Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Deep Learning en videojuegos

Deep Learning in video games

Daniel Suárez Labena

La Laguna, *7* de *septiembre* de 2020

D. **Jose Demetrio Piñeiro Vera**, con N.I.F. 43.774.048-B profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43.826.207-Y profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Deep Learning en videojuegos”

ha sido realizada bajo su dirección por D. **Daniel Suárez Labena**,

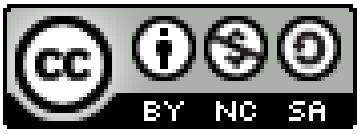
con N.I.F. 54.115.091-R.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 07 de septiembre de 2020.

Agradecimientos

Este TFG se lo dedico a mi familia, que me ha apoyado todo este tiempo y me ha insistido para que siga adelante. También a mis amigos que me han proporcionado un gran apoyo moral en todo momento y finalmente a mis tutores por aconsejarme cuando más lo necesitaba y guiarme por el buen camino.

Licencia



Resumen

Hoy en día una gran parte de la población mundial son usuarios de videojuegos, ya sea jugando o consumiendo contenido multimedia de los mismos (streaming, videos , etc). Muchos de estos juegos requieren de inteligencias artificiales que interactúen con el jugador o entre ellas. Una forma de conseguir esta labor es haciendo uso de un conjunto de algoritmos de aprendizaje por refuerzo que utilizan métodos de "deep learning".

El objetivo de este TFG consiste en aplicar algoritmos de aprendizaje por refuerzo a un juego propio de tanques desarrollado en Unity. Se pretende obtener una inteligencia artificial capaz de jugar de manera adecuada, con un comportamiento realista.

Palabras clave: Deep Learning, Unity, MlAgents

Abstract

Nowadays a large part of the world population are video game users, either playing or consuming their multimedia content (streaming, videos, etc.). Many of these games require artificial intelligences to interact with the player or with each other. One way of doing this is by making use of a set of reinforcement learning algorithms that use "deep learning" methods.

The objective of this TFG is to apply reinforcement learning algorithms to a game of tanks developed in Unity. It is intended to obtain an artificial intelligence capable of playing properly, with realistic behavior.

Keywords: Deep learning, Unity, MlAgents

Índice general

Índice de figuras

Índice de tablas

Capítulo 1 Introducción

1.1 Motivación

La principal motivación de este proyecto surge, como es de esperar, por un interés por el campo de los videojuegos así como su desarrollo. Específicamente en la inteligencia artificial que se encuentra en los mismos.

Puesto que hoy en día los videojuegos forman parte de la vida de la población mundial se han convertido en una industria muy importante que mueve cantidades de dinero elevadas. Además debido a que los videojuegos están muy ligados a la inteligencia artificial, es prácticamente imprescindible el tener conocimiento de esta para poder desarrollar videojuegos de calidad y adentrarse en esta industria.

En base a estas razones he decidido abordar este proyecto, para así poder obtener unas nociones básicas sobre la inteligencia artificial que rodea a los videojuegos así como las técnicas de *deep learning* que son comúnmente empleadas en los mismos.

1.2 Objetivos

El principal objetivo de este trabajo es obtener una inteligencia artificial que pueda jugar de manera realista a un juego propio de tanques por equipos y en 3D realizado en Unity.

Como es de esperar antes de realizar esto es necesario crear el juego en Unity, que en este caso se va a tratar un juego sencillo en el que cada tanque puede disparar con una fuerza y un ángulo determinados así como moverse hacia delante y hacia atrás. El objetivo de estos tanques es destruir a los tanques enemigos mediante disparos hasta que solo quede un equipo con vida.

Finalmente una vez se hayan realizado todos estos pasos será necesario analizar de manera adecuada los resultados obtenidos. Para determinar si los resultados han sido adecuados se utilizará el sistema de puntuación conocido como Elo, se profundizará en estos aspectos más adelante.

Capítulo 2 Historia y estado actual

2.1 Aprendizaje automático

El *machine learning* o aprendizaje automático es una disciplina científica del ámbito de

la inteligencia artificial cuyo principal objetivo consiste en desarrollar técnicas que permitan a las máquinas aprender de manera autónoma. Para lograr esta labor el algoritmo se encarga de construir un modelo matemático usando como base unos datos de ejemplo. Por lo tanto, se puede definir como un proceso de inducción del conocimiento, es decir, un método que permite obtener por generalización un enunciado general a partir de enunciados que describen casos particulares.

2.1.1 Historia

El comienzo de la historia del *machine learning* se remonta al siglo XVIII en el que Thomas Bayes postuló el conocido como teorema de Bayes, que fue uno de los hitos matemáticos que sentó las bases de esta tecnología.

Varios años después, en la década de 1950 el matemático Alan Turing publica el artículo "Computing machinery and Intelligence" [1] en el que plantea la novedosa pregunta de si es posible que una máquina pueda pensar por sí misma, con la que surgió la idea de la creación de computadoras con inteligencia artificial.

Un poco más adelante, durante las décadas de 1950 y 1960, varios científicos investigaron la creación de redes neuronales artificiales, un modelo computacional inspirado en el comportamiento observado en su homólogo biológico. Uno de los experimentos realizados en este campo fue el diseñado por Marvin Minsky [2] y Dean Edmonds, que consiguieron crear el primer programa informático capaz de aprender a resolver un laberinto. Este fue un hecho revolucionario en la época en el ámbito de la inteligencia artificial, puesto que por primera vez se había conseguido crear una máquina capaz de resolver una tarea por sí misma.

Aunque el experimento tuvo un éxito sin precedentes dejó al descubierto algunos de los principales problemas de esa época, como son la falta de potencia de cómputo o de gestión de datos masivos de ejemplo. Este hecho llevó a un estancamiento en el desarrollo de inteligencias artificiales derivando en una serie de décadas conocidas como el "invierno de la inteligencia artificial" en las que los avances en este campo fueron casi nulos.

A finales del siglo XX gracias a la llegada de internet, las cantidades masivas de datos y una notable mejoría en cuanto a la potencia de las máquinas, surge el "renacer de la inteligencia artificial". Durante este periodo se consiguió un hito a nivel mundial para la inteligencia artificial, cuando el sistema de IBM Deep Blue [3], fue capaz de derrotar al campeón de ajedrez mundial en aquellos momentos, Garry Kasparov.

Poco más adelante y adentrándonos en el nuevo milenio, se acuña el término *deep learning* por Geoffrey Hinton, dando nombre así a una nueva subcategoría dentro del aprendizaje automático cuya principal característica es el uso de arquitecturas de redes neuronales profundas que son capaces de aprender mucho mejor modelos más planos. Un ejemplo del uso de esta tecnología es el desarrollado por un equipo de "DeepMind Technologies" en el año 2013 en el que consiguieron una inteligencia artificial capaz de jugar a varios juegos de la consola Atari 2600 tomando únicamente como entradas los píxeles de la pantalla. y utilizando una función de recompensa para guiar sus acciones.

Durante los últimos años esta tecnología no ha hecho más que avanzar de manera exponencial llegando al estado actual, en que nos encontramos algún tipo de inteligencia artificial en muchos de los dispositivos que utilizamos diariamente.

2.1.2 Estado actual

En la actualidad, estamos en un punto en el que con frecuencia se crean algoritmos de aprendizaje automático. Pese a las diferencias que pueden tener entre sí, se agrupan en una taxonomía dependiendo de la salida de los mismos. Algunos de los principales grupos de algoritmos de aprendizaje automático que podemos encontrar actualmente son: aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semi supervisado y aprendizaje por refuerzo.

El aprendizaje supervisado es el más sencillo de todos. Básicamente construye un modelo matemático a partir de un conjunto de datos que contiene tanto las entradas como la salida esperada. Este conjunto de datos es el que se conoce como datos de entrenamiento. Esta estrategia es comúnmente utilizada en algoritmos de clasificación o regresión. Un ejemplo claro sería el de detectar el spam en los gestores de e-mail.

A diferencia de el supervisado, los algoritmos de aprendizaje no supervisado aprenden de unos datos de prueba que no han sido etiquetados, clasificados y categorizados. Es decir, que no cuentan con una salida esperada para las entradas de los datos. Por lo tanto el objetivo que siguen los mismo es modelizar la estructura o distribución de los datos para aprender más sobre ellos, para ello, identifican elementos comunes en los datos y reaccionan en función de la presencia o ausencia de dichos elementos comunes en cada nuevo elemento de los datos. Comúnmente se utilizan para la agrupación de elementos en clústeres basándose en sus similitudes.

Progresamos hacia el aprendizaje semi supervisado, el cual se trata de un término medio entre el supervisado y el no supervisado. Es decir que algunos de los datos de ejemplo utilizados cuentan con una salida mientras que otros no.

Finalmente nos encontramos con los denominados algoritmos de aprendizaje por refuerzo, que serán los utilizados en este proyecto. Estos algoritmos observan el entorno que les rodea y, tal y como su nombre indica, obtienen una recompensa o refuerzo a partir de las acciones que realizan. Su objetivo es extraer qué acciones deben ser realizadas para poder maximizar esta recompensa en cada momento.

2.2 Inteligencia artificial en los videojuegos

Se considera un videojuego a cualquier juego electrónico que sea visualizado a través de una pantalla. Durante los últimos años estos juegos han sufrido un aumento de popularidad meteórico. Esto en gran parte se ha debido al hecho de la generalización del uso de internet, que ha permitido llegar a un público más extenso debido al uso masivo de redes sociales, contenido multimedia o revistas digitales en las que se comparte este contenido.

La inteligencia artificial en los videojuegos tiene una serie de características que la diferencian de la aplicación de esta tecnología en otros campos. La más evidente es que está orientada al entretenimiento, por lo que da lugar a interesantes usos de la misma no vistos en otros campos.

2.2.1 Historia

Desde el inicio de la historia de los videojuegos la inteligencia artificial ha estado ampliamente ligada a los mismos. De hecho se podría decir que el inicio de la inteligencia artificial en esta industria se remonta a sus inicios, puesto que, el considerado por muchos el primer juego de la historia, OXO (1952) [4], ya contaba con una IA capaz de enfrentarse al jugador.

