Universitat
de les Illes Balears

TRABAJO DE FIN DE GRADO

UNA IA QUE JUEGA A POKÉMON

Gerard Naharro López

Grau d'Enginyeria Informàtica

Escola Politècnica Superior

Año académico 2022-23

UNA IA QUE JUEGA A POKÉMON

Gerard Naharro López

Trabajo de Fin de Grado

Escola Politècnica Superior

Universitat de les Illes Balears

Año académico 2022-23

Paraules clau del treball: TFG, memoria, Pokémon, Gym retro, Deep Reinforcement Learning, Inteligencia Ar

Tutor: Gabriel Moyà Alcover y José María Buades Rubio

A mis padres, por haberme ayudado a ser quién soy.

ÍNDICE GENERAL

ACRÓNIMOS

RL Reinforcement Learning

IA Inteligencia Artificial

NPC Non Playable Character

IDE Entorno de Desarrollo Integrado

VSC Visual Studio Code

ROMs Read Only Memories

ROM Read Only Memory

PPO Proximal Policy Optimization

ML Machine Learning

SVM Super Vector Machine

TFG Trabajo Fin de Grado

CNN Convolutional Neural Network

GBA Game Boy Advance

DRL Deep Reinforcement Learning

RAM Random Access Memory

GPU Graphics Processing Unit

FPS Frames Per Second

LOOCV Leave-One-Out Cross-Validation

CPU Central Processing Unit

RESUMEN

En el presente proyecto se abordará el desafío de desarrollar dos agentes de Inteligencia Artificial (IA) capaces de aprender a jugar y ganar combates en el videojuego Pokémon Rojo Fuego utilizando el aprendizaje por refuerzo. Se utilizará una metodología ágil con incrementos semanales para gestionar el trabajo de manera efectiva. En el análisis, se identificarán los requisitos funcionales y no funcionales, y se establecerá un entorno tecnológico adecuado, incluyendo el uso de librerías como Gym Retro y Stable Baselines. El desarrollo, se centrará en la creación de una base de datos de combates, la implementación de redes neuronales como NATURE-CNN e IMPALA-CNN, y la construcción de un entorno funcional. En la experimentación y resultados, se llevarán a cabo las fases de entreno y validación, se aplicarán métricas definidas y se comparará el rendimiento de las redes utilizando técnicas como *bootstrapping* y permutaciones, con el objetivo de evaluar su desempeño. Finalmente se extraerán conclusiones sobre los experimentos llevados a cabo. En general, el proyecto será una oportunidad enriquecedora para aplicar conocimientos en IA y Machine Learning (ML) en el contexto de un videojuego, resaltando la importancia de las redes neuronales y las técnicas de aprendizaje por refuerzo en el desarrollo de sistemas inteligentes.

CAPÍTULO 1

INTRODUCCIÓN

El aprendizaje automático y la Inteligencia Artificial (IA) son dos de los campos de la informática que han experimentado un crecimiento más significativo en los últimos años. En este sentido, los videojuegos se han convertido en una plataforma ideal para la investigación y el desarrollo de nuevas técnicas de IA. En particular, el videojuego Pokémon Rojo Fuego, lanzado en 2004 para la consola Game Boy Advance (GBA), ha sido objeto de interés para la realización de este trabajo de fin de grado debido a su complejidad y a la gran cantidad de decisiones estratégicas que deben tomarse durante el juego. Por tanto, se explorará cómo se puede utilizar el aprendizaje por refuerzo para entrenar a un agente de IA para jugar y ganar en Pokémon Rojo Fuego.

1.1 IA

Hoy en día, la Inteligencia Artificial (IA), es cada vez más un tema habitual de conversación debido a los diferentes avances en ella y a su impacto innegable en nuestra vida diaria. Eso, sumado a apariciones que han adquirido gran relevancia como el ChatGPT, han hecho que deje de ser un tema de nicho a abarcar un gran número de personas.

Es un campo de la informática que se centra en la creación de sistemas que pueden realizar tareas que normalmente requieren inteligencia humana, como el aprendizaje, la toma de decisiones, la resolución de problemas y la percepción del lenguaje natural. Se basa en el procesamiento de grandes cantidades de datos y en la aplicación de algoritmos y técnicas estadísticas para aprender de esos datos y mejorar su desempeño con el tiempo.

La podemos encontrar en nuestra vida diaria de muchas formas diferentes como por ejemplo: smartphones, smartwatches, chatbots, videojuegos.

Figura 1.1: Pantalla de título de: Pokémon Rojo Fuego versión japonesa

1.2 Videojuegos

Por otra parte, encontramos la industria de los videojuegos, uno de los mayores referentes en ocio y cultura de la actualidad, estos involucran la interacción del jugador con un sistema electrónico que controla imágenes y sonidos en una pantalla. Hoy en día existen de muchos tipos, tanto en naturaleza del propio juego, como en formas de jugarlo. Han evolucionado significativamente desde su creación en la década de 1970, originariamente se interactuaba con ellos a través de un teclado o un mando, hoy en día a estas posibilidades se incluyen de más sofisticadas como pantallas táctiles, visores y sensores de realidad virtual, etc. Tal es la popularidad de estos que se crean hasta torneos de *e-sports* (deportes electrónicos) con una base de fans apasionada que sigue los lanzamientos y noticias de la industria de los videojuegos. A medida que la tecnología sigue avanzando, se espera que los videojuegos continúen evolucionando y ofreciendo experiencias de juego cada vez más inmersivas y emocionantes.

1.2.1 Pokémon

El fenómeno de los videojuegos, ha logrado crear a lo largo de los años grandes entregas, que han emocionado y atraído a diferentes tipos de personas desde todas partes del mundo. Una de las sagas más famosas ha sido Pokémon, la cual se inició con los lanzamientos de Pocket Monsters Aka and Midori (Pokémon Red y Pokémon Green) en Japón en 1996, para la consola Game Boy. Tal fue el frenesí que crearon estos, que poco después se tuvo que lanzar en todo el mundo, creando un legado de actualmente 96 juegos que sigue vigente a día de hoy.

Sus videojuegos pertenecen al género JRPG por turnos. Caracterizados por sus combates de entrenador contra entrenador, donde cada uno debe tener desde un mínimo de 1 Pokémon hasta un máximo de 6. Cada uno de los Pokémon tiene diferentes atributos:

- Puntos de salud

- Ataque
- Defensa
- Ataque especial
- Defensa especial
- Velocidad
- Precisión
- Evasión
- Tipo

Existen ventajas y desventajas entre tipos Pokémon (agua es fuerte contra fuego, fuego a su vez lo es contra planta...) y perderá el jugador cuya totalidad del equipo Pokémon quede a 0 puntos de vida.

1.3 IA en los videojuegos

Ambos mundos (IA y videojuegos) no son excluyentes ni mucho menos, todo lo contrario. Desde poco después los inicios de los videojuegos, la IA ha estado presente, en mayor o menor medida, en muchos de ellos, actuando como Non Playable Character (NPC) o como tu rival a batir (el equipo contrario en un juego de fútbol, por ejemplo). Hoy en día, se ha dado un paso más adelante pudiendo ocupar incluso la figura del jugador humano, no diciéndole explícitamente a la IA lo que tiene que hacer, si no dejándola aprender mediante una serie de refuerzos positivos o negativos, esto es conocido como el Reinforcement Learning (RL) .

1.4 Reinforcement Learning

El concepto Reinforcement Learning (o RL), se asemeja a un sencillo método que se ha utilizado durante muchos años, por ejemplo en el adiestramiento de perros: pedirle hacer algo y reforzar positivamente o negativamente dependiendo de si la respuesta ha sido la esperada o no. En RL se define algo que se debe maximizar (por ejemplo una puntuación) y en base a realizar mucho y muchos intentos y los refuerzos positivos y negativos, la propia IA aprenderá que acciones la han llevado a obtener las mejores puntuaciones.

1.5 Objetivos

El objetivo principal de este trabajo de fin de grado, consistirá en utilizar el aprendizaje por refuerzo para nutrir a dos agentes de IA de las pautas necesarias para que ellas

aprendan a combatir de la mejor manera posible al Pokémon Rojo Fuego de la Game Boy Advance (GBA). Además de compararlas ambas para ver cuál de los dos es más idónea para este experimento.

Por tanto, para poder lograr nuestro objetivo principal deberemos cumplir una serie de objetivos previos:

- Creación de una base de datos.
- Acceso a los diferentes atributos que definen a un Pokémon durante una pelea.
- Creación de un entorno en Python que permita la experimentación.
- Realización pruebas preliminares.
- Entreno de la red neuronal y validación del mismo.
- Análisis de los resultados.

En conclusión, en el presente apartado de introducción se ha expuesto de manera general el propósito y el alcance del Trabajo Fin de Grado (TFG), que se enfoca en el desarrollo de dos agentes de Inteligencia Artificial (IA) que aprendan a jugar a Pokémon Rojo Fuego. Se ha establecido la relevancia de este proyecto, considerando el interés tanto en el campo de la Inteligencia Artificial (IA) como en la comunidad de jugadores de videojuegos. Además, se han delineado los potenciales temas tratados en este trabajo, como el avance en el campo del aprendizaje por refuerzo y la comprensión de la capacidad de las máquinas para aprender y mejorar su desempeño en entornos complejos. En esta introducción, hemos tratado los conceptos necesarios para poder iniciar el análisis detallado de los requisitos, la metodología, la temporización y el entorno tecnológico que se abordarán en el siguiente apartado, brindando una visión integral del proyecto y sentando las bases para su desarrollo exitoso.

CAPÍTULO 2

ANÁLISIS

Durante el capítulo de análisis, se abordarán algunos aspectos fundamentales del Trabajo Fin de Grado (TFG). Se describirán los requisitos necesarios para llevar a cabo el proyecto, tanto a nivel técnico como de conocimientos previos. Además, se presentará la metodología utilizada, basada en una metodología ágil adaptada, así como la temporización y la planificación ágil con incrementos semanales adoptada para la ejecución del trabajo. Por último, se analizará el entorno tecnológico y las herramientas utilizadas para implementar y evaluar la IA en el contexto del juego Pokémon Rojo Fuego, junto a una explicación de los conceptos previos más importantes para el entendimiento completo de este proyecto. Este análisis proporcionará una base sólida para comprender los elementos clave que sustentan el desarrollo exitoso del proyecto.

2.1 Requisitos

A la hora de desarrollar software y/o aplicaciones, una parte fundamental son los requisitos, generalmente estos se suelen dividir en dos: requisitos funcionales y requisitos no funcionales [1].

Los requisitos funcionales se refieren a las funcionalidades y comportamientos específicos que un sistema o software debe cumplir para satisfacer las necesidades del usuario. Estos requisitos describen las acciones que el sistema debe ser capaz de realizar, cómo debe responder a las entradas del usuario y qué resultados debe producir. Los requisitos funcionales se centran en el "qué" y definen las características y capacidades que se esperan del sistema.

Por otro lado, los requisitos no funcionales se refieren a los aspectos de calidad y rendimiento del sistema. Estos requisitos abordan características como la eficiencia, la usabilidad, la seguridad, la escalabilidad y el rendimiento del sistema. Los requisitos no funcionales se centran en el "cómo" y definen los criterios que el sistema debe

cumplir en términos de calidad, rendimiento y restricciones técnicas. Estos requisitos son igualmente importantes ya que afectan la experiencia del usuario, la eficiencia operativa y la capacidad del sistema para adaptarse a diferentes entornos y situaciones.

2.1.1 Requisitos funcionales

- **Interacción con el juego (RF1)**: La IA debe ser capaz de interactuar con el juego Pokémon Rojo Fuego, realizar acciones dentro del entorno virtual y recibir información del estado del juego, como los Pokémon enemigos o la posición del cursos en el menú de elección de acción.

- **Aprendizaje por refuerzo (RF2)**: El sistema debe implementar algoritmos de aprendizaje por refuerzo, para que la IA pueda aprender y mejorar su desempeño a lo largo del tiempo mediante la retroalimentación recibida del juego.

- **Definición de objetivos (RF3)**: Se deben establecer los objetivos específicos que la IA debe buscar en el juego, para poder ganar batallas contra entrenadores. Estos objetivos proporcionarán la dirección y el enfoque para el proceso de aprendizaje.

2.1.2 Requisitos no funcionales

- **Rendimiento (RNF1)**: La IA debe ser capaz de tomar decisiones y realizar acciones de manera rápida, para que se produzca una reproducción del juego lo más parecida posible a la que haría un humano.

- **Escalabilidad/Generalización (RNF2)**: El sistema debe tener la capacidad de adaptarse a diferentes niveles de dificultad del juego y a escenarios más complejos a medida que la IA mejora y se enfrenta a desafíos más difíciles.

2.2 Temporización

En el presente apartado de temporización, se detallará la distribución temporal del Trabajo Fin de Grado (TFG) en seis fases clave que abarcan desde febrero hasta junio. Estas fases han sido cuidadosamente diseñadas para asegurar un avance progresivo y eficiente del proyecto, permitiendo alcanzar los objetivos establecidos de manera sistemática y oportuna. La estructuración de las fases proporcionará una visión clara y organizada del desarrollo del proyecto, facilitando su ejecución exitosa y el cumplimiento de los plazos establecidos. La planificación temporal exacta dividida a lo largo de los meses se puede ver en la Fig 2.1

Las 6 grandes fases son las siguientes:

- **Investigación y creación del entorno**: Una primera fase que incluirá los aspectos de formación en los campos a tratar y en las herramientas a utilizar, además de la creación del entorno de trabajo y la instalación de las librerías necesarias.

