Escuela
Politécnica
Superior

Reinforcement learning para videojuegos

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:
Luis Pérez-Brotons Ballester
Tutor/es:
Miguel Ángel Cazorla Quevedo
Álvaro Belmonte Baeza

Junio 2023

Universitat d'Alacant
Universidad de Alicante

Reinforcement learning para videojuegos

Autor
Luis Pérez-Brotons Ballester

Tutor/es
Miguel Ángel Cazorla Quevedo
Ciencia de la computación e Inteligencia Artificial
Álvaro Belmonte Baeza
Ciencia de la computación e Inteligencia Artificial

Grado en Ingeniería Robótica

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Junio 2023

Preámbulo

"Este proyecto nace del profundo interés en las nuevas tecnologías y en particular del interesante campo de la inteligencia artificial. El objetivo es proponer una investigación que sea relevante, así como aplicable a una de mis grandes pasiones, los videojuegos."

Agradecimientos

Este trabajo no habría sido posible sin el apoyo continuo de mis tutores Miguel Cazorla y Álvaro Belmonte, a los que les quiero agradecer la oportunidad de haberme permitido trabajar en este proyecto y haberme ayudado y aportado soluciones cuando más lo necesitaba.

También quiero agradecerle a mi familia y amigos por haberme acompañado en todo el camino hasta aquí y haberme convertido en la persona que soy hoy.

Igualmente, quiero dar las gracias a mi novia Marina por estar siempre a mi lado en las subidas y bajadas que he tenido a lo largo de este proyecto.

A mi madre y a mi padre por todo lo que me han dado,
sin los cuales no habría podido llegar hasta aquí

Incluso si pierdes en la batalla,
si mejoras lo que has hecho antes,
te has superado a ti mismo.

Marshal
Miembro del Alto Mando de la región Teselia
Pokemon Negro y Blanco.

Índice general

Índice de figuras

Índice de cuadros

1 Introducción

La inteligencia artificial (IA) se ha estado estudiando durante décadas y todavía es un campo donde queda mucho por descubrir. Esto se debe en parte a lo grande y amplio que este tema. La IA abarca desde las máquinas capaces de pensar por sí mismas, algoritmos de búsqueda utilizados para jugar a juegos, aplicaciones para el tratamiento de lenguajes naturales o sistemas de reconocimiento de objetos y del habla.

La primera vez que se usó el término de inteligencia artificial fue en 1956, cuando John McCarthy habló sobre este tema en la conferencia de Darmouth. Sin embargo, los primeros trabajos comenzaron años antes. En 1950, Alan Turing escribió una publicación sobre si las máquinas podrían ser capaces de simular el comportamiento humano y realizar acciones inteligentes[1]. A pesar de este primer acercamiento, el objetivo final todavía estaba lejos. Esto se debía a que por aquel entonces, los ordenadores no estaban aún preparados para ello. Además, el coste de la informática era demasiado elevado, solo las empresas tecnológicas que tuviesen un gran capital podían permitirse trabajar con ella. En los años venideros la IA comenzó a florecer. Pasada la primera mitad del siglo XX, la capacidad de almacenamiento y velocidad de los ordenadores comenzó a cobrar importancia, haciéndolos más accesibles y demostrando el potencial que podían ofrecer a un público ligeramente más amplio. No obstante, aún faltaba un largo camino por recorrer ya que la potencia que ofrecían seguía siendo limitada. En la década de los 80, la IA comenzó su etapa de expansión, en la que se vio la mejora de muchos algoritmos que permitieron, por ejemplo, crear los primeros sistemas expertos capaces de simular el razonamiento humano. Pocos años después, durante la década de los 90 y principios del 2000, ocurrieron hechos históricos como la derrota de Gary Kasparov, considerado el mejor jugador de ajedrez de la historia, contra Deep Blue, una supercomputadora desarrollada por la empresa IBM. Además, ese mismo año la empresa Dragon Systems desarrolló un software de reconocimiento de voz, lo que fue considerado como otro gran paso adelante.