Con la aparición de nuevos juegos más complejos, también mejoraron las inteligencias artificiales que se encontraban en los mismos. Por ejemplo es el caso de Pong que era capaz de calcular la trayectoria de una bola para así poder contrarrestar el tiro del enemigo o el caso del famoso Pac Man que contó con uno de los primeros sistemas de búsqueda de rutas para los enemigos.

Obviamente, estas son inteligencias artificiales muy básicas, lo que nos hace preguntarnos hasta qué punto las máquinas toman decisiones, en lugar de que los programadores tomen decisiones a través de los scripts. En otras palabras, la máquina no sabe por qué se toma una decisión u otra, y aún más importante, no aprende de sus errores o del comportamiento de los jugadores humanos.

En la década de 1980, surgieron otros títulos que lograron llevar la inteligencia artificial un paso más allá en los videojuegos. Son juegos de aventura y estrategia, como Rogue o Elite, que pueden generar mapas al azar basadas en algoritmos. De esta manera se empezaron a crear los conocidos como escenarios generados proceduralmente, que hasta día de hoy son ampliamente utilizados.

Durante los años 90, se empiezan a popularizar los juegos de estrategia en tiempo real o RTS, como Starcraft o Age of Empires. Estos incluyen muchas técnicas de IA y en tiempo real (planificación de rutas, máquinas de estado, gestión de recursos, etc...). Los enemigos ya podían diseñar tácticas y estrategias basadas en los movimientos del jugador o usar sus recursos de manera eficiente, de esta manera se obtenía un comportamiento más cercano a una verdadera inteligencia.

Otro género que también ha aportado su granito de arena en la mejora de las IA es el de la conducción. Por lo general, estos juegos siempre han utilizado a adversarios no humanos que compiten contra el jugador y es de esperar que estos adversarios tengan un comportamiento realista (choques inesperados, algunas salidas de la pista, etc). Esto ha llevado a un desarrollo bastante notable de estos adversarios con los años, un ejemplo destacable es el del videojuego Forza Motorsport en el que la IA era capaz de crear un estilo de conducción del piloto para así interactuar con él de la manera más adecuada.

Durante los últimos años ha avanzado considerablemente el uso de IA en videojuegos y es bastante inusual encontrar un juego actual que no cuente con algún tipo de IA.

2.2.2 Estado actual

Como ya se mencionó en el apartado anterior, es común utilizar la inteligencia artificial para generar comportamientos sensibles, adaptativos o inteligentes principalmente en personajes no jugadores (NPC) similares a la inteligencia humana. Aunque estas prácticas siguen estando presentes en los juegos actuales, también se han encontrado nuevos usos que han permitido realizar hazañas increíbles que antes parecían casi imposibles.

Un ejemplo claro de estas nuevas tecnologías es el caso de la plataforma Stadia [5] de

Google. Se trata de un servicio de retransmisión de videojuegos en la nube que permite al usuario jugar a un juego sin que este se esté ejecutando en su máquina, puesto que esto último se realiza en los centros de datos de Google. El principal problema de estos servicios, como se puede esperar, es la latencia entre las entradas que origina el jugador y las acciones que ocurren en el juego. La solución que han aplicado los desarrolladores a este problema es usar técnicas de *deep learning* para aprender cómo los jugadores juegan a sus juegos y de esta manera intentar predecir las acciones de los jugadores con el fin de reducir el tiempo de respuesta.

La IA también se está usando para mejorar juegos ya existentes. Es común que juegos antiguos se remastericen para nuevas generaciones de máquinas con unos mejores gráficos y este siempre ha sido un proceso que se ha realizado de manera "manual". Sin embargo, hoy en día se están empezando a utilizar herramientas como GAN (generative adversarial network) que pueden reducir un trabajo de semanas o meses a unas cuantas horas. Nvidia ha adoptado esta tecnología desarrollada por Ian Goodfellow y ha creado su propia inteligencia artificial conocida como StyleGan [6]. Se trata de un proyecto de código abierto gracias a el cual algunos desarrolladores independientes han logrado realizar remasterizaciones no oficiales de juegos como Doom o Final Fantasy VII.

Finalmente uno de los usos que empieza a ganar más relevancia con el tiempo y que seguramente de mucho de qué hablar en el futuro es el uso de esta tecnología en el propio desarrollo de juegos. Un ejemplo clave es la inteligencia artificial "Angelina" [7] desarrollada por Michael Cook en 2011. Se trata de una IA capaz de crear sus propios juegos desde cero de manera realmente sencilla. Tan solo necesita como entradas cierta información como fotografías o texto para crear un juego completamente nuevo, con escenarios y niveles originales. Grandes compañías también han podido apreciar el potencial de esta tecnología y están investigando posibles implementaciones para sus juegos triple A, puesto que puede abaratar en gran medida el presupuesto necesario para el desarrollo de un videojuego entre otras cosas.

Capítulo 3 Aprendizaje en Unity

3.1 Unity

Unity [8] es un motor de videojuegos creado por Unity Technologies y se trata de uno de los motores más utilizados a nivel mundial. Gran parte de su éxito se debe a la facilidad y versatilidad de uso con la que cuenta además de que cuenta con licencias de uso y publicación gratuitas para desarrolladores freelance.

Una de las principales características de Unity es que facilita el trabajo a muchos de sus desarrolladores es la cantidad enorme de plataformas a las que permite ser exportado. Por ejemplo, un juego que inicialmente hemos creado para la plataforma windows puede ser fácilmente exportado a linux, mac o incluso móviles y consolas.

Además cuenta con una gran tienda de los denominados “assets” o recursos que pueden ser publicados por la comunidad. Una gran cantidad de estos son gratuitos y pueden ser incorporados en un proyecto de manera muy sencilla.

3.2 ML-Agents

El “toolkit” de ML-Agents [9] es un plugin de código abierto para Unity que permite que los juegos y simulaciones sirvan de entorno de aprendizaje para el entrenamiento de agentes inteligentes.

3.2.1 TensorFlow

TensorFlow [10] se trata de una librería de aprendizaje automático de código abierto desarrollada por Google. Las implementaciones de los algoritmos que utiliza ML-Agents están basadas en esta librería. Esto significa que los modelos producidos por el kit de herramientas ML-Agents están en un formato que solo entiende TensorFlow (.nn). En un principio el uso de esta librería en el plugin queda completamente invisible al usuario final, a no ser que este quiera implementar un nuevo algoritmo para este plugin.

Una de las ventajas con las que cuenta TensorFlow es el componente conocido como TensorBoard, que permite visualizar de manera sencilla gráficas con los resultados de los entrenamientos, esto permite una mejor interpretación y comparación de los mismos. Estas gráficas serán utilizadas en el capítulo de “Resultados y evaluación”.

3.2.2 Entornos de aprendizaje

En cualquier entorno de ML-Agents se encuentran tres tipos básicos de objetos, estos

son los agentes, el cerebro y la academia [11].

Se entiende como agente a cada unidad capaz de percibir su entorno, procesar esas percepciones y actuar en consecuencia. Cada uno de ellos actúa de manera independiente al resto, es decir, cuenta con sus propias observaciones, acciones y recompensas. Cada acción que realiza el agente está siempre decidida por el cerebro al que está vinculado.

El cerebro define los estados y las acciones que se han de tomar por lo tanto, como se mencionó con anterioridad, es el encargado de tomar las decisiones de las acciones a realizar para la lista de agentes que estén vinculados al mismo.

El último objeto esencial de cada entorno es la academia. Este se trata de el objeto más importante de todos ya que en cierta medida es una representación del mismo entorno, por lo que es único en cada entorno. Su principal función es la comunicación entre Unity y la API de Python de la librería *machine learning* a utilizar (por defecto Tensorflow) , esta comunicación se realiza mediante un módulo conocido como comunicador externo.

En la siguiente figura se pueden apreciar de manera más visual todo lo comentado con anterioridad.

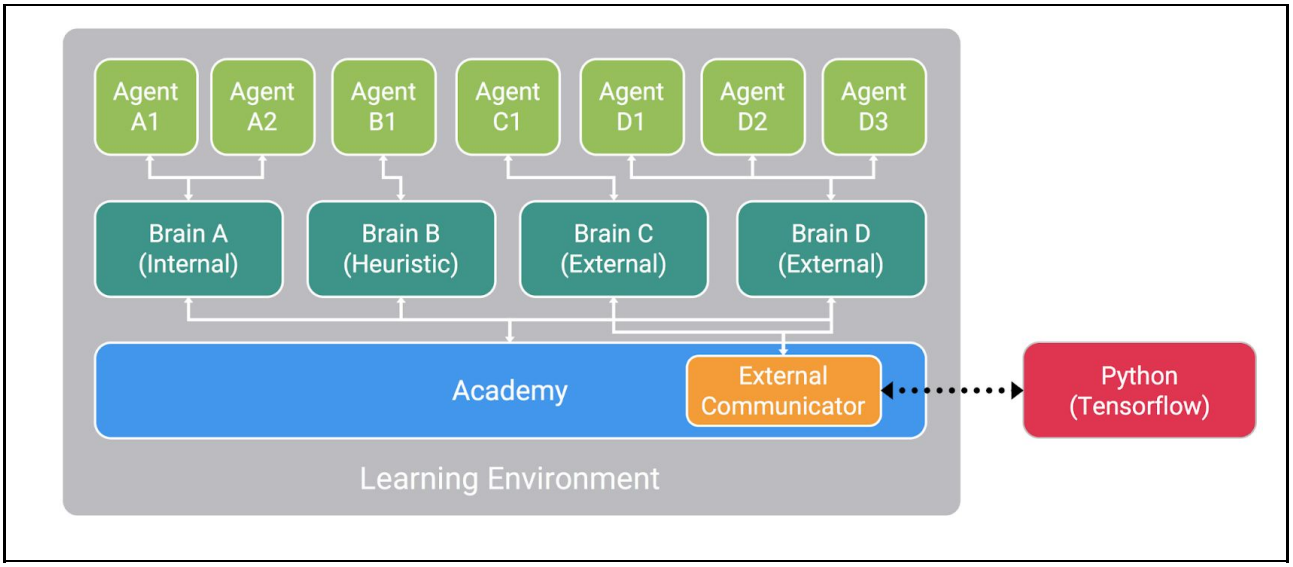


Figura 3.1: Entorno de aprendizaje ML-Agents

3.2.3 Algoritmos de aprendizaje

ML-Agents cuenta con dos algoritmos diferentes para poder realizar aprendizaje por refuerzo, estos son el Proximal Policy Optimization (PPO) [12] y el Soft Actor-Critic (SAC) [13].

