manera pero entrenados de distinta, pueden llegar a obtener una inteligencia completamente diferente, mostrando así que no haría falta construir dos tipos de agentes distintos para dos tareas distintas:
    - El desarrollo de un agente que fuera capaz de navegar por el laberinto y dependiendo de las pistas encontradas decidiera el mejor camino:
        - Sigue a: Hilos
        - Su meta: La salida
    - El desarrollo de un agente que fuera básicamente igual por dentro pero con distintos objetivos:
        - Sigue a: Huellas
        - Su meta: El agente enemigo
- Construir un laberinto donde se consiguiese determinar el camino más corto entro dos puntos usando un algoritmo de búsqueda del camino más corto.

Uno de los retos más grandes para lograr estos objetivos es la construcción de un escenario que no solo sirviera como suelo para el desarrollo de los agentes, si no que en proporción a los agentes y al aprendizaje sobre el que se iban a desarrollar estuviera correctamente construido, puesto que un mal escenario puede dar lugar a un comportamiento indeseado de los agentes.

Además, es necesario encontrar una manera para construir un algoritmo que sepa y conozca los valores necesarios del escenario para poder proporcionar una ayuda al aprendizaje de los agentes.

Por, ello, los objetivos iniciales mejor especificados son estos:

- Un agente, la representación de Teseo, que estando en el centro del laberinto, busque los hilos que le conducirán a una salida mientras intenta esquivar al Minotauro que le matará si lo encuentra.
- Un agente, el Minotauro, que estando lejos del centro busque huellas de humano que le conduzcan a Teseo y matarlo. La única diferencia notable, además de los objetivos que persigue o su logro final, es que el Minotauro no tiene ningún enemigo que lo mate, pero si Teseo logra llegar al final, su recompensa negativa será proporcional a como si lo hubieran matado.
- Un laberinto, que sea proporcional en toda su forma y que desde el cual de cualquier lugar se pueda llegar al punto más lejano del mismo.
- Un algoritmo de camino más corto, donde busque en todo el laberinto cuál sería el camino más corto para llegar desde un punto A hasta un punto B, además, el algoritmo deberá de ser el más óptimo de resultado en cuanto coste/objetivo.

- Un fichero de código que funcione como un cerebro, que construya el escenario y adecúe en todo momento el mejor lugar para cada agente y además haga aparecer objetos como los hilos o las huellas de Teseo.

Si se consiguiera desarrollar un proyecto con estas características, se conseguiría demostrar la teoría de que no hacen falta construir distintos sujetos para personajes que sirvan de distintos propósitos puesto que con una buena elaboración del entorno y de sus propios objetivos bastaría.

2.2 Herramientas usadas

Estas son las herramientas, aplicaciones o programas que se han usado para la elaboración de este proyecto:

- La herramienta en la cual se basa todo, Unity[2], concretamente la versión 2019.2.17f1.
- El lenguaje de programación que usa el motor Unity es C#[3].

Aunque todo lo que se desarrolla en este proyecto proviene de Unity, se han de instalar algunas cosas que Unity proporciona, de manera gratuita, pero que no están de base en el propio producto:

- La herramienta de aprendizaje automático[4], concretamente la versión Release 2.
- Para el control de los datos y una visualización sencilla de estos, se puede usar Anaconda Prompt[5].

2 Véase la página oficial de Unity para más información: https://unity.com/es
3 Véase la página oficial de documentación de C#: https://docs.microsoft.com/es-es/dotnet/csharp/
4 Véase la página oficial para aprendizaje automático en Unity: https://unity3d.com/es/machine-learning
5 Véase la página oficial de Anaconda Prompt: https://docs.anaconda.com/anaconda/user-guide/getting-started/

3

3 Gestión del proyecto

En este capítulo se analizan los riesgos y la gestión del tiempo asignada al comienzo de la creación del proyecto, y se compara con la realidad.

3.1 Gestión del alcance

En un principio se decidió basar el proyecto en distintas maneras de búsquedas de camino correcto en un entorno virtual, comparando distintos algoritmos para ver cuál era el método de decisión de cada uno en tiempo real, el uso del aprendizaje automático fue una idea que apareció más tarde, aunque ya estuviera apuntado como una posibilidad en el alcance original del proyecto, se estipuló que posiblemente fuera un trabajo arriesgado y no diese tiempo a completarlo.

La realidad fue todo lo contrario, la elección del algoritmo de búsqueda de camino más corto está mayoritariamente establecido en la comunidad de Unity, puesto que es una duda que aparece en la mayoría de proyectos del calibre. En el capítulo 6.4 se explica con más detenimiento la decisión tomada.

Una vez solucionado el problema del camino más corto, se volvió atrás para ver en qué podía basarse mejor el proyecto y fue entonces cuando la idea de aprendizaje automático sobresalió.

El alcance original del proyecto, aunque enfocado de manera incorrecta, tenía matices bien expresados, estos eran:

- El mejor “pathfinding” (búsqueda del camino más corto):
  - Puntos a favor:
    - Mucha documentación.
    - Fácil de entender.
  - Puntos en contra:
    - Actualmente miles de proyectos y tutoriales explicando cómo hacerlo.
    - Poco original.
- Aprendizaje automático en Unity:
  - Puntos a favor:
    - Complemento nuevo.
    - Muy interactivo.
  - Puntos en contra:
    - Difícil de usar.
    - Mucha carga de trabajo.

Los algoritmos de camino más corto es algo que se estudia y se sigue estudiando en la informática, pero es un problema antiguo. Esto quiere decir que muchas veces se ha hablado de este tema y al uso, no es un problema original a no ser que sea algo novedoso en este campo. El aprendizaje automático, en cambio, actualmente es un tema candente en la comunidad de científicos en el campo de la computación y por ello se decidió darle más importancia.

3.2 Sistemas de información y sistema de gestión

Una de las cuestiones más importantes para la realización de un proyecto es la organización, por mucho que el proyecto sea bueno, con una mala organización difícilmente saldrá un buen proyecto, como refuerza Ángel Nájera Pérez (s/f).

Con una buena organización se consigue predecir y disminuir los riesgos posibles, pero es necesaria una planificación concisa y fácil de usar para determinar si es rentable dedicar un tiempo a decidir el sistema de información.

El sistema de información general ha sido Google Drive. Si bien es cierto que hay otras plataformas como Github que resultan más eficaces para proyectos de gran calibre, la rápida edición de documentos en línea y la facilidad de distinguir versiones de un mismo archivo junto con la sencillez que tiene el cambiar de ordenador sin necesidad de descargar todo de nuevo por la opción de editar texto en línea, ha terminado dictando el sistema de información.

Pero Google Drive, aún con las mejoras que ofrece, no puede determinar una hoja de ruta de manera tan concisa como otras plataformas, y por ello, para la planificación se ha usado Trello[6], el cual tiene una versión totalmente gratuita que satisface completamente la demanda de planificación del proyecto.

3.3 Gestión de los costos y riesgos

Al enfrentarse a un trabajo grande que consta de cerca de 300 horas y una fecha de entrega determinada, los riesgos y activos negativos que se pueden sufrir pueden llegar a ser determinantes. Además, la gestión de los costos es necesaria para no dedicar demasiado a una parte y dejar el resto a medias.

Una buena gestión de los costos y riesgos debe de tener una dedicación de horas proporcional en cuanto a distintas partes del proyecto y, con mayor importancia, a la documentación, puesto que la memoria es el producto final del desarrollo del trabajo de fin de grado, como sostiene Axel Becerril, Arquitecto de Software en Jaque Software Developers: “El producto no es el software [...] Nadie en este mundo despierta un día y dice «Quiero exactamente este conjunto de instrucciones corriendo en una computadora». Lo que sí ocurre es que alguien descubre que tiene una necesidad y busca una solución.” (2016:1) y es la manera de encontrar la solución lo que interesa, no el qué, si no el cómo.

Los riesgos que hay que tener en cuenta son en gran medida los de la investigación de los contenidos, arreglar una equivocación de implementación cuesta tiempo, pero un error de planificación de investigación puede salir aún más caro. No solo se debería de desechar gran parte del contenido realizado, también es un trabajo incompleto, pues se deben de investigar y aprender nuevas opciones que rellenen el hueco que se ha dejado vacío.

Una de las mejores maneras que se han encontrado para afrontar este peligro es la posibilidad de informarse y hacer tutoriales que ayuden a entender de una manera general las herramientas del proyecto, más específicamente, dedicar tiempo en un principio a realizar

6 Véase la página oficial: https://trello.com/es

tutoriales como los de Immersive Limit[7] o los de la propia página oficial de Unity[8] ayudó en gran medida al entendimiento completo de la herramienta de aprendizaje automático.

3.4 Dedicación

El diagrama de Gantt construido en el alcance original del proyecto fue este (se entiende como S1 el primer fin de semana de febrero):

	S1	S2	S3	S4	S5	S6	S7-S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22-S23
Selección de tema	X																
Documentarse y planificar		X															
Primer prototipo						X											
Segundo prototipo (60-70% hecho)								X									
Prototipo final										X							
Comienzo de la memoria formal											X						
Entrega														X			
Subir el trabajo a ADDI + Poster															X		
Defensa del TFG																	X

Tabla 1: Diagrama de Gantt inicial

El diagrama de Gantt de hitos, aunque no del todo equivocado, no tuvo en cuenta la aparición de un acontecimiento sin precedentes como lo fue la pandemia, y como la informática no se limita solo al ámbito de la ciencia entendido como algo impersonal, también tiene mucho que ver el entorno personal, por motivos personales se tuvieron que retrasar algunos hitos.

Por ello, el diagrama de Gantt de dedicación final fue este:

7 Véase la página oficial del tutorial: https://www.immersivelimit.com/tutorials/reinforcement-learning-penguins-part-1-unity-ml-agents
8 Véase la página oficial del tutorial de Unity: https://github.com/Unity-Technologies/ml-agents/blob/0.13.1/docs/Basic-Guide.md

	S1	S2	S3	S4	S5	S6-S9	S10	S11	S12	S13	S14-S17	S18	S19	S20	S21	S22-S23
Selección de tema	X															
Documentarse y planificar				X												
Tutoriales de Unity					X											
Primer prototipo							X									
Segundo prototipo (60-70%)										X						
Prototipo final												X				
Arreglos- limpieza del código													X			
Comienzo de la memoria																
Memoria último visto bueno													X			
Subir el trabajo a ADDI + Poster														X		
Defensa del TFG																X

Tabla 2: Diagrama de Gantt definitiva

Se ha de comentar también que el desarrollo de la memoria, aunque no formalmente, se hizo desde el primer minuto del desarrollo del proyecto, apuntando todos los fallos, modelos usados, código usado y muchas más cosas para poder usarla en esta memoria.

La dedicación de horas completa es esta:

	Estimación	Realidad	Desviación
Planificación del proyecto y el alcance del mismo	10h	10h	0h
Gestión y seguimiento del proyecto	10h	15h	-5h
Gestión- TOTAL	20h	25h	-5h
Conocer sobre la herramienta Unity	10h	5h	+5h
Aprender sobre aprendizaje automático en Unity	10h	8h	+2h
Realizar tutoriales	30h	40h	-10h

Diseño y planificación del trabajo de manera técnica: Formato de los nombres de archivo, de las funciones,...	2h	2h	0h
Diseño, planificación y creación del escenario	5h	6h	-1h
Aprender y decidir mejor pathfinding	10h	7h	+3h
Implementación del pathfinding	10h	10h	0h
Escritura de código	40h	35h	+5h
Implementación del aprendizaje automático	50h	40h	+10h
Análisis de los resultados junto con la espera de la realización de estos	10h	20h	-20h
Limpieza del código	3h	3h	0h
Descarga y subida de archivos a Google Drive	7h	6h	+1h
Tiempo extra para errores o problemas inesperados	13h	20h	-7h
Desarrollo- TOTAL	200h	202h	-2h
Memoria	70h	60h	+10h
Elaboración de la presentación	10h	10h	0h
Documentación- TOTAL	80h	70h	+10h
TOTAL	300h	297h	+3h

Tabla 3: Dedicación de horas

Se puede asumir que el desarrollo del proyecto, en cuanto a los correspondiente de la dedicación, ha sido un logro, aunque se han corrido muchos riesgos. Existía un tiempo extra para errores o problemas inesperados ya que después de la realización de tantos proyectos se sabe que la probabilidad de que algo malo ocurra es alto, pero lamentablemente ha llevado más

tiempo del deseado y aún habiendo salido con una derivación positiva, no es tan positiva como se hubiera gustado que lo fuera.

En general, se podría decir que es un resultado positivo, y refuerza la necesidad de una buena planificación para el desarrollo de este trabajo, puesto que aún teniendo tiempo extra el tiempo que ha sobrado ha sido muy limitado.

4

4 Conceptos básicos

En este capítulo se introducen conceptos básicos que complementan al entendimiento del proyecto de manera total, ideas que se usarán mucho a lo largo del documento y, por tanto, es necesario conocerlas.

4.1 Aprendizaje automático

El aprendizaje automático, también conocido como Machine Learning, está en relación directa con la inteligencia artificial, pero no es lo mismo, referenciando a John McCarthy, premio Turing 1971 y mundialmente conocido como el padre de la inteligencia artificial.

El aprendizaje automático trata, *grosso modo*, sobre programas que son capaces de aprender mediante datos, a diferencia de la inteligencia artificial que se emplea para programas que tienen un comportamiento similar al de los humanos.

"Lo que se denomina aprendizaje consiste en la capacidad del sistema para identificar una gran serie de patrones complejos determinados por una gran cantidad de parámetros." (Redacción APD, 2019). Cuando se habla de programas que aprenden, no se hace referencia a programas lo suficientemente inteligentes como para aprender por ellos mismos, si no a programas que contienen algoritmos lo suficientemente buenos como para, con una entrada de datos, pueda identificar la mejor solución.

Los tres tipos de aprendizaje automático más conocidos son:

1. Aprendizaje supervisado
    - Se usa para un aprendizaje basado en ejemplos y en soluciones correctas. En primera instancia se tienen ejemplos y las soluciones correctas de los mismos, después, con los mismos ejemplos, se ejecuta el algoritmo y se intenta aproximar al mayor porcentaje de aciertos posible, comparándolos con las soluciones correctas dadas al comienzo (Véase el ejemplo de la figura 1). Es, por tanto, necesaria la intervención de humanos en el proceso. Los ejemplos son varios, por ejemplo para la evasión de spam en correos.

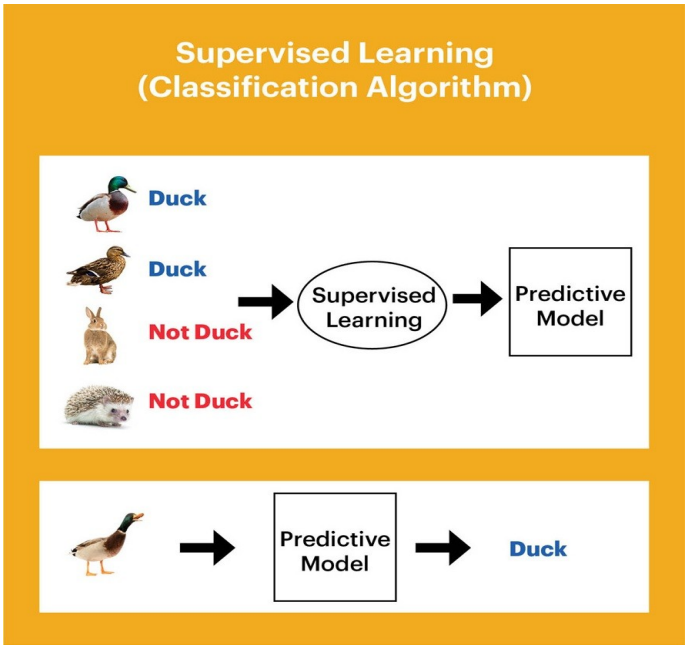

Figura 1: Ejemplo de aprendizaje supervisado. Por learncuriously (2018).

2. Aprendizaje no-supervisado

   - Al contrario del aprendizaje supervisado, los humanos no son necesarios, o su intervención debería ser la del mínimo posible. Se usa mayoritariamente para estimación en el mundo de la estadística. Explicado de manera general, en este caso nadie le dice al programa cuál es la solución correcta y debería ser capaz de identificar o clasificar cada solución por sí misma. (Véase el ejemplo de la figura 2)

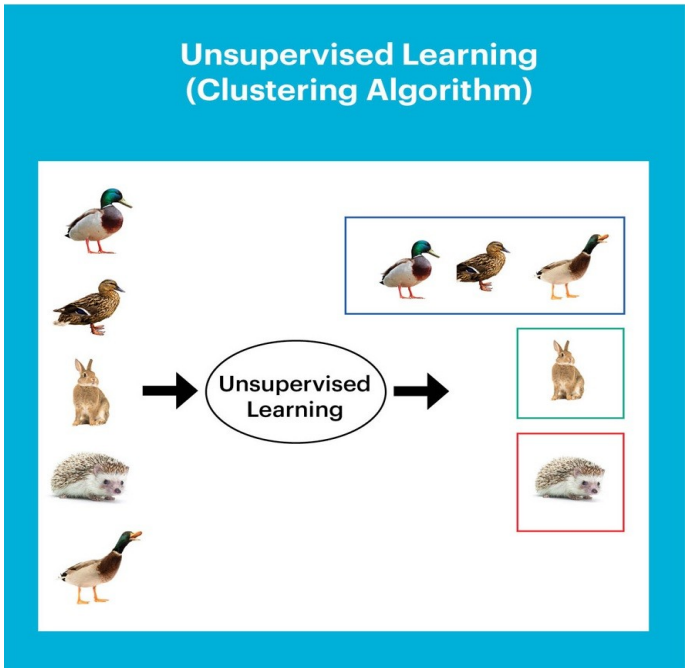

Figura 2: Ejemplo de aprendizaje no-supervisado. Por learncuriously (2018).

3. Aprendizaje por refuerzo

   - Es el aprendizaje que se ha usado para este proyecto. Aprenden mediante propia experiencia, gracias a un sistema (véase el diagrama de la figura 3) que penaliza o premia, o una combinación de las dos posibilidades. El objetivo es enseñar al agente a maximizar la premiación posible para obtener un resultado que satisfaga las intenciones del desarrollador. Sus ejemplos son conocidos, por ejemplo la creación de bots para el videojuego DOTA 2[9].