Sin embargo, a pesar de los avances conseguidos en estos años, alcanzar una inteligencia artificial fuerte está aún muy distante. En la actualidad, vivimos en la era del Big Data, una época donde se tiene la capacidad de recopilar gran cantidad de datos, los cuáles serían demasiado para que una persona los procesase. Es por esto que la inteligencia artificial sigue en evolución. [2][3]

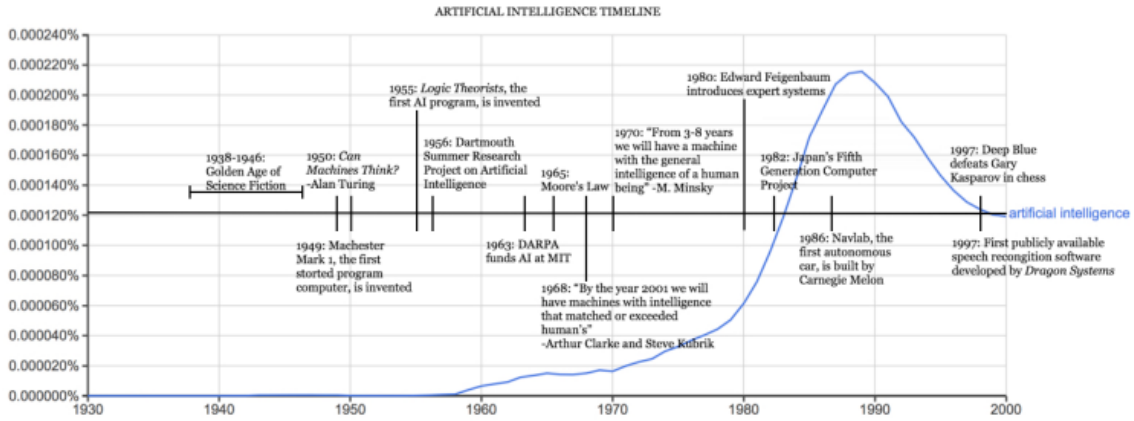


Figura 1.1: Historia de la Inteligencia Artificial en el siglo XX

La inteligencia artificial ha supuesto una revolución en diversos campos al proporcionar a las máquinas la capacidad de aprender y tomar decisiones de manera autónoma. Han nacido agentes inteligentes que pueden realizar diferentes tareas en el mundo real. Dentro de este amplio campo, el aprendizaje por refuerzo se destaca como un enfoque prometedor para entrenarlos. Al combinar los principios de la IA y la psicología del comportamiento, el aprendizaje por refuerzo permite que los agentes aprendan interactuando con su entorno recibiendo señales de recompensa y adaptando sus acciones para maximizar esos beneficios. Esta poderosa técnica ha sido probada en grandes campos que van desde los juegos hasta la robótica y el control de procesos.

Más en detalle, el aprendizaje por refuerzo, en adelante RL, del inglés reinforcement learning, consiste en una colección de métodos computacionales que están motivados principalmente por su capacidad para resolver problemas prácticos. Esta técnica tuvo éxito en el pasado en problemas de baja dimensionalidad ya que los algoritmos que se usaban carecían de gran capacidad de memoria o de potencia computacional. Sin embargo, en los últimos años y gracias al auge del Deep Learning se han podido desarrollar redes neuronales profundas mucho más poderosas. Éstas ofrecían la posibilidad de encontrar automáticamente representaciones compactas de baja dimensionalidad en datos de alta dimensionalidad, lo que ha generado una mejora sustancial en muchas de las áreas del Machine Learning, como por ejemplo la detección de objetos, el reconocimiento de voz o la traducción de idiomas. Este progreso ha servido para usar algoritmos de Deep Learning dentro del RL que definen el campo de Deep RL [4].