El PPO utiliza una red neuronal para aproximar la mejor acción a realizar en base a las observaciones del agente en cada estado, como es de esperar, para poder realizar esta labor es necesario utilizar señales de recompensa que el agente ha de maximizar. De esta manera permiten identificar qué acción otorga mejores resultados en cada estado. A

este mapeo de observaciones y acciones se le conoce como política.

A diferencia del PPO el SAC es un algoritmo “off policy”, lo que quiere decir que aprende de cualquier experiencia obtenida en el pasado. Estas se almacenan en un buffer y se extraen de manera aleatoria durante el entrenamiento. Esto hace que el SAC sea más eficiente en cuanto a los datos de entrenamiento que PPO, generalmente requiriendo entre 5 y 10 veces menos datos de entrenamiento para conseguir un resultado similar. Por lo tanto esto hace que SAC sea una buena opción en entornos en los que sea complicado conseguir datos de entrenamiento.

Ambos algoritmos están implementados en TensorFlow y se comunican con Unity como se mencionó en el apartados anteriores.

Capítulo 4 Desarrollo del videojuego

En este apartado se hablará de cómo ha sido el desarrollo del videojuego que como ya se había comentado brevemente en el capítulo de “Introducción”, se trata de la creación de un juego propio de tanques.

4.1 Descripción

Se trata de un juego de tanques por equipos en 3D y tercera persona, en el que el objetivo es derrotar al equipo enemigo destruyendo sus tanques mediante disparos. Cuenta con unos gráficos y texturas bastante simples, puesto que la finalidad del mismo no es que sea agradable a la vista sino que sea un buen entorno para el desarrollo de la inteligencia artificial.

4.2 Tanques

Los tanques son *Gameobjects* bastante sencillos, se basan en un cubo para representar el cuerpo del tanque y un cilindro para representar el cañón, estos cilindros cuentan con un objeto vacío llamado spawnPoint que se utiliza como punto inicial para el lanzamiento de los proyectiles que dispara dicho tanque, por lo tanto se sitúa en el extremo del cañón. Cada uno además cuenta con un texto que se sitúa siempre encima del mismo que representa su salud restante, en caso de que este haya sido destruido simplemente desaparecerá de la escena. Los objetos mencionados anteriormente así como su jerarquía se pueden visualizar en el siguiente esquema:

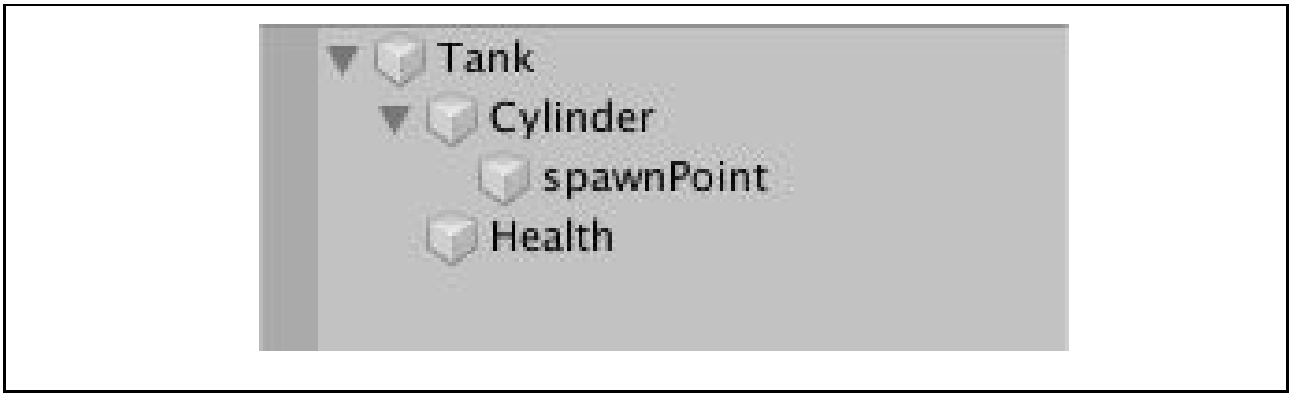


Figura 4.1: Esquema del *GameObject* de los tanques

Los tanques son capaces de moverse hacia adelante y hacia atrás, rotar sobre el eje “Y” y disparar siempre hacia su frente. Solo pueden disparar un proyectil cada vez, dicho

de otro modo, es necesario que el primer proyectil impacte con algún objeto para que el tanque sea capaz de disparar de nuevo. Tienen una fuerza de disparo además de un ángulo que determinan la trayectoria del proyectil disparado describiendo una parábola. Cada uno cuenta con 30 puntos de salud y por cada impacto que reciben su salud disminuye en 10 puntos, es decir que una vez han recibido 3 disparos son destruidos. Los tanques pueden dañar tanto a sus aliados como a sí mismos por lo que es necesario disparar de manera inteligente para poder obtener la victoria.

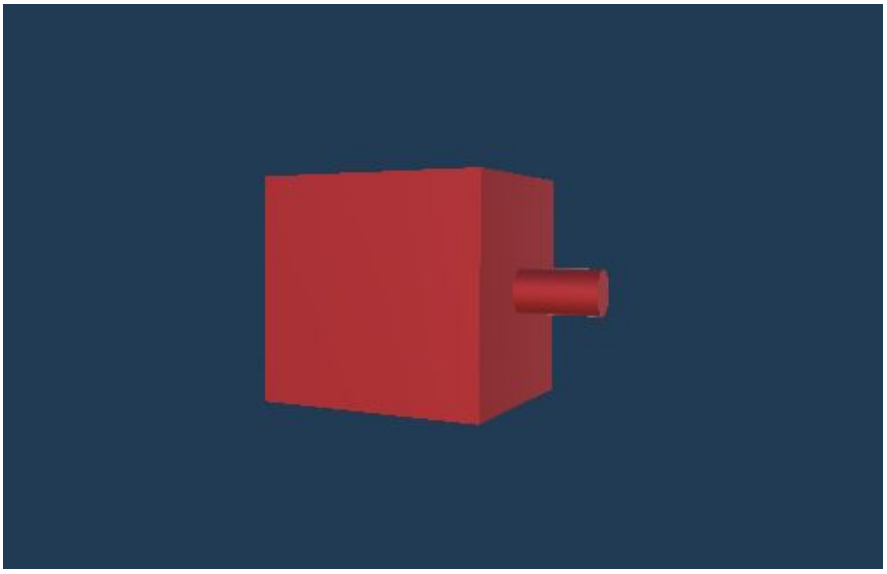


Figura 4.2: Representación de un tanque del equipo rojo

4.3 Proyectiles

Los proyectiles son *GameObjects* muy sencillos, de hecho, se trata simplemente de una esfera de color negro. Este proyectil es capaz de detectar las colisiones y en caso de que esta haya sido realizada contra un objeto del tipo tanque este tanque recibe daño.

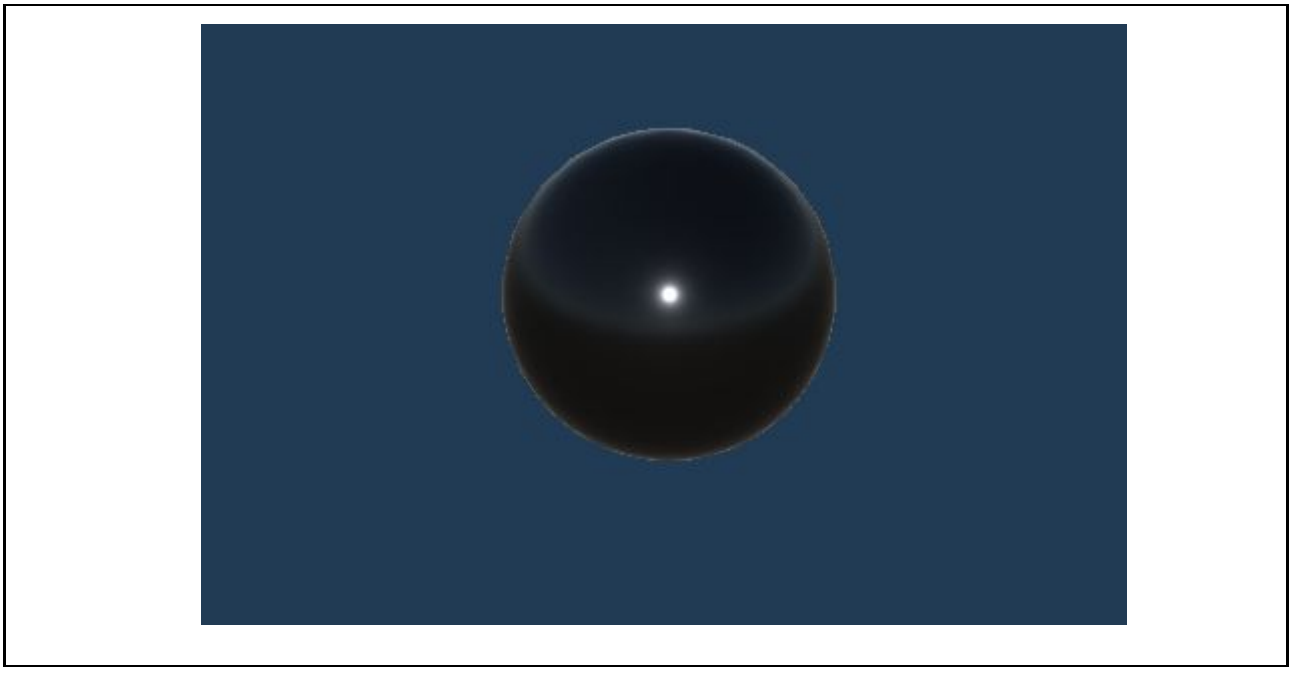


Figura 4.3: Representación de un proyectil

4.4 Área

El área en la que se encuentran los agentes es un plano rectangular delimitado por unas paredes invisibles (para poder visualizar mejor a los tanques) que impiden que los tanques escapen de la misma así como los proyectiles que estos disparan. Esta varía dependiendo del tipo de entrenamiento que se realice, por lo que se profundizará en los cambios realizados para cada entrenamiento en el apartado 5.2.3 “Tipos de entrenamiento”.