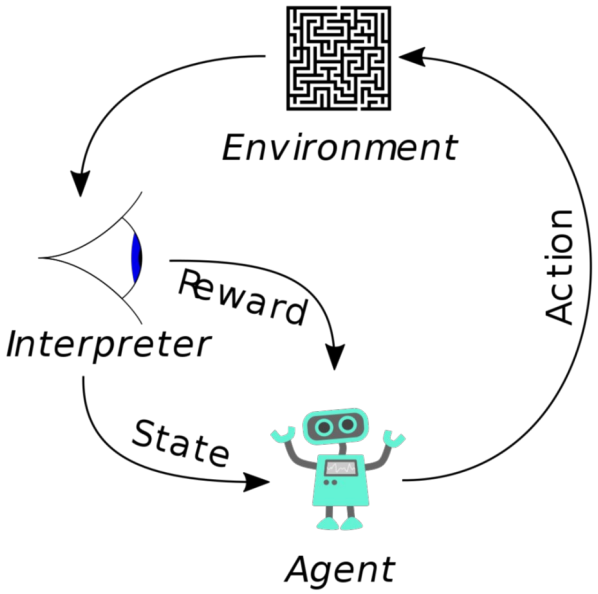

Figura 3: Diagrama de aprendizaje por refuerzo. Por Megajuice (2017).

Mediante el aprendizaje por refuerzo que se usa en Unity se crea una red neuronal artificial que funciona de la manera más óptima que aprendió el agente. La red neuronal es, como su nombre indica, una red de neuronas artificiales que se organizan mediante capas. Funciona de esta manera (explicado de manera sencilla) : se introduce un valor desde un extremo de la red, la llamada red de entrada, y después de pasar por distintas capas de neuronas se llega al otro extremo de la red, la llamada red de salida, donde dará la solución correspondiente a ese valor de entrada. Se puede observar la figura 4 para ver la representación de una red de neuronas artificiales en manera de dibujo.

Mediante el aprendizaje por refuerzo se consigue una red de este tipo que se puede añadir al agente para que funcione como se le ha entrenado y poder comprobar si el aprendizaje ha dado sus frutos.

9 Leer blog oficial sobre el tema: https://openai.com/blog/openai-five/

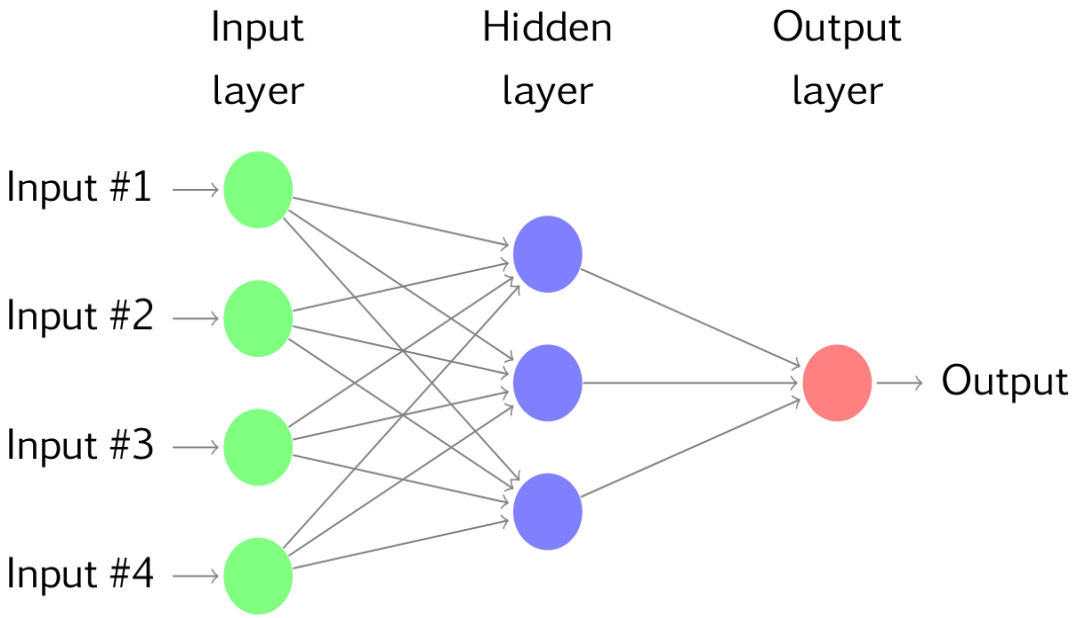

Figura 4: Representación de red neuronal artificial. Por otexts (s/f).

4.2 Agentes

Los agentes son lo que se considera el responsable o músculo del aprendizaje por refuerzo. Son los que toman las decisiones basándose en el sistema de premios que se ha construido, de alguna manera, se podría decir que son la representación del algoritmo construido en el mundo virtual.

De modo de ejemplo, se puede considerar el caso de este proyecto. Existe Teseo, y existe el Minotauro, los dos son "objetos" en el mundo virtual que deciden por ellos mismos, y los dos son considerados agentes. Si el Minotauro mata a Teseo, se podría considerar que el agente Minotauro ha conseguido puntos positivos, porque ha logrado uno de sus objetivos, en cambio el agente Teseo consigue puntos negativos, puesto que le ha ocurrido todo lo contrario.

Es imposible hablar de aprendizaje por refuerzo sin mencionar a los agentes, debido a que el Agente es el ente virtual que decide y toma decisiones y el aprendizaje por refuerzo es el método de aprendizaje que se ha usado para que esto ocurra.

El truco está en enseñar a los agentes mediante premios positivos y negativos con los cuales aprenderá el comportamiento más óptimo. Los dos agentes, el agente Minotauro y el agente Teseo contienen, de base, un código muy parecido, la intención es que mediante diferentes experiencias se comporten como agentes distintos, Teseo siendo más cuidadoso, con miedo de encontrarse al Minotauro y el Minotauro siendo completamente agresivo, buscando a Teseo y encarándolo.

De esta manera, se podría demostrar que no es necesario construir distintos agentes o completamente diferentes para cada objetivo en un mundo virtual, una ligera e inteligente modificación de estos basta para encontrar resultados totalmente opuestos y viables.

4.3 Refuerzo positivo y negativo

El refuerzo ha sido siempre un tema muy discutido en el mundo de la educación y la pedagogía. Desde experimentos con perros y su comida hasta teorías renovadoras del conductismo humano, el refuerzo positivo o negativo ha influenciado en muchos aspectos técnicos del aprendizaje.

Sin adentrarse demasiado en el mundo de la pedagogía, el aprendizaje automático, como su nombre indica, es aprendizaje, por lo que los métodos de enseñanza de la educación pueden usarse para el aprendizaje automático.

De este pensamiento aparece el refuerzo positivo y negativo, el psicólogo Burrhus Frederic Skinner fue el que definió estos conceptos. El refuerzo, o más concretamente el reforzamiento, se puede definir como el procedimiento gracias al cual mediante un estímulo la probabilidad de que se repita una conducta aumenta: "(1971) […] When a bit of behavior is followed by a certain kind of consequence, it is more likely to occur again, and a consequence having this effect is called a reinforcer" (McConnell, 1990: 248). Mientras que el refuerzo positivo va seguido de una recompensa positiva, el refuerzo negativo conlleva la eliminación de un evento desagradable, que no es lo mismo que el castigo positivo o negativo que conllevan un premio positivo o un premio negativo, respectivamente. Un refuerzo positivo podría ser premiar a un estudiante con un punto extra en el examen, mientras que un refuerzo negativo sería dejarle sentarse en última fila cerca de sus amigos después de haber estado sentado en primera fila tras haberse portado mal.

Como se puede intuir, el refuerzo negativo es difícil de definir porque depende completamente de la subjetividad del sujeto. "Given that reward is a synonym for positive reinforcement and that negative is defined as 'something disagreeable' many students assume that negative reinforcement should be the technical term that is synonymous with punishment (Tauber, 1988)" (McConnell, 1990: 248) pero no tiene por qué ser un premio negativo. En el aprendizaje por refuerzo que se usa en este proyecto se centra en la técnica de los castigos positivos y negativos.

Un mal castigo negativo conlleva un comportamiento indeseado, es siempre preferible usar premios positivos ignorando los premios negativos para los inexperienciados en el concepto. Un fácil ejemplo de un mal castigo negativo es, por ejemplo, el agente Teseo de este proyecto. Si se le recompensa negativamente solo cuando el Minotauro lo atrapa, decidirá no moverse del lugar donde aparece porque sabe que si se mueve tiene más probabilidades de que el Minotauro le atrape ¿una posible solución? Darle un castigo negativo muy pequeño en cada paso, por lo que no tiene más remedio que ir a buscar la salida para obtener un premio positivo.

El término del refuerzo pasó de la psicología a la informática de la mano de Minsky "Reinforcement Learning (RL), a term borrowed from animal learning literature by Minsky (1954, 1961)" (Keerthi y Ravindran, 1994: 852), y desde entonces el aprendizaje por refuerzo ha crecido a pasos agigantados y promete un futuro esperanzador.

5

5 Unity y el toolkit ML-Agents

En este capítulo se aporta la información necesaria para entender Unity, además, se explicará cómo funciona el aprendizaje automático en este motor de videojuegos.

5.1 Unity- Introducción

Unity es un motor de videojuegos[10] que soporta de forma nativa el lenguaje de programación C#. Su primera aparición fue en 2002 del resultado de unir dos desarrolladores trabajando en motores gráficos (Haas, 2014: 4), y se ha convertido en uno de los motores de videojuegos de más éxito debido al apoyo que puede brindar a desarrolladores independientes buscando un motor gratuito, fiable y eficaz.

Debido a esto, Unity está haciéndose cada vez más famoso "Mobile gaming is the fastest-growing segment in the overall gaming industry, expected to rise at an average annual rate of 17%+ between 2017 and 2021. Unity will be an important contributor to future growth, providing the digital infrastructure necessary to take the mobile gaming segment $100B+ in the next few years." (CBINSIGHTS, 2018) y no solamente entre la comunidad de desarrolladores de videojuegos, gracias a su capacidad de adaptación a todo tipo de problemas también está creciendo en la comunidad de animaciones tridimensionales o en la realidad aumentada "Unity 3D will become essential for AR [Realidad aumentada][11] game development in the future" (Sung, Hae, Jeong, Jung, Laina y Westlin 2014: 21).

Unity es una de las herramientas más usadas para trabajos de fin de grado en la ingeniería informática, debido a la fácil aplicación de algoritmos y problemas aprendidos durante los años cursados de la carrera, por eso no es de extrañar observar tantos trabajos realizados con este motor de videojuegos y siendo tan diferentes entre sí. Obsérvese la figura 5 para ver la interfaz básica de Unity.

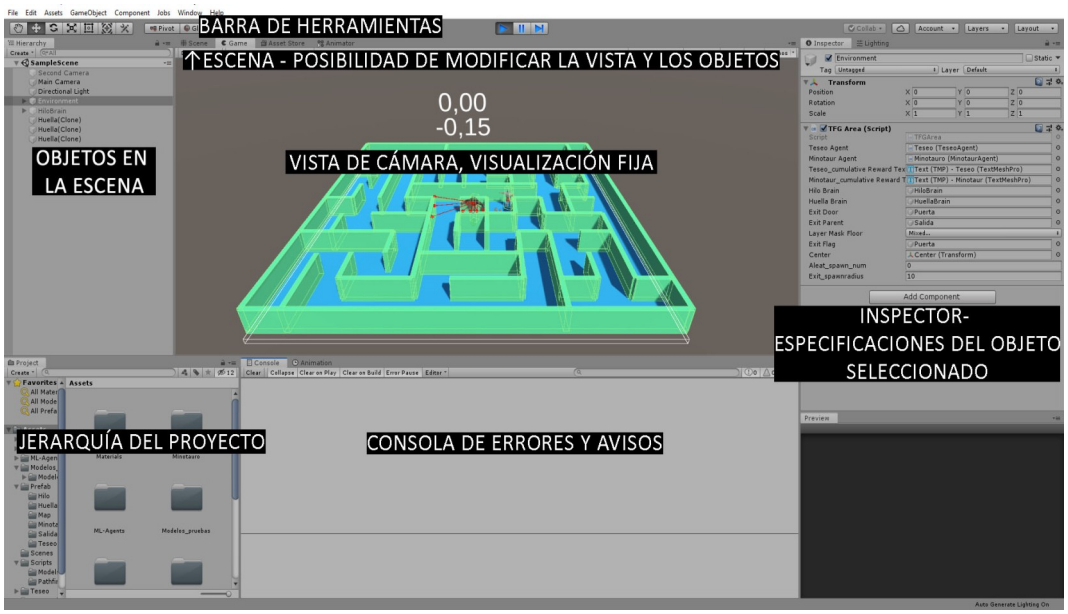

Figura 5: Interfaz básica de Unity

10 Véase la página oficial de Unity: https://unity.com/es y la explicación oficial de un motor de videojuegos: https://unity3d.com/what-is-a-game-engine

11 Lo que hay dentro del corchete no pertenece a la cita original.

5.2 Machine Learning en Unity

El kit de herramientas llamado ML-Agents publicado por Unity en 2018, implementa un pipeline[12] de Python para entrenamiento de agentes. El funcionamiento general de un entorno de aprendizaje en Unity se puede ver en la figura 6.

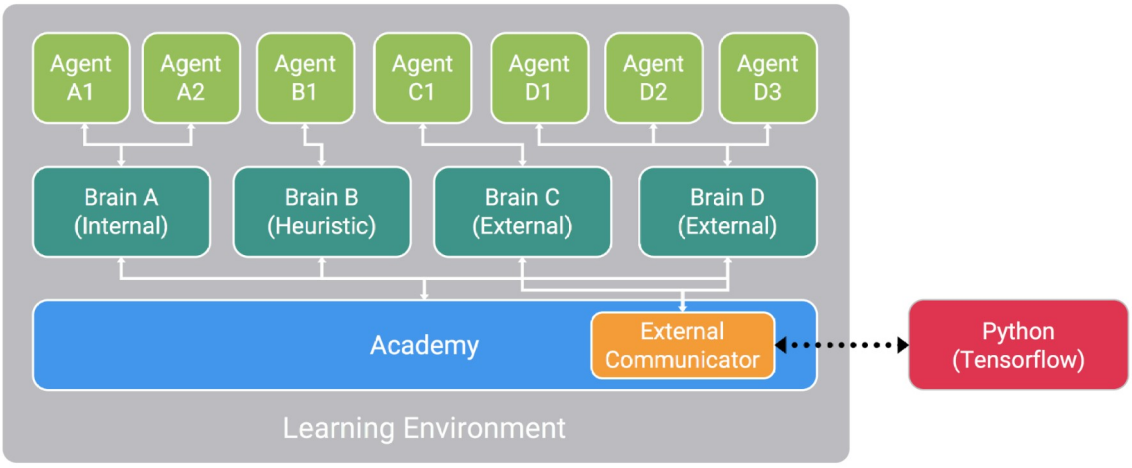

Figura 6: Entorno de aprendizaje con ML-Unity. Por Juliani (2017).

Como se puede observar, Los agentes están conectados a distintos tipos de Brain que a su vez están conectados con un Academy. El Academy, mediante un External Communicator se conecta con el pipeline de Python que hemos mencionado antes.[13]

- Agent: La explicación de los agentes se estudia en el capítulo 4.2. Las decisiones de un agente están conectados a su Brain.
- Brain: Un Brain define un estado y son responsables de las decisiones que va a tomar el agente. Existen distintos tipos, para este proyecto se usará el tipo External donde la decisión de las acciones se toma mediante un socket[14] que comunica el entorno virtual con la interfaz del pipeline de Python. También se comentará el tipo Heuristic en el capítulo 6.5.
- Academy: El Academy es un objeto que contiene todos los Brain como sus hijos. En la versión 0.14 de ML-Unity el Academy se convirtió en un singleton y ya no era necesario crear un programa para este apartado, debido a eso no hay referencias a ello en el proyecto, pero que se sepa que existe y es importante.

12 Véase esta página para obtener más información: https://medium.com/datos-y-ciencia/pipeline-python-20c84e255444
13 Toda la información que se comentará en este apartado se puede encontrar en la página oficial de Unity: https://blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents/
14 Mediante el término socket dos procesos intercambian cualquier flujo de datos, en este caso la API de Python y el entorno virtual.

Además de estos componentes para el entorno de aprendizaje en Unity, el toolkit ML-Agents también incluye distintos tipos de entrenamientos para agilizar o para mejorar el proceso. Entre ellos lo más importantes son:

- Single-agent: El aprendizaje por defecto, se enseña a un solo agente en el proceso de aprendizaje. En este proyecto se usa este tipo de entrenamiento debido a que interesa saber el tiempo de aprendizaje propio de un escenario simple de aprendizaje automático.
- Simultaneous Single-Agent: Una multitud de agentes independientes conectados a un mismo Brain que agilizan proporcionalmente el ritmo de aprendizaje del agente.
- Adversarial Self-Play: Dos agentes conectados a un mismo Brain con un sistema de premios inversamente proporcional, se usó este tipo de entrenamiento para entrenar los bots de DOTA 2.

5.3 State of the art

Actualmente el toolkit (kit de herramientas) ML-Agents está mejor que nunca, habiendo sacado su primera versión estable después de 2 años de beta a fecha de 12 de mayo (Mattar, Shih, Berges, Elion y Goy 2020) y ayudando a distintas organizaciones en plena crisis de la pandemia (Fort, Crespi, Elion, Kermanizadeh, Wani y Lange 2020).

Después de más de 8400 proyectos de Github realizados, se puede asegurar que esta herramienta solo está en fase de comienzo. Los desarrolladores afirman que van a seguir dando su apoyo al proyecto y que ya tienen en mente distintas metas: Como el entrenamiento de agentes en la nube (para la gente que no quiere usar Python) o reconstruyendo el núcleo del kit de la herramienta para que sea DOTS[15].

Por el momento, Unity está siendo explorado en distintas áreas fuera del mundo de los videojuegos como la sanidad (Pires, Santos, Andrade, Caurin y Siqueira, 2014) y está teniendo éxito. Por lo que no sería de extrañar observar esta herramienta siendo introducida en facultades dentro o fuera del área de la informática, como dice Clive Downie, director de marketing de Unity, "Education is interesting to us, as is the ability for Unity to be something that students realize is as transformational as a pen and paper used to be" (Valentine, 2019). Además, dentro del área de los videojuegos, estando completamente presente en el futuro de la industria gracias a su accesibilidad para la creación de videojuegos móviles.

15 Véase la explicación oficial de Unity sobre DOTS (data oriented tech stack): https://unity.com/es/dots

6

6 Desarrollo

Este es el capítulo central del trabajo, donde se discute el desarrollo que ha hecho posible la realización de este trabajo de fin de grado.