El creciente interés en estas nuevas técnicas se debe al desafío de diseñar sistemas inteligentes que sean capaces de operar en entornos dinámicos del mundo real. Por ejemplo, creando robots más autónomos y menos dependientes de trabajar en espacios controlados y con información anticipada. Para ello se requieren métodos de decisión que sean efectivos en presencia de incertidumbre y que puedan actuar en tiempo real. Bajo estas circunstancias, el aprendizaje es esencial para conseguir comportamientos útiles. Es por eso que el reinforce-

ment learning puede tener ciertas ventajas sobre el resto de técnicas de inteligencia artificial. Por ejemplo, su gran capacidad de generalización, lo que le permite transferir su conocimiento y experiencias adquiridas a situaciones similares. O también su capacidad de trabajar en entornos estocásticos al modelar la incertidumbre y aprender a partir de la interacción con el entorno, mediante las cuales puede desarrollar estrategias robustas frente a la variabilidad del entorno. [5]

2 Marco Teórico

Por todo lo mencionado, el aprendizaje por refuerzo se ha convertido en un área de investigación muy importante donde participan activamente gran cantidad de investigadores. En este proyecto se presentará este campo dentro de la perspectiva de los videojuegos.

Dentro de la industria de los videojuegos la inteligencia artificial es un campo muy estudiado. Se analizan las complejas interacciones entre los agentes y los entornos del juego. Varios de ellos brindan problemas interesantes y complicados para que los agentes los resuelvan, lo que hace a los videojuegos perfectos para la investigación de la IA. Se han utilizado como banco de pruebas para analizar algoritmos porque proporcionan un entorno controlado con conjuntos de reglas y niveles de complejidad muy diferentes. Permiten a los investigadores demostrar las propiedades de sus algoritmos en entornos de un solo agente (donde un solo agente compite para terminar con éxito el juego) y en sistemas de múltiples agentes (donde un equipo de agentes deben coordinarse para lograr un objetivo).

Los desafíos más interesantes de este paradigma son, en primer lugar, desarrollar agentes inteligentes en juegos donde el espacio de estados sea muy extenso. En segundo lugar, la dificultad de encontrar una política adecuada para entornos dinámicos y cambiantes. Una política se refiere a una asignación de cierta distribución de probabilidad sobre todas las acciones posibles. Y por último, la complejidad de generalizar los sistemas de aprendizaje ya que la mayoría de videojuegos se desarrollan en entornos muy específicos.

Concretamente en el aprendizaje por refuerzo, los agentes aprenden el comportamiento óptimo a base de prueba y error. Al interactuar con el entorno, se puede aplicar con éxito a las tareas de toma de decisiones secuenciales. Considerando un proceso de decisión episódico, el agente elige una acción de acuerdo con la política en un estado y dada una distribución inicial se puede asignar el retorno esperado de la política. Entonces el entorno recibe la acción, produce una recompensa y pasa al siguiente estado según la probabilidad de transición. El proceso continúa hasta que el agente alcanza un estado terminal o se alcanza un número máximo de pasos. El objetivo final es maximizar la recompensa. [6]

En la actualidad, se han experimentado grandes avances en este sector gracias al uso de técnicas de Deep Learning. Se han desarrollado diversos proyectos y algoritmos con el fin de conseguir agentes inteligentes que sean capaces de aprender a jugar de manera autónoma sin la intervención del ser humano. En esta sección se van a dar a conocer algunos de los proyectos más relevantes.