Capítulo 5 Desarrollo de la IA

5.1 Agentes

Para la programación de agentes inteligentes en ML-Agents es necesario definir las observaciones que recoge del entorno, las acciones que este puede realizar y el reinicio de los agentes cuando comienza un nuevo episodio [14] . Tambíen en el caso de este juego puesto que hay varios agentes en cada escena y se destruyen unos a otros es necesario manejar la destrucción o muerte de estos agentes de manera correcta.

5.1.1 Observaciones

Las observaciones que obtienen los agentes del entorno son cruciales para obtener un buen desempeño, no han de ser ni excesivas, puesto que esto puede ralentizar el entrenamiento, ni escasas, puesto que el agente no sería capaz de aprender la tarea que tiene asignado.

En el caso de los agentes de mi proyecto cuentan con un número de observaciones bastante limitado para que puedan aprender de manera más rápida. Estas son: la vida del tanque, la fuerza y ángulo de disparo, la rotación del tanque con respecto al área, si está disparando y finalmente una visión artificial utilizando raycast.

La vida del tanque es una observación importante, puesto que le permite desarrollar nuevos comportamientos en función del valor de la misma, es decir, jugar de manera más segura cuando la vida del tanque es baja y en cambio arriesgar un poco más cuando su vida es alta.

Tanto la fuerza como el ángulo del disparo son cruciales en este proyecto, puesto que el agente los utiliza para poder calcular la trayectoria que tendrá el disparo a realizar. Además necesita saber si está disparando en ese momento (se considera que sigue disparando hasta que el proyectil no impacte con algún objeto) puesto que no puede volver a disparar hasta que su disparo inicial haya concluido.

Puesto que la rotación es un factor importante en este juego, puesto que un agente solo puede percibir lo que tiene inmediatamente delante (como se explicará más adelante), es ideal que este tenga una noción de cuál es su rotación actual o si está rotando actualmente.

Finalmente cuenta con una visión mediante raycast, es decir, dispara rayos en varias direcciones con los que detecta colisiones y de esta manera es capaz de descubrir la posición de los otros agentes. En concreto utiliza 6 rayos hacia cada lado y uno hacia su frente, es decir 13 rayos en total con lo que obtiene un ángulo de visión de 80º. En la siguiente figura se puede apreciar de manera visual lo comentado.

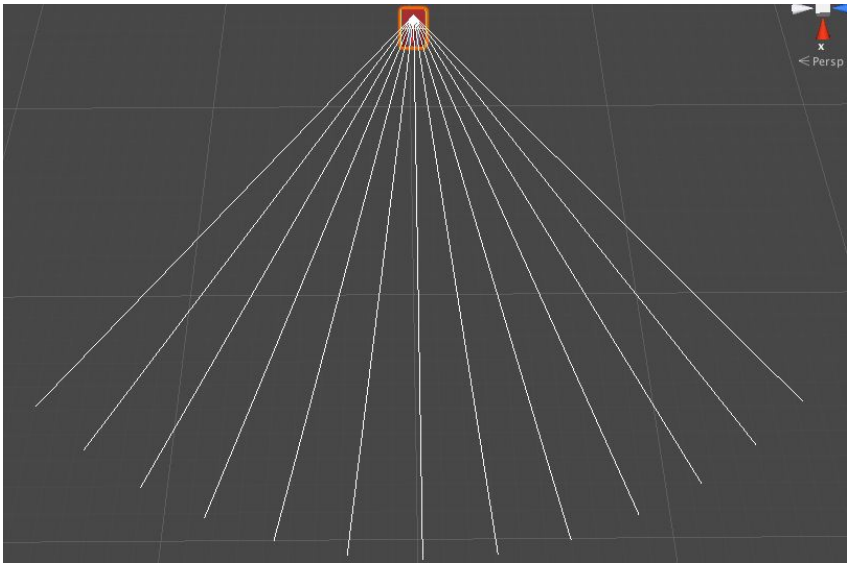


Figura 5.1: Visión por Raycast

Estos rayos permiten determinar tanto la distancia como el objeto con el que han impactado, por ello, están configurados para solo detectar los tanques aliados o enemigos, los obstáculos y los muros de área. Finalmente el agente cuenta con una memoria conocida como *stacked inputs*, que le permite almacenar datos de las observaciones obtenidas por estos rayos en los pasos anteriores, lo que lo posibilita a calcular velocidades y trayectorias de los tanques enemigos con el fin de que sus disparos sean más acertados.

Además de las observaciones comentadas, en los entrenamientos que cuentan con comunicación entre agentes (se explicará en el apartado 5.2.3, “Tipos de entrenamiento”) se añaden observaciones adicionales, para permitir esta comunicación. Básicamente se trata de la visualización de enemigos que han sido vistos por algún aliado, es decir, que si un tanque aliado encuentra a un tanque enemigo en el campo de juego utilizando su visión por raycast, todos los agentes de su equipo recibirán una observación con la posición de este enemigo. Para complementar esta observación se añade también a las lista de observaciones la posición local del tanque en el área, lo que le permite al agente situarse en el campo de juego y, más importante aún, calcular la trayectoria desde su posición hasta un enemigo que haya sido visualizado por un aliado, puesto que obviamente sin contar con su propia posición sería imposible.

5.1.2 Acciones

Cada agente puede tomar acciones en base a las observaciones que ha recibido previamente. En este caso las acciones que pueden tomar son las que vienen naturalmente heredadas de las mecánicas del juego, es decir, cada agente puede avanzar, retroceder, girar, aumentar o disminuir su fuerza y ángulo de disparo y disparar.

5.1.3 Reinicio del agente

Los reinicios del agente son realizados cada vez que comienza un nuevo episodio. En

este caso son bastante intuitivos, simplemente se trata de reiniciar los parámetros de los mismos a su valor inicial, o sea, su vida vuelve a ser la vida máxima, su ángulo y fuerza de disparo se reasignan a su valor inicial y finalmente se sitúan en una posición y rotación aleatoria en su área de aparición (dependiendo del tipo de entrenamiento puede abarcar todo el área o tan solo el spawner para cada equipo).

5.1.4 Muerte de un agente

Manejar la muerte de un agente es uno de los principales problemas que se plantea al crear un juego por equipos de este estilo. Hay varias opciones para solventar este problema, pero finalmente se optó por la desactivación del *GameObject* que contiene al agente. De esta manera el agente deja de recibir observaciones y de aprender hasta que sea reiniciado en el siguiente episodio y además desaparece de la escena para no entorpecer el entrenamiento de los tanques que quedan con vida.

5.2 Entrenamiento

Los entrenamientos se consideran los periodos en los que los agentes utilizan los datos de ejemplo para ir creando el modelo matemático que define sus acciones.

5.2.1 Opciones del entrenamiento

La primera opción crucial que hay que tener en cuenta a la hora de entrenar los agentes es qué algoritmo de aprendizaje elegir. Tal y como se comentó en el apartado 5.2.3 “Algoritmos de aprendizaje”, Ml-Agents cuenta con dos algoritmos, el PPO y el SAC, siendo la principal ventaja del SAC sobre el PPO el menor número de datos de ejemplo necesarios para aprender. Puesto que en este proyecto no hay dificultad alguna en obtener datos de aprendizaje ya que se generan durante los entrenamientos, se ha optado por utilizar el algoritmo PPO.

Cada episodio de los agentes se ha configurado a 5000 pasos o hasta que todos los agentes de un equipo hayan sido destruidos y cada 5 pasos en cada episodio el agente decide una nueva acción a realizar. Tras finalizar cada episodio el agente es reiniciado como se comentó previamente, lo que hace que cada episodio se trate de una partida completamente nueva. El número de pasos necesarios para completar el entrenamiento varía dependiendo del tipo de entrenamiento a realizar por la complejidad del mismo, pero por lo general no suele superar los 25.000.000 pasos por lo que por norma general se toma este como el límite de pasos a realizar.

Los entrenamientos se realizan utilizando 5 instancias concurrentes de Unity y en cada una de ellas se encuentran 10 áreas independientes, por lo tanto se están ejecutando 50 partidas en cada momento. Esto permite acelerar notablemente el proceso de aprendizaje aunque, como es obvio, requiere mayor uso del hardware del ordenador.

5.2.2 Recompensas y Self Play

El *Self play* consiste en enfrentar al agente contra una versión de sí mismo. Esto permite un aprendizaje óptimo puesto que ambos cuentan con niveles de habilidad

similares [15].

Los entrenamientos se realizan haciendo uso de la opción de *Self play* de ML-Agents, dicha opción funciona de manera especialmente bien en los juegos conocidos como "suma cero", en los que la pérdida de recompensa de un agente es equivalente a la ganancia del otro. Este es el caso exacto de este proyecto, puesto que si un equipo gana, el equipo ganador obtiene la misma recompensa que el equipo perdedor solo que en el caso del perdedor esta es negativa.

Representar este sistema de recompensas de manera tradicional puede ser bastante difícil y requerir de funciones de recompensa bastante complejas, pero haciendo uso del *Self play* se puede limitar a definir unos equipos y unas condiciones de victoria o derrota. De esta manera cuando un equipo muere completamente se considera que ha perdido y se le da a todos los tanques de ese equipo la recompensa '-1' en cambio al otro equipo se le otorga una recompensa de '1' puesto que ha ganado. En caso de que el episodio sea completado sin que ningún equipo muera se considera un empate y ambos equipos se quedan con la recompensa inicial de '0'.

Finalmente en los modos de juego que incluyen equipos se ha añadido una pequeña recompensa adicional, que a pesar de salirse un poco del estándar de juego "suma cero", ha demostrado otorgar mejores resultados en este proyecto en específico. Esta consiste en pequeñas recompensas de '0.2' cuando un tanque elimina a un enemigo y una recompensa negativa de '-0.2' cuando esté elimina a uno de sus aliados con el fin de penalizar el fuego amigo.