Figura 2.1: Esquema visual de la temporización del proyecto.

- **Creación de la Base de Datos**: La fase donde nos dedicaremos a reunir todo el conjunto de estados que formarán nuestra base de datos y con la que podremos entrenar a la IA.

- **Codificación del programa a ejecutar**: Creación de los archivos de código necesarios para poder ejecutar nuestro entorno de experimentación.

- **Experimentación**: Fase de ejecución de los diversos experimentos que necesitaremos para poder entrenar nuestra IA y donde obtendremos resultados que se analizarán en la fase siguiente.

- **Análisis de los resultados**: En esta fase, definiremos métricas para poder evaluar que tan buenos han sido nuestros experimentos, y donde podremos obtener conclusiones sobre el trabajo realizado.

- **Documentación del proyecto**: Fase de recopilación de toda la información generada a lo largo del proyecto, y su posterior documentación en la memoria del Trabajo Fin de Grado (TFG).

Como se puede observar en la Fig 2.1, hay dos momentos de solapamiento. Esto es debido a que las fases de codificación del programa y de documentación no son directamente dependientes de la finalización las fases de creación de la base de datos y de análisis de los resultados. De esta manera, podemos aprovechar esto como una ventaja temporal, y dedicar más horas a las iteraciones que correspondan a esas semanas, avanzando varias fases de manera paralela y aprovechando de manera más eficiente el tiempo.

2.3 Metodología

La metodología que se utilizará en este proyecto será una modificación de una planificación ágil con incrementos semanales realizados por una sola persona, ya que creemos que puede ser un enfoque efectivo para llevar a cabo el proyecto de manera organizada y progresiva. Estos incrementos irán acompañados de reuniones tutorizadas por dos personas interesadas, con el objetivo de recibir recibir *feedback* sobre las mejoras (o desmejoras) que se han producido en el proyecto. Dichas reuniones, se llevarán a cabo cada lunes, se presentará el trabajo hecho a lo largo de la iteración y se establecerá un nuevo objetivo para desarrollar hasta la siguiente reunión.

En este tipo de planificación, se establecerán metas y objetivos claros para cada semana o iteración, dividiendo el trabajo en incrementos manejables. Cada incremento semanal se enfocará en un aspecto específico del proyecto, como podría ser la implementación de un algoritmo de aprendizaje por refuerzo, el acceso a variables dentro del juego o la integración con él.

Este enfoque permite tener una visión general del proyecto desde el principio, al mismo tiempo que permite realizar ajustes y mejoras en cada iteración. Además, la temporización semanal proporciona un ritmo constante y ayuda a mantener el progreso y la motivación a lo largo del trabajo, ya que puedes ver avances (en mayor o menor medida) en el proyecto con gran asiduidad.

Es importante tener en cuenta que la temporización puede variar, dependiendo de la complejidad de la iteración específica y de los recursos disponibles, pudiendo aumentar así algunas hasta dos o incluso tres semanas en las iteraciones más complejas o extensas. Es fundamental asignar tiempo suficiente para la investigación, el diseño, la implementación y las pruebas, y ser flexible para adaptarse a posibles desafíos y contratiempos. De esta manera, estaremos preparados para poder hacer frente a todos los impedimentos que pueden llegar a surgir en un proyecto de estas dimensiones.

Con todo esto, se fomenta la iteración constante, la retroalimentación y la adaptabilidad, lo que puede ayudar a garantizar un desarrollo continuo y un resultado final exitoso del Trabajo Fin de Grado (TFG).

2.4 Entorno Tecnológico

Para poder llevar a cabo este TFG, necesitamos montar un entorno tecnológico específico antes de estar plenamente preparados para empezar con las fases de desarrollo propiamente dichas.

En una primera recopilación de necesidades a grandes rasgos, podemos darnos cuenta que habrá algunas herramientas que nos serán obligadas, como la elección de un lenguaje de programación, junto a su correspondiente Entorno de Desarrollo Integrado (IDE) y a una o varias librerías para ayudarnos con el código. El lenguaje elegido

será Python, debido a que para esta tarea, la librería que más se adapta a nuestras necesidades es Gym Retro [2], la cuál está diseñada para Python. Con esto, junto a otras librerías, dispondremos de una buena base.

Una vez establecida la base, se puede entrar más en detalle en el resto de aspectos necesarios.

2.4.1 MiniConda

Miniconda es una distribución ligera de Conda. Conda es un sistema de gestión de paquetes de software y un entorno de virtualización ampliamente utilizado en el ámbito de la ciencia de datos y la programación en Python. Proporciona un instalador mínimo de Conda que incluye solo el administrador de paquetes y las dependencias básicas, lo que permite una instalación más rápida y una mayor flexibilidad en la configuración del entorno [3].

Se utiliza principalmente para crear y administrar entornos virtuales de desarrollo de software. Un entorno virtual es un ambiente aislado que contiene su propio conjunto de bibliotecas y dependencias, lo que permite trabajar en proyectos con diferentes requisitos y versiones de paquetes sin interferencias entre ellos. Esto es especialmente útil cuando se trabaja en proyectos de ciencia de datos y aprendizaje automático, donde es común utilizar diferentes versiones de bibliotecas y paquetes.

Por tanto, el primer paso en la creación del entorno será crear un *Virtual environment* o entorno virtual, donde instalaremos las librerías Gym Retro y Stable Baselines junto a las dependencias que necesiten y una versión de Python compatible con ellas.

2.4.2 IDE

Un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) es una herramienta de software que proporciona a los desarrolladores un entorno completo para escribir, depurar y compilar código. Un IDE generalmente incluye un editor de código, un depurador, un compilador o intérprete, y herramientas adicionales para facilitar el desarrollo de software. Proporciona un flujo de trabajo eficiente al agrupar todas estas herramientas en una sola interfaz, lo que permite a los programadores trabajar de manera más productiva y colaborativa en sus proyectos.

PyCharm es el elegido para este proyecto, debido a que es un IDE específicamente diseñado para el desarrollo de aplicaciones en el lenguaje de programación Python. Desarrollado por JetBrains, ofrece una amplia gama de características y herramientas especializadas para ayudar a escribir, depurar y mantener el código de manera eficiente. Además de su potente editor de código con resaltado de sintaxis y autocompletado inteligente, PyCharm ofrece características avanzadas como el análisis estático del código, la depuración interactiva, el soporte para control de versiones y la integración con frameworks muy diversos y útiles. En líneas generales, podemos decir que su enfoque se basa en la productividad y la calidad del código.

Como además necesitaremos escribir o manipular archivos que no sean necesariamente archivos Python, utilizaremos también Visual Studio Code (VSC). Se trata de un IDE altamente popular y versátil desarrollado por Microsoft. Aunque comparte el nombre con la familia de productos Visual Studio, VSC es una aplicación independiente y ligera que se centra en ofrecer una experiencia de desarrollo eficiente y personalizable. Con una amplia gama de extensiones y complementos disponibles, permite trabajar con varios lenguajes de programación, proporcionando un potente editor de código con características como resaltado de sintaxis, autocompletado inteligente, navegación de código y refactorización. Además, ofrece integración con sistemas de control de versiones, depuración interactiva, terminal integrada y una amplia gama de herramientas y servicios que facilitan la creación y el mantenimiento de aplicaciones en diversos entornos de desarrollo.

2.4.3 R

R es un lenguaje de programación ampliamente utilizado en el ámbito del análisis de datos y la estadística y se caracteriza por su enfoque en la manipulación, análisis y visualización de datos.

Una de sus principales fortalezas es su extensa colección de paquetes y librerías, que proporcionan una amplia gama de funcionalidades para el análisis de datos. Estos paquetes son desarrollados tanto por la comunidad de usuarios de R como por instituciones académicas y empresas, lo que brinda una gran variedad de herramientas especializadas y técnicas avanzadas.

Destaca por su capacidad para realizar análisis estadísticos complejos, como regresión lineal, análisis de supervivencia, análisis factorial y muchos otros. Además, ofrece una amplia gama de herramientas de visualización, lo que facilita la creación de gráficos y representaciones visuales de datos.

Otra ventaja es su interoperabilidad con otros lenguajes de programación, lo que permite combinarlo con Python, Java, C++ y otros para aprovechar sus respectivas fortalezas. Además, R cuenta con una comunidad activa de usuarios y un amplio soporte en línea, lo que facilita el aprendizaje y la resolución de problemas.

2.4.4 Librerías

Las librerías, también conocidas como bibliotecas, son conjuntos de código predefinido que se pueden añadir a los lenguajes de programación para extender sus funcionalidades y facilitar el desarrollo de aplicaciones. Estas librerías contienen funciones, clases y métodos que abordan tareas específicas y comunes, lo cual nos ahorrará tiempo y esfuerzo al no tener que escribir todo el código desde cero.

Las librerías pueden abarcar una amplia variedad de áreas, desde el procesamiento de imágenes y el análisis de datos hasta el desarrollo web y el aprendizaje automático. Al

Figura 2.2: Representación artística del aprendizaje por refuerzo [4].

añadir una librería a un lenguaje de programación, se puede aprovechar su funcionalidad y utilizarla en proyectos de manera sencilla. Existen muchas utilizadas de manera asidua por la comunidad de desarrolladores de software mundial como por ejemplo: NumPy, Pandas, TensorFlow, Pytorch, etc.

Por lo que respecta a este proyecto, las más interesantes serás dos: GymRetro y Baselines (concretamente su versión estable), ya que ambas ofrecen funcionalidades muy interesantes para los campos que se van a tratar, y están pensadas para utilizarse (sobretodo) de forma conjunta.

La librería Gym Retro, es una herramienta con un gran potencial que permite a los investigadores y desarrolladores de IA explorar y entrenar agentes de RL en entornos de videojuegos clásicos. Gym Retro se basa en la popular librería Gym de OpenAI y proporciona una interfaz sencilla y unificada para interactuar con una amplia colección de juegos retro. Esta librería permite cargar y emular Read Only Memories (ROMs) de juegos clásicos, lo que nos facilitará experimentar y desarrollar algoritmos de aprendizaje en entornos bien establecidos. Podremos aplicar técnicas avanzadas de RL y probar diferentes estrategias en nuestro juego, lo que resultará de gran ayuda, sobretodo en las fases de pruebas preliminares. Además, Gym Retro también ofrece herramientas para la evaluación de rendimiento y la comparación de algoritmos, lo que lo convierte en una herramienta idónea para nosotros.

Irá fuertemente ligada a la librería Baselines (basada también en OpenAI Gym), ya que esta se encuentra diseñada para simplificar el desarrollo y la implementación de algoritmos de aprendizaje por refuerzo en entornos de simulación. Dispone de una colección de algoritmos de aprendizaje por refuerzo clásicos y de vanguardia, que son conocidos por su estabilidad y buen rendimiento.

2.5 Conceptos previos

A partir de ahora, ahondaremos mucho más en el desarrollo propiamente dicho de este proyecto, lo cuál requerirá de forma obligada el uso de vocabulario y terminología técnica del mundo de la Inteligencia Artificial (IA) y el Reinforcement Learning (RL). Para facilitar la familiarización con este mundo de la informática y sus tecnicismos se explicarán algunos de los conceptos más importantes a continuación.

2.5.1 Inteligencia Artificial

La IA es un campo de estudio que se enfoca en el desarrollo de sistemas y programas capaces de realizar tareas que requieren de inteligencia humana. Se basa en el concepto de que las máquinas pueden aprender de la experiencia y ajustar su comportamiento en función de esa experiencia.

Existen diferentes enfoques en este campo. Uno de ellos es el aprendizaje automático (Machine Learning), que permite a las máquinas aprender a través de datos y algoritmos sin ser programadas explícitamente para cada tarea. El aprendizaje automático se divide en dos categorías principales: el aprendizaje supervisado, donde se proporcionan ejemplos etiquetados para entrenar a los modelos, y el aprendizaje no supervisado, donde los modelos descubren patrones o estructuras en los datos sin etiquetas.

Otro enfoque importante es el aprendizaje profundo (Deep Learning), que es una rama del aprendizaje automático que utiliza redes neuronales artificiales de múltiples capas. El aprendizaje profundo ha logrado avances significativos en áreas como el reconocimiento de imágenes, el procesamiento del lenguaje natural y la visión por computador.

La Inteligencia Artificial (IA) se utiliza en una amplia gama de aplicaciones, incluyendo asistentes virtuales, sistemas de recomendación, reconocimiento de voz, vehículos autónomos, diagnósticos médicos, análisis de datos, juegos, entre otros. Estas aplicaciones se benefician del procesamiento masivo de datos y de la capacidad de las máquinas para identificar patrones y tomar decisiones basadas en ellos.

Sin embargo, esta también plantea desafíos y preguntas éticas, como la privacidad de los datos, la transparencia en la toma de decisiones, la responsabilidad de los sistemas autónomos y el impacto en el empleo. Es fundamental abordar estos aspectos para garantizar que la Inteligencia Artificial (IA) se desarrolle y utilice de manera responsable y ética.