6.1 La historia: El laberinto de Teseo y el Minotauro

La elección de la leyenda de Teseo y el Minotauro (García, 1983) para este trabajo se debe a la popularidad que alberga, es uno de los mitos más conocidos de la antigua Grecia. Junto a la popularidad, existe siempre la curiosidad, las ganas de contar un cuento en distintos medios como en la televisión, en el cine[16] y en este caso, en un entorno virtual.

Los laberintos han sido desde siempre uno de los juegos más comunes como pasatiempos, como en periódicos, cuentos, etc. …, "la entrada en el laberinto supone el riesgo de perderse" (Santarcangel, 1997: 13) y por tanto, el llegar al final provoca en el jugador un alivio, un sentimiento de realización.

Un laberinto es un conjunto de caminos posibles donde la única posibilidad de encontrar un camino correcto es recorriendo cada uno hasta encontrar el que llega al final.

Cada vez que hay un problema de esta índole despierta en la mente del informático una manera de informatizar y mecanizar una búsqueda de mejor camino de manera automática. Debido a esto se escogió un laberinto como escenario posible.

La existencia de los personajes de Teseo y el Minotauro permite construir agentes mediante el aprendizaje automático. Si se analiza detenidamente, los dos personajes funcionan de la misma manera. Teseo quiere llegar al final, y la manera en lo que lo hace es siguiendo el hilo. El Minotauro, en cambio, quiere llegar a Teseo para matarlo y para hacerlo sigue las huellas que deja en el suelo Teseo.

De manera resumida, la historia de este cuento épico se remonta a la antigua Grecia, el hijo del rey Eges, Teseo, se ofrece voluntario como ofrenda del Minotauro. Ariadna le aconseja usar un hilo con el cual pueda encontrar el camino de vuelta del laberinto una vez mate al Minotauro.

La versión de este proyecto no contempla el asesinato del Minotauro, pero sí lo demás. Teseo, que de pronto aparece en mitad del laberinto, tiene un hilo que le lleva a la salida y debe seguirlo si quiere salir. El Minotauro, en cambio, tiene que matarlo antes de que consiga salir.

6.2 Adecuación del entorno

Uno de los conceptos claves para la realización de un buen aprendizaje automático es, entre otras muchas cosas, el entorno, por una sencilla razón: El aprendizaje por refuerzo se desarrolla mediante agentes que se construyen interactuando con el entorno que les rodea.

Para confirmar esta afirmación se realizaron distintas pruebas, en una el agente estaba en un entorno sin obstáculos, en la otra tenía una pared en medio del escenario como obstáculo. En los dos casos el cubo era el objetivo.[17] Se pueden ver los dos casos en la figura 7.

16 Véase la película clásica de 1960 *Teseo contro il Minotauro* (Silvio Amadino): https://www.filmaffinity.com/es/
17 Se puede encontrar el tutorial para hacerlo en:
https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Create-New.md

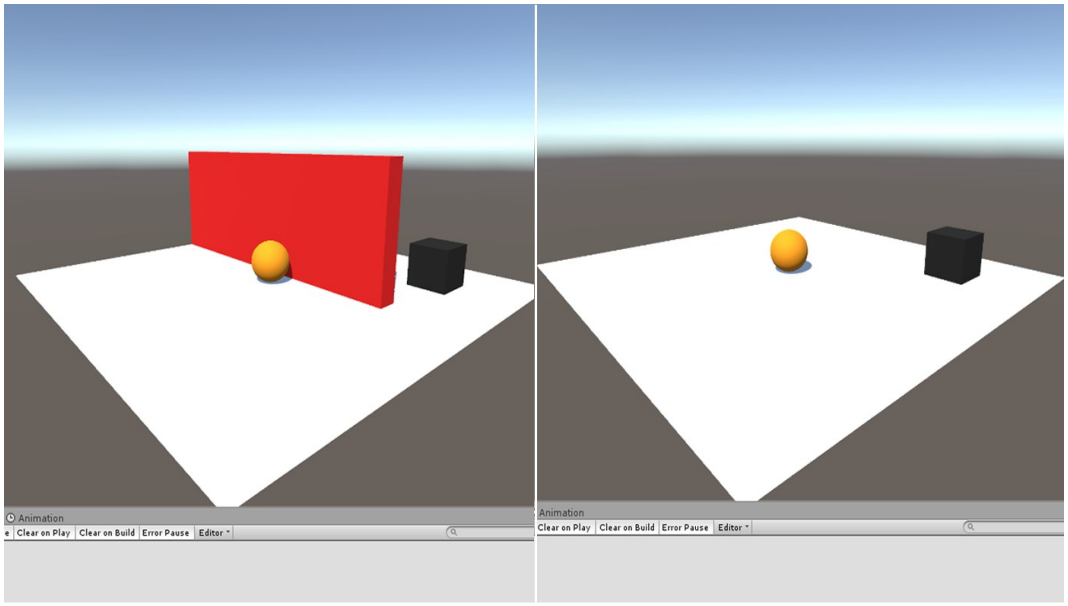

Figura 7: Ejemplo de la importancia del entorno, escenario con obstáculo y el escenario sin obstáculo, respectivamente.

Los dos agentes se construyeron de la misma manera y contienen el mismo código por dentro. El nivel de recompensa medio máximo es 1, el menor es 0. Como se puede analizar en la figura 8, en el primer escenario el agente es rápido y aprende a llegar al objetivo en 140 mil pasos que son, aproximadamente, 47 minutos, en el segundo, en cambio, tarda más o a veces hasta no llega como se puede ver en la figura 9. Esto se debe a que el agente no ha aprendido con los ojos, si no con la distancia y velocidad que alberga, no tiene sensores ni manera de identificar dónde hay un objetivo y dónde no, simplemente sabe que si está mirando en la dirección del objetivo tiene que subir la velocidad. Así se analiza que el escenario no cumple solamente un papel clave a la hora del desarrollo, también condiciona la posibilidad y necesidades del agente para que llegue a su objetivo. Al final, al no ser un cambio tan diferencial aprenderá a esquivar el obstáculo, pero hay que tener en cuenta que es solo uno, en el laberinto habrá muchas paredes y no se puede permitir una ejecución demasiado larga porque no hay tanto tiempo de desarrollo, es necesario encontrar una forma de que el agente identifique las paredes y sepa esquivarlas.

Figura 8: Resultados del ejemplo sin obstáculos

Figura 9: Resultados del ejemplo con obstáculo.

En este proyecto se opta por un laberinto estático (se discuten mejoras y posibilidades en el capítulo 6.2) por lo que es necesario, antes de empezar a desarrollarlo, pensar en qué se quiere construir, qué posibilidades debe tener el agente, cómo se quiere que el agente se mueva por el escenario y más cosas de las que se hablan a continuación.

Debido a estas razones, se escogieron 4 requisitos que el laberinto debería cumplir:

- Que no sea lo suficientemente grande como para que el entrenamiento dure mucho ni demasiado pequeño para que el agente no pueda aprenderse el laberinto.

- Que el laberinto sea difícil, para que el agente no aprenda bien los caminos.
- Que el laberinto sea lo suficientemente grande para que el Minotauro y Teseo no reaparezcan cerca.
- Que el laberinto tenga un centro marcado y una salida que sea modificable, para que no aprenda siempre el mismo camino.

El laberinto seleccionado se basó en un juego de laberinto de Amazon, Maze World - Labyrinth Game[18] (véase la figura 10) debido a que cumple todos los requisitos necesarios mencionados anteriormente:

- Su tamaño es correcto, el agente puede perderse pero no lo suficiente como para llegar a un punto sin salida.
- El laberinto no es fácil, contiene paredes aleatorias que no se juntan con ningún otro punto lo que hace que sea más complicado.
- El laberinto contiene un centro correctamente señalado, para el agente Teseo, y varios caminos en la parte más exterior donde podrá aparecer el Minotauro.
- El laberinto no tiene una salida marcada, por lo que se podrían crear nuevas salidas sin problemas.

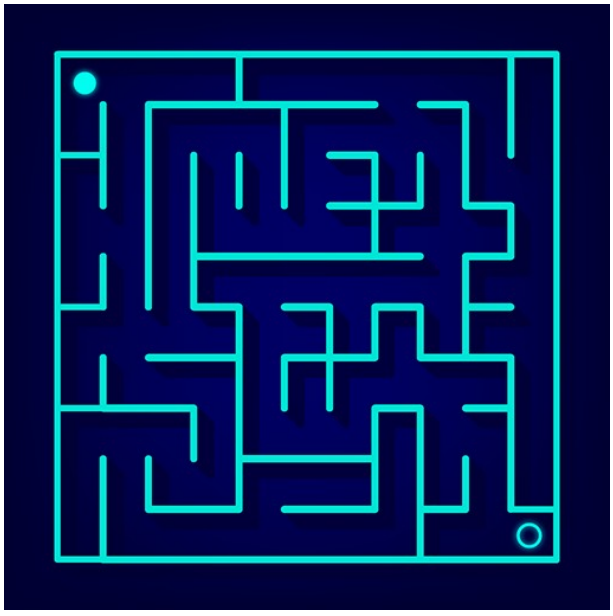

Figura 10: Laberinto de Maze World - Labyrinth Game.

El desarrollo del laberinto se puede encontrar en el anexo de la página 68.

18 Véase la página de Amazon: https://www.amazon.com/SNK-IT-Solutions-Maze-World/dp/B0777MRDFP

Los archivos de código creados fueron los siguientes:

- Pathfinding (explicado en el capítulo 6.4)
  - Node.cs
  - Grid.cs
  - Pathfinding.cs
- Teseo (explicado en el capítulo 6.6)
  - TeseoAgent.cs
- Minotauro (explicado en el capítulo)
  - MinotaurAgent.cs
- Escenario
  - TFGArea.cs

TFGArea.cs se encarga de organizar el escenario y el juego. Desde este archivo de código se decide dónde colocar el Minotauro, Teseo y la salida. Es el código central, desde el cual se conectan los demás códigos fuente.

6.3 Modelos y Prefabs

Para el desarrollo de este proyecto se han usado distintos tipos de modelos 3D, algunos construidos y otros obtenidos de manera gratuita. Los prefabs[19] se pueden definir como objetos de Unity que anteriormente han sido construidos con sus propios componentes, valores y diseños. Una vez se construye un gameobject (cualquier tipo de objeto en Unity; cámara, escenario, luces, etc.) para guardarlo en una carpeta es necesario crear un prefab del mismo objeto, así, la siguiente vez que se quiera usar este objeto simplemente se abre el prefab y ya se tiene directamente todo el gameobject construido. Para ver los modelos y prefabs usados dirigirse al anexo de la página 64.

6.4 Pathfinding

El laberinto está construido, pero no tiene todas las capacidades que debería tener un laberinto, por ejemplo, no tiene salida.

Al construir agentes mediante aprendizaje por refuerzo se requiere un cierto punto de aleatoriedad para que el agente no se aprenda el camino, puesto que una salida y entrada estáticas solo van a hacer que el agente se aprenda el camino y realmente no investigue para salir. La capacidad de que el agente sepa moverse correctamente por el laberinto es uno de los objetivos cruciales de este proyecto.

19 Véase la página oficial de Unity: https://docs.unity3d.com/es/530/Manual/Prefabs.html

La aleatoriedad no puede darse en el propio Teseo, dado que debe estar en el centro del laberinto al empezar la ejecución. El Minotauro, en cambio, puede estar en cualquier parte del laberinto y la salida, debería, estar en cualquier extremo del escenario. Entonces podemos asumir estos roles por cada apartado de la realización:

- Teseo: Aparece en el centro del laberinto.
- Minotauro: Aparece en cualquier parte del laberinto
- Salida: Aparece en un extremo del escenario.

Se debe asumir que estas características marcarían un correcto funcionamiento del proyecto. Pero a la hora de la realización del aprendizaje automático se debe enseñar a los agentes poco a poco, es difícil que el agente Teseo aprenda a ir hasta la salida si ésta está lo más alejada posible de él, debido a que al comienzo no tiene constancia de que la salida vaya a darle una recompensa positiva. Por eso es importante marcar un plan de trabajo:

- Inicio: La salida estará cerca de Teseo para que aprenda que es el objetivo. Cuando aprenda y dé un resultado positivo se continuará al siguiente apartado.
- Intermedio: Se irá alejando poco a poco la salida, cuando dé un resultado positivo se continuará al siguiente apartado.
- Final: El final estará en el extremo del laberinto.

Todo esto debe funcionar mientras hay un hilo marcando el camino desde Teseo hasta la salida, se ha decidido que en vez de solo un hilo haya trozos de hilo debido a 2 razones: El aprendizaje automático en Unity funciona mejor con objetos simples en el escenario, cuando Teseo toque un trozo de hilo se le puede premiar de mejor manera, y si se desea que el agente Teseo y el agente Minotauro sean lo más parecidos posibles por dentro, como el Minotauro perseguirá huellas se puede usar esta misma técnica pero con Teseo persiguiendo trozos de hilo. Se pueden observar los modelos 3D de huella e hilo en el escenario en la figura 11.

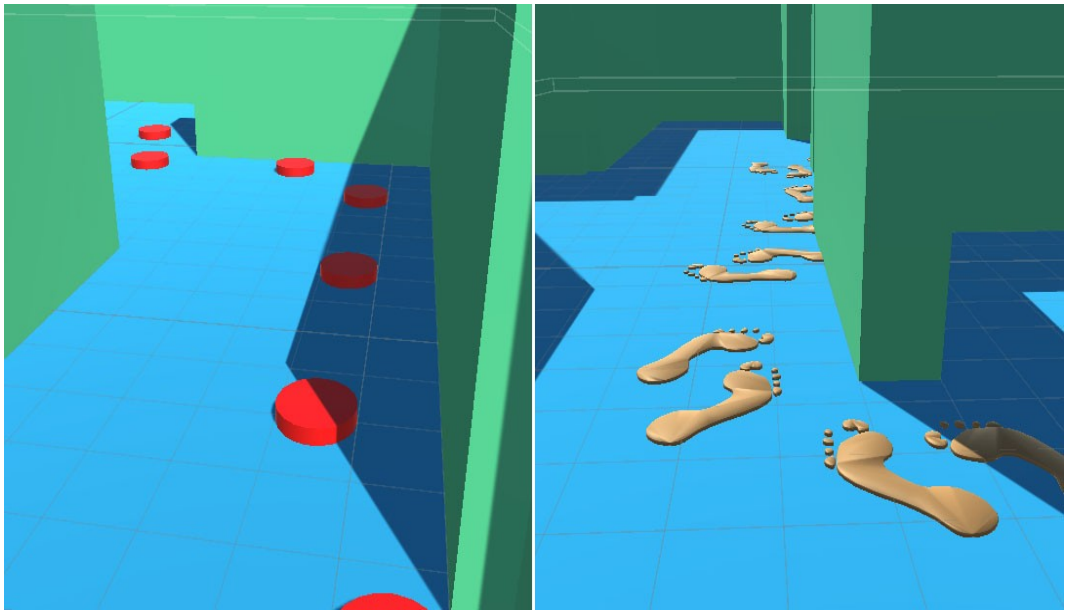

Figura 11: Modelos 3D de Hilo y Huella en el escenario.

Una vez definido las posiciones que puede tener la salida aparecen dos opciones para la construcción del hilo:

- Posiciones estáticas: No es recomendables, debido a que las salidas y los hilos deberían tener una posición definida, y no da elección para la aleatoriedad.
- Posiciones dinámicas: La salida puede aparecer en cualquier parte del escenario, pero ello conlleva que un algoritmo calcule el camino desde Teseo a la salida y coloque trozos de hilo.

La elección fue la de las posiciones dinámicas. Entonces, lo único que quedaba era decidir el algoritmo que calculase el camino desde Teseo y la salida.

La búsqueda de ruta (pathfinding en inglés) se trata de un algoritmo conocido en el mundo de la informática. El algoritmo de Dijkstra, escrito por Edsger W. Dijkstra, pionero de la rama de la computación, en 1959, es el algoritmo desde el cuál podemos partir debido a que es considerado el algoritmo base para la búsqueda de caminos.

Este algoritmo funciona con nodos en un grafo, la idea es decodificar el suelo del escenario en rectángulos donde cada rectángulo represente un nodo y tenga conexión con los nodos vecinos, de esta manera el suelo del escenario es decodificado en un grafo grande de nodos. Mediante esto se puede aplicar el algoritmo de Dijkstra para buscar la mejor ruta entre dos puntos.

Si bien es un algoritmo útil, en los videojuegos se usa una variación de este, el algoritmo A*. Con este algoritmo cada nodo tiene un peso asignado respecto a la distancia recorrida desde el nodo que da el comienzo hasta el mismo, cuyo valor es determinado por una heurística. Si el peso de todos los nodos fuera 0, el algoritmo sería igual que el algoritmo de Dijkstra (Lester, 2005). En la figura 12 se puede observar un ejemplo de algoritmo A*.

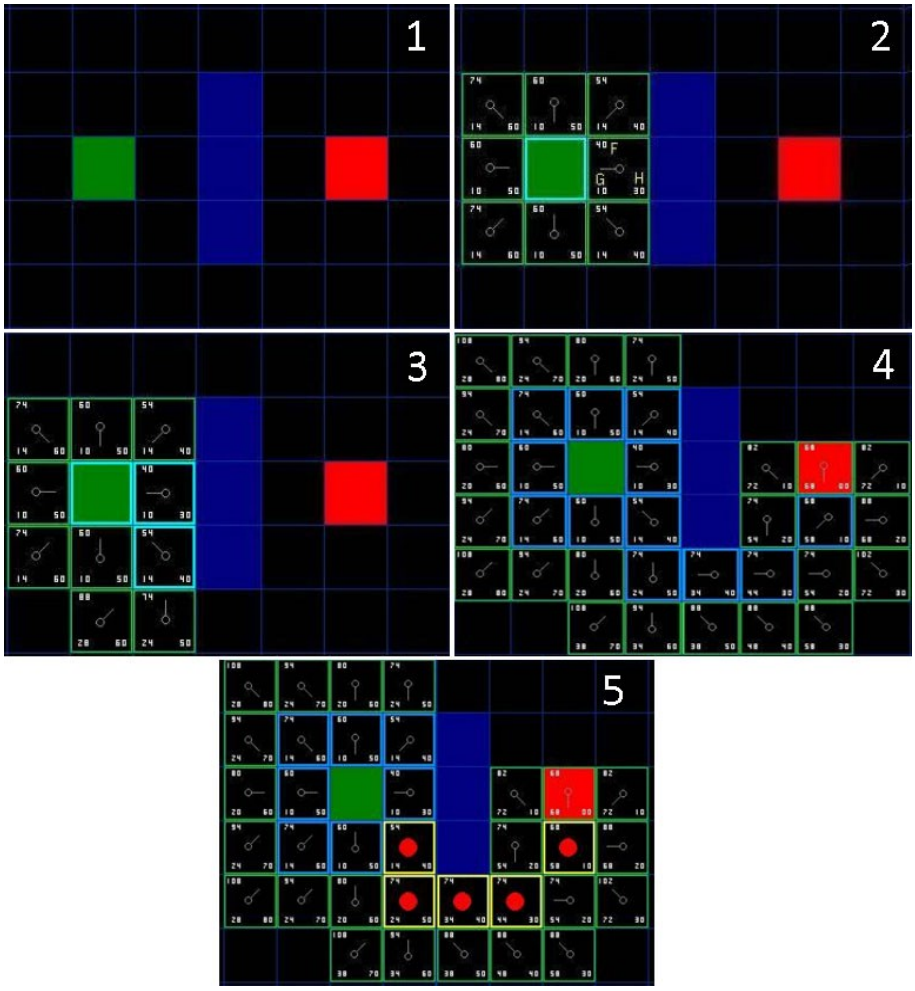

Figura 12: Ejemplo de funcionamiento del algoritmo A. Por Lester (2005).*

En este trabajo se ha usado como base para la realización del algoritmo de A* un tutorial de vídeo extraído de YouTube del canal Daniel[20] que proporciona toda la documentación necesaria para la aplicación del algoritmo en el motor de videojuegos Unity.