Este sistema de recompensas puede parecer sencillo, pero es lo recomendado en los entrenamientos que utilizan *Self Play*.

A la hora de ejecutar entrenamientos con *Self Play* el módulo Ml-Agents proporciona un sistema de puntuación conocido como Elo que se trata de un modelo matemático que pretende calcular la habilidad relativa de un jugador (o en nuestro una inteligencia artificial). Por defecto cada entrenamiento de Ml-Agents cuenta con un Elo inicial de 1200 y lo correcto sería que este aumentará constantemente a lo largo del mismo.

5.2.3 Tipos de entrenamiento

Durante este TFG se han planteado diversos problemas para así poder comprobar la forma en la que se desenvuelven los agentes al enfrentarse a ellos. Cada uno de los distintos problemas abordados cuenta con variaciones con respecto a los demás, lo que hace que tengan que tener entrenamientos ligeramente diferentes. Por ello he dividido la explicación de los entrenamientos en 5 grupos: Entrenamiento individual, entrenamiento por equipos, entrenamiento con obstáculos, entrenamiento con comunicación y finalmente entrenamiento con comunicación y obstáculos.

El entrenamiento individual, como es de esperar, es el más sencillo de todos. Cada entorno se basa en un área completamente limpia y sin obstáculos en la que dos agentes (uno en cada equipo) aparecen en posiciones y rotación aleatorias e intentan buscar y destruir al otro.

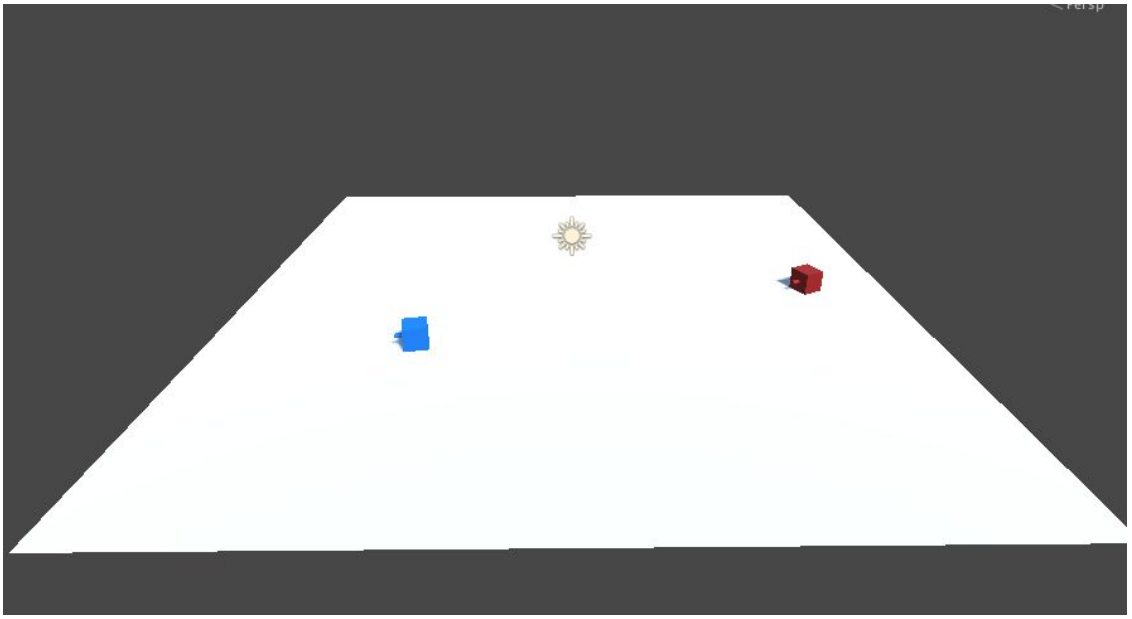


Figura 5.2: Área de entrenamiento individual

En el entrenamiento por equipos se añaden uno o más agentes a cada equipo, incluyendo así la posibilidad del “fuego amigo”. Este proceso de adición de agentes se hace de manera controlada utilizando una tećnica conocida como *curriculum learning* que permite cambiar parámetros del entorno del entrenamiento a medida que este avanza para así poder empezar con situaciones sencillas e ir escalando a las más complicadas (en este caso de menos tanques a más tanques). Además se añade un área de aparición o spawner para cada equipo en la que los tanques de ese equipo aparecen en posiciones y rotación aleatoria al inicio de cada episodio.

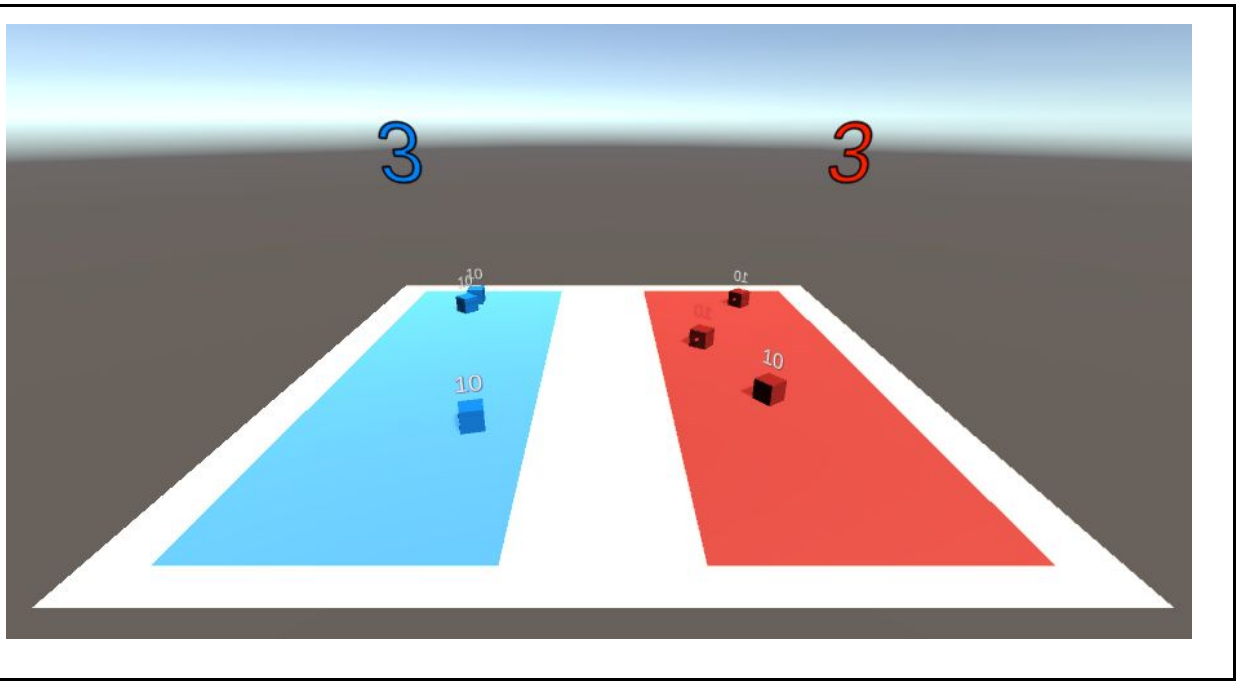


Figura 5.3: Área de entrenamiento por equipos y con comunicación

Seguidamente nos encontramos con el entrenamiento con obstáculos, en el que tal y como su nombre indica se añaden obstáculos al entorno que dificultan la visualización de tanques enemigos. Como en previos entrenamientos se hace uso de técnicas de *curriculum learning* que permiten un aprendizaje más efectivo y rápido, en este caso esta técnica se aplica a los propios obstáculos que no están presentes en las etapas iniciales del entrenamiento (con el fin de que los agentes aprendan las cosas básicas) y se incorporan durante las etapas finales del mismo.

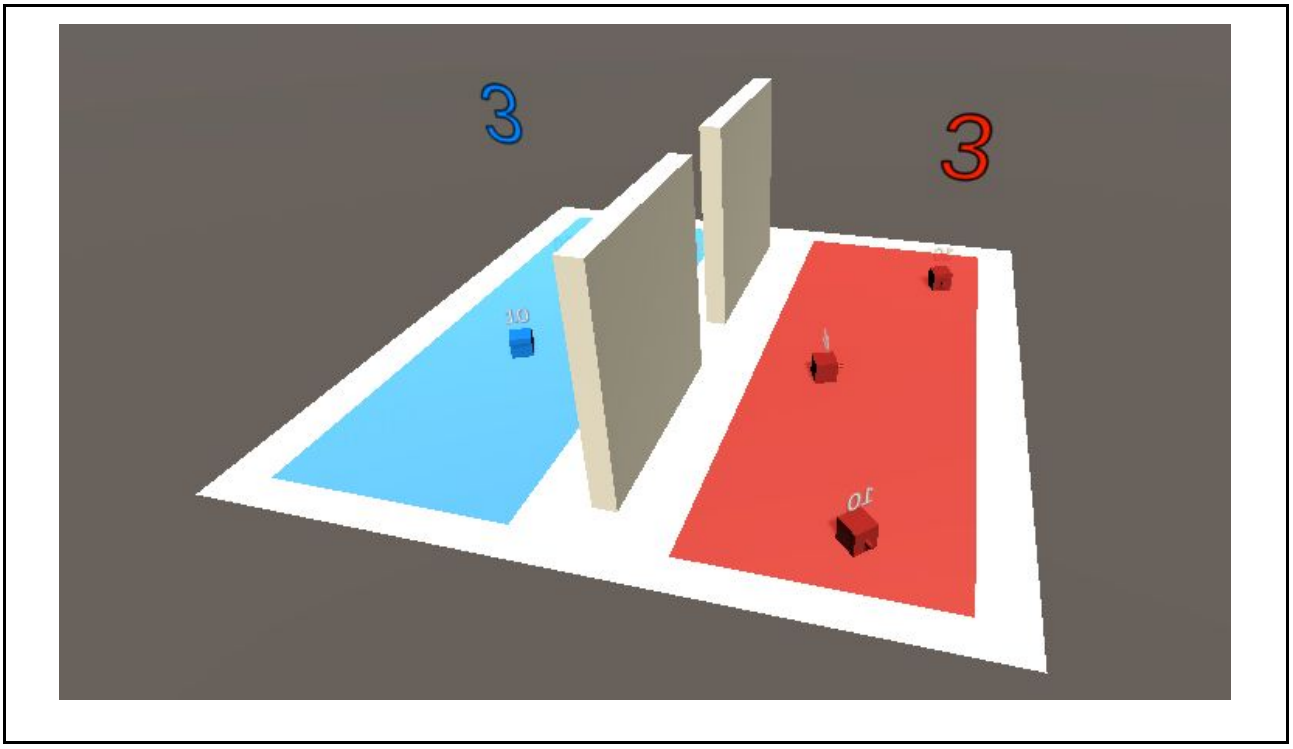


Figura 5.4: Área de entrenamiento con obstáculos

El entrenamiento con comunicación es muy similar al entrenamiento por equipos, de hecho se desarrollan en un entorno con las mismas características. La diferencia viene en los agentes que se utilizan en este entorno, que tal y como se explicó en el apartado 5.1.1 “Observaciones”, cuentan con unas modificaciones en las observaciones que les permiten comunicarse la posición de los enemigos.