2.5.2 Aprendizaje Automático

Para afrontar el problema de crear una IA que aprenda a pelear en Pokémon Rojo Fuego, hemos optado por utilizar Machine Learning (ML), más concretamente Deep

Reinforcement Learning (DRL) (el cuál será explicado más adelante). Esto es debido, al gran número de posibilidades que existe dentro de un combate Pokémon (como se puede observar en la Fig 2.3), lo cuál nos hace inviable un enfoque diferente al de la generalización mediante aprendizaje por refuerzo profundo.

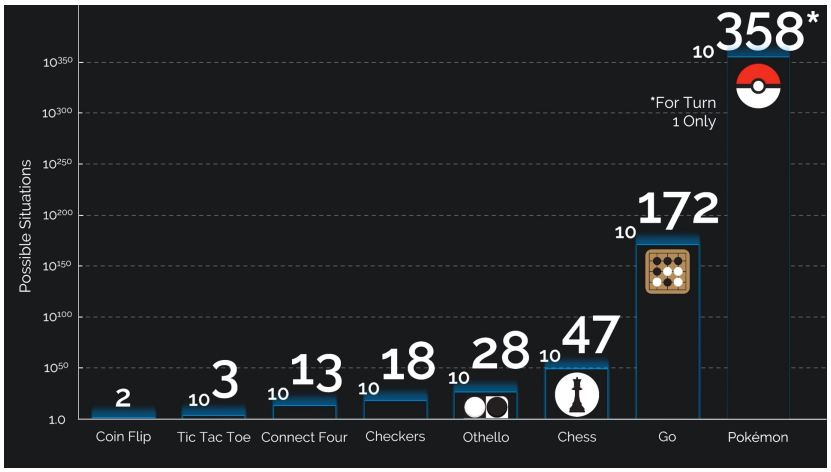


Figura 2.3: Comparativa posibilidades en diferentes juegos [5].

El Machine Learning (ML), es una rama de la Inteligencia Artificial (IA) que se enfoca en desarrollar algoritmos y modelos que permiten a las máquinas aprender de los datos y mejorar su rendimiento en una tarea específica sin ser programadas explícitamente.

El proceso de aprendizaje automático se basa en la idea de que las máquinas pueden analizar patrones y características en los datos para extraer información relevante y tomar decisiones o hacer predicciones. Para lograr esto, se utilizan algoritmos y modelos que son entrenados con un conjunto de datos de entrenamiento, que consiste en ejemplos etiquetados o no.

En el aprendizaje supervisado, el conjunto de datos de entrenamiento incluye ejemplos etiquetados, donde se proporciona una respuesta o etiqueta esperada para cada instancia. El algoritmo aprende a partir de estos ejemplos para generar un modelo capaz de predecir etiquetas para nuevas instancias. Ejemplos comunes de algoritmos de aprendizaje supervisado incluyen las Super Vector Machine (SVM), los árboles de decisión, la regresión lineal y los clasificadores de Naive Bayes.

En el aprendizaje no supervisado, el conjunto de datos de entrenamiento no incluye etiquetas. En cambio, el algoritmo busca patrones y estructuras en los datos, como agrupamientos o dimensiones latentes, para organizar y comprender la información.

2.5.3 Aprendizaje por refuerzo

El aprendizaje por refuerzo o RL es un enfoque del aprendizaje automático en el que un agente autónomo aprende a tomar decisiones óptimas en un entorno desconocido a través de la interacción y retroalimentación continua. Se basa en el concepto de recompensa y castigo, donde el agente recibe una señal de recompensa o penalización según sus acciones en el entorno.

El proceso de aprendizaje por refuerzo implica tres elementos clave: el agente, el entorno y la política. El agente es el sistema de inteligencia artificial que aprende y toma decisiones. El entorno es el contexto en el que el agente actúa y con el que interactúa. La política es la estrategia que el agente sigue para determinar sus acciones en función de las observaciones del entorno.

El agente explora el entorno realizando acciones y recibe una señal de recompensa en función de su desempeño. El objetivo del agente es aprender una política óptima que maximice la acumulación de recompensas a largo plazo. Para lograr esto, el agente utiliza algoritmos de aprendizaje por refuerzo que ajustan su comportamiento en función de la retroalimentación recibida.

Se basa en el concepto de "prueba y error". El agente toma acciones, observa los resultados y ajusta su comportamiento para mejorar las recompensas obtenidas. A través de múltiples iteraciones, el agente aprende a tomar decisiones cada vez más óptimas en diferentes situaciones del entorno.

Todo el conocimiento se deposita en una tabla (ver la parte de arriba de la Fig 2.6), se utiliza el término "tabla" para referirse a una matriz de datos, donde cada fila representa una muestra de entrenamiento y cada columna corresponde a una característica o variable.

El aprendizaje por refuerzo se ha aplicado en una amplia variedad de campos, como la robótica, los juegos, la optimización de recursos, la toma de decisiones y muchas otras áreas donde la interacción con el entorno es fundamental.

2.5.4 Redes Neuronales

Las redes neuronales son modelos matemáticos y computacionales inspirados y basadas en el funcionamiento del cerebro humano. Consisten en una colección interconectada de nodos llamados neuronas artificiales o unidades, que trabajan en conjunto para procesar y analizar datos.

La estructura básica de una red neuronal está compuesta por capas de neuronas, que se dividen en tres tipos principales: capa de entrada (input layer), capas ocultas (hidden layers) y capa de salida (output layer). Cada neurona está conectada a las neuronas de la capa anterior y a las de la capa siguiente, formando una red de conexiones.

Cada conexión entre las neuronas tiene un peso asociado (weight), que representa la importancia de esa conexión en el cálculo del resultado final. Durante el entrenamiento de la red neuronal, estos pesos se ajustan gradualmente para optimizar el rendimiento y la capacidad de generalización del modelo.

La información fluye a través de la red neuronal de manera secuencial. Los datos de entrada se propagan hacia adelante a través de las capas ocultas, donde se realiza el procesamiento mediante una función de activación. Finalmente, se obtiene una salida en la capa de salida, que puede ser una clasificación, una predicción numérica u otra tarea dependiendo del objetivo.

El aprendizaje de estas se realiza a través de algoritmos de aprendizaje, como podría ser el descenso del gradiente, que ajustan los pesos de las conexiones para minimizar la diferencia entre las salidas esperadas y las salidas reales de la red.

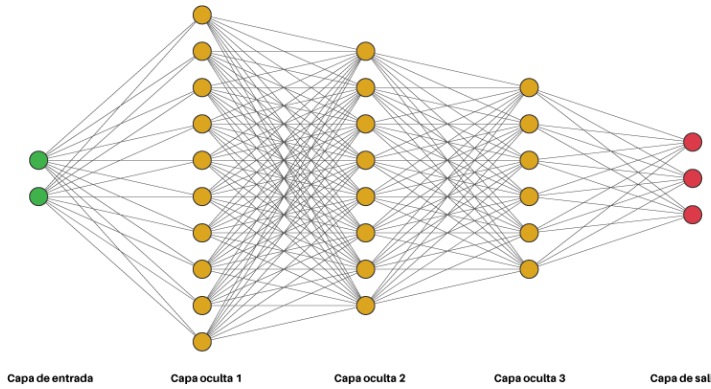


Figura 2.4: Representación de una red neuronal con 3 capas ocultas [6].

Las redes neuronales se utilizan en una amplia gama de aplicaciones, incluyendo reconocimiento de imágenes, procesamiento del lenguaje natural, visión por computador, robótica, juegos, análisis de datos y mucho más. Su capacidad para aprender y reconocer patrones complejos en los datos ha llevado a importantes avances en muchas áreas y ha impulsado el desarrollo de la inteligencia artificial [7].

2.5.5 Redes Convolucionales

Las redes convolucionales, también conocidas como Convolutional Neural Network (CNN), son un tipo especializado de redes neuronales diseñadas específicamente para el procesamiento eficiente de datos estructurados, como imágenes y señales.

Su principal característica distintiva es su capacidad para capturar y aprender características relevantes a través de filtros convolucionales y operaciones de pooling. Estas operaciones permiten que la red extraiga y combine características [8].

La capa convolucional es el componente clave de las redes convolucionales. Consiste en un conjunto de filtros o kernels que se aplican a la entrada de datos para realizar la operación de convolución. Cada filtro se desliza sobre la entrada, realizando multiplicaciones y sumas ponderadas para producir una respuesta convolucionada. Estas respuestas convolucionadas se utilizan para detectar características específicas, como bordes, texturas o formas, en diferentes regiones de la imagen.

Después de la capa convolucional, se aplica una capa de pooling para reducir la dimensionalidad de la representación y generar una representación más compacta. La operación de pooling reduce el tamaño espacial de la salida al seleccionar el valor máximo (pooling máximo) o promedio (pooling promedio) dentro de una región local. Esto proporciona invariancia espacial y reduce la sensibilidad a pequeñas variaciones en la posición de las características.

Las redes convolucionales también pueden tener capas totalmente conectadas al final de la arquitectura para realizar tareas de clasificación, detección de objetos, segmentación semántica u otras tareas específicas. Estas capas se encargan de transformar la representación aprendida en una salida final, utilizando diferentes técnicas para producir probabilidades o valores de clasificación.

Una ventaja significativa y la razón por la que hemos hecho elección de estas es su capacidad para aprender automáticamente características relevantes de los datos, eliminando así la necesidad de diseñar manualmente características a mano. De esta manera, nos evitamos la extracción de características hecha por nosotros mismos.

Figura 2.5: Representación de la entrada de la red convolucional.

Además, otra de la razones de su elección ha sido que las redes convolucionales se benefician del uso de técnicas de aprendizaje profundo, como el entrenamiento con grandes conjuntos de datos y el uso de arquitecturas más profundas. Estas técnicas permiten aprender representaciones más complejas y lograr un rendimiento superior

en diversas tareas de visión por computadora.

2.5.6 Aprendizaje por refuerzo profundo

Sin embargo, para cumplir nuestro objetivo, debido a la gran cantidad de posibilidades que encontramos en nuestro juego, el RL y su matriz de datos no es algo factible, por tanto utilizaremos DRL.

El aprendizaje por refuerzo profundo es una extensión del aprendizaje por refuerzo que utiliza redes neuronales profundas para representar y aproximar la función de valor o la política del agente (ver la parte de abajo de la Fig 2.6). En lugar de depender de una representación manual de las características o variables del entorno, utiliza redes neuronales para aprender automáticamente las representaciones relevantes del entorno. Esto permite que el agente aprenda de manera más eficiente y generalice mejor a situaciones nuevas y diferentes [9]. Algunas de las arquitecturas más comunes utilizadas en el aprendizaje por refuerzo profundo son las Convolutional Neural Network (CNN).

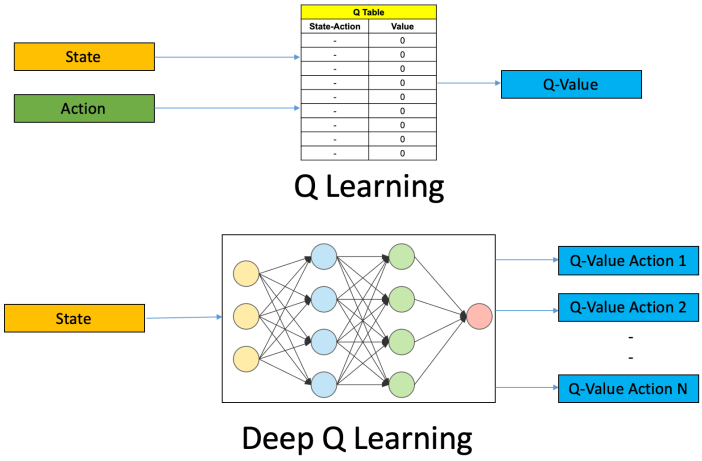


Figura 2.6: Reinforcement Learning (RL) vs Deep Reinforcement Learning (DRL) [10].

En conclusión, el análisis detallado realizado en este apartado ha permitido comprender y abordar de manera integral los requisitos, la metodología, la temporización y el entorno tecnológico necesarios para el desarrollo del Trabajo Fin de Grado (TFG). Se ha establecido una variación de una planificación ágil con incrementos semanales, lo que ha facilitado la gestión efectiva del tiempo y los recursos, y ha proporcionado una visión clara y organizada del progreso del proyecto. Además, se ha creado un entorno tecnológico plenamente capacitado del desarrollo del proyecto. Todo esto, nos deja el camino despejado para poder adentrarnos en la sección de desarrollo, donde se expondrá como se ha creado la base de datos, junto al entorno de interacción entre la IA y el videojuego y cuáles han sido las redes neuronales utilizadas.

CAPÍTULO 3

DESARROLLO

En el presente apartado de Desarrollo, se detallará el proceso de creación de la base de datos de combates, una parte fundamental, haciendo especial hincapié en el proceso de recopilación y en los criterios utilizados. Se especificarán también las redes neuronales utilizadas para el entrenamiento y aprendizaje de la IA. Finalmente, se explicará la construcción del entorno del proyecto para la interacción entre la IA y el videojuego, esto implica la configuración de herramientas y bibliotecas que permitan la comunicación y el control del juego.

3.1 Base de datos

Describiremos en detalle el proceso de creación de la base de datos utilizada en el desarrollo de este proyecto. La base de datos juega un papel fundamental, ya que proporciona los datos necesarios para entrenar y poder evaluar la IA en su capacidad para jugar a Pokémon Rojo Fuego. En esta sección, se explicará cómo se ha recopilado, organizado y preparado la información de los combates y garantizando su calidad y representatividad. Además, se abordarán los criterios utilizados para la selección de los datos, la metodología de adquisición y los desafíos encontrados durante este proceso. El análisis detallado de la creación de la base de datos permitirá comprender la fundamentación y la calidad de los datos utilizados en el entrenamiento y evaluación de la IA, sentando así las bases para el éxito del proyecto.