Para ello, necesitamos:

- 3 archivos de códigos:
  - Una clase Nodo que sirva de manera de constructor y cálculo de la heurística.
  - Un código fuente Grid que transforme el mapa a un grafo de nodos y calcule los vecinos de cada nodo junto a los valores que ayuden al cálculo del peso de cada nodo.
  - Un código fuente Pathfinding que, una vez se tenga el escenario transformado a un grafo de nodos donde cada nodo tiene un peso asignado, calcule mediante el algoritmo A* el mejor camino posible.

20 Véase la bibliografía para encontrar el vídeo.

Primero, es necesario aplicar componentes al escenario para que funcione. El escenario contendrá, de manera estática, suelo y paredes. Los agentes deben ser capaces de caminar por el suelo sin ningún problema pero no pueden atravesar paredes ni subirse encima. Para que el código fuente Grid transforme correctamente el escenario necesitamos aplicarle una máscara de etiquetas para objetos que son obstáculos. Para ello asignamos a todas las paredes la etiqueta Pared y capa Wal (figura 13) y marcamos en el propio código fuente que si el nodo está en contacto con una pared con esa etiqueta o capa debe marcarse como prohibido. Se puede ver la solución en la figura 14 y en la figura 15. El color gris indica los nodos disponibles y el color amarillo indica nodos imposibles de seleccionar debido a que son paredes.

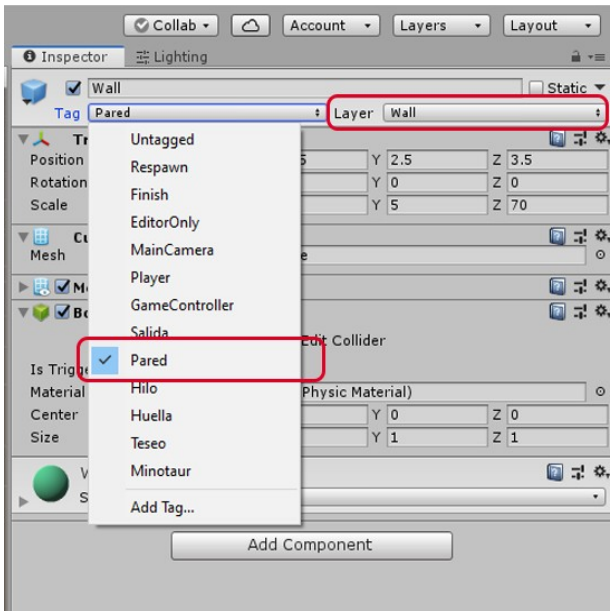

Figura 13: Etiqueta pared y capa wall: distinción al aplicar el algoritmo A.*

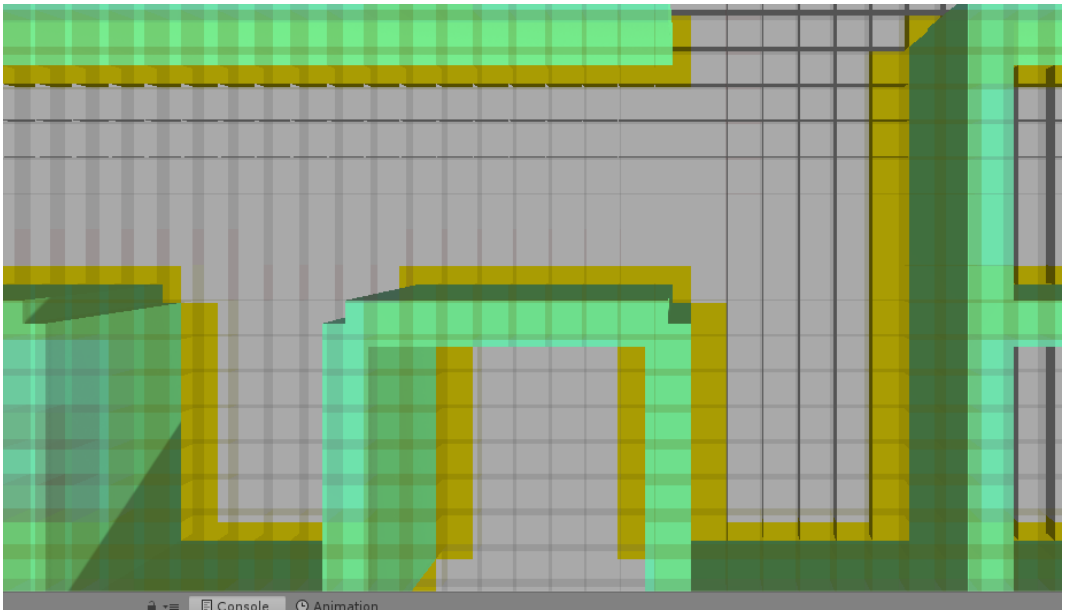

Figura 14: Ejemplo 1 del algoritmo A en el escenario.*

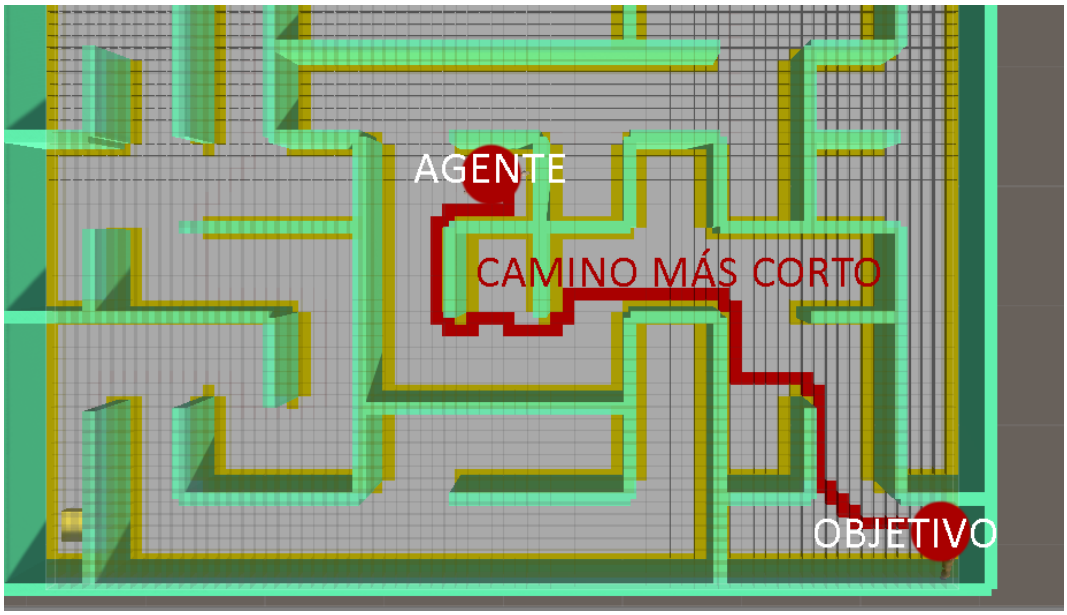

Figura 15: Ejemplo 2 del algoritmo A en el escenario.*

La heurística que se usa será la de la distancia de Manhattan por ser una heurística admisible (Korf, 2000).

Una vez se haya decidido el camino correcto, es hora de colocar los trozos de hilo. El prefab de hilo (visto en el capítulo 6.3) marcará los trozos de hilo que debe de seguir el agente Teseo, este prefab se colocará dependiendo de una variable cambiante que marque cada cuántos nodos

instanciaremos el mismo. La variable comenzará con el valor 3, en la etapa fácil y cuanto más se dificulte la ejecución el valor irá aumentando.

6.5 La clases usadas: Clase MonoBehaviour y Agente

La clase MonoBehaviour[21] es la clase más conocida de Unity por ser la clase base al usar el lenguaje C#.

Estas son las funciones y características propias de la clase que se usan para este proyecto:

- `private void Start()`
  - Cuando empieza la ejecución se llama a esta función. Generalmente se usa para inicializar variables locales.
- `private void Update()`
  - Esta función se ejecuta cada fotograma.
- `private void FixedUpdate()`
  - A diferencia de `Update()`, esta función se ejecuta junto a un cambio en las físicas. Puede ejecutarse más de 2, 1 o 0 veces cada fotograma, dependiendo de si ha ocurrido un cambio de físicas o no[22].
- `private void OnCollisionEnter(Collision collision)`
  - En Unity, cuando dos objetos que contienen una hitbox colisionan se ejecuta este programa, recibiendo como parámetro el objeto con el que ha chocado. Gracias a esta función podemos comprobar que el Minotauro ha tocado a Teseo y puede matarlo, por ejemplo.

El agente Teseo y el agente Minotauro son una clase llamada Agent que otorga el propio kit de herramientas de ML-Agents[23].

Estas son las funciones y características propias de la clase que se usan para este proyecto:

- `public void AddReward(float increment)`
  - Esta función sirve para añadir un valor como premio. Sirve tanto como para castigar o premiar al agente y el agente acumula premios hasta el final del episodio. Si por ejemplo en todo el episodio se ha ejecutado `AddReward(0.2f)` 3 veces, como premio total del episodio quedaría 0.6f.
- `public void SetReward(float reward)`
  - Esta función es parecida a `AddReward(…)` pero en vez de añadir un valor al premio total acumulado lo reemplaza por el valor pasado como parámetro. Así, si el premio

21 Véase la documentación oficial de Unity: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html
22 Véase la página de Unity: https://answers.unity.com/questions/10993/whats-the-difference-between-update-and-fixedupdat.html
23 Véase la página de Unity: https://docs.unity3d.com/Packages/com.unity.ml-agents@1.0/api/Unity.MLAgents.Agent.html

acumulado es 0.6f y se llama a `SetReward(-1.0f)` el premio total acumulado cambiará a -1.0f.

- `public virtual void CollectObservations(VectorSensor sensor)`
  - Esta función se usa para reunir las observaciones que pueden interesar al agente. Por ejemplo, para guardar la distancia en la que está el agente comparado con el objetivo, así el agente tendrá constancia de cuánta diferencia había entre el mismo y el objetivo al finalizar el episodio.
- `public void EndEpisode()`
  - Una vez el agente ha terminado, esta función acaba el episodio y lo reinicia. Después llama a `OnEpisodeBegin()`.
- `public virtual void Heuristic(float[] actionsOut)`
  - Gracias a esta función se abre la posibilidad de poder controlar el agente mediante cualquier dispositivo de entrada que tengamos conectado, sea un mando, un teclado o un ratón. Mediante esta función se puede comprobar que el agente funciona y se mueve como se ha deseado. El agente aprende a moverse por su propia cuenta comprobando distintas combinaciones de esta función.
- `public virtual void Initialize()`
  - Esta función es la correspondiente a la función `Start()` de las clases MonoBehaviour de Unity. Cuando el agente es habilitado se llama a esta función, generalmente se usa para inicializar variables locales.
- `public virtual void OnActionReceived(float[] vectorAction)`
  - Cuando el agente realiza una acción, por ejemplo la de `Heuristic()`, se llama a esta función y dependiendo de los valores enviados como parámetros realiza una función. Aquí se mueve el agente de manera completa, se calcula la posición a la que se tiene que mover, a qué dirección, etc.. .
- `public virtual void OnEpisodeBegin()`
  - Al comienzo de cada episodio se llama a esta función. Puede servir para asegurar distintos objetos del escenario que deben cambiar de episodio a episodio, como por ejemplo borrar todos los hilos existentes, puesto que en el nuevo episodio la salida puede estar en otra posición.
- `public int MaxStep`
  - Este parámetro marca el tamaño máximo de cada episodio. Se puede usar para penalizar al agente si no se está acercando al objetivo, por ejemplo, debido a que hay casos donde el agente no recibe un premio positivo ni negativo y si ocurre esto el agente preferirá quedarse quieto antes de arriesgarse a poder fallar, así el agente se moverá a buscar la solución correcta en vez de quedarse inmóvil.
- [Componente] `RayPerceptionSensorComponent3D`

- Son los ojos del agente, los sensores que marcan lo que el agente ve. Se puede configurar al gusto del desarrollador con opciones como marcar qué objetos ve y a cuáles no prestar atención, a qué dirección mirar o cuántos sensores tiene activos.

6.6 El agente Teseo

El agente Teseo tiene un solo objetivo: Salir del laberinto. Para ello dispondrá de unos hilos que le guíen a la salida. La solución, entonces, será seguir los hilos hasta llegar al final. Además de esto, el Minotauro intentará atraparlo, así que tiene que tener cuidado con no encontrarse con él.

El agente Teseo dispondrá de 5 mil pasos para lograr el objetivo por episodio, por cada paso que haga se le recompensará de manera proporcional negativamente para que el agente no se quede quieto e intente buscar hilos y la salida. Cuando el episodio acabe, se reiniciará la estancia.

Para ello se ha pensado en 5 fases por los que el agente puede aprender la solución final:

1. En la primera fase, el agente Teseo aparecerá en el centro del escenario, con una rotación aleatoria y su objetivo, la salida, aparecerá en la vuelta de la esquina, los dos con una posición de reaparición estática. El agente Teseo y su objetivo estarán lo suficientemente cerca como para encontrarse rápido pero no tanto como para que no haya espacio para hacer aparecer hilos entre ellos. En principio, cada 3 celdas de pathfinding aparecerá un hilo. Además, el Minotauro aparecerá en una esquina del recorrido pero estará estático, para que Teseo aprenda que acercarse a él no es buena idea. En la figura 16 se puede ver esta representación.

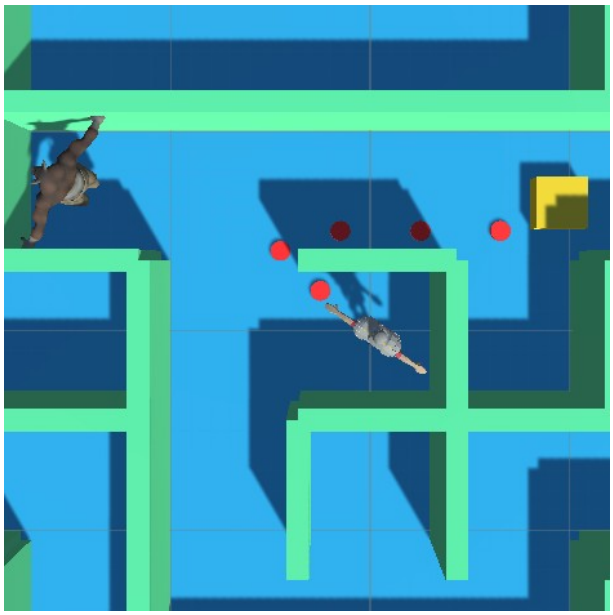

Figura 16: Fase 1 del aprendizaje del agente Teseo.

2. En la segunda fase, el objetivo, la salida, aparecerá aleatoriamente en un radio de entre 2 y 8 unidades del centro, de la misma manera será la reaparición del Minotauro. Gracias a la fase 1 el agente Teseo ha aprendido a que seguir los hilos es positivo pero como antes estaba siempre en una posición concreta el agente Teseo ahora tiene que aprender a no ir a la posición exacta, si no a buscar y a seguir los hilos. Teseo ahora no puede tocar las paredes, así aprende a centrarse más en seguir su objetivo sin chocarse con ellos. Obsérvese la figura 17.

3. En la tercera fase, el agente Teseo ya ha aprendido a seguir los hilos aunque no salgan siempre en el mismo sitio. Ahora el radio de reaparición del objetivo agranda de entre 2 a 8 unidades a de entre 4 a 10 unidades, lo mismo pasa con el Minotauro. Además cuando se reinicie la estancia el objetivo no cambiará de posición, por lo que el agente Teseo tiene que aprender a llegar hasta su posición correctamente. Véase la figura 17.

4. En la cuarta fase el agente Teseo ya es rápido y sabe manejarse por el escenario, ahora además de un radio de aparición más grande para el objetivo, de entre 12 a 20, los hilos aparecerán cada 4 celdas de pathfinding. El objetivo vuelve a aparecer aleatoriamente al reiniciar la estancia, como el Minotauro. Desaparece la letalidad de las paredes para que el aprendizaje se asemeje más al resultado definitivo. Obsérvese la figura 17.

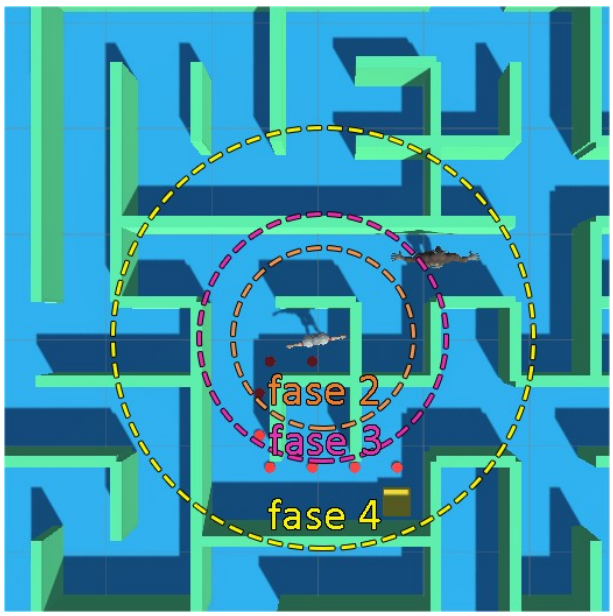

Figura 17: Fase 2, 3 y 4 del aprendizaje del agente Teseo.