Finalmente en entrenamiento con obstáculos y comunicación, tal y como su nombre indica, se trata de una variación del entrenamiento con obstáculos original pero realizado con los agentes con comunicación explicados anteriormente.

Capítulo 6 Resultados y evaluación

En este capítulo se detallarán los procesos realizados para los distintos entrenamientos de este proyecto así como los resultados que se han obtenido. Todas las gráficas representadas a lo largo del mismo cuentan con el Elo en el eje 'Y' y el número de pasos en el eje 'X'.

El hardware utilizado para los entrenamientos consiste en un Intel Core i7-7500U como CPU, una Nvidia 940MX como tarjeta gráfica y 8 Gb de ram.

6.1 Entrenamiento individual

El primer entrenamiento aplicado a los agentes es el entrenamiento individual descrito en capítulos anteriores. Pese a su baja complejidad representó un reto mayor mayor del esperado durante las primeras fases de desarrollo del proyecto, en gran medida ocasionado por una incorrecta elección de observaciones y/o configuraciones erróneas. Finalmente después de pulir el proceso a base documentación y pruebas se consiguió obtener una configuración óptima para resolver este tipo de problema.

Los resultados obtenidos finalmente son bastante buenos y representan un correcto aprendizaje por parte de los agentes, cuya habilidad en este juego no hacía más que aumentar como se puede ver en la siguiente gráfica.

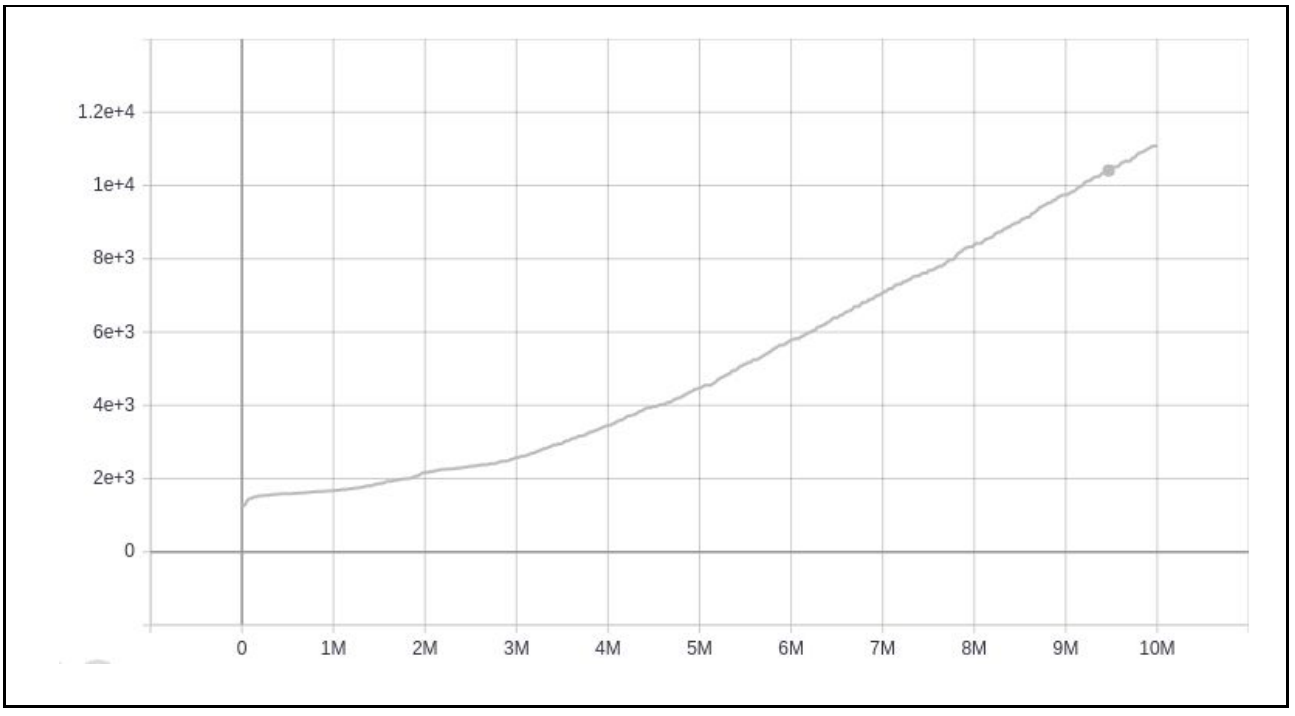


Figura 6.1: Gráfica de entrenamiento individual

En concreto el entrenamiento abarcó diez millones de pasos, que transcurrieron durante el intervalo de cuatro horas, alcanzando finalmente un Elo de 11.083, lo que es un aumento considerablemente grande teniendo en cuenta que el Elo inicial es de 1200.

La inteligencia artificial obtenida como resultado no es del todo realista desde un punto de vista humano, puesto que, la mejor estrategia que los agentes pudieron deducir consiste en buscar al tanque enemigo y una vez encontrado acercarse a él hasta una distancia muy reducida, para así siempre asegurarse de que su disparo sea certero. Esta estrategia es bastante efectiva pero al mismo tiempo arriesgada, ya que quedan muy expuestos a ser impactados por proyectiles enemigos y podría ser fácilmente contrarrestada por otra estrategia que emplee disparos a mayor distancia.

6.2 Entrenamiento por equipos

El entrenamiento por equipos, a pesar de las amplias similitudes que comparte con el individual, plantea algunos retos que no existían en este, siendo el principal de ellos cómo manejar la muerte de un agente para que no afecte al entrenamiento de los que quedan con vida. Tal y como se comentó durante el apartado 5.1.4 se optó por desactivar el *GameObject* que contenía al agente y darle una recompensa negativa para que pueda identificar los errores que ha cometido.

Para este entrenamiento se utilizó la técnica de *curriculum learning* comentada anteriormente, de manera que durante los primeros diez millones de pasos sólo había un tanque por cada equipo, los siguientes cinco millones de pasos habían dos tanques por equipo y finalmente durante los últimos cinco millones de pasos este número se aumentó a tres tanques. De esta manera se obtuvo un entrenamiento de una duración total de 20 millones de pasos y 8 horas.

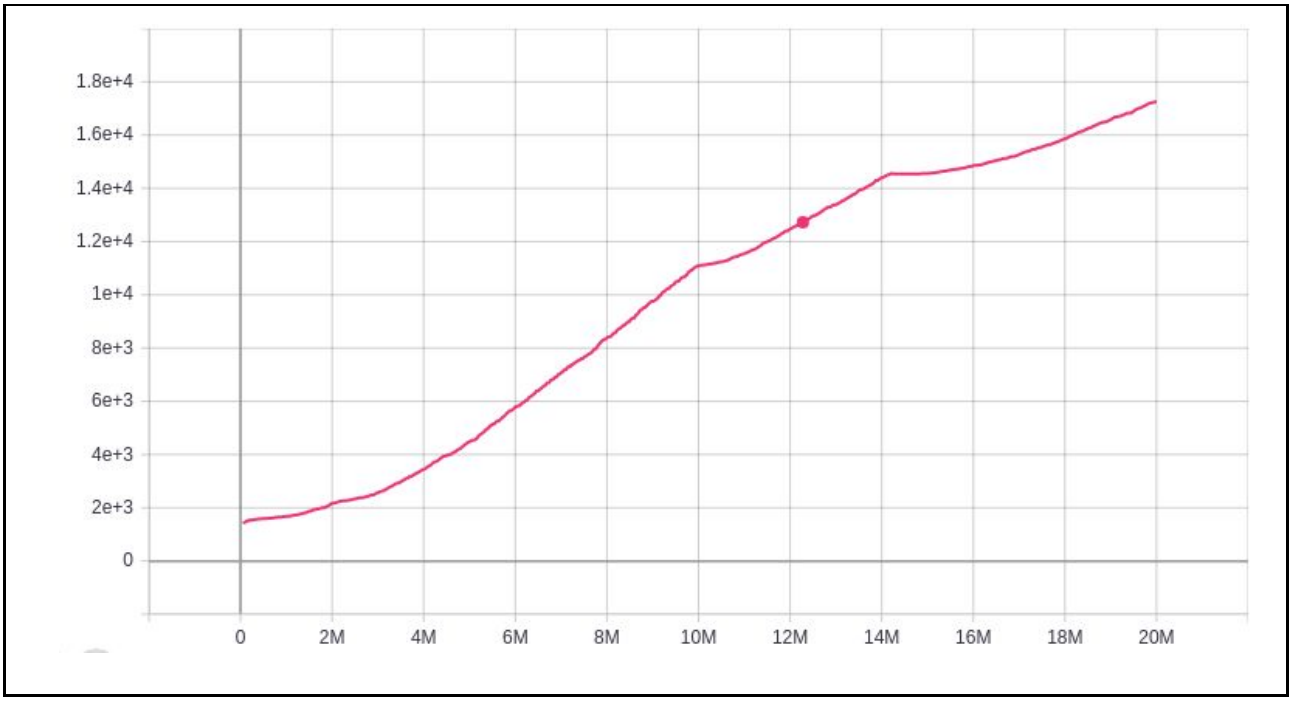


Figura 6.2: Gráfica de entrenamiento por equipos

En la gráfica se pueden observar las distintas alteraciones que se producen en las marcas de 10 y 15 millones, siendo estas originadas por el cambio en el número de agentes por equipo y rompiendo la constante creciente del entrenamiento debido al cambio en el entorno que estos suponen.