3.1.1 GymRetro Integration Tool

La herramienta de integración de Gym Retro es una utilidad desarrollada por OpenAI que facilita la integración y el uso de la biblioteca Gym Retro en entornos de desarrollo.

Nos permite interactuar con los juegos, capturar estados de juego, tomar acciones y recibir recompensas. Además, proporciona una interfaz coherente y fácil de usar para

cargar y gestionar los juegos retro, así como herramientas adicionales para la visualización y el análisis de los datos generados durante el entrenamiento.

Utiliza como base para poder correr las ROMs de Game Boy Advance (GBA) un emulador de dicha consola, por tanto será necesario descargarlo a parte.

3.1.2 Combate/Estado

Los estados de juego son representaciones instantáneas del estado actual de un juego en un momento dado. En el contexto de los videojuegos, un estado de juego captura toda la información relevante necesaria para describir y determinar la situación actual en la que se encuentra un jugador y el entorno del juego.

Son fundamentales en el desarrollo de algoritmos de Inteligencia Artificial (IA) y en el aprendizaje por refuerzo, ya que proporcionan la entrada necesaria para que los agentes tomen decisiones y aprendan a partir de ellas.

La base de datos, estará compuesta por todos los combates encontrados desde el inicio del juego hasta el cuarto líder de gimnasio, lo cuál nos brindará un total de 98 combates o estados diferentes.

Se entiende por combate o estado, el primer momento de la pelea donde se nos permite comenzar a realizar acciones. Teniendo así el combate entero por disputar y suprimiendo cualquier posible diálogo precombate entre los dos combatientes, centrándonos en que el estado que le quedará a la IA sea íntegramente jugable.

En estos combates, mantendremos el orden del equipo Pokémon siempre igual y todos con la vida al máximo, de esta manera, los combates tendrán siempre las mismas condiciones de inicio por parte nuestra, homogeneizando las variables que dependen de nuestro equipo Pokémon y facilitando este apartado a nuestra IA.

3.1.3 Proceso de recopilación de datos

Para recopilar todos estos combates, no ha existido otra alternativa que jugar manualmente al juego y avanzar en él como si fuera una partida normal. Buscando pelear con todos los entrenadores que encontremos durante el transcurso de la aventura, sin alterar el orden del equipo Pokémon y restaurando la vida de todos ellos después de cada combate.

Cada vez que se iniciaba el primer momento jugable dentro de un combate, creábamos un estado con la herramienta de integración. De esta manera se podía volver exactamente a ese punto del juego en cualquier momento.

Una vez llegados hasta el cuarto líder de gimnasio y acumulando un total de 98 combates diferentes en la base de datos, se dio por completada la fase de recopilación de los combates.

3.1.4 Criterios para la selección

Para la selección del conjunto de combates, es muy importante tener en cuenta que necesitaremos combates contra una gran variedad de entrenadores, puesto que debemos intentar cubrir peleas de todo tipo: contra equipos de pocos y de muchos Pokémon, contra muchos tipos diferentes de Pokémon, contra muchos rangos de niveles de Pokémon diferentes y contra entrenadores más duros o más débiles.

En nuestra base de datos, encontramos combates que cumplen todas estas características, puesto que en el intervalo de juego que hemos jugado y durante el cual hemos recopilado todos los combates, incluimos los inicios del juego que funcionan como una fase introductoria, con Pokémon débiles, diferentes combates contra el antagonista de la historia principal el cuál siempre va a la par con tus niveles de Pokémon y tiene un equipo competitivo, 4 Líderes de gimnasio los cuáles también son siempre bastante competitivos y tienen diferentes tipos de Pokémon, además de todos los combates intermedios que nos podemos encontrar mientras avanzamos en el juego, los cuales acostumbran a ser de una dificultad media.

Entrenador	Bulbasaur [1]	Squirtle [2]	Charmander [3]
	 Planta Veneno	 Agua	 Fuego
Rival	Nivel 5		

Figura 3.1: Ejemplo de un combate en la base de datos con Pokémon de diferentes tipos y bajo nivel [11].

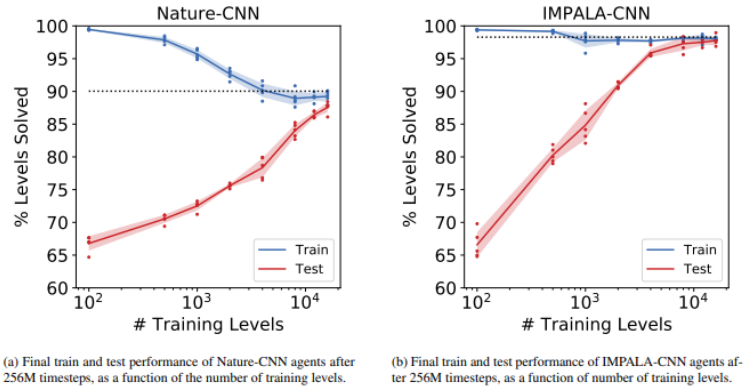
Entrenador	Koffing	Muk	Koffing	Weezing
	 Veneno	 Veneno	 Veneno	 Veneno
Líder Koga	Nivel 37	Nivel 39	Nivel 37	Nivel 43

Figura 3.2: Ejemplo de un combate en la base de datos con mayor número de Pókemon, todos del mismo tipo y de mayor nuvel [11].

3.2 Redes Utilizadas

Como ya hemos discutido anteriormente, este proyecto necesitaremos hacer uso del Deep Reinforcement Learning (DRL) y de una Convolutional Neural Network (CNN). Pero para mejorar el experimento, no se utilizará solo una red neuronal, si no que se utilizarán dos para comparar entre ellas cuál es la que obtiene un mejor rendimiento en nuestro juego.

Por tanto, las redes elegidas serán la red NATURE-CNN y la red IMPALA-CNN, la cuál ha demostrado obtener mejores resultados que la anterior cuando es aplicada a ciertos juegos utilizando DRL [12]. Esto, se puede observar en la figura Fig 3.3, donde a misma cantidad de entreno, la red IMPALA-CNN presenta mejoras significativas tanto en porcentaje de niveles completados, como en el número de entrenos necesarios para conseguir cierto porcentaje de niveles completados. Por tanto, compararemos ambas para ver si también hay mejoría en nuestro entorno de Pokémon.



(a) Final train and test performance of Nature-CNN agents after 256M timesteps, as a function of the number of training levels.

(b) Final train and test performance of IMPALA-CNN agents after 256M timesteps, as a function of number of training levels.

Figura 3.3: Comparativa de la red Nature-CNN vs la red IMPALA-CNN en el juego CoinRun [12].

3.2.1 NATURE-CNN

La arquitectura NATURE-CNN, se trata de una arquitectura específica de la red CNN. Fue presentada en un artículo titulado “Human-level control through deep reinforcement learning” publicado en la revista Nature en 2015 [13].

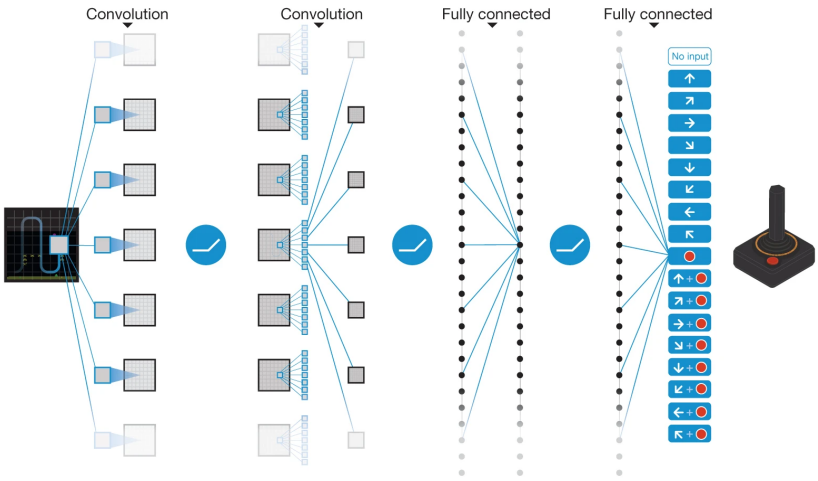


Figura 3.4: Ilustración esquemática de la red NATURE-CNN [13].

Esta arquitectura, fue diseñada específicamente para tareas de Deep Reinforcement Learning (DRL) desafiantes, entre ellos juegos de la consola Atari 2600. Fue diseñada para aprender a extraer conocimiento a partir de imágenes de entrada brutas.

Su rendimiento, comparada con los métodos de Reinforcement Learning (RL) más potentes de aquel momento fue abrumador. Probada en muchos juegos de la plataforma Atari 2600 y comparado su rendimiento contra los anteriores métodos, arrojó una gran ventaja a favor esta red convolucional, la cuál presentó mejores marcas en casi todos los juegos, con una diferencia notable en comparación a sus métodos predecesores.

De modo que, esta red será ideal de utilizar en nuestro entorno, debido al gran rendimiento que ha demostrado tener en videojuegos.

3.2.2 IMPALA-CNN

La arquitectura IMPALA-CNN (Importance Weighted Actor-Learner Architecture) es una arquitectura de la red Convolutional Neural Network (CNN) propuesta en el artículo “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures” en 2018 [14].

Utiliza una arquitectura de red neuronal llamada arquitectura *actor-learner*, donde hay un *actor* que interactúa con el entorno de juego y un *learner* que aprende de las experiencias de los actores.

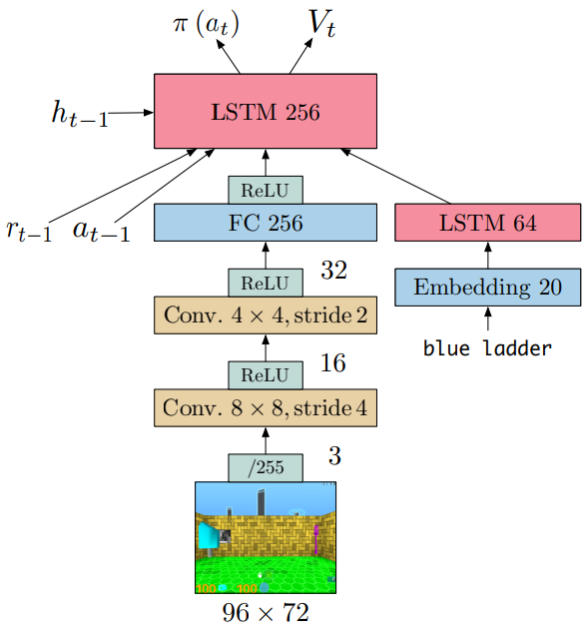


Figura 3.5: Arquitectura del modelo de red IMPALA-CNN [14].

La red IMPALA-CNN emplea una combinación de capas convolucionales y capas completamente conectadas para procesar las entradas del juego, que pueden ser imágenes de píxeles o características relevantes. Las capas convolucionales extraen características espaciales y patrones de las imágenes de entrada, mientras que las capas completamente conectadas procesan estas características y generan las acciones a tomar.

Ambas redes fueron probadas y comparadas en el entorno CoinRun, obteniendo la red IMPALA-CNN mejores resultados, como se puede oberservar en la Fig 3.3 [13]. Por tanto, también será una buena elección debido a su gran desempeño en videojuegos.

3.3 Construcción del entorno

Durante el apartado de construcción del entorno de este Trabajo Fin de Grado (TFG), se abordará la gestión del proyecto de Python y la integración de las librerías necesarias para crear un entorno funcional. El objetivo principal es establecer un ambiente adecuado que permita desarrollar y ejecutar el sistema de Inteligencia Artificial (IA) basado en Deep Reinforcement Learning (DRL) para jugar a Pokemon Rojo Fuego. Esto implica la instalación y configuración de las herramientas y bibliotecas pertinentes, así como la organización y estructuración del proyecto para garantizar un flujo de trabajo efectivo y eficiente. A lo largo de este apartado, se describirán los pasos y decisiones tomadas para construir un entorno sólido y optimizado.

3.3.1 Estructura archivos Python

Una vez instaladas todas las librerías necesarias, podemos iniciar el proyecto en Python. Este contendrá dos carpetas: *Baselines* con los archivos necesarios para poder aplicar algoritmos de Reinforcement Learning (RL) y Deep Reinforcement Learning (DRL), *Custom Integrations* de la cual hablaremos en la siguiente sección y 6 archivos. La base de estos archivos se puede encontrar en un repositorio de *scripts* para juegos retro de uso libre pensados para Gym Retro y Stable Baselines [15].

Utilizando como base parte de esos scripts y haciendo algunas modificaciones, podremos preparar el archivo que utilizaremos para entrenar nuestra IA, que será *PokemonRF_trainer.py*. Al ejecutar este *Script*, deberemos indicarle por parámetro algunos argumentos, como qué red neuronal utilizar, en qué estados entrenar, qué algoritmo utilizar, qué política...

Ejecutando en la consola este archivo iniciaríamos el entreno y la validación.