5. En la quinta fase, la denominada fase final, se deben asentir las bases finales para estimar un buen funcionamiento. El Minotauro y la salida aparecerán en los caminos extremos del escenario, mientras que el agente Teseo aparecerá en el centro. Así será la ejecución final así que es importante ver si el aprendizaje desarrollado hasta el momento ha servido o no. Se puede observar en la figura 18.

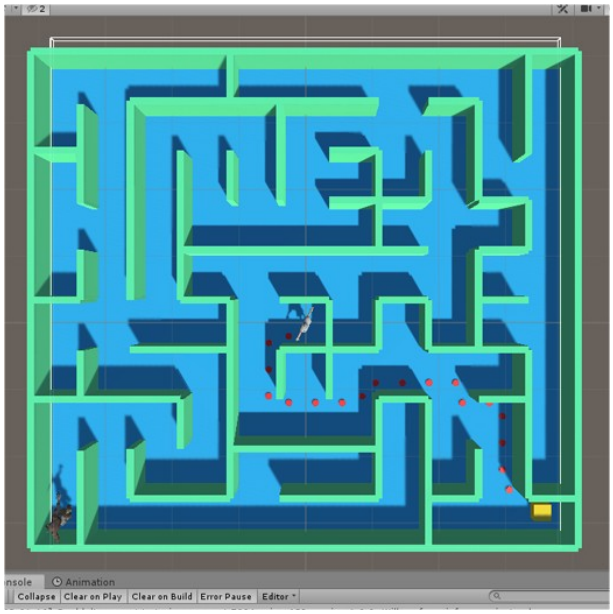

Figura 18: Fase 5 del aprendizaje del agente Teseo.

Una vez entrenado el agente, con un resultado positivo en todas las fases, se puede denominar la ejecución como un éxito, el agente ya ha aprendido qué debe de hacer.

6.7 El agente Minotauro

El agente Minotauro, a diferencia del agente Teseo, no ve las huellas o rastros que le conducen al objetivo con tanta frecuencia, por tanto es importante entrenarlo con cada vez menos huellas para que se acostumbre a no tener siempre pistas marcándole el paso.

El agente Minotauro, como el agente Teseo, dispondrá de 5 mil pasos para lograr el objetivo en cada episodio, por cada paso que haga se le recompensará de manera proporcional negativamente. Para que el agente no se quede quieto. Si en 5 mil pasos el agente no consigue llegar al objetivo se reiniciará la estancia.

Para el agente Minotauro se decidió hacer dos aprendizajes distintos, uno más parecido al escenario final de los agentes y el otro más parecido al aprendizaje tenido por Teseo, para poder comprobar las diferencias que pueden dar dos aprendizajes distintos.

El agente entrenado como Teseo fue llamado Minotauro-1 y al ser muy parecido al aprendizaje de Teseo, tuvo estas fases:

1. En la primera fase, el agente Minotauro aparecerá en el centro del escenario, con una rotación aleatoria y su objetivo, Teseo, aparecerá en la vuelta de la esquina, los dos con una posición de reaparición estática. El agente Minotauro y su objetivo estarán lo suficientemente cerca como para encontrarse rápido pero no tanto como para que no haya espacio para hacer aparecer huellas. En principio, cada 3 celdas de pathfinding aparecerá una huella. Véase la figura 19.

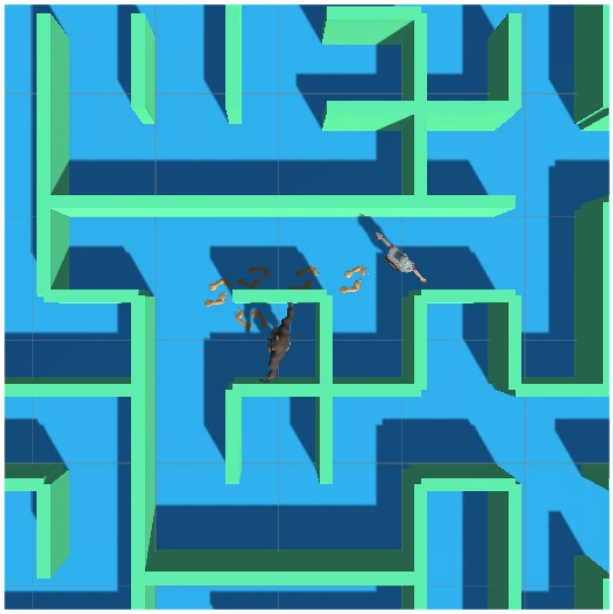

Figura 19: Fase 1 del aprendizaje del agente Minotauro.

2. En la segunda fase, Teseo, aparecerá aleatoriamente en un radio de 10 a 15 unidades del centro. Gracias a la fase 1, el agente Minotauro ha aprendido a que seguir las huellas es positivo pero como antes estaba siempre en una posición concreta, el agente Minotauro ahora tiene que aprender a no ir a la posición exacta, si no a buscar y a seguir las huellas. Esto se puede observar en la figura 20.
3. En la tercera fase, el agente Minotauro ya ha aprendido a seguir las huellas aunque no salgan siempre en el mismo sitio. Ahora el radio de reaparición del objetivo agranda y además cuando se reinicie la estancia el objetivo no cambiará de posición, por lo que el Minotauro tiene que aprender a llegar hasta su posición rápidamente. Además, se reduce la frecuencia de aparición de las huellas, ahora cada 4 celdas de pathfinding aparecerá 1. Véase la figura 20.
4. En la cuarta fase el agente Minotauro ya es rápido y sabe manejarse por el escenario, ahora además de un radio de aparición más grande para el objetivo, las huellas aparecen cada 5 celdas de pathfinding. El objetivo vuelve a aparecer aleatoriamente al reiniciar la estancia. Se puede observar esto en la figura 20.

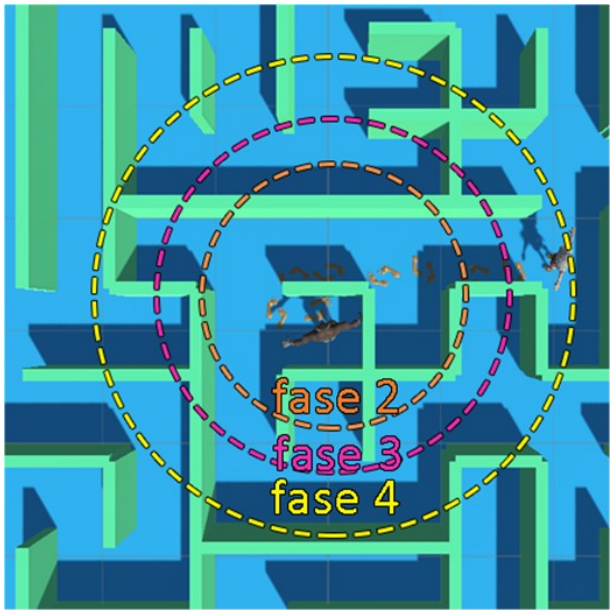

Figura 20: Fase 2,3 y 4 del aprendizaje del agente Minotauro-1.

5. En la quinta fase, la denominada fase final, se deben asentir las bases finales para estimar un buen funcionamiento. El agente Minotauro seguirá apareciendo en el centro del escenario, y su objetivo, el agente Teseo, en un extremo del mapa. Cada 6 celdas de pathfinding aparecerá una huella por lo que el Minotauro a veces se verá perdido debido a que no encontrará cerca ninguna huella, es importante que el agente aprenda a manejar ese tipo de situaciones lo mejor posible puesto que en la ejecución final pueden ocurrir casos así. Véase la figura 21.

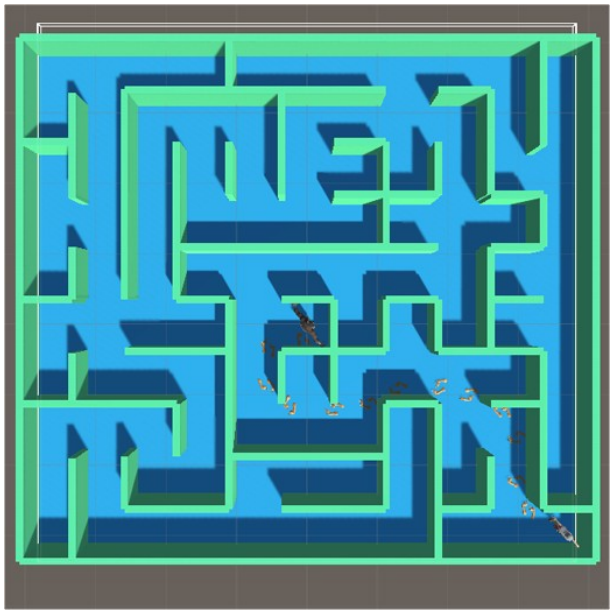

Figura 21: Fase 5 del aprendizaje del agente Minotauro-1.

El agente que fue entrenado de distinta manera se llamó Minotauro-2 y tuvo estas fases:

1. La primera fase es igual que la del Minotauro-1, es la manera más óptima para que el agente aprenda bien su objetivo y que las huellas le benefician. Véase la figura 19.
2. Para la segunda fase el agente Minotauro seguirá apareciendo en el centro y el objetivo Teseo empezará a reaparecer en un sitio aleatorio del escenario a una distancia de entre 5 a 20 unidades. Así el agente aprende a seguir las huellas. Las huellas aparecerán cada 3 celdas de pathfinding. Se puede observar en la figura 22.

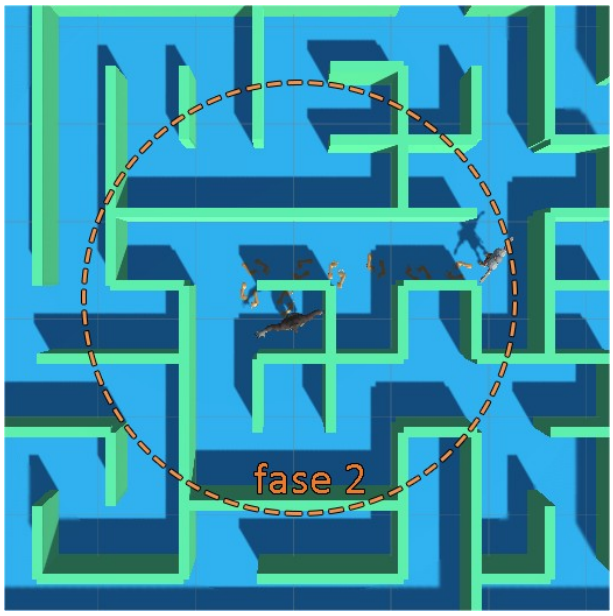

Figura 22: Fase 2 del aprendizaje del agente Minotauro-2.

3. En la tercera fase el agente Minotauro ha relacionado las huellas con Teseo y sabe que seguirlas le dará una buena recompensa, así que el propio agente Minotauro empieza a reaparecer en sitios aleatorios del mapa. Aún así, en esta fase se disminuye la distancia de reaparición del objetivo a una distancia de entre 5 a 12 unidades debido a que las paredes le matarán. Esto se hace para que el agente no se acostumbre a quedarse mirando a las paredes, porque así no puede ver el objetivo y si hay huellas en el camino. También se disminuye la aparición de huellas a cada 4 celdas de pathfinding. Véase la figura 23.
4. Para la cuarta fase la letalidad de las paredes se mantiene, se aumenta la distancia de reaparición del objetivo a 15 en vez de a 12 y las huellas aparecerán solo cada 5 celdas de pathfinding. Véase la figura 23.

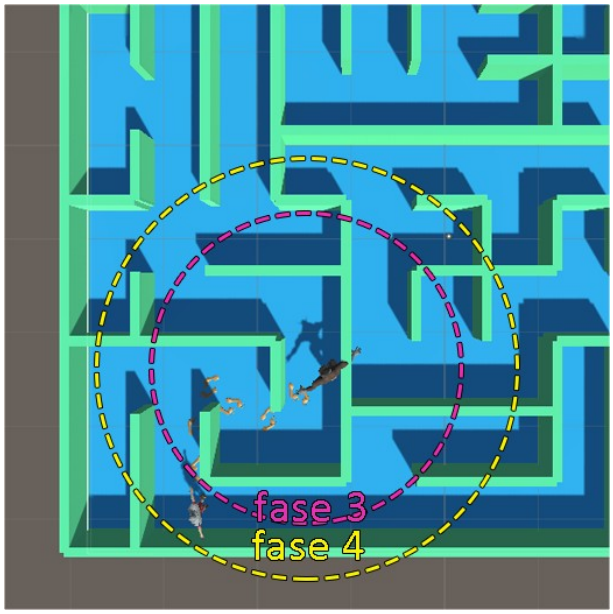

Figura 23: Fase 3 y 4 del aprendizaje del agente Minotauro-2.

5. La última fase es la definitiva, Teseo aparecerá en el centro mientras que el agente Minotauro aparecerá en un extremo del mapa. La pared ya no lo mata pero ya ha aprendido que quedarse mirando la pared no es una buena idea así que no resulta un problema. Se puede observar en la figura 24.

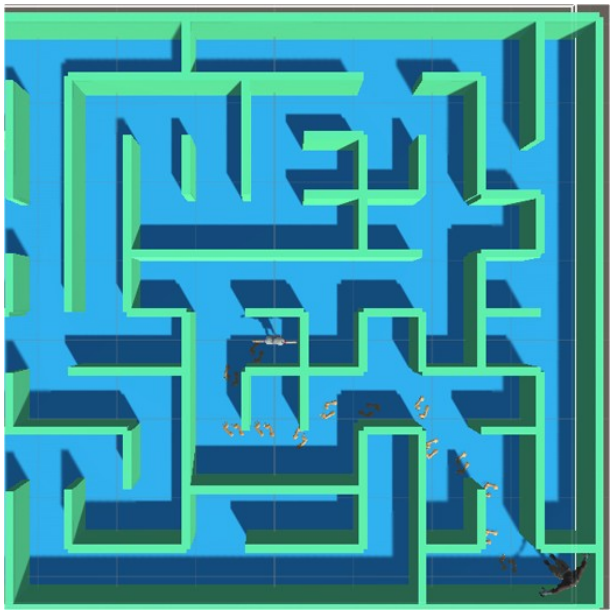

Figura 24: Fase 5 del aprendizaje del agente Minotauro-2.

Una vez entrenados los dos tipos de agente, con un resultado positivo en todas las fases, se puede denominar las ejecuciones como un éxito, el agente ya ha aprendido qué debe de hacer.

6.8 Config

A la hora de entrenar un agente en Unity existe un archivo comúnmente llamado config (trainer-config o simplemente training configuration) que define los parámetros del entrenamiento que se va a realizar.

Como en cualquier aprendizaje automático, el aprendizaje no se realiza simplemente decidiendo que se quiere realizar aprendizaje por refuerzo, también existen muchas combinaciones posibles para una ejecución totalmente personalizada.

Por ejemplo, si se desea que un agente realice 10mil pasos para terminar el proceso de entrenamiento en vez de los 5mil que son por defecto, se puede configurar, hasta más de 30 configuraciones distintas que empeorarán o mejorarán el aprendizaje deseado dependiendo del escenario y el agente a entrenar. En este apartado, se analizarán las que han resultado más importantes para el desarrollo del proyecto, para obtener la definición del resto de los parámetros se puede dirigir a la página oficial de Unity[24].

- Tipo de entrenamiento (ppo o sac): Es el parámetro más importante para el entrenamiento. Unity recomienda, si el usuario no es un entendido de la materia, usar PPO.
    - PPO (Proximal Policy Optimization): La idea principal detrás de este tipo de entrenamiento es evitar tener una actualización de política demasiado grande entre entrenamientos, "The central idea of Proximal Policy Optimization is to avoid having too large policy update" (Simonini, 2018). Gracias a esto, se consigue estabilizar el entrenamiento limitando las políticas mediante una función llamada Clipped Proximal Policy Optimization Algorithm[25]. En este proyecto se entrenan los dos agentes mediante PPO.
    - SAC (Soft Actor Critic): A diferencia de PPO, SAC es considerado *off-policy*, que significa que el agente puede aprender también de versiones pasadas. Cada experiencia pasada se guarda en un buffer y será seleccionado aleatoriamente para el aprendizaje, esto no es lo que interesa en este proyecto y por ello se ha descartado este aprendizaje.
- summary_freq: Es un parámetro que permite definir cuántas experiencias se deben de completar para generar y mostrar los datos estadísticamente. Sirve para los gráficos que se verán en el capítulo 7. Define la granularidad de los gráficos de Tensorboard[26], si se quisieran usar.
- time_horizon: Define cuántos pasos se deben de completar para guardarlos en el búfer de experiencia. Así, si este parámetro tiene el valor 35, pasados 35 pasos se guardará la experiencia para el desarrollo del agente.

24 Véase la documentación de Unity: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Configuration-File.md
25 Véase https://vitalab.github.io/article/2019/05/09/PPO.html
26 Obsérvese https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Using-Tensorboard.md

- batch_size: Número de experiencias de demostración utilizadas para una iteración de una actualización de *descenso de gradiente* (un cálculo que permite ajustar los parámetros para minimizar la desviación de salida).
- buffer_size: Número de experiencias que reunir antes de actualizar la política.
- hidden_units: Número de unidades en las capas ocultas de la red neuronal. El valor corresponde a cuántas unidades "neuronales" están completamente conectadas en cada capa de la red neuronal. En el capítulo 4.1 se habló de esto.
- beta: Fuerza de la regularización de la entropía, lo que hace que la política se más aleatoria.
- max_steps: Cuántos pasos se han de realizar antes de concluir el episodio, por defecto son 5 mil.

Una de las particularidades del documento de configuración de ML-Agents es la posibilidad de modificar para cada aprendizaje particular distintos parámetros reescribiéndolos. Así, se pueden usar los valores por defecto que ofrece Unity y solo modificar los necesarios para el aprendizaje que se quiera.

Estos son los valores por defecto que se usarán para el aprendizaje:

```
default:
    trainer: ppo
    batch_size: 1024 // ← Se modificará
    beta: 5.0e-3 // ← Se modificará
    buffer_size: 10240 // ← Se modificará
    epsilon: 0.2
    hidden_units: 128
    lambd: 0.95
    learning_rate: 3.0e-4
    learning_rate_schedule: linear
    max_steps: 5.0e5 // ← Se modificará
    memory_size: 128
    normalize: false
    num_epoch: 3
    num_layers: 2
    time_horizon: 64 // ← Se modificará
    sequence_length: 64
    summary_freq: 10000 // ← Se modificará
    use_recurrent: false
    vis_encode_type: simple
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99

// Los parámetros cambiados para el aprendizaje del agente Minotauro

MinotaurLearning:
    summary_freq: 5000
```

```
    time_horizon: 128
    batch_size: 128
    buffer_size: 2048
    hidden_units: 256
    beta: 1.0e-2
    max_steps: 1.0e6
// Los parámetros cambiados para el aprendizaje del agente Teseo
TeseoLearning:
    summary_freq: 5000
    time_horizon: 128
    batch_size: 128
    buffer_size: 2048
    hidden_units: 256
    beta: 1.0e-2
    max_steps: 1.0e6
```

6.9 Curricula

En un escenario simple el entrenamiento no necesita una modificación de valores. Por ejemplo, si se quiere entrenar un agente que siempre juegue en un mismo escenario no se necesita dificultar el entrenamiento para que aprenda más de lo que necesita.