Finalmente la inteligencia artificial obtenida tiene un comportamiento bastante similar a la del apartado anterior, aunque algo más compleja puesto que es capaz de reconocer a los tanques de su propio equipo y no dispararles mientras que sí lo hace a los tanques enemigos.

6.3 Entrenamiento con obstáculos

Para el entrenamiento con obstáculos se parte de inteligencia artificial obtenida en el entrenamiento por equipos y se realiza un nuevo entrenamiento durante 5 millones de pasos adicionales con los obstáculos añadidos al área de juego, por tanto la red neuronal obtenida ha entrenado durante 25 millones de pasos en total lo que ha tardado alrededor de unas 10 horas en completarse.

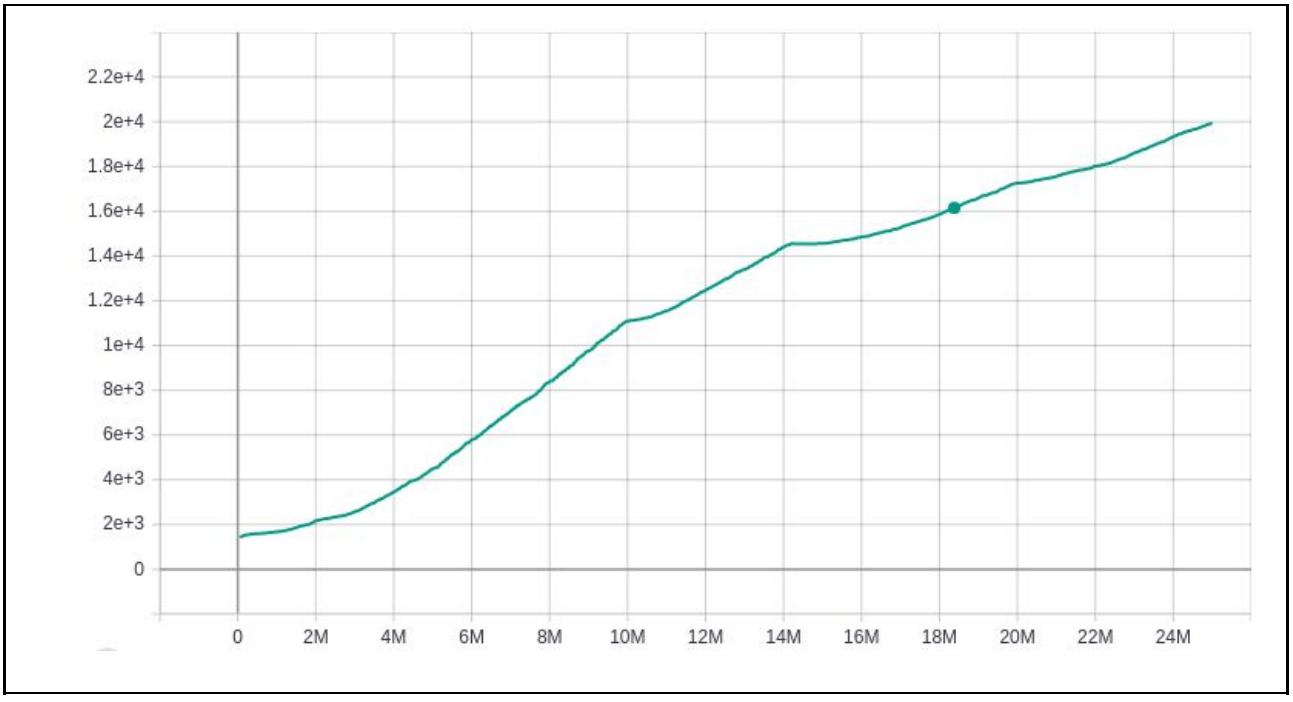


Figura 6.3: Gráfica de entrenamiento con obstáculos

Se puede apreciar la incorporación de los obstáculos en el paso 20 millones, que llevó a que durante unos momentos el aumento de Elo se ralentizará, pero rápidamente volvió a su ritmo habitual.

El resultado obtenido ha sido un comportamiento bastante previsible dadas las circunstancias, en el que los agentes simplemente se limitan a moverse por el mapa hasta localizar a un tanque enemigo, para así proceder a acercarse a él mismo y eliminarlo. Hubiera sido interesante el lograr que los agentes aprendieran a cubrirse con los obstáculos, pero no ha sido posible obtener este comportamiento.

6.4 Entrenamiento con comunicación

El entrenamiento con comunicación es muy similar al de por equipos, debido a que la única variación que se hace es la adición de nuevas observaciones. Al igual que en el entrenamiento por equipos se utilizan los mismos incrementos de tanques por cada intervalo de pasos, lo que hace que la gráfica tenga similitudes notables con la anterior.

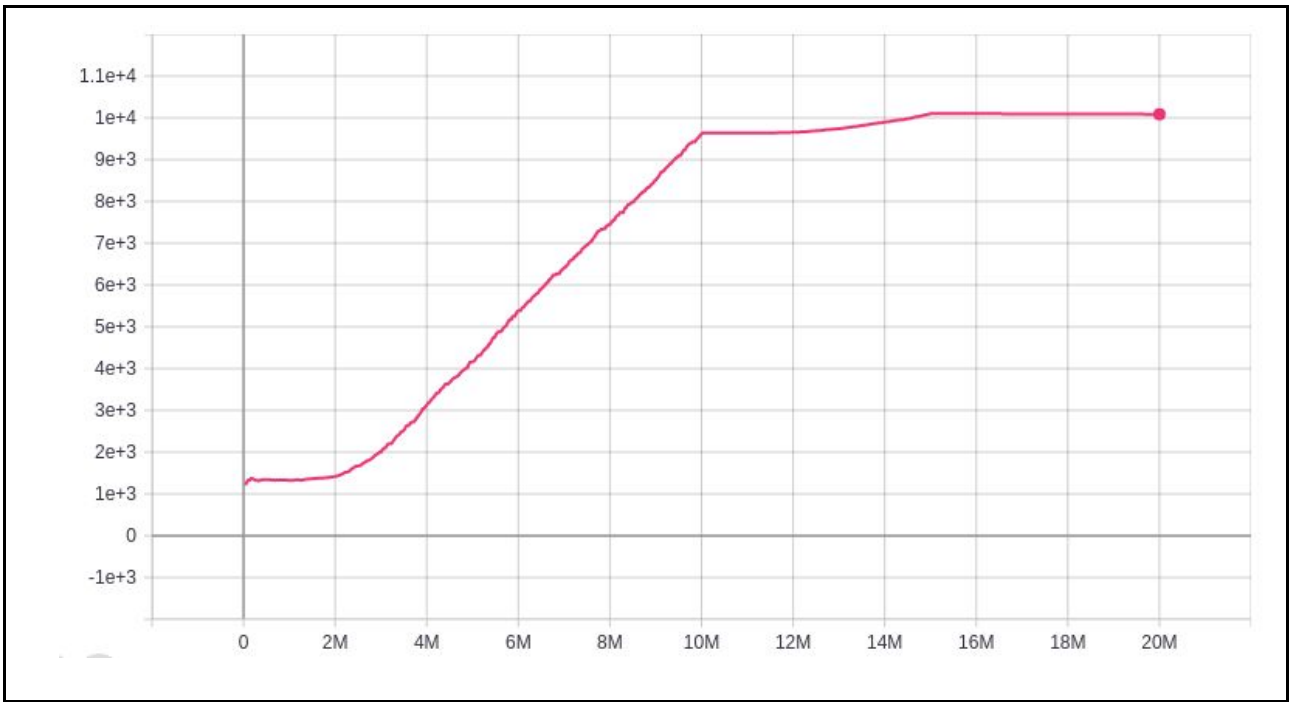


Figura 6.4: Gráfica de entrenamiento con comunicación

Se puede apreciar que el crecimiento de esta gráfica es algo más lento que en apartados anteriores, lo que es entendible debido a que el número de observaciones con el que cuentan es superior y esto dificulta notablemente el entrenamiento. También es fácil observar las mismas alteraciones que en el entrenamiento por equipos en las marcas de 10 y 15 millones de pasos, puesto que son las adiciones de nuevos agentes a cada equipo.

A pesar de contar con más observaciones los resultados en el comportamiento de los tanques no son muy diferentes a los conseguidos en el entrenamiento por equipos sin comunicaciones, aunque se pondrá a prueba durante el apartado de “Pruebas finales”.

6.5 Entrenamiento con comunicación y obstáculos

Para facilitar el entrenamiento con comunicación y obstáculos se va a partir de la inteligencia artificial obtenida al realizar el entrenamiento con comunicación de esta manera al igual que en el entrenamiento con obstáculos original tan solo los últimos 5 millones de pasos de un total de 25 millones tendrán los susodichos obstáculos en el área, llevando el entrenamiento hasta las 12 horas totales.