3.3.2 Integración del juego

Para la integración del juego debemos conseguir la Read Only Memory (ROM) de Pokémon Rojo Fuego y colocarla en la carpeta de *Custom Integrations*.

Junto a eso, debemos incluir nuestra base de datos (los combates), para que también sean accesibles y una serie de archivos que debemos escribir para poder darle ciertas instrucciones a las librerías.

El primero de todos será el archivo que contendrá datos de variables. Aquí, debemos indicar una a una las direcciones y los tipos de las variables de la memoria Random Access Memory (RAM) del juego que vamos a necesitar para poder crear las condiciones de *reward* (recompensa) y de *done* (finalizado) para poder utilizar nuestros algoritmos de Deep Reinforcement Learning (DRL). Estas direcciones estarán en formato hexadecimal y las extraeremos con un mapeo de la memoria RAM [16] [17].

El siguiente archivo a crear será el archivo de metadatos, el cual simplemente definirá cuál será el estado (combate) que se abrirá por defecto en caso de que no indiquemos ninguno al ejecutar el entorno.

El archivo de origen Lua será donde debemos crear la función que determinará en base a las variables introducidas en el archivo de datos, cuándo se cumple o no la condición de *done*.

Finalmente en el archivo de escenario indicaremos si queremos recortar la imágen que será la entrada de nuestra red neuronal (en nuestro caso no, ya que queremos utilizar la imágen bruta de entrada), en que archivo se encuentra la función que determina el valor de *done* y en base a qué variables se otorgan *reward* o *penalty* (castigo).

3.3.3 Optimizaciones

La librería Baselines, dejó de recibir soporte hace algún tiempo, debido a que el proyecto se migró hacia versiones posteriores las cuales nosotros no podemos utilizar por compatibilidades entre librerías y versiones. La versión que tenemos que utilizar presenta un *bug* en uno de los archivos. Debemos poner el escalado en *False*, debido a que en los modelos que utilizaremos, la imagen de entrada ya se normaliza utilizando la Graphics Processing Unit (GPU) (lo cuál es mucho más eficiente). Si no arreglamos esto, la imagen será normalizada dos veces, lo cual trae dos problemas principales: errores en la precisión del modelo y su entreno y errores de rendimiento. Una vez hemos hecho el *bugfix*, nuestros entornos correrán el juego (aproximadamente) al doble de Frames Per Second (FPS) de lo que correrían sin el arreglo.

En el apartado de desarrollo hemos abordado de manera detallada la generación de la base de datos de combates, la selección de las redes neuronales utilizadas, así como la construcción del entorno necesario para llevar a cabo este proyecto. La creación de la base de datos ha sido crucial para obtener una amplia variedad de situaciones de juego para poder entrenar el modelo de IA de manera efectiva. La elección de las redes neuronales, como son la red NATURE-CNN y la red IMPALA-CNN, es adecuada para el objetivo del proyecto. Y finalmente, la construcción del entorno nos ha permitido tener creada la forma de conectar los diferentes algoritmos y redes con el juego. En conjunto, estas etapas de desarrollo han sentado las bases para para poder empezar con el apartado de experimentación, acercándonos cada vez más a un agente capaz de jugar y ganar en este desafiante juego de manera autónoma.

CAPÍTULO 4

EXPERIMENTACIÓN

Para poder llevar a cabo la experimentación, se hará uso de todo el entorno de desarrollo construido, para poder crear los agentes de IA y hacer que aprendan a jugar a Pokémon Rojo Fuego. En esta etapa, se establecerán las métricas específicas para más adelante poder medir el rendimiento y el progreso del agente. Además, se definirán y llevarán a cabo una serie de experimentos rigurosos y controlados para poder analizar en profundidad el comportamiento y las habilidades adquiridas por las IA. Estos experimentos estarán diseñados de manera sistemática, con condiciones predefinidas y metodologías establecidas, permitiendo una comparación precisa y una comprensión más profunda de la eficacia del enfoque de aprendizaje por refuerzo utilizado. A través de esta experimentación detallada, estaremos preparados para extraer conclusiones significativas y se proporcionará una evaluación completa del desempeño y las capacidades de los agentes en el contexto del juego Pokémon Rojo Fuego.

4.1 Métricas

Una métrica, se refiere a una medida cuantitativa utilizada para evaluar o cuantificar un aspecto específico de un sistema, proceso, evento o fenómeno. En el contexto de la ciencia, la tecnología o el análisis de datos se utilizan para obtener una comprensión objetiva y medible del rendimiento, la eficiencia, la calidad o cualquier otro criterio relevante de evaluación. Proporcionan una forma de cuantificar resultados o características observables, lo que permite realizar comparaciones, tomar decisiones informadas y evaluar el progreso hacia un objetivo deseado. Se pueden expresar en términos de números, proporciones, porcentajes, puntuaciones, tiempos, distancias u otras unidades de medida apropiadas para el dominio específico.

Por tanto, lo primero que deberemos hacer es establecer una o diversas métricas que nos vayan a servir más adelante para poder realizar las comparaciones pertinentes entre las dos redes neuronales que vamos a utilizar.

Aquí podemos encontrar mucha información, principalmente el tiempo, los resultados finales de la vida de cada Pokémon una vez acabado el combate, la recompensa total acumulada y si el combate se ha ganado o no. Cuando tenemos en cuenta estos datos, lo primero que puede venir a la mente es utilizar el número de combates que ha ganado cada red. Esto parece una buena métrica, aunque por si sola algo escueta, así que la acompañaremos junto al total de vidas de cada Pokémon una vez finalizado el combate.

Utilizando estas dos métricas, podremos comparar cuánto gana cada red y cómo de bien gana, basándonos en las vidas restantes de su equipo Pokémon en comparación con la otra. De esta manera en combates que ambas hayan ganado, miraríamos cuál es la que ha presentado mejor rendimiento en base a cuánta vida tiene su equipo todavía (significando mayor vida restante, mayor rendimiento).

4.2 Hardware utilizado

Sabiendo que el proceso de experimentación consistirá en entrenar dos redes neuronales diferentes en un entorno con muchas posibilidades y dónde utilizamos de entrada para la red neuronal la imagen bruta del juego directamente, podemos suponer con bastante certeza que el proceso de entreno consumirá bastante tiempo. Es por ello que dividiremos el entreno de cada red neuronal en 4 ordenadores diferentes para acelerar el proceso.

Por tanto, dividiremos el conjunto total de combates de la base de datos entre los 4 ordenadores. Como 98 no es divisible entre 4 de manera que a cada ordenador se le asigne el mismo número de combates, asignaremos 35 a dos de ellos, 16 a otro y 12 al último. Estas asignaciones se hacen siguiendo unos criterios que se explicarán a continuación.

El hardware utilizado y su asignación de combates será la siguiente:

- **Ordenador 1**: Este ordenador será utilizado íntegramente para realizar cálculos, esto junto a su potencia de cómputo hace que sea uno de los dos que va a tener asignados 35 combates. Este concretamente se encargará de entrenar entre el 0 y el 35.
- **Ordenador 2**: Al igual que el anterior, este ordenador esta dotado de gran potencia, junto a una completa disponibilidad para realizar cálculos, por tanto será el otro al que le asignaremos 35 combates, más concretamente los combates del 35 al 70.
- **Ordenador 3**: En este encontramos un caso más particular, y es que aún que goza de completa disponibilidad para los entrenos, dispone de algo menos de potencia que los anteriores. Por tanto a este le asignaremos 16, los comprendidos entre el 70 y el 86.
- **Ordenador 4**: Finalmente, el último ordenador, a nivel de capacidad de cálculo similar al anterior, pero el cuál no se encuentra completamente disponible solo para realizar la tarea de entrenar, debido a que es un ordenador personal. Por

tanto, este será el que menor asignación tenga, solamente 12, los comprendidos entre el 86 y el 98.

En cada uno de los ordenadores abriremos diversos entornos para el entreno, aprovechando todos los núcleos de la Central Processing Unit (CPU) de cada ordenador. De modo que si por ejemplo un ordenador dispone de 4 núcleos y 20 combates, a cada núcleo se le asignará un entorno con 5 combates. De esta manera utilizaremos toda la capacidad de cómputo de la que dispone cada uno de los ordenadores, reduciendo todo lo posible el tiempo que se necesitará para llevar a cabo la etapa de la experimentación.

4.3 Definición de los experimentos

Una vez preparado todo, solo nos queda por delante realizar el experimento y entrenar los dos agentes de IA, para posteriormente valorar su desempeño en el juego. Recordemos que al querer comparar dos redes neuronales diferentes, deberemos realizar exactamente el mismo experimento, simplemente cambiando la red neuronal a entrenar, creando así dos agentes diferentes, cada una creada con una red neuronal diferente. Ahora entraremos en profundidad sobre los aspectos que definen al experimento a realizar.

4.3.1 Recompensa y castigo

En el contexto del aprendizaje por refuerzo RL, las recompensas (*rewards*) y las penalizaciones (*penalties*) son elementos fundamentales para guiar el comportamiento de un agente inteligente.

La recompensa es una señal numérica positiva que se proporciona al agente cuando toma una acción que se considera favorable o que lo acerca a su objetivo. Representa una retroalimentación positiva y refuerza el comportamiento deseado. Puede variar en magnitud, dependiendo de la tarea o entorno específico. Por ejemplo, en un juego, una recompensa puede otorgarse por ganar una partida, obtener puntos o alcanzar un logro específico.

Por otro lado, las penalizaciones son señales numéricas negativas que se asignan al agente cuando toma una acción no deseada o que lo aleja de su objetivo. Proporcionan una retroalimentación negativa y desincentivan comportamientos no deseados. Al igual que las recompensas, las penalizaciones pueden variar en magnitud según el contexto. Por ejemplo, en un juego, una penalización puede ser otorgada por perder una vida, cometer un error estratégico o tomar una acción subóptima.

Tanto las recompensas como las penalizaciones son utilizadas por el algoritmo de aprendizaje por refuerzo para guiar al agente hacia la toma de decisiones óptimas. El objetivo del agente es maximizar las recompensas acumuladas a lo largo del tiempo y minimizar las penalizaciones. Mediante la exploración y el aprendizaje, el agente busca encontrar una estrategia que maximice las recompensas esperadas y evite las

penalizaciones.

En nuestro caso, como queremos que nuestra IA de exactamente la misma prioridad a atacar que a no recibir daño, asignaremos un punto de recompensa a bajarle la vida a cualquiera de los Pokémon enemigos y un punto de castigo a que nos bajen la vida a cualquiera de los Pokémon aliados. Sin embargo, como las recompensas se asignan típicamente al aumentar una variable (como podría ser aumentar el número de monedas recogidas) y en nuestro caso tratamos con una variable de puntos de vida, es decir, que baja en vez de subir, para conseguir el efecto deseado debemos asignar un castigo **negativo** a bajarle la vida a cualquiera de los Pokémon enemigos.

4.3.2 Estado finalizado

En el aprendizaje por refuerzo, el término *done* o estado finalizado se refiere a una señal binaria que indica si se ha alcanzado un estado final o terminal en el entorno. Esta señal se utiliza para determinar si un episodio o una secuencia de acciones ha llegado a su fin.

Cuando el agente realiza una serie de acciones en el entorno, interactúa con este y espera alcanzar un objetivo o una meta específica. Se utiliza para indicar si se ha logrado o no ese objetivo. Si este estado es verdadero, significa que el episodio ha finalizado, y se considera que se ha alcanzado el estado final deseado. En cambio, si es falso, el episodio aún no ha terminado y el agente puede continuar tomando acciones en busca de su objetivo.

Esto es muy importante para el proceso de aprendizaje del agente. Cuando se alcanza un estado final, se puede proporcionar una recompensa o penalización final al agente, dependiendo del resultado obtenido. Esta retroalimentación adicional ayuda al agente a aprender y ajustar su comportamiento para futuros episodios.

Además, el estado finalizado también se utiliza para reiniciar el entorno y comenzar un nuevo episodio. Una vez que se alcanza el estado final y se proporciona la retroalimentación correspondiente, el agente puede volver a iniciar el proceso desde un estado inicial y continuar explorando y aprendiendo.

En nuestro caso, como un combate se acaba cuando todos los Pokémon de un entrenador tienen su vida a 0, haremos una función que devuelva *True* cuando uno de los dos equipo Pokémon tenga todas las vidas a 0, y que en caso contrario (al menos un Pokémon en cada equipo sigue vivo) devuelva *False*.

4.3.3 Algoritmo

Dentro del aprendizaje por refuerzo, existen muchos algoritmos diferentes, en este caso hemos seleccionado el algoritmo Proximal Policy Optimization (PPO) debido a su optimización proximal de la política, concretamente su segunda versión.

PPO2 (Proximal Policy Optimization) es un algoritmo de aprendizaje por refuerzo que se utiliza para entrenar modelos de inteligencia artificial en entornos de toma de de-

cisiones secuenciales. Fue desarrollado por OpenAI y se ha convertido en uno de los algoritmos más populares para resolver problemas complejos en aprendizaje por refuerzo [18].

Este se basa en la idea de optimizar de manera proximal las políticas de decisión, lo que significa que busca mejorar la política actual sin alejarse demasiado de ella. Esto se logra mediante la limitación del tamaño de las actualizaciones de política para evitar cambios drásticos y garantizar una optimización más estable.

Durante el entrenamiento, recopila muestras de interacción con el entorno y utiliza estas muestras para actualizar la política. El proceso de actualización se realiza en lotes, donde se calculan las ventajas y se estima la función de valor para mejorar la política.