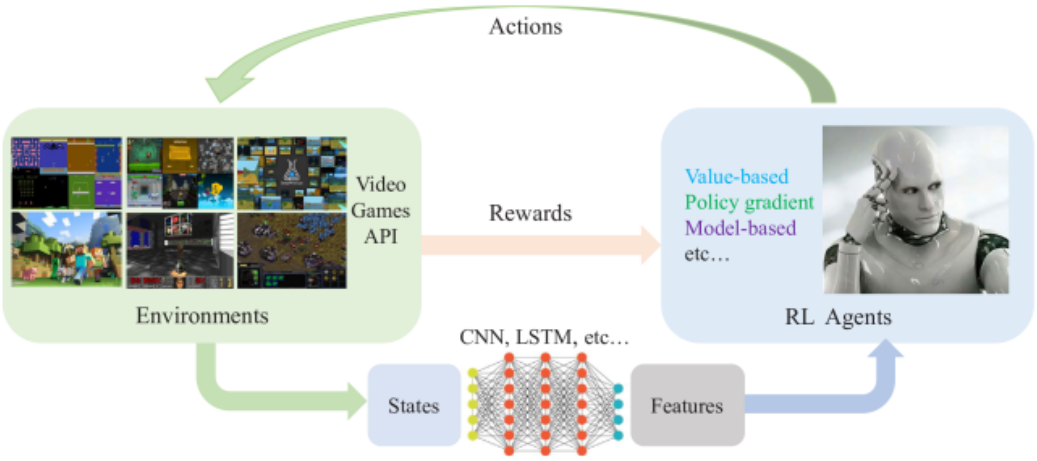


Figura 2.1: Diagrama típico que sigue el aprendizaje por refuerzo en videojuegos

2.1 Dota 2

Dota 2 es un videojuego multijugador en tiempo real desarrollado por Valve Corporation en 2013, el cual tiene de media entre 800000 y 1000000 jugadores concurrentes. Es un juego que presenta desafíos interesantes para el aprendizaje por refuerzo debido a la observabilidad parcial y la alta dimensionalidad de los espacios de observación y acción. Para trabajar en este entorno se construyó un sistema de entrenamiento distribuido para entrenar al agente, al cual lo llamaron OpenAi Five. Para el entrenamiento definieron una política la cual parametrizaron como una Red Neuronal Recurrente con aproximadamente 159 millones de parámetros. Esta red está formada por una única capa de 4096 unidades LSTM. [7]

En cuanto al espacio de observaciones que definieron, en lugar de usar todos los píxeles de la pantalla como entrada, que sería la información que recibirían los jugadores humanos, realizaron una simplificación. También, decidieron que el modelo tuviera acceso a la totalidad de la información disponible en cada paso del entrenamiento, a diferencia de los jugadores humanos, que necesitan mover el mapa para conocer la situación completa de su estado. Esto lo hicieron para disminuir la cantidad de recursos necesarios para computar toda la información.

Durante el transcurso del entrenamiento y para medir su progreso, OpenAi Five jugó contra numerosos jugadores aficionados y profesionales, así como contra equipos profesionales. Después de diez meses de entrenamiento (de julio de 2018 a abril de 2019), OpenAi Five se lanzó a la red para jugar partidas competitivas donde consiguión un 99.4% de victorias frente a la comunidad. Además, se enfrentó a los campeones mundiales de Dota 2, ganando al mejor de tres (2 -0), demostrando que el modelo había conseguido un alto nivel. Está fue la primera vez que un sistema de IA vencía a un campeón mundial de eSports. [8]