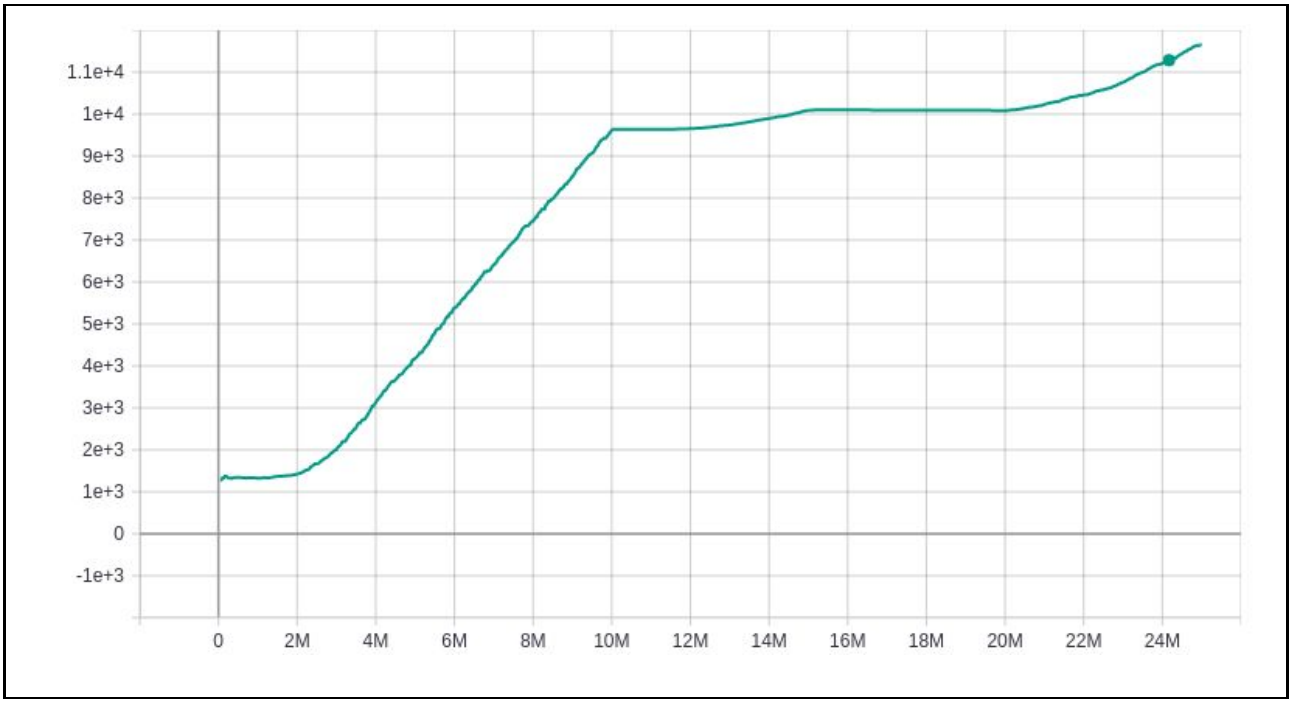


Figura 6.5: Gráfica de entrenamiento con comunicación y obstáculos

Como era de esperar a partir del paso 20 millones se produce una ligera alteración en el crecimiento de la gráfica debido a la adición de los obstáculos, aunque, en este caso curiosamente la velocidad de aprendizaje aumenta al añadirlos.

Parece que en esta situación el comportamiento es algo más efectivo que en entrenamientos anteriores

6.6 Pruebas finales

Finalmente se han realizado una serie de pruebas para así poder obtener unos resultados empíricos más representativos sobre la utilidad de las comunicaciones en el juego. Para ello se han creado nuevas áreas en las que se han enfrentado por equipos a dos inteligencias artificiales diferentes de las generadas anteriormente, una contando con comunicación entre agentes y la otra sin esta.

Con el fin de obtener unos resultados lo más objetivos posibles, el proceso ha consistido en enfrentar ambas inteligencias artificiales durante 1000 partidas anotando en cada una de ellas cuál ha sido el vencedor, de manera que se pueda diferenciar que inteligencia artificial es superior. Este proceso se ha realizado tanto en un área con obstáculos como en una sin obstáculos.

Los resultados obtenidos tras realizar las 1000 partidas sobre un área sin ningún obstáculo son los siguientes:

Resultados	Victorias	Derrotas	% Victoria	% Derrota
Sin comunicación	162	838	16.2%	83.8%
Con Comunicación	838	162	83.8%	16.2%

Tabla 6.1: Resultados enfrentamiento sin comunicación contra con comunicación

Tal y como se puede apreciar el margen es bastante grande poniendo en ventaja de manera clara a la inteligencia artificial que contaba con comunicaciones entre los agentes. Además tal y como podemos ver en la siguiente tabla, este gran margen no hace más que agravarse cuando estas dos inteligencias artificiales se enfrentan en un área en la que hay obstáculos.

Resultados	Victorias	Derrotas	% Victoria	% Derrota
Sin comunicación	68	932	6.8%	93.28%
Con Comunicación	932	68	93.2%	6.8%

Tabla 6.2: Resultados enfrentamiento sin comunicación contra con comunicación en un área con obstáculos

Estos resultados obtenidos eran los esperables puesto que los agentes con comunicación cuentan con unas observaciones que les otorgan una gran ventaja con respecto a los otros, sobre todo en el entorno con obstáculos ya que la visualización de enemigos se complica y estos agentes son capaces de comunicar las posiciones de los mismos cuando tan sólo un tanque del equipo lo ha visualizado.

Capítulo 7 Conclusiones y líneas futuras

Tras analizar los resultados obtenidos, se puede afirmar que se ha conseguido el objetivo de lograr una inteligencia artificial capaz de jugar a este juego. A pesar de ello, el comportamiento alcanzado no es en todos los casos realista desde un punto humano y es fácil diferenciar a una de estas inteligencias artificiales de un jugador real.

Como balance personal, este TFG me ha ayudado a darme cuenta de la increíble potencia y versatilidad que tiene esta tecnología, en la que hasta ahora era completamente inexperto. Me ha permitido obtener conocimientos y experiencias que me serán muy útiles durante mi vida laboral que seguramente esté relacionada con el ámbito de la inteligencia artificial y/o los videojuegos.

Como posibles líneas futuras del proyecto podría ser la adición de nuevos cambios en el entorno que hagan que cambie el estilo de juego de los agentes tales como power-ups, mapas con relieve o más de dos equipos en cada área. También, puesto que es posible que los comportamientos más complejos y realistas no se dieran en este proyecto debido a que el espacio de acciones y observaciones a explorar es muy amplio, sería interesante el intentar optimizar aún más los entrenamientos realizando más experimentos en paralelo, así como sesiones de entrenamiento más largas y/o cambios en las observaciones y recompensas con el fin de intentar descubrir estos comportamientos. Finalmente sería factible el incorporar la inteligencia artificial obtenida a un juego real en el que pudiera enfrentarse a humanos para así poder comprobar su desempeño en estas situaciones.

Capítulo 8 Summary and Conclusions

After analyzing the results obtained, it is confirmed that the objective of obtaining an artificial intelligence capable of playing this game has been achieved. Despite this, the behavior achieved is not in all cases realistic from a human point of view and it is easy to differentiate one of these artificial intelligences from a real player.

As a personal balance, this TFG has helped me to realize the incredible power and versatility that this technology has, in which until now I was completely inexperienced. It has allowed me to obtain knowledge and experiences that will be very useful during my working life that is surely related to the field of artificial intelligence and / or video games.

Possible future lines of the project could be the addition of new changes in the environment that change the style of play of the agents, such as power-ups, relief maps or more than two teams in each area. Also, since it is possible that the most complex and realistic behaviors did not occur in this project due to the fact that the space of actions and observations to be explored is very wide, it would be interesting to further optimize the training by carrying out more experiments in parallel, as well as longer training sessions and / or changes in observations and rewards in order to try to discover these behaviors. Finally, it would be feasible to incorporate the artificial intelligence obtained into a real game in which it could face humans in order to check its performance in these situations.

Capítulo 9 Presupuesto

Todas las tecnologías utilizadas durante el proyecto son gratuitas y/o de código abierto y no se van a tener en cuenta los recursos básicos para poder realizarlo (ordenador, conexión a internet, electricidad, etc). Por lo tanto este presupuesto se va a basar en el sueldo aproximado que debería de haber cobrado un ingeniero informático junior para realizar este trabajo.

Para poder concluir este TFG se han empleado alrededor de unas 400 horas de trabajo, en las que se incluyen los períodos de aprendizaje de los agentes. En base a la información recopilada se estima que el sueldo medio de un ingeniero informático junior por cada hora ronda los 8€, por lo tanto el presupuesto aproximado total de este TFG asciende a unos 3200€.

Capítulo 10 Apéndice A

10.1 Repositorio del proyecto

El proyecto de Unity que contiene tanto el código como las configuraciones de ML-Agents ha sido subido a un repositorio público de Github.

- https://github.com/alu0101040882/DeepLearningUnityTFG

Bibliografía

[1] Alan M. Turing, Mind 49: 433-460, “Computing Machinery and Intelligence”

https://www.csee.umbc.edu/courses/471/papers/turing.pdf

[2] Marvin Minsky, Proceedings of the IRE, “Steps toward Artificial Intelligence”

http://www-public.imtbs-tsp.eu/~gibson/Teaching/CSC4504/ReadingMaterial/Minsky61.pdf

[3] IBM, “Deep Blue”

https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

[4] Wikipedia, “OXO”

https://es.wikipedia.org/wiki/OXO

[5] Google, “Stadia”

https://stadia.google.com/home

[6] Nvidia, “StyleGan”

https://github.com/NVlabs/stylegan

[7] Michael Cook, IEEE Transactions on Computational Intelligence and AI in Games, “The ANGELINA Videogame Design System, Part I”

http://research.gold.ac.uk/17349/1/COM-Cook2016a.pdf

[8] Unity-Technologies, “Plataforma de Unity”

https://unity.com/es/products/unity-platform

[9] Unity Technologies, “Unity ML-Agents Toolkit”

https://github.com/Unity-Technologies/ml-agents

[10] TensorFlow, “TensorFlow Overview”

https://www.tensorflow.org/overview

[11] Arthur Juliani, Unity3D Blogs, “Introducing: Unity Machine Learning Agents Toolkit (2017)”

https://blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents/

[12] Schulman, Wolski, Prafulla, Radford and Klimov (2017), arXiv:1707.06347, “Proximal Policy Optimization”

https://arxiv.org/pdf/1707.06347.pdf

[13] Haarmoja, Zhou, Abbeel and Levine (2018), arXiv:1801.01290, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”

https://arxiv.org/pdf/1801.01290.pdf

[14] Unity Technologies, “Making a New Learning Environment”

https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Create-New.md

[15] Andrew Cohen, Unity3D Blogs, “Training intelligent adversaries using self-play with ML-Agents (2020)”

https://blogs.unity3d.com/es/2020/02/28/training-intelligent-adversaries-using-self-play-with-ml-agents/