Una de las ventajas del algoritmo PPO2 es su capacidad para manejar políticas estocásticas, lo que permite una exploración más efectiva del espacio de acciones. Además, utiliza técnicas de optimización avanzadas para garantizar una mejora gradual y estable de la política.

Este algoritmo ha sido elegido para este proyecto debido a sus características y ventajas en el aprendizaje por refuerzo.

Una de las razones es que utiliza una estrategia de optimización basada en la política. Esto permite una mejora continua y gradual a medida que se recopilan más datos y se obtiene retroalimentación del entorno de juego, maximizando así las recompensas obtenidas.

Otra ventaja es su estabilidad. El algoritmo está diseñado para limitar los cambios en la política de acciones en cada iteración, evitando cambios demasiado grandes que podrían llevar a un rendimiento inestable. Esto ayuda a mantener un proceso de aprendizaje más suave y confiable, lo cual es crucial para garantizar resultados consistentes y predecibles en el entrenamiento.

4.3.4 Validación

Dentro del *script* de entreno hemos programado un Leave-One-Out Cross-Validation (LOOCV) para llevar a cabo las fases de entreno y test. Este método es una técnica de validación cruzada utilizada en el aprendizaje automático y la evaluación de modelos. En este enfoque, se realiza una validación iterativa dejando un solo punto de datos fuera del conjunto de entrenamiento y utilizando el resto de los datos para entrenar el modelo. Luego, se evalúa el rendimiento del modelo al predecir el punto de datos excluido. Este proceso se repite para cada punto de datos del conjunto de datos original, generando así un conjunto de evaluaciones individualizadas. Al final, se promedian los resultados para obtener una medida general del rendimiento del modelo [19].

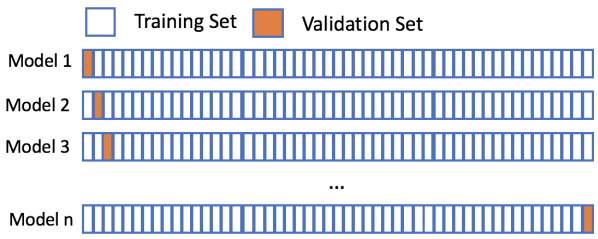


Figura 4.1: *Leave-One-Out Cross-Validation* [20].

4.3.5 Obtención de resultados

Además del Leave-One-Out Cross-Validation (LOOCV) y de toda la infraestructura de manejo de las librerías, debemos de recolectar toda la información que se vaya produciendo a medida que entrenamos la IA y que la validamos, ya que toda esta información será necesaria más adelante cuando, una vez finalizada la experimentación, queramos ver los resultados y comparar las dos arquitecturas a utilizar. Para ello iremos generando un *dataframe*, que se irá completando cada vez que validemos con el estado excluido del entreno, contendrá la marca temporal, la vida de cada uno de los Pokémon, el total de recompensa, el combate en el que se ha validado y si el combate lo hemos ganado o no. Una vez se haya finalizado todo el entreno, este *dataframe* se guardará en formato excel en memoria secundaria para poder estudiar los datos del conjunto de validación.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Time	HP1	HP2	HP3	HP4	HP5	HP6	EnemyHP1	EnemyHP2	EnemyHP3	EnemyHP4	EnemyHP5	EnemyHP6	Reward	State	Won
2	2023-03-30 12:26:36.868038	45	0	0	0	0	0	0	0	0	0	0	0	-517.0	ENTRENADOR5	1
3	2023-03-30 12:30:27.997835	42	12	0	0	0	0	0	0	0	0	0	0	62.0	ENTRENADOR6	1
4	2023-03-30 12:33:24.598038	36	0	15	0	0	0	0	0	0	0	0	0	50.0	ENTRENADOR7	1
5	2023-03-30 12:37:19.745553	41	25	23	17	0	0	0	0	0	0	0	0	31.0	ENTRENADOR8	1
6	2023-03-30 12:41:58.487485	46	25	25	17	0	0	0	0	0	0	0	0	81.0	ENTRENADOR9	1

Figura 4.2: Ejemplo del archivo excel generado.

4.3.6 Experimentos preliminares

Para comprobar que el entorno carga sin problemas todo nuestra integración propia del juego programaremos un agente aleatorio, junto a una serie de impresiones por pantalla para comprobar que accedemos correctamente a todas las variables del juego y que nuestras variables propias de recompensa y estado finalizado se actualizan correctamente. Este agente automático no utilizará ningún algoritmo ni ninguna red neuronal, simplemente en cada instante de tiempo seleccionará una acción aleatoria del repertorio de botones disponibles en el juego.

Una vez comprobada la validez de nuestra integración del juego, podemos generar un pequeño conjunto de pruebas con diferentes estados para así proceder a encontrar posibles fallos dentro de nuestra implementación del *Leave-One-Out Cross-Validation*, o algún posible error manejando los entornos del juego junto a los algoritmos y redes utilizados para Deep Reinforcement Learning (DRL). Una vez todo funciona correcta-

mente, estamos listos para avanzar hacia la experimentación definitiva.

4.3.7 Experimentos definitivos

Una vez tenemos todo lo anterior listo, alta una última cosa por asegurar, y es la obtención de suficientes muestras para, posteriormente, poder comparar ambas redes neuronales. Como ya hemos estipulado anteriormente, el método que utilizaremos para el entreno y la validación se trata del Leave-One-Out Cross-Validation (LOOCV), pero encontramos un problema aquí para generar suficientes muestras. Si ejecutásemos el experimento ahora mismo, solo obtendríamos 98 resultados de validación, uno de cada combate, lo cuál no son suficientes muestras. Por tanto repetiremos el proceso k veces. Asignando una $k = 31$ repetiríamos 31 veces el proceso del Leave-One-Out Cross-Validation (LOOCV), es decir, entrenar y validar del 1 al 98 contaría como una iteración, y esto lo repetiríamos $k - 1$ veces más. De esta manera obtendremos 3.038 muestras de validación totales de cada red neuronal, o lo que es lo mismo 31 muestras de validación de cada uno de los combates. Aseguraremos una cantidad de muestras lo suficientemente representativa como para poder comparar ambas redes neuronales.

Primero entrenaremos una red utilizando todos los ordenadores posibles, una vez todos hayan finalizado, procederemos a repetir exactamente el mismo entreno pero cambiando a la otra red. De esta manera, cuando esta segunda red haya finalizado su entreno, estaremos listos para poder analizar los resultados de la fase de experimentación.

En conclusión, el capítulo de experimentación ha sido fundamental para obtener todos los datos que nos permitirán evaluar y analizar el desempeño de las IA en el juego Pokémon Rojo Fuego. A través de la definición de métricas adecuadas, se podrá medir de manera cuantitativa el rendimiento y la capacidad de aprendizaje de cada una de las IA. Los experimentos rigurosos y controlados han permitido obtener resultados confiables y comparables.

CAPÍTULO 5

RESULTADOS

En el capítulo de resultados, se presentarán los hallazgos obtenidos al comparar el rendimiento y los resultados de dos redes neuronales utilizadas. Estas redes, la arquitectura NATURE-CNN y la arquitecura IMPALA-CNN, fueron entrenadas igual y serán evaluadas utilizando las métricas definidas en el capítulo anterior, proporcionando así una base sólida para el análisis comparativo. A lo largo de este capítulo, se examinarán y compararán los resultados obtenidos por ambas redes, destacando sus fortalezas, debilidades y diferencias significativas en términos de rendimiento, capacidad de aprendizaje y adaptación al juego. Además, se analizará cómo estas diferencias pueden afectar el desempeño general de la IA en el juego y se proporcionarán conclusiones basadas en los resultados obtenidos. Este capítulo será fundamental para evaluar y comprender el impacto de las elecciones de arquitectura de red en el rendimiento de la IA en el juego Pokémon Rojo Fuego y sentará las bases para la discusión y recomendaciones finales del Trabajo Fin de Grado (TFG).

5.1 Obtención y procesamiento de los datos

Una vez finalizados ambos entrenos de las redes neuronales, podemos recoger todos los datos generados de estos. Estos incluyen los archivos excel con los datos de las métricas que utilizaremos para evaluar y un archivo final que incluye los datos y parámetros de cada red neuronal para poder utilizar estas IA sin necesidad de volver a entrenarlas de nuevo.

Como el entreno estaba dividido entre diversos ordenadores diferentes y cada ordenador también tenía divididos el número de combates entre sus núcleos, tenemos muchos archivos excel diferentes, que contienen los resultados obtenidos al validar los combates. Primero debemos juntar todos estos archivos en uno solo y comprobar que la fusión se haga correctamente. Tendremos que hacer esto dos veces: primero fusionando todos los archivos de una red en un solo archivo y posteriormente, todos

los archivos de la otra red en uno solo también.

5.2 Análisis de los resultados

Ahora que ya disponemos de los datos de forma conjunta, debemos analizarlos para comparar el rendimiento basándonos en las métricas que ya hemos establecido, para esto tendremos que hacer un análisis estadístico con R.

5.2.1 Comparativa de ambas redes

Para poder comparar ambas redes, primero crearemos un subconjunto de cada una de las dos redes, incluyendo solo los combates que ganan (al menos uno de nuestros Pokémon estará vivo, y los enemigos todos muertos) y descartando los combates que pierden ya que ambas redes presentaban un rendimiento muy similar en términos de combates totales ganados y perdidos. Una vez hecha esta división crearemos una columna nueva en nuestros *dataframes* que será la vida total, es decir la suma de las vidas de todos nuestros Pokémon. Esta será la variable que compararemos.

Para poder compararlas, será necesario hacer un contraste de hipótesis para poder evaluar nuestra hipótesis nula H_0 de que la vida media de ambas redes es igual $\mu_1 = \mu_2$ (siendo μ_1 la vida media de la red IMPALA-CNN y μ_2 la vida media de la red NATURE-CNN). Un contraste de hipótesis, también conocido como prueba de hipótesis, es un procedimiento estadístico utilizado para evaluar la evidencia a favor o en contra de una afirmación o suposición sobre una población o un fenómeno en particular. Se utiliza para tomar decisiones basadas en datos y evidencia empírica.

El objetivo principal de un contraste de hipótesis es probar una hipótesis nula (H_0) en contra de una hipótesis alternativa (H_1). La hipótesis nula establece que no hay diferencia o efecto en la población o fenómeno en estudio, mientras que la hipótesis alternativa propone que sí existe una diferencia o efecto [21]. En nuestro caso tendremos:

$$\begin{cases} H_0: & \mu_1 = \mu_2 \\ H_1: & \mu_1 \neq \mu_2 \end{cases}$$

Se establece un nivel de significancia, que representa la probabilidad aceptada de cometer un error al rechazar incorrectamente la hipótesis nula. Luego, se calcula una estadística de prueba utilizando los datos disponibles y se compara con una distribución de probabilidad conocida (como la distribución normal) para determinar si la estadística de prueba es lo suficientemente inusual como para rechazar la hipótesis nula.

Si la evidencia estadística es lo suficientemente fuerte, se rechaza la hipótesis nula en favor de la hipótesis alternativa. Esto implica que hay suficiente respaldo para afirmar que existe una diferencia o efecto en la población o fenómeno en estudio. Por otro lado, si la evidencia estadística no es lo suficientemente fuerte, se falla en rechazar la

hipótesis nula y se concluye que no hay suficiente evidencia para afirmar la existencia de una diferencia o efecto.

Para poder realizar el contraste de hipótesis, primero debemos estudiar nuestros datos y conocer su distribución. Como podemos observar en la Fig 5.1 y en la Fig 5.2 las distribuciones distan bastante de una distribución normal.

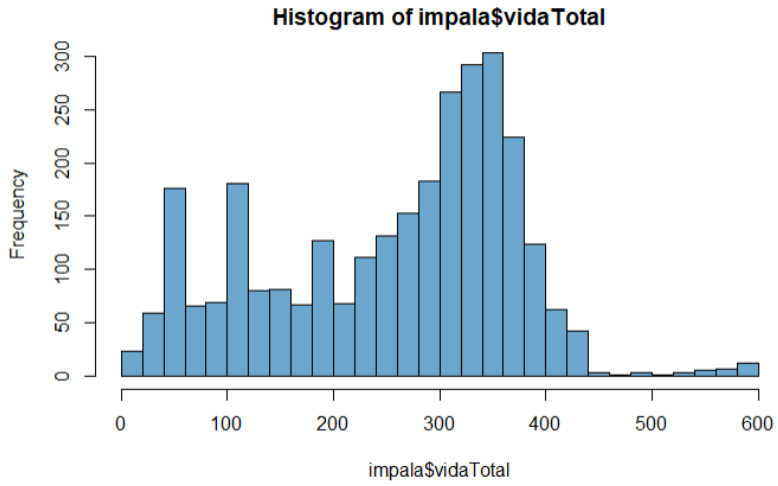


Figura 5.1: Distribución de la vida total de la red IMPALA-CNN.