Unity ofrece una manera de dificultar el escenario para que el agente aprenda de manera progresiva hasta llegar a un resultado que sería imposible de lograr (o muy lento) si se entrenase desde el comienzo con esos valores, esto se consigue mediante un archivo YAML[27] que se denomina como curriculum[28].

Como se ha definido al hablar de los agentes, se puede definir el entrenamiento en distintas fases. Gracias a la herramienta que nos ofrece Unity, el curriculum, se pueden categorizar y diferenciar distintos tipos de valores para distintos niveles de entrenamiento. Por ejemplo, si se entrena un coche en una carrera para que nunca se choque, se puede empezar con un entorno pequeño, no más de 200 metros de circuito en el que aprende que chocarse con las paredes es malo. Una vez aprende eso, se le pueden añadir obstáculos al circuito para que también aprenda a que chocarse con objetos no es bueno y así hasta conseguir un nivel óptimo de conducción, haciendo que el agente vaya aprendiendo poco a poco, consiguiendo evitar una carga de trabajo muy grande para el agente en un comienzo.

La idea de la realización de esto proviene de un artículo escrito Yoshua Bengio (premio Turing 2018), Jérôme Louradour, Ronan Collobert y Jason Weston en 2009. El artículo sostiene que el entrenamiento gradual y organizado es necesario para conseguir un funcionamiento óptimo del aprendizaje automático, "Humans and animals learn much better when the examples are not randomly presented but organized in a meaningful order which illustrates gradually more concepts, and gradually more complex ones". En el artículo mencionado, se comprueba que aumentar gradualmente la dificultad aumenta la velocidad del entrenamiento. Al final es lógico, un niño aprende primero a estar de pie y luego a caminar, si se le intenta enseñar a caminar sin

27 Véase https://yaml.org/
28 Véase la documentación de Unity: https://blogs.unity3d.com/es/2017/12/08/introducing-ml-agents-v0-2-curriculum-learning-new-environments-and-more/

todavía aprender a conseguir un equilibrio con su cuerpo para no caer, el aprendizaje será más duro y existe la posibilidad de que no consiga aprenderlo.

El formato del curriculum es simple, estas características componen la configuración del archivo[29]:

- measure: Lo que se va a usar para medir el proceso de aprendizaje, existen dos opciones: reward o progress. Reward usa el premio y ganancias recibidas mientras que progress funciona mediante el cálculo del ratio de pasos realizados dividido con los pasos máximos posibles de hacer. Los dos pueden ser válidos dependiendo del escenario que se construya.
- thresholds: Es la característica más importante del archivo, se decide el umbral mínimo del measure para pasar a la siguiente fase. Se entiende mejor con el ejemplo a continuación.
- min_lesson_length: Define la cantidad de episodios que han de ocurrir y que hayan superado el threshold para pasar a la siguiente fase.
- signal_smoothing: Con valores posibles de true o false, define si se deberían tener en cuenta los valores anteriores para el peso del progreso actual. Normalmente esta opción suele ser true.

Las tres primeras características definen la utilidad que tiene el curriculum. Por ejemplo, si el measure es reward con el primer threshold siendo 0.6 y junto con un min_lesson_length de 60, cuando ocurran 60 episodios seguidos donde el premio de cada episodio sea mejor que 0.6 se continuará a la siguiente fase.

Se habla de fases, pero mediante estas características no se puede conseguir un cambio directo en el escenario, por ello existe otra característica llamada parameters:

- parameters: Dentro de esta opción se definen los parámetros que controlarán el cambio en el escenario. Se pueden definir tantos como se quiera y se consigue su valor dentro del código realizando una llamada al propio curriculum. Si se tienen P thresholds deberían existir P+1 valores de parámetros para cada fase, debido a que el inicio de la ejecución no entra dentro del valor del threshold.

Un ejemplo de parameters podría ser el de cuántos enemigos se quieren hacer aparecer en el escenario, comenzando con 1 y terminando gradualmente con 4. Además, se puede definir también la velocidad del jugador para que cada nivel que progrese sea más rápido y tenga más posibilidades de escapar, por poner un ejemplo. La configuración total de un curriculum sería de este estilo:

```
measure: reward
thresholds: [0.1, 0.3, 0.5]
```

29 Véase https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-ML-Agents.md si se quiere una explicación más detallada.

```
min_lesson_length: 100
signal_smoothing: true
parameters:
      enemigos_posibles: [1.0, 2.0, 3.0, 4.0]
      velocidad_jugador: [12.0, 12.5, 13.0, 13.5]
```

Nótese que cuando empieza la ejecución del aprendizaje se empieza con los valores de `enemigos_posibles=1.0` y `velocidad_jugador=12.0`, solo cuando se llegue a un thresholds de 0.1 cambiarán a `enemigos_posibles=2.0` y `velocidad_jugador=12.5`, y así progresivamente hasta llegar al último.

Para realizar una llamada desde el código interior del proyecto se usan estos parámetros:

- `agente_actual`: Define qué agente se está entrenando, `0.0` para Teseo y `1.0` para Minotauro.
- `fase`: En qué fase se está, empieza en `1.0` y termina en `5.0`, como definido en el capítulo 6.6 y el capítulo 6.7.
- `aleat_spawn`: Si va a existir una aleatoriedad de reaparición o no. Si el valor es `0.0` significa que la reaparición de los objetos en el escenario va a ser estática y si es un `1.0` que la reaparición de los objetos será aleatoria.
- `radius_spawn`: Esto indica el valor máximo que tendrá el radio de aparición del objeto.
- `min_radius_spawn`: En relación directa con `radius_spawn`, indica el mínimo valor de reaparición que va a tener respecto a su centro, para que no aparezca al lado de Teseo por ejemplo.
- `reward_hilo`: La recompensa que dará en total la recogida de los hilos, llegar al objetivo le dará una recompensa de su 80%, si por ejemplo tiene un `reward_hilo` de `4.0`, si coge todos los hilos obtendrá una recompensa de `4.0` pero si llega al objetivo conseguirá un `4.0` + `3.2` (0.8*`4.0`) = `6.2`.
- `reward_huella`: Lo mismo que `reward_hilo` pero para las huellas.
- `probabilidad_rastro`: La probabilidad en la que reaparecerán los rastros (hilos o huellas). Por ejemplo si el valor es `3.0`, cada 3 celdas aparecerá un rastro.
- `letal_pared`: Indica si la pared debe de ser letal, con un valor de `1.0`, o no letal, con un valor de `0.0`.

Los archivos completos de curricula desarrollados para este proyecto, así como los `thresholds` elegidos después de muchas pruebas por ser los más óptimos están en el Anexo de la página 71.

7

7 Pruebas concluyentes

En este capítulo se analizan las pruebas satisfactorias realizadas y los valores dados.

7.1 Agente Teseo

El aprendizaje del agente Teseo es rápido y conciso. Gracias a la facilidad de dificultar el aprendizaje, se consigue un agente que presta atención a los hilos que pueda haber en el escenario y los sigue sin perderlos de vista.

Como se puede ver en los resultados, figura 25, el agente comienza explorando. Al principio consigue una buena solución llegando a la salida pero no ha entendido todavía que eso es lo que debe hacer. Se puede observar que en el paso 10000 la recompensa del agente cae a un 0.12, esto se debe a que el agente ha intentado explorar otras alternativas para descubrir si hay más estímulos que le den una recompensa mayor, como descubre que no los hay, vuelve a intentar realizar la ejecución que mejor recompensa le ha otorgado.

Una vez consigue una recompensa media de 5-6.5, pasa a la fase 2 y, como es obvio, su media de recompensa vuelve a bajar. El agente ahora tiene que aprender que la salida no estará siempre en un mismo sitio, por lo que poco a poco va aprendiendo y consigue llegar a la fase 3.

En la fase 3 ocurre algo excepcional, el agente ya ha aprendido que seguir los hilos le dará una buena recompensa, así que independientemente de la distancia a la que aparece la salida, el agente sigue intentando alcanzarla culminando en un resultado aún más positivo.

Una vez llega hasta la fase 4 la recompensa de los hilos aumenta mucho debido a que la salida estará todavía más lejos de Teseo, posicionada en caminos que Teseo todavía no ha sido capaz de explorar. Pero eso no frena al agente y, como ya conoce cuál es la mejor solución, la media de recompensas no disminuye. Se puede comprobar cómo la media de recompensas oscila entre el 8-12, un resultado inmejorable.

Una vez realizadas las suficientes ejecuciones para cambiar de fase, se llega a la última y definitiva fase, la fase 5. Aquí la media de recompensas empieza a bajar debido a que ahora la salida está más lejos que nunca y son caminos complicados con gran cantidad de giros. Aun y todo, el agente Teseo cae hasta una media de recompensa del 5 y medio, pero desde esa posición comienza a ganar fuerza y su mejora es notable. Ahora el agente ha aprendido lo que debe hacer y no hay dificultad que le haga frente.

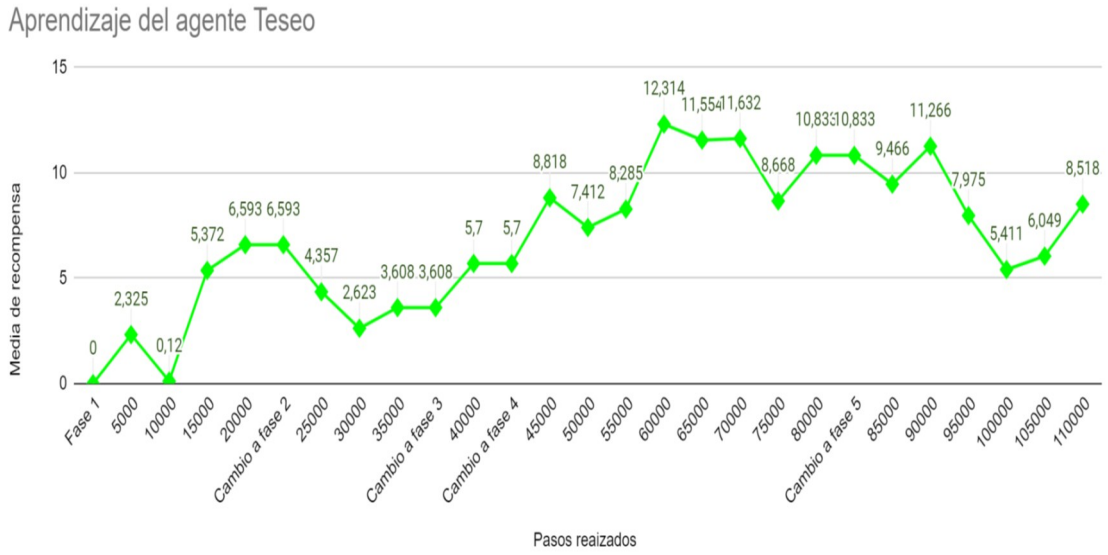

Figura 25: Aprendizaje del agente Teseo. Tiempo de ejecución aproximado: 40 minutos.

7.2 Agente Minotauro

Los diferentes aprendizajes de Minotauro-1 y Minotauro-2 otorgan dos experiencias totalmente distintas. Un agente entrenado como Teseo, obtendrá un resultado de recompensas y pasos parecido, pero en el escenario definitivo empezará en un lugar que no es el centro (porque en el centro aparecerá Teseo) y por ello tendrá dificultades para encontrar el camino. En cambio el agente que es entrenado como una versión del escenario final, aprenderá distinto que Teseo, posiblemente le cueste más tiempo aprender, pero en el escenario definitivo se moverá mejor.

Esto es el resultado del aprendizaje de Minotauro-1, figura 26:

El aprendizaje Minotauro-1 fue el más inestable, a pesar de tener un aprendizaje casi calcado de Teseo, el cambio de velocidad del agente para que fuera un poco más rápido y una mayor dimensión del agente causó un aprendizaje inconstante.

A diferencia de los demás aprendizajes, en Minotauro-1 el agente no aprende rápido cuál es la solución y en consecuencia tarda mucho tiempo en aprender el mejor camino posible. Hasta el paso 80000 no encuentra la solución y desde esa posición comienza a mejorar. El agente, aún así, sigue siendo bastante inestable, al cambiar a la fase 2 en vez de comenzar a fallar sigue consiguiendo un resultado positivo y al llegar a la fase 3, en cambio, su media de recompensas disminuye.

Después de tantos intentos en un lugar cerrado el agente comienza a conocer cada rincón correctamente por lo que cuando el objetivo sale de esa zona el agente Minotauro empieza a perderse, pero cuando empieza a ver que siguiendo las huellas llega al objetivo empieza a tener un resultado positivo.

Llegar a la fase 4 provoca un cambio muy grande en el agente Minotauro, ahora tiene que recorrer más pasillos y caminos y se empieza a perder, muchas veces empieza a ir por caminos

que no tienen huellas porque intenta encontrar rápidamente al objetivo y no se da cuenta de la relación que tienen, por esta razón se puede ver unos cambios de ganancia de recompensas tan bruscos.

El agente Minotauro ha aprendido a seguir las huellas y ahora ha llegado a la última fase, la fase en la que el agente tiene que llegar hasta Teseo. Al principio le cuesta, se puede observar cómo tiene momentos donde falla, pero nada más empezar a aprender la media de recompensas sube y consigue un resultado positivo.

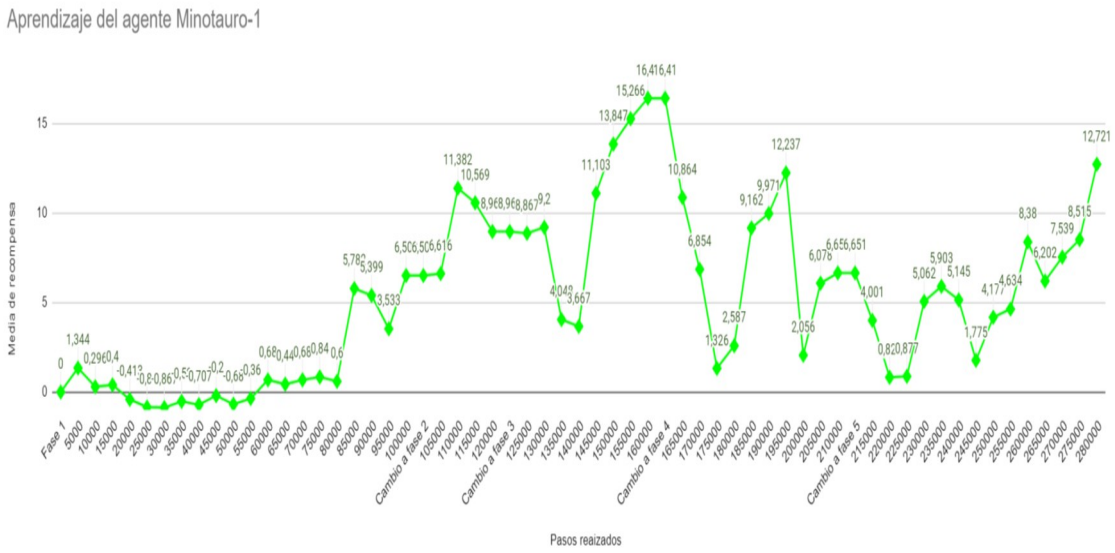

Figura 26: Aprendizaje del agente Minotauro-1. Tiempo de ejecución aproximado: 1 hora y 30 minutos.

Esto es el resultado del aprendizaje de Minotauro-2, figura 27:

El aprendizaje al principio es positivo, en seguida aprende cuál es su objetivo y su resultado mejora favorablemente. Al principio, como se discute en el capítulo 6.7, el agente Minotauro está en el centro y como siempre empieza en el mismo sitio aprende su entorno, pero al cambiar a la fase 2, el objetivo aparece aleatoriamente y ya no le otorga una recompensa ir al mismo sitio constantemente. Como se puede comprobar en el paso 30000, la media de recompensa sufre una bajada completa, es casi como si el agente empezase a aprender de nuevo, y es lo que ocurre. Hay una similitud entre los pasos 30000- 40000 de este agente con el del comienzo del agente Teseo, y es que consiguen una buena solución por coincidencia pero siguen explorando distintos caminos para encontrar posibles mejores recompensas y como no los encuentran, vuelven a realizar la ejecución que mejor resultado les ha dado.

Al cambiar a la fase 3 todo cambia. El agente Minotauro ya no está en el centro, se encuentra perdido y le es difícil conseguir una buena recompensa, pero sigue intentándolo. Ahora el threshold (capítulo 6.9) es distinto, es mucho menor debido a que al chocarse con las paredes se le recompensa negativamente. Una disciplina dura que poco a poco le hace encontrar la salida y conseguir suficientes ejecuciones positivas para llegar a la fase 5.

En la fase 5, la fase definitiva, las paredes ya no son letales, el agente Minotauro puede seguir las huellas sin tener miedo de chocarse con los muros. En un principio el propio agente no entiende este cambio y consigue una media de recompensas muy bajas, pero a medida que intenta llegar a objetivo empieza a conseguir un resultado positivo hasta ya conseguir un resultado extremadamente positivo. Se puede observar cómo el agente comprende el juego y empieza a conseguir un buen resultado.