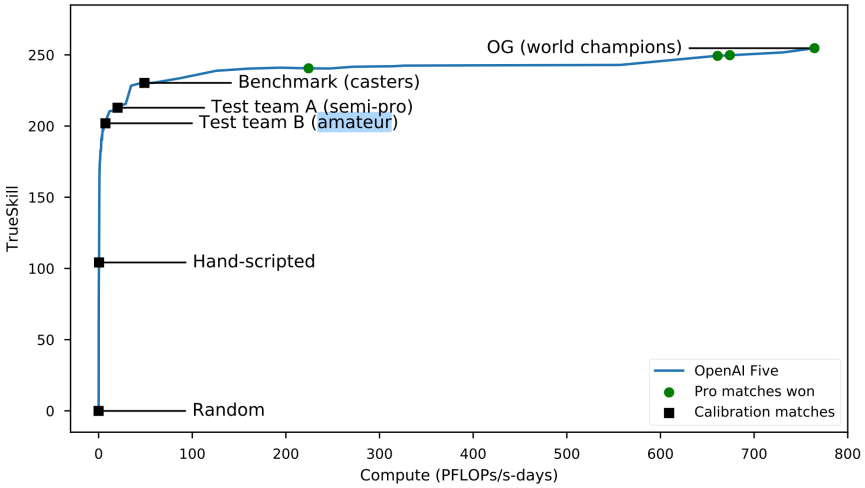


Figura 2.2: Curva de entrenamiento de OpenAiFive. Se marcan los niveles de jugadores humanos para otorgar información extra

2.2 Starcraft

Al igual que Dota 2, Starcraft también es un juego multiagente donde los jugadores toman decisiones en tiempo real. Este dominio genera importantes desafíos debido a su largo espacio de estrategias y acciones que se pueden llegar a tomar en cada instante. Cada partida consiste en miles de pasos y de acciones tomadas en una duración aproximada de 30 minutos.

En StarCraft, cada jugador puede elegir entre tres tipos de razas distintas, cada uno con mecánicas diferentes. Es por eso que se entrenaron tres agentes distintos cada uno de ellos entrenados durante 44 días. Para su evaluación, se guardaron tres modelos distintos: uno durante un entrenamiento supervisado (AlphaStar Supervised), otro después de 27 días(AlphaStar Mid) y otro al finalizar el entrenamiento (AlphaStar Final).

Respecto al método de entrenamiento, AlphaStar usa una combinación de diferentes técnicas. En primer lugar utiliza Redes Neuronales Recurrentes para hacer frente al espacio de acciones tan extenso del juego. En segundo lugar, emplean Reinforcement learning basado en algoritmos “off-policy”, que consisten en actualizar la política en base la experiencia generada en las políticas anteriores. De esta manera se podrá actualizar la red de manera asíncrona. Y finalmente Imitation Learning para predecir mejor las acciones de las políticas.

Los resultados demostraron la capacidad de aprendizaje de estos modelo consiguiendo estar AlphaStar Supervised en el percentil 84% de todos los jugadores humanos, el modelo intermedio en el 99.5% y AlphaStar Final en el 99.85%. [9]