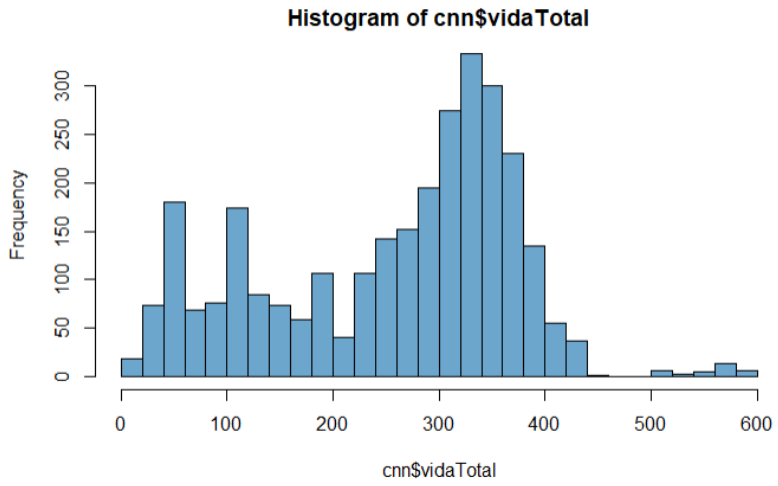


Figura 5.2: Distribución de la vida total de la red NATURE-CNN.

Viendo los resultados, tendremos que aplicar técnicas estadísticas para poder obtener distribuciones normales y poder así aplicar el test t de Student. Este test es una prueba

estadística utilizada para determinar si hay diferencias significativas entre las medias de dos grupos independientes.

5.2.2 Bootstrapping

El *bootstrapping* es una técnica estadística utilizada para estimar la incertidumbre asociada a un estadístico de interés sin hacer suposiciones sobre la distribución subyacente de los datos. Fue propuesto por primera vez por Bradley Efron en 1979 y se ha convertido en una herramienta muy útil en el análisis estadístico y la inferencia.

El objetivo principal es aprovechar al máximo la información contenida en una muestra de datos observada para obtener estimaciones robustas y precisas de los parámetros poblacionales o de las características de interés. En lugar de depender de supuestos sobre la distribución de los datos, se basa en la idea de que la muestra observada es una representación aproximada de la población subyacente [22].

El proceso del *bootstrapping* implica generar B muestras de reemplazo a partir de la muestra original, donde cada muestra tiene el mismo tamaño que la muestra original. Estas muestras de reemplazo se generan seleccionando aleatoriamente observaciones de la muestra original, permitiendo que las mismas observaciones aparezcan en múltiples muestras. Luego, se calcula el estadístico de interés para cada una de las muestras de reemplazo.

Al obtener una colección de estadísticas de interés a partir de las muestras de reemplazo, se puede construir una distribución empírica de los valores del estadístico. Esta distribución proporciona información sobre la variabilidad y la incertidumbre asociada con el estadístico de interés.

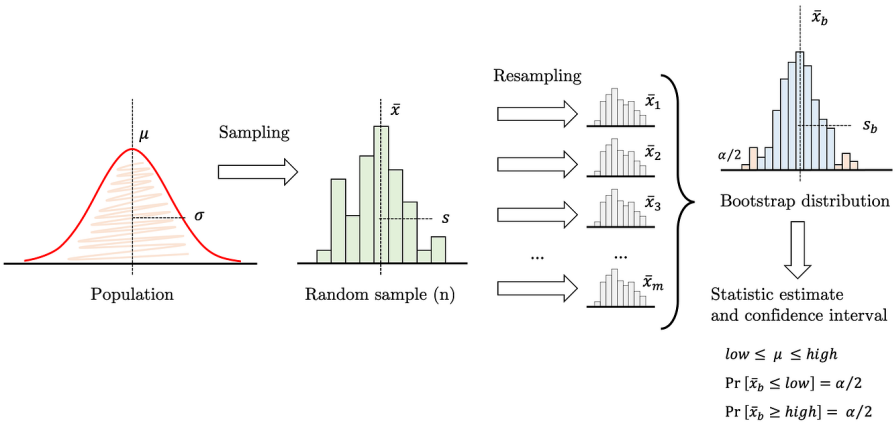


Figura 5.3: Representación de una distribución bootstrap, 5 muestras aleatorias y la distribución bootstrap final [23].

Esto ofrece varias ventajas. En primer lugar, es una técnica no paramétrica, lo que significa que no requiere suposiciones específicas sobre la forma de la distribución de los datos. Esto lo hace muy flexible y aplicable en una amplia gama de situaciones. Además, puede proporcionar estimaciones de intervalo de confianza, p-valores y otros indicadores de incertidumbre que son útiles para la toma de decisiones estadísticas.

Por tanto, aplicaremos *bootstrapping* a cada una de nuestras redes y en cada uno de nuestros combates, diferenciando así entre las dos redes y entre los 98 combates. Para cada uno de ellos se han utilizado $B = 10{,}000$ muestras. Obteniendo así distribuciones normales para cada red y en cada combate.

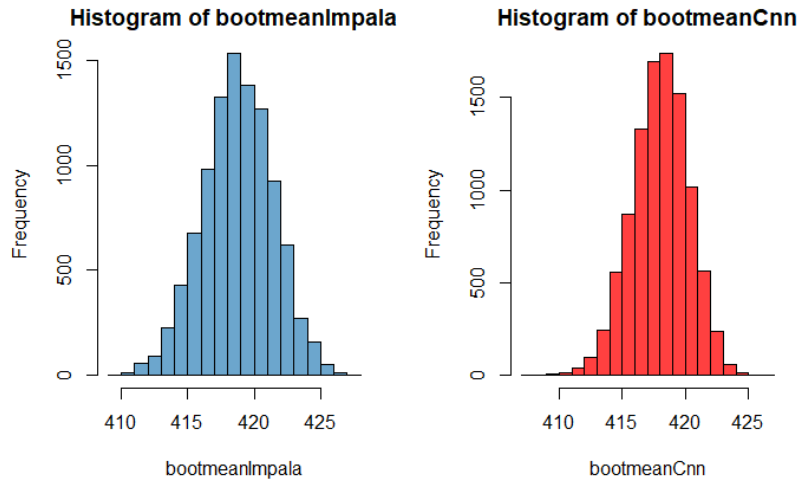


Figura 5.4: Distribuciones de la vida total de ambas redes en el combate 95.

5.2.3 Permutaciones

Las permutaciones son una técnica estadística que se utiliza para explorar todas las posibles combinaciones de un conjunto de elementos. En el contexto del análisis de datos, son útiles para evaluar la variabilidad de un estadístico o para generar distribuciones que sirven de referencia en pruebas de hipótesis [22].

La idea fundamental detrás de esto es reorganizar aleatoriamente los datos sin alterar sus valores individuales. Esto implica generar todas las posibles formas de ordenar o distribuir los datos y luego calcular el estadístico de interés para cada una de estas permutaciones.

Al realizar las permutaciones, se pueden obtener múltiples estadísticas para cada permutación, lo que permite construir una distribución nula o de referencia. Esta distribución representa la variabilidad esperada del estadístico bajo la hipótesis nula de que no hay diferencia o asociación entre las variables de interés.

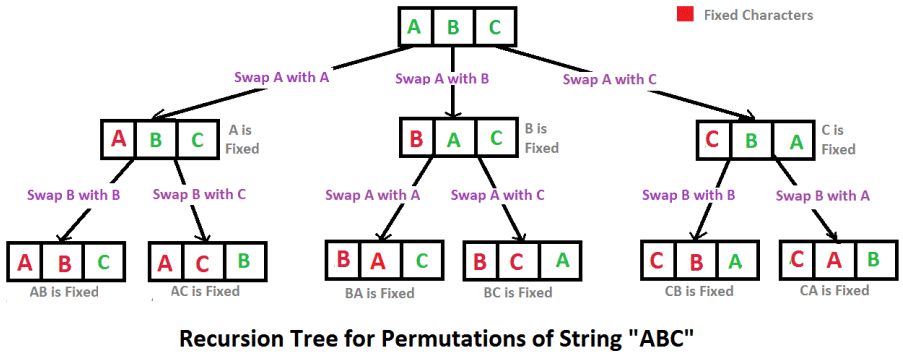


Figura 5.5: Árbol de permutaciones del string “ABC” [24].

Una ventaja es que no requieren supuestos sobre la distribución de los datos subyacentes. Además, son especialmente útiles en situaciones donde los métodos analíticos tradicionales pueden no ser aplicables o están limitados.

La idea detrás de aplicar este método también para comparar ambas redes además del *bootstrapping* nace de la posibilidad de que al calcular las vidas medias con el método anterior, tengamos casos en los que en la mayoría de combates se tiene un peor rendimiento pero se tiene uno con un rendimiento muy bueno, que desequilibra la media hacia ese lado (por ejemplo una red que tiene de vidas al finalizar un combate 3,6,8 y 5 mientras que la otra tiene 2,4,3 y 100). De esta manera, a parte de cubrir la media de las vidas aplicando *bootstrapping*, también cubriremos la posibilidad de casos extremos, obteniendo así un resultado mucho más completo y fiable.

Por tanto, calcularemos permutaciones de los resultados de vidas finales en cada combate y de cada red, comparando uno a uno en cuántos casos tiene más vida una red que otra.

5.2.4 Comparativa

Una vez aplicadas las técnicas de *bootstrapping* y permutación, ya hemos obtenido los resultados deseados, ahora debemos compararlos y extraer conclusiones de estas comparativas.

Por un lado, tenemos las comparativas de vida media aplicando *bootstrapping*, donde obtenemos resultados que nos hacen rechazar nuestra hipótesis nula de que obtienen la misma vida media en pos de aceptar la alternativa de que son diferentes en 96 de 98 combates. En estos 96 combates donde tenemos indicios estadísticos de diferencia de medias encontramos que la red IMPALA-CNN es mejor en 41 de ellos y la red NATURE-CNN en 55, resultados muy próximos en los cuáles encontramos ligeras mejoras generales en la red NATURE-CNN. Los dos combates que no presentan evidencias estadísticas de diferencia de medias es debido a su gran similitud: 322,53 y 322,57 de vida media en uno y 558,32 y 558,28 de vida media en el otro. Por tanto podríamos decir

que hay en dos combates donde tienen el mismo rendimiento.

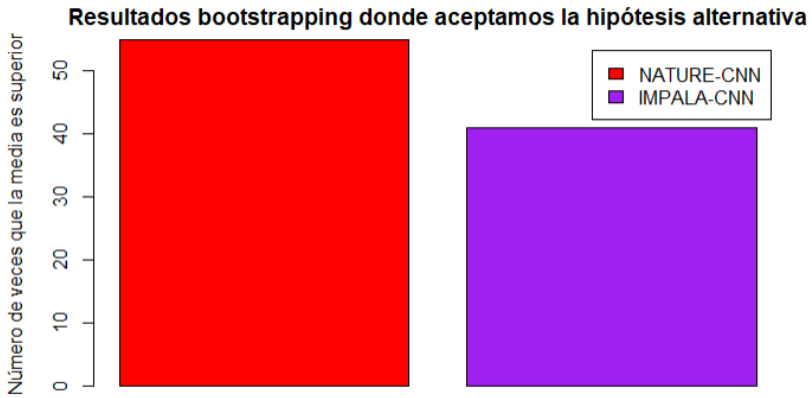


Figura 5.6: Gráfico de los resultados del *bootstrapping* donde tenemos evidencias estadísticas para aceptar la hipótesis alternativa.

Por otro lado, en las comparativas aplicando permutaciones para evitar encontrar casos que desbalanceen la media y nos hagan creer que una red es mejor que otra aunque en general tenga un rendimiento peor, encontramos que la red IMPALA-CNN tiene mejor rendimiento en 40.897 combinaciones, la red NATURE-CNN en 42.651 combinaciones y un rendimiento igual en 5.655 combinaciones.

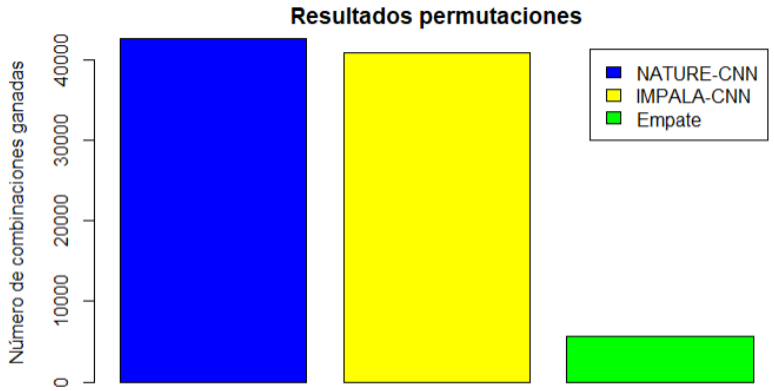


Figura 5.7: Gráfico de los resultados de las combinaciones.

Como se puede observar, ambas comparativas nos indican que la red NATURE-CNN tiene un mejor rendimiento en lo que respecta a acabar con mayor vida los combates. Aunque esta mejoría es leve en ambos casos (sobretodo cuando comparamos mediante permutaciones), al llegar a la misma conclusión con ambas, podemos determinar que para nuestro entorno de juego de Pokémon Rojo Fuego, con la experimentación realizada y para los combates de los que disponíamos en la base de datos, la red NATURE-CNN da mejores resultados.

En conclusión, en el presente apartado de resultados se han comparado y analizado las métricas obtenidas al evaluar las dos redes neuronales en el contexto del juego Pokémon Rojo Fuego. Gracias a lo obtenido en la experimentación, se ha podido evaluar el desempeño de las redes en términos de su capacidad para ganar lo mejor posible en el juego.