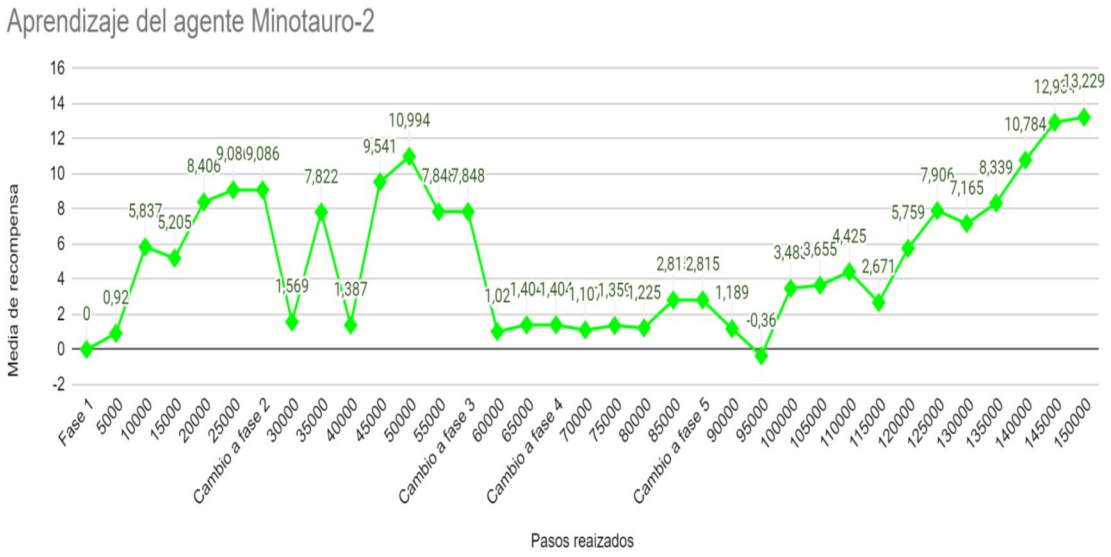

Figura 27: Aprendizaje del agente Minotauro-2. Tiempo de ejecución aproximado: 1 hora y 5 minutos.

7.3 Escenario definitivo

Después de muchas pruebas, se llegó a la conclusión de que no se podía conseguir un resultado definitivo porque no había multitud de caminos para llegar a un mismo sitio. Muchas veces el Minotauro ganaba quedándose quieto en mitad del camino y como no había otros caminos para llegar a la salida, Teseo tenía que enfrentarse a él obligatoriamente.

Debido a esto se idealizó un mapa con los caminos más decisivos eliminados, figura 28, y con estos cambios se procedió a realizar las pruebas.

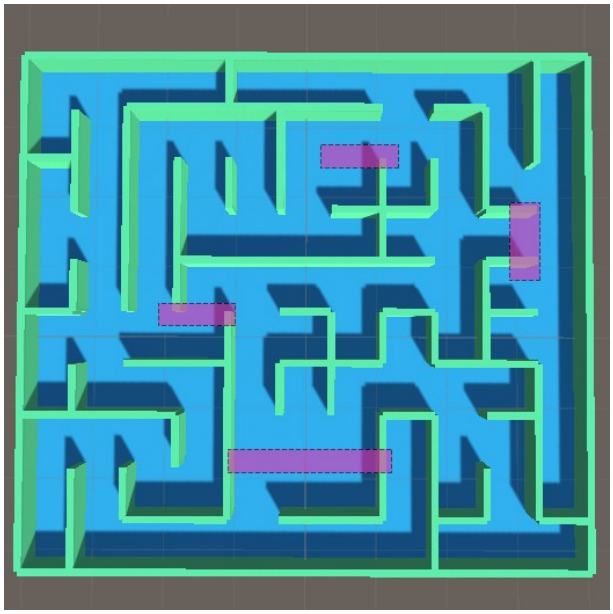

Figura 28: Cambios en el escenario definitivo.

Se realizaron 20 pruebas, 10 con el aprendizaje Minotauro-1 y otras 10 con Minotauro-2. Se recuperaron los datos obtenidos y estos son los resultados (figura 29):

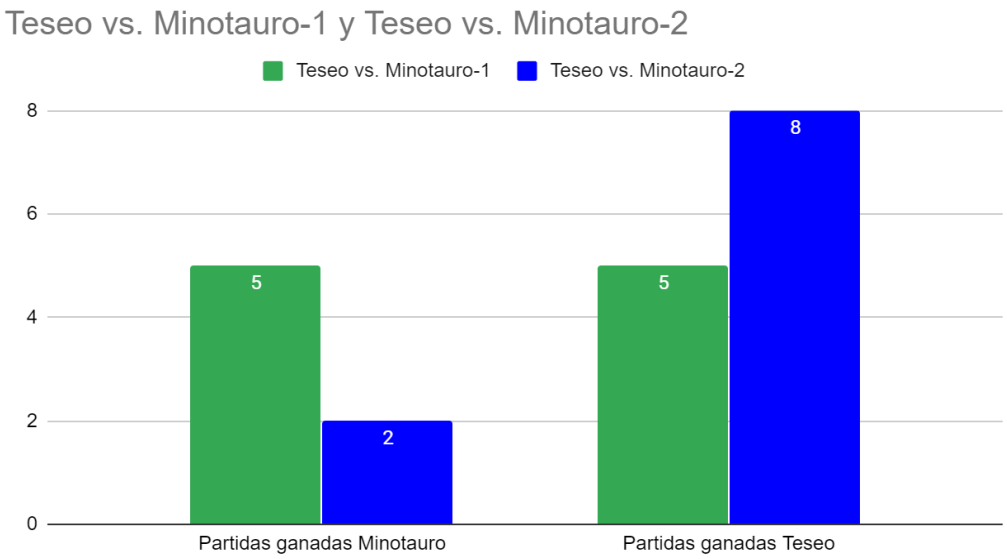

Figura 29: Resultados de ejecución definitiva.

Como se puede observar, el aprendizaje Minotauro-1 tuvo un resultado esperado. Al ser entrenado de la misma manera que Teseo fue un enfrentamiento muy igualado. En cambio con Minotauro-2 pasó algo inesperado, el agente Minotauro entrenado con este aprendizaje no era tan agresivo como el enseñado con Minotauro-1, por lo que muchas veces estuvo a punto de atrapar a Teseo pero no lo consiguió. Para observar las pruebas realizadas, dirigirse al anexo de la página 73. Con estos resultados, se puede afirmar que el mejor aprendizaje es Minotauro-1.

8

8 Conclusiones

Después del desarrollo del proyecto, se necesita hacer una perspectiva al futuro y explicar lo que le deparará a este proyecto en los siguientes años.

8.1 Objetivos logrados

Los agentes construidos funcionan correctamente, se ha aprendido lo propuesto y ahora se alberga un mayor conocimiento para el desarrollo de proyectos parecidos. El escenario cumple su objetivo y la búsqueda de camino más corto funciona de manera positiva.

La apretada gestión del proyecto no ha dado lugar a distintos tipos de agente con los que tal vez se pudiera haber comprobado un mejor resultado, pero como se ha entrado en el marco de lo establecido el trabajo se puede dar por finalizado.

8.2 Comentario sobre el trabajo realizado

El desarrollo del proyecto ha sido duro debido a que muchas veces había que esperar ejecuciones de horas enteras para ver si el agente había aprendido correctamente o no, pero ha valido la pena puesto que se empezó en este proyecto sin tener ningún conocimiento sobre esta herramienta y se ha terminado con una completa visión y entendimiento del kit.

Todavía queda mucho por probar, y más importante, mucho por aprender. La herramienta muestra su utilidad en proyectos como este tamaño y en manos de una empresa con mayores recursos podría crearse algo revolucionario.

8.3 Visión comercial del futuro

Después del desarrollo del proyecto, se necesita hacer una perspectiva al futuro y explicar lo que le deparará.

Por ahora, el proyecto es útil para mostrar y comprobar las habilidades aprendidas con esta herramienta, pero no contiene la posibilidad de entrar al mercado debido a que sirve más para mostrar las capacidades del desarrollador.

Si en un futuro existiera una oportunidad comercial de un producto de parecidas características, se podría mostrar el trabajo realizado en este proyecto.

8.4 Mejoras y visión para un futuro

Uno de los problemas más importantes para este proyecto fue la necesidad de entrenar a los dos agentes por separado. Existen posibilidades de entrenar a los dos agentes a la vez, como se comentó en el capítulo 5.2, pero es necesario haber obtenido un conocimiento mucho mayor en la herramienta.

Si se viera necesario, una mejora podría ser implementar esta opción y daría lugar a interacciones más directas entre Teseo y el Minotauro. Por el momento, se puede dar como positivo el trabajo realizado y se han abierto posibilidades a crear nuevos proyectos con los conocimientos plasmados en este documento.

Bibliografía

[1] Becerril, A. (14 de junio, 2016). *La importancia de la documentación de software*. Recuperado de https://medium.com/universo-mutante/la-importancia-de-la-documentaci%C3%B3n-de-software-c863cdf6a251. Consultado en junio del 2020.

[2] Bengio Y.; Louradour J.; Collobert R. y Weston J. (2009). *Curriculum Learning*. Recuperado de https://qmro.qmul.ac.uk/xmlui/bitstream/handle/123456789/15972/Bengio%2C%202009%20Curriculum%20Learning.pdf?sequence=1. Consultado en junio del 2020.

[3] CBINSIGHTS. (20 de septiembre, 2018). *The $120B Minga Industry Is Being Built On The Backs Of These Two Engines*. Recuperado de https://www.cbinsights.com/research/game-engines-growth-expert-intelligence/#:~:text=Unity%20Technologies%20was%20founded%20in,entire%20global%20game%20engine%20market. Consultado en junio del 2020.

[4] Daniel. (10 de enero, 2018). *Unity - A Star Pathfinding Tutorial* [Vídeo]. YouTube: https://www.youtube.com/watch?v=AKKpPmxx07w. Consultado en junio del 2020.

[5] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*(1), 269–271.

[6] Fort J.; Crespi A.; Elion C.; Kermanizadeh R.; Wani P. y Lange D. (8 de mayo, 2020). *Exploring new ways to simulate the coronavirus spread*. Recuperado de https://blogs.unity3d.com/2020/05/08/exploring-new-ways-to-simulate-the-coronavirus-spread/?_ga=2.142492978.1004374938.1590936225-16231767.1590686867. Consultado en junio del 2020.

[7] García E. D. (1983). *El mito de Teseo en la literatura*. Recuperado de https://dialnet.unirioja.es/servlet/articulo?codigo=144016. Consultado en junio del 2020.

[8] Haas, J. K. (marzo, 2014). *A History of the Unity Game Engine*. Recuperado de https://digitalcommons.wpi.edu/cgi/viewcontent.cgi?article=4206&context=iqp-all. Consultado en junio del 2020.

[9] Juliani A. (2017). *A visual depiction of how a Learning Environment might be configured within Unity ML-Agents Toolkit* [Figura]. Recuperado de https://blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents/. Consultado en junio del 2020.

[10] Keerthi S. S. y Ravindran B. (diciembre, 1994). *A tutorial survey of reinforcement learning*. Recuperado de https://link.springer.com/article/10.1007/BF02743935. Consultado en junio del 2020.

[11] Korf R. E. (2000). *Recent Progress in the Design and Analysis of Admissible Heuristic Functions*. Recuperado de https://www.aaai.org/Papers/AAAI/2000/AAAI00-212.pdf. Consultado en junio del 2020.

[12] Learncuriously (2018). *Machine Learning vs Human Learning Part 1: Types of ML and Their Human Learning Theory Equivalents* [Figura]. Recuperado de https://learncuriously.wordpress.com/2018/12/22/machine-learning-vs-human-learning-part-1/. Consultado en junio del 2020.

[13] Lester P. (2005). *A* Pathfinding for Beginners* [Figura]. Recuperado de http://csis.pace.edu/~benjamin/teaching/cs627/webfiles/Astar.pdf. Consultado en junio del 2020.

[14] Lester, P. (18 de julio, 2005). *A* Pathfinding for Beginners.* Recuperado de http://csis.pace.edu/~benjamin/teaching/cs627/webfiles/Astar.pdf. Consultado en junio del 2020.

[15] Li, M. y Tan, S. (30 de junio-2 de julio, 2008). The Market Structure of the Video Game Industry: A Platform Perspective. Trabajo presentado en el *International Conference on Service Systems and Service Management*, Melbourne, VIC, Australia. Recuperado de https://ieeexplore.ieee.org/abstract/document/4598486. Consultado en junio del 2020.

[16] Mattar M.; Shih J.; Berges V. P.; Elion C. y Goy C. (12 de mayo, 2020). *Announcing ML-Agents Unity Package v1.0!*. Recuperado de https://blogs.unity3d.com/2020/05/12/announcing-ml-agents-unity-package-v1-0/?_ga=2.207949074.1004374938.1590936225-16231767.1590686867. Consultado en junio del 2020.

[17] McCarthy, J. (2007). *What Is Artificial Intelligence?*. Recuperado de http://jmc.stanford.edu/artificial-intelligence/what-is-ai/index.html. Consultado en junio del 2020.

[18] McConnell J. V. (1990). *Negative Reinforcement and Positive Punishment*. Recuperado de https://journals.sagepub.com/doi/abs/10.1207/s15328023top1704_10. Consultado en junio del 2020.

[19] Megajuice (2017). *Archivo:Reinforcement learning diagram.svg* [Figura]. Recuperado de https://es.wikipedia.org/wiki/Archivo:Reinforcement_learning_diagram.svg. Consultado en junio del 2020.

[20] Nájera Perez, Á. (s/f). *Los factores del éxito de un proyecto*. Recuperado de https://wolfproject.es/los-factores-del-exito-de-un-proyecto/. Consultado en junio del 2020.

[21] Otexts. (s/f). *A neural network with four inputs and one hidden layer with three hidden neurons* [Figura]. Recuperado de https://otexts.com/fpp2/nnetar.html. Consultado en junio del 2020.

[22] Peckham, E. (17 de octubre, 2019). *How Unity built the world's most popular game engine*. Recuperado de https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/. Consultado en junio del 2020.

[23] Pires F. A.; Santos W. M.; Andrade K. de O.; Caurin G. A. P. y Siqueira A. A. G. (14 de mayo-16 de mayo, 2014). *Robotic platform for telerehabilitation studies based on unity game engine*. Trabajo presentado en el *2014 IEEE 3nd International Conference on Serious Games*

and Applications for Health (SeGAH), Rio de Janeiro, Brazil. Recuperado de https://ieeexplore.ieee.org/document/7067094/authors#authors. Consultado en junio del 2020.

[24] Redacción APD (4 de marzo, 2019). *¿Qué es Machine Learning y cómo funciona?*. Recuperado de https://www.apd.es/que-es-machine-learning/#:~:text=Machine%20Learning%20o%20Aprendizaje%20autom%C3%A1tico,de%20datos%20en%20su%20sistema. Consultado en junio del 2020.

[25] Santarcangeli P. (1997). *El libro de los laberintos: historia de un mito y de un símbolo*. Ediciones Siruela: Madrid. Recuperado de https://books.google.es/. Consultado en junio del 2020.

[26] Shummon Maass, L. E. (1 de julio de 2019). *Artificial Intelligence in Video Games*. Recuperado de https://towardsdatascience.com/artificial-intelligence-in-video-games-3e2566d59c22. Consultado en junio del 2020.

[27] Simonini T. (3 de septiembre, 2018). *Proximal Policy Optimization (PPO) with Sonic the Hedgehog 2 and 3*. Recuperado de https://towardsdatascience.com/proximal-policy-optimization-ppo-with-sonic-the-hedgehog-2-and-3-c9c21dbed5e. Consultado en junio del 2020.

[28] Sung, L. K.; Hae, J. S.; Jeong, H. K.; Jung, M. J.; Laine, T. H. y Westlin J. (6 de marzo-8 de marzo, 2014). *Using Unity 3D to Facilitate Mobile Augmented Reality Game Development*. Trabajo presentado en el *2014 IEEE World Forum on Internet of Things (WF-IoT)*, Seúl, Corea del sur. Recuperado de https://www.researchgate.net/publication/271462295_Using_Unity_3D_to_facilitate_mobile_augmented_reality_game_development. Consultado en junio del 2020.

[29] Valentine R. (17 de abril, 2019). *"Picking the right door" with the future of Unity*. Recuperado de https://www.gamesindustry.biz/articles/2019-04-17-picking-the-right-door-with-the-future-of-unity. Consultado en junio del 2020.

Anexos

Anexo A: Modelos3D usados

Estos son los modelos que se han conseguido para este proyecto:

- El modelo de Minotauro, obtenido desde la página TurboSquid en este enlace. Véase la figura 30.

Figura 30: Modelo 3D del Minotauro.

- El modelo de Teseo, obtenido desde la página TurboSquid en este enlace. Se puede observar en la figura 31.

Figura 31: Modelo 3D de Teseo.

- El modelo para las huellas de Teseo, obtenido desde la página TurboSquid en este enlace. La figura 32.

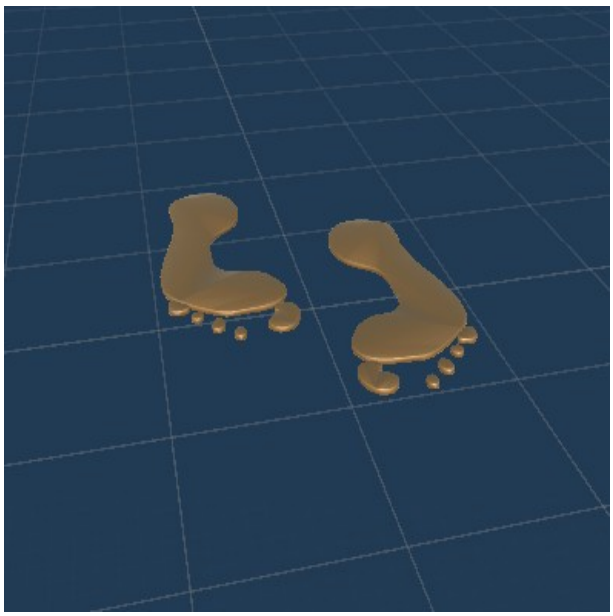

Figura 32: Modelo 3D de la huella.

- Para el modelo del hilo se ha construido un cilindro rojo para representar los trozos de hilo. Se puede observar en la figura 33.

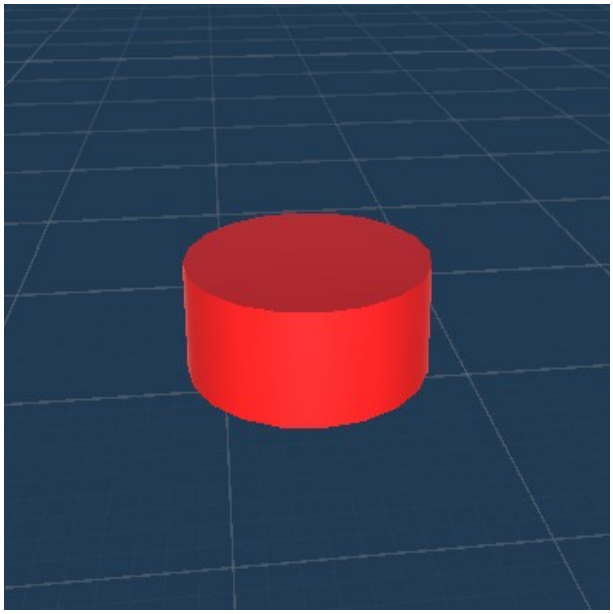

Figura 33: Modelo 3D del hilo.

- Para la salida se ha construido un cubo sencillo que represente la salida del laberinto. La figura 34.

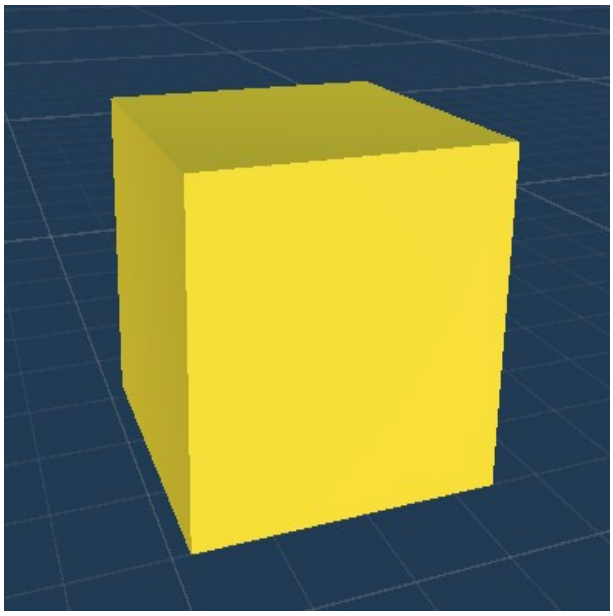

Figura 34: Modelo 3D de la representación de la salida.

Para cada modelo hay un prefab definido. Para el proyecto entero se ha construido un prefab del entorno donde están contenidos todos los objetos necesarios para ejecutar el proyecto. Véase la figura 35.

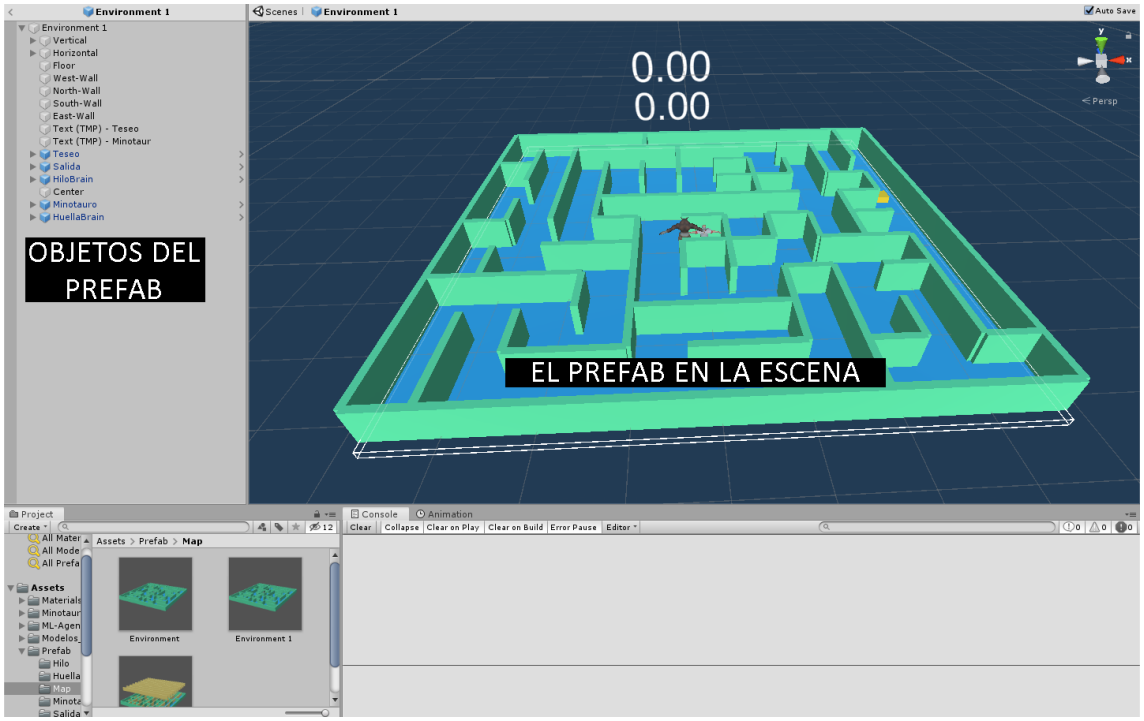

Figura 35: Prefab del escenario en la escena.

Anexo B: Construcción del laberinto

El desarrollo del laberinto fue el siguiente:

1. En primera instancia, se creó un plano 3D de 78x78 unidades[30]. Figura 36.

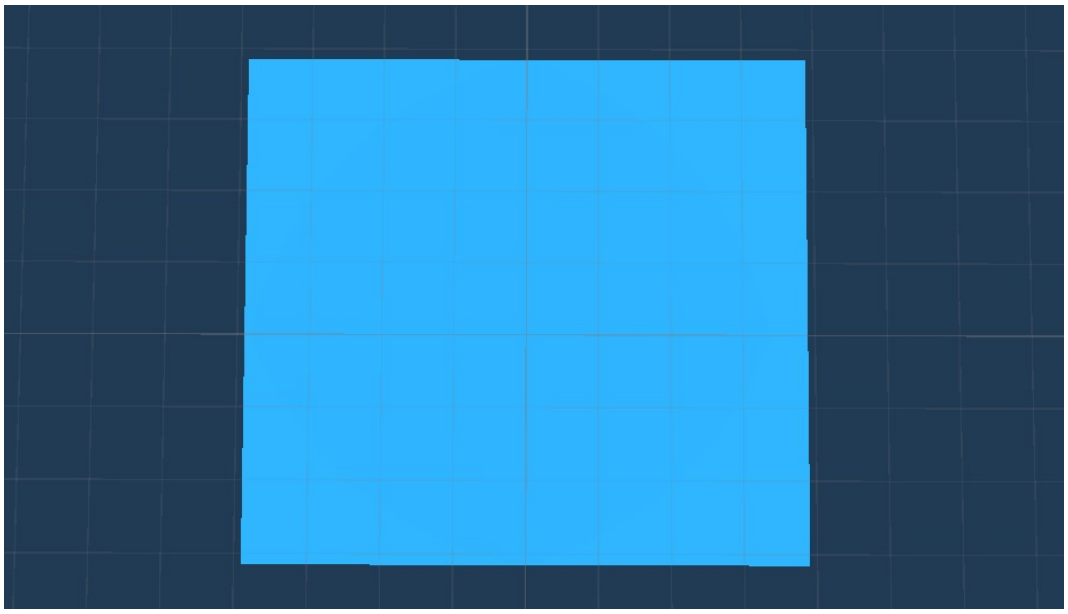

Figura 36: Construcción del escenario: Parte 1.

2. Después se crearon las paredes exteriores con 1 unidad de anchura y 5 unidades de altura. Figura 37.

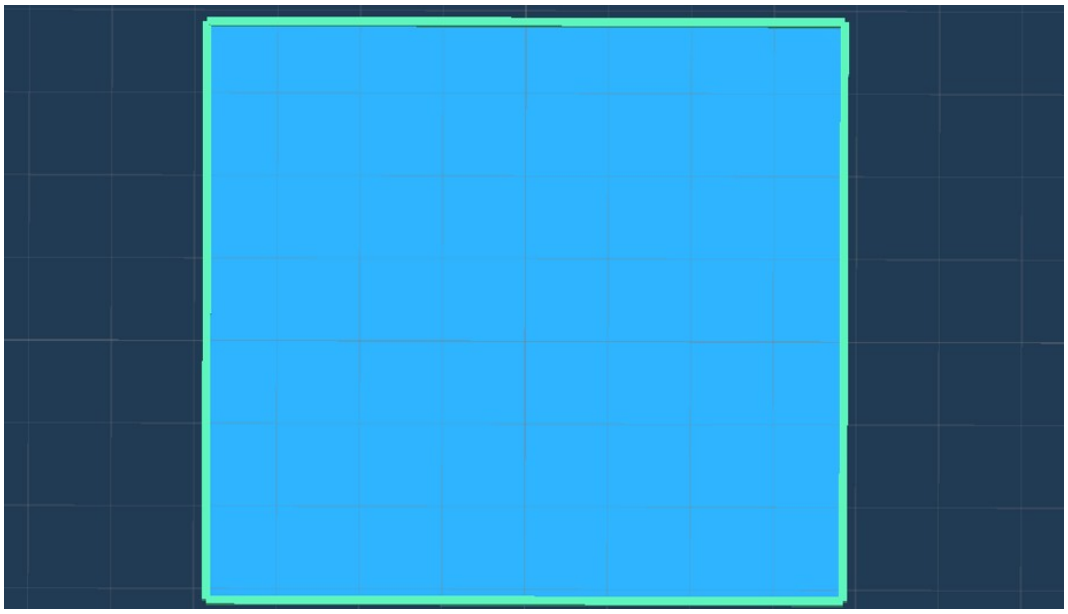

Figura 37: Construcción del escenario: Parte 2.

30 Las unidades en Unity tienen una relación de: 1 unidad = 1 metro.

3. Las paredes del escenario contenían un camino proporcional y bien estructurado, por lo que fue fácil dividir el escenario en cubículos. Figura 38.

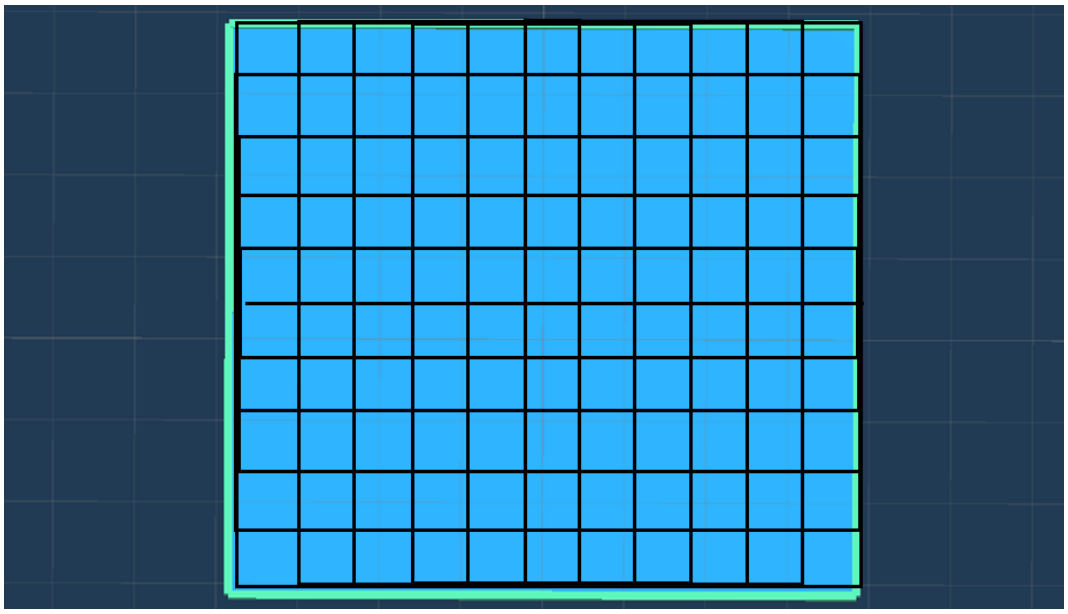

Figura 38: Construcción del escenario: Parte 3.

4. Una vez teniendo los cubículos, se construyeron las paredes verticales. Figura 39.

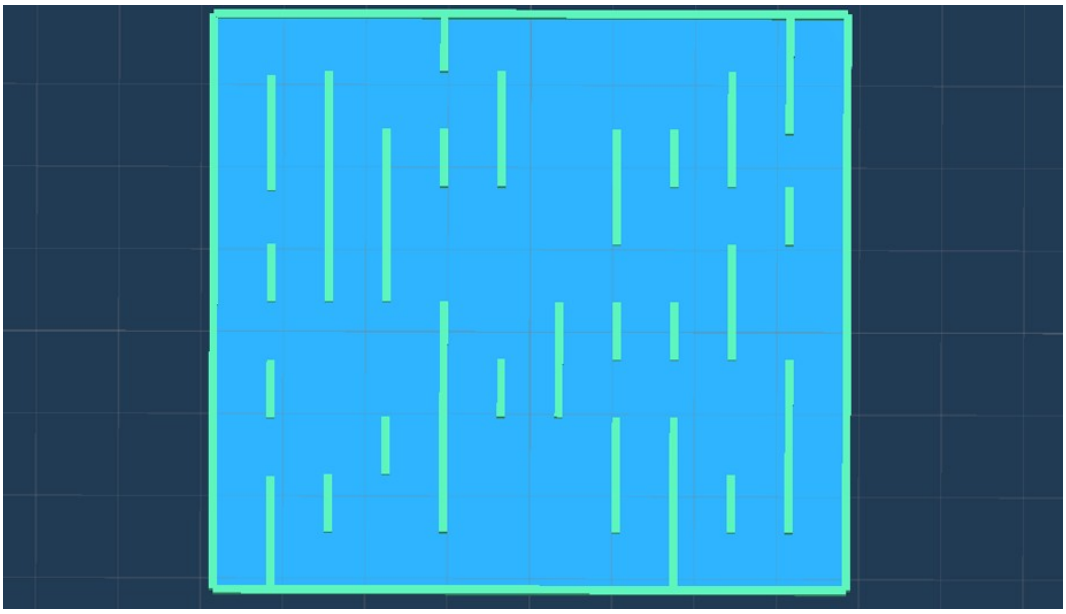

Figura 39: Construcción del escenario: Parte 4.

5. Y para finalizar, las horizontales. Figura 40.

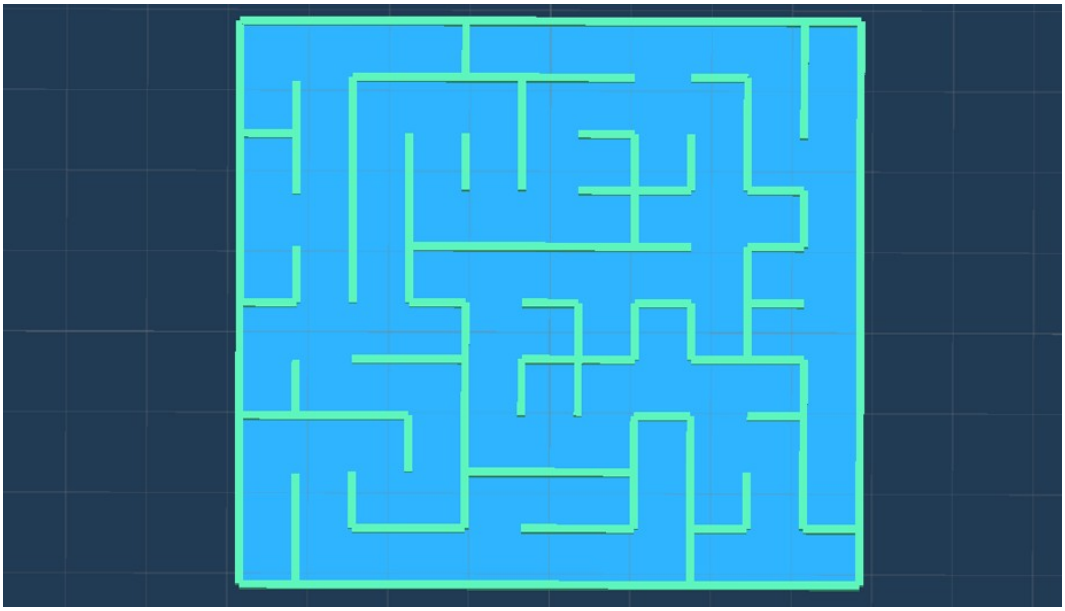

Figura 40: Construcción del escenario: Parte 5.

Y así, se consiguió construir un escenario que fuese un laberinto útil, sencillo y óptimo para el desarrollo de este proyecto.

Anexo C: Archivos de curricula del proyecto

```
// Modificación del entorno para el agente Teseo
TeseoLearning:
  measure: reward
  thresholds: [4.1, 4.4, 5.1, 5.0]
  min_lesson_length: 80
  signal_smoothing: true
  parameters:
    agente_actual: [0.0, 0.0, 0.0, 0.0, 0.0]
    fase: [1.0, 2.0, 3.0, 4.0, 5.0]
    aleat_spawn: [0.0, 1.0, 0.0, 1.0, 1.0]
    radius_spawn: [5.0, 8.0, 10.0, 20.0, 30.0]
    min_radius_spawn: [2.0, 2.0, 4.0, 12.0, 24.0]
    reward_hilo: [4.0, 8.0, 10.0, 10.0, 10.0]
    probabilidad_rastro: [3.0, 3.0, 3.0, 4.0, 4.0]
    letal_pared: [0.0, 1.0, 1.0, 0.0, 0.0]

// Modificación del entorno para el agente Minotauro-1
MinotaurLearning:
  measure: reward
  thresholds: [4.1, 4.4, 5.1, 5.0]
  min_lesson_length: 80
  signal_smoothing: true
  parameters:
    agente_actual: [1.0, 1.0, 1.0, 1.0, 1.0]
    fase: [1.0, 2.0, 3.0, 4.0, 5.0]
    aleat_spawn: [0.0, 1.0, 0.0, 1.0, 1.0]
    radius_spawn: [5.0, 8.0, 10.0, 20.0, 30.0]
    min_radius_spawn: [2.0, 2.0, 4.0, 12.0, 24.0]
    reward_huella: [4.0, 8.0, 10.0, 10.0, 10.0]
    probabilidad_rastro: [3.0, 3.0, 3.0, 4.0, 4.0]

// Modificación del entorno para el agente Minotauro-2
MinotaurLearning:
```

```
measure: reward
thresholds: [6.5, 6.5, 8.0, 7.5]
min_lesson_length: 50
signal_smoothing: true
parameters:
  agente_actual: [1.0, 1.0, 1.0, 1.0, 1.0]
  fase: [1.0, 2.0, 3.0, 4.0, 5.0]
  aleat_spawn: [0.0, 1.0, 0.0, 1.0, 1.0]
  radius_spawn: [10.0, 20.0, 12.0, 15.0, 35.0]
  min_radius_spawn: [5.0, 5.0, 5.0, 5.0, 25.0]
  reward_huella: [6.0, 8.0, 10.0, 10.0, 10.0]
  probabilidad_rastro: [3.0, 3.0, 4.0, 5.0, 5.0]
  letal_pared: [0.0, 0.0, 1.0, 1.0, 0.0]
```

Anexo D: Archivos y recursos del proyecto

Para acceder a la carpeta que contiene todo el proyecto, acceder a este enlace.

Importante leer el Readme.txt si se quiere instalar y probar el proyecto.

Los vídeos que muestran las pruebas realizadas se pueden encontrar en este enlace.