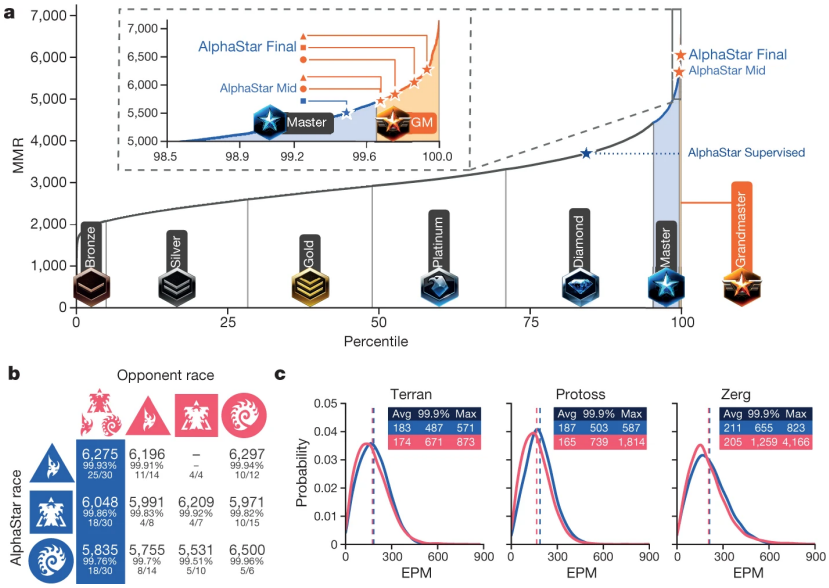


Figura 2.3: Resultados del entrenamiento de AlphaStar

2.3 VizDoom

El último proyecto a destacar es VizDoom. El cual hace uso de visual reinforcement learning para aprender a jugar a Doom, un juego shooter en primera persona en un mundo tridimensional. Mediante el uso del buffer de pantalla, el agente es capaz de percibir, interpretar y aprender de manera efectiva sobre el mundo, permitiéndole tomar decisiones tácticas y estratégicas sobre dónde ir y qué acciones tomar. Los factores más limitantes que afectan al rendimiento de VizDoom son el número de actores (objetos y bots), la resolución y el cálculo del buffer de profundidad.

Hicieron 2 experimentos. El primero se realizó en un entorno sencillo, donde el escenario era simplemente un rectángulo donde aparecía un monstruo en una posición aleatoria. El agente podía moverse a izquierda o derecha o disparar. El episodio terminaba cuando se eliminaba al monstruo o se superaban los 300 frames. El problema se modelaba como un proceso de decisión de Markov y se usaba Q-learning para aprender la política. La red neuronal usada estaba formada por 2 capas convolucionales con 32 filtros cuadrados, de 7 y 4 píxeles de ancho. En cuanto al estado del juego, estaba representado por los frames con sus imágenes en RGB.

Para el entrenamiento decidieron experimentar con diferente número de skip-frames. Es decir, no actuaban sobre cada uno de ellos, sino que se saltaban varios y en aquellos que se omitían se repetiría la acción anterior. Gracias a esta técnica demostraron que saltarse frames influenciaba directamente sobre la velocidad del entrenamiento y la robustez del agente. Se evidenció que en el rango de 4 a 10 se obtenía el mejor balance entre la velocidad del entrenamiento y la actuación final.

Este primer experimento fue muy sencillo, por lo que decidieron subir la complejidad. Para el siguiente ensayo, el agente aparecía en una zona del escenario aleatoria y existía un factor que hacía que la vida del agente fuera bajando lentamente, pero sin pausa. Para sobrevivir, tenía que recoger botiquines y evitar pociones venenosas. Estos objetos aparecen de forma aleatoria en el mapa. Las acciones que puede elegir el agente son moverse hacia adelante o hacia atrás y girar a sus lados. En referencia a la arquitectura de la red neuronal, es muy similar a la primera, pero en este caso tiene 3 capas convolucionales con filtros cuadrados de 7, 5 y 3 píxeles de ancho. Al estado del juego le añadieron los puntos de vida. Además integraron como entrada a la red una memoria de los últimos 4 estados.

Realizaron un entrenamiento de 1 millón de pasos y cada 5000 pasos evaluaban al agente. Todo el conjunto del entrenamiento duró 29 horas y sus resultados fueron decepcionantes. Ya que pese a que parecía que el agente si que había entendido las mecánicas del juego, hasta para los modelos finales del entrenamiento, en algunas posiciones iniciales no logró vivir más que un jugador que realiza acciones al azar. [10]

2.4 Pokemon

La franquicia Pokemon es una de las más longevas y exitosas en el mundo de los videojuegos y ofrece un entorno ideal para explorar las aplicaciones del Reinforcement Learning. Las batallas Pokemon, con sus elementos estratégicos, decisiones tácticas y espacios de estados complejos ofrecen un entorno desafiante para aplicar estas técnicas. Los agentes de RL, al igual que los jugadores, deben aprender y adaptarse a medida que se enfrentan a oponentes cambiantes y buscan maximizar sus posibilidades de victoria. Además, otra de las características interesantes de las batallas Pokemon es que el entorno es parcialmente observable y el espacio de estados es continuo y de alta dimensión, con múltiples atributos y características que influyen en el rendimiento de los Pokemon. Por estos motivos puede ser un campo interesante de investigación para el RL. En esta sección, se presentarán los resultados de dos proyectos que exploraron la aplicación del aprendizaje por refuerzo en Pokemon.

2.4.1 Competitive Deep