Los resultados obtenidos han demostrado que ambas redes han logrado aprender a jugar y ganar en Pokémon Rojo Fuego, aunque con variaciones ligeras en su desempeño. Estas diferencias, inclinan la balanza a favor de la red NATURE-CNN.

CAPÍTULO 6

CONCLUSIONES

En este Trabajo Fin de Grado (TFG), se ha abordado el desafío de desarrollar dos agentes de Inteligencia Artificial (IA) capaces de aprender a jugar y ganar en el juego Pokémon Rojo Fuego utilizando el enfoque del Deep Reinforcement Learning (DRL). A lo largo de los diferentes apartados, se ha realizado un análisis exhaustivo de los requisitos del proyecto, se ha llevado a cabo una modificación de una planificación ágil con incrementos semanales, se ha construido un entorno de desarrollo funcional y se han utilizado diversas librerías y herramientas para poder implementar, entrenar y evaluar el sistema.

En el apartado de análisis, se han identificado los requisitos funcionales y no funcionales necesarios para su cumplimiento. Además, se ha utilizado una variación de una metodología de desarrollo ágil que ha permitido una planificación detallada y una gestión efectiva del tiempo y los recursos.

En el apartado de desarrollo, se ha creado una base de datos de combates que ha servido como punto de partida para el entrenamiento de las redes neuronales. Se han elegido diferentes arquitecturas de redes, como son la NATURE-CNN y la IMPALA-CNN, y se ha construido un entorno de juego funcional para simular las interacciones entre el agente y el entorno.

En el apartado de experimentación, se han definido métricas de evaluación para medir el desempeño de las redes y se han llevado a cabo experimentos comparativos para analizar los resultados obtenidos.

En el apartado de resultados, se han presentado los hallazgos obtenidos a través de la experimentación y se han discutido los resultados en relación con las métricas definidas. Se ha realizado un análisis estadístico y se han aplicado pruebas de hipótesis para determinar la significancia de las diferencias observadas.

Con todo esto, ha sido posible llevar a cabo este proyecto y cumplir el objetivo de

crear y comparar dos IAs completamente funcionales mediante DRL para el videojuego Pokémon Rojo Fuego de la GameBoy Advance.

6.1 Evaluación del resultado

Para la evaluación, hemos realizado una comparación exhaustiva entre dos redes neuronales utilizando técnicas estadísticas como el *bootstrapping* y las permutaciones. Estas técnicas nos permitieron evaluar el rendimiento de las redes de manera robusta y objetiva.

Mediante el *bootstrapping*, generamos múltiples muestras de datos de entrenamiento mediante remuestreo con reemplazo. Esto nos permitió obtener distribuciones normales de las métricas de evaluación, permitiéndonos así el uso del test t de Student para comparar las medias de la vida total y determinar si las diferencias observadas eran estadísticamente significativas.

Además, utilizamos las permutaciones. Al permutar los datos de manera aleatoria, evitamos la posibilidad de que un resultado muy grande desviase la media, algo que podía ocurrir haciendo *bootstrapping*. Comparando las métricas observadas, pudimos determinar si las diferencias eran notables respecto a la otra técnica empleada.

Los resultados obtenidos a través de estas técnicas indicaron que ambas redes mostraron un rendimiento similar en general. Sin embargo, la red NATURE-CNN demostró un rendimiento ligeramente superior en términos de las métricas evaluadas. Aunque las diferencias no fueron estadísticamente significativas en absolutamente todos los casos, los análisis revelaron una tendencia consistente hacia un mejor desempeño de la red NATURE-CNN.

Estos hallazgos sugieren que la arquitectura NATURE-CNN puede ser más adecuada para el juego Pokémon Rojo Fuego y el aprendizaje por refuerzo profundo, teniendo en cuenta todo el contexto de este proyecto. Siendo así contrario, al rendimiento mostrado por ambas redes en el entorno COIN-RUN (ver la Fig 3.3), donde la red IMPALA-CNN era superior.

Cabe destacar que consideramos que, los objetivos iniciales que comprendían la creación de dos agentes de IA que aprendiesen a jugar y ganar combates en Pokémon Rojo Fuego y la comparativa entre ambos para discernir cuál de ellos tenía un mejor rendimiento han sido cumplidos con éxito.

6.2 Posibles mejoras

Si hubiéramos tenido más tiempo para los experimentos, habríamos tenido la oportunidad de mejorar significativamente este proyecto. Una de las áreas clave en las que podríamos haber invertido más tiempo es en el entrenamiento de las redes neuronales. Dado que las redes tienen la capacidad de aprender y adaptarse a medida que reciben

más datos, un tiempo de entrenamiento más prolongado habría permitido a las redes explorar un espacio de soluciones más amplio y adquirir un conocimiento más profundo sobre el juego.

En particular, con un entrenamiento más prolongado, las redes podrían, presumiblemente, haber sido capaces de aprender la tabla de tipos de Pokémon (Fig 6.1), lo que habría sido una ventaja significativa en los combates. Al reconocer y aprovechar las ventajas y desventajas de tipo de cada Pokémon, las redes habrían sido capaces de tomar decisiones estratégicas más informadas durante el juego. Esto les habría permitido maximizar su rendimiento y aumentar sus posibilidades tanto de ganar, como de hacerlo de la mejor manera posible.

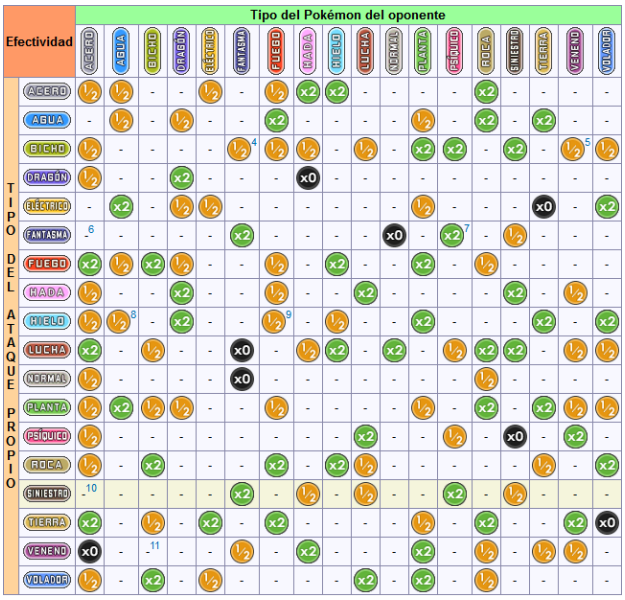


Figura 6.1: Tabla de tipos Pokémon, donde los símbolos hacen referencia al multiplicador de daño [25].

Además, más tiempo para los experimentos habría permitido realizar ajustes y optimizaciones adicionales en la arquitectura de las redes y en los parámetros de entrenamiento. Podríamos haber explorado diferentes configuraciones y técnicas de mejora, como la búsqueda de hiperparámetros, con el fin de lograr un mejor rendimiento y una mayor eficiencia en el aprendizaje.

Cabe destacar también, que con más tiempo disponible podríamos incluso haber expandido la base de datos de combates hasta que incluyese todos los combates disponibles en el juego, de manera que nuestro conjunto de datos habría sido el mejor, más variado y más representativo de lo que se encuentra dentro del juego.

Por último, con un tiempo adicional, habríamos tenido la oportunidad de realizar más pruebas y validaciones exhaustivas. Esto nos habría permitido obtener resultados más

sólidos y confiables, y nos habría brindado una comprensión más completa de las capacidades y limitaciones de las redes en el contexto específico del juego Pokémon Rojo Fuego.

6.3 Competencias y conocimientos adquiridos

Trabajar con redes neuronales en un videojuego ha sido una experiencia fascinante, ya que nos ha brindado la oportunidad de aplicar los conocimientos adquiridos en el Grado de Ingeniería Informática sobre Machine Learning (ML) e Inteligencia Artificial (IA), además de descubrir y aprender a aplicar conceptos como el RL o el DRL. A través de este proyecto, hemos podido profundizar nuestra comprensión de los mecanismos y operaciones que ocurren en el procesamiento de información dentro de una red neuronal, así como comprender como se podrían aplicar en los videojuegos. Esta experiencia nos ha permitido fusionar teoría y práctica, y nos ha otorgado una visión más sólida y aplicada de las capacidades de las redes neuronales en un contexto tan emocionante como el de los videojuegos.

Además de los aspectos mencionados anteriormente, es importante destacar la aplicación de nuestros conocimientos en programación para comprender y mejorar el código existente en este Trabajo Fin de Grado (TFG). La habilidad de interpretar y analizar el código fue fundamental, y también nos permitió desarrollar nuevas mejoras en el proyecto. La elección de Python y R como lenguajes de programación fue beneficiosa, ya que nos permitió familiarizarnos con ellos, así como adquirir conocimientos sobre diversas bibliotecas utilizadas, como GymRetro y Stable Baselines y recordar a la par que consolidar diferentes conceptos estadísticos y matemáticos. Todo esto ha ampliado nuestro conjunto de habilidades y nos ha permitido aprovechar al máximo las funcionalidades que ofrecen para la implementación de la Inteligencia Artificial (IA) en el contexto de los videojuegos.

Enlace al código del proyecto aquí.

BIBLIOGRAFÍA

[1] Northware, "Requerimientos en el desarrollo de software y aplicaciones," Jul 2022. [Online]. Available: https://www.northware.mx/blog/requerimientos-en-el-desarrollo-de-software-y-aplicaciones/#:~:text=Los%20requerimientos%20funcionales%20especifican%20lo,c%C3%B3mo%20debe%20comportarse%20el%20sistema. 2.1

[2] OpenAI, "Gym retro documentation." [Online]. Available: https://retro.readthedocs.io/en/latest/getting_started.html 2.4

[3] Anaconda, "Miniconda." [Online]. Available: https://docs.conda.io/en/latest/miniconda.html 2.4.1

[4] OpenAI, "Welcome to stable baselines docs! - rl baselines made easy." [Online]. Available: https://stable-baselines.readthedocs.io/en/master/ 2.2

[5] aed3, "Fsai-extras," https://github.com/aed3/FSAI-Extras, 2021. 2.3

[6] M. Sotaquirá, "¿qué es una red neuronal?" [Online]. Available: https://www.codificandobits.com/blog/que-es-una-red-neuronal/ 2.4

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, may 2017. [Online]. Available: https://doi.org/10.1145/3065386 2.5.4

[8] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015. 2.5.5

[9] C. V. Nicholson, "A beginner's guide to deep reinforcement learning." [Online]. Available: https://wiki.pathmind.com/deep-reinforcement-learning#neural 2.5.6

[10] P. Baheti, "Deep reinforcement learning: Definition, algorithms amp; uses." [Online]. Available: https://www.v7labs.com/blog/deep-reinforcement-learning-guide 2.6

[11] P. Wiki, "Guía de pokémon rojo fuego y pokémon verde hoja." [Online]. Available: https://pokemon.fandom.com/es/wiki/Gu%C3%ADa_de_Pok%C3%A9mon_Rojo_Fuego_y_Pok%C3%A9mon_Verde_Hoja 3.1, 3.2

[12] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," 2019. 3.2, 3.3

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015. [Online]. Available: https://doi.org/10.1038/nature14236 3.2.1, 3.4, 3.2.2

[14] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," 2018. 3.2.2, 3.5

[15] P. Mat, "retro-scripts," https://github.com/MatPoliquin/retro-scripts, 2022. 3.3.1

[16] Bulbapedia, "Pokémon data structure (generation iii)." [Online]. Available: https://bulbapedia.bulbagarden.net/wiki/Pok%C3%A9mon_data_structure_(Generation_III) 3.3.2

[17] DataCrystal, "Pokémon firered and leafgreen:ram map." [Online]. Available: https://datacrystal.romhacking.net/wiki/Pok%C3%A9mon_FireRed_and_LeafGreen:RAM_map 3.3.2

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. 4.3.3

[19] G. Webb, C. Sammut, C. Perlich, T. Horváth, S. Wrobel, K. Korb, W. Noble, C. Leslie, M. Lagoudakis, N. Quadrianto, W. Buntine, L. Getoor, G. Namata, J. Jin, J.-A. Ting, S. Vijayakumar, S. Schaal, and L. De Raedt, *Leave-One-Out Cross-Validation*, 01 2010. 4.3.4

[20] [Online]. Available: https://biol607.github.io/lectures/crossvalidation.html#31 4.1

[21] G. Guyatt, R. Jaeschke, N. Heddle, D. Cook, H. Shannon, and S. Walter, "Basic statistics for clinicians: 1. hypothesis testing," *CMAJ*, vol. 152, no. 1, pp. 27–32, 1995. [Online]. Available: https://www.cmaj.ca/content/152/1/27 5.2.1

[22] F. Bittman, *Bootstrapping an integrated approach with python and stata*. Degruyter, 2021. 5.2.2, 5.2.3

[23] A. Rodríguez, "Introduction to bootstrapping in data science-part 1," Feb 2022. [Online]. Available: https://towardsdatascience.com/introduction-to-bootstrapping-in-data-science-part-1-6e3483636f67 5.3

[24] GeeksforGeeks, "Write a program to print all permutations of given string," Apr 2023. [Online]. Available: https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/ 5.5

[25] PokeMew, "Tabla de tipos," Oct 2018. [Online]. Available: https://pokemew20.wordpress.com/tabla-de-tipos/ 6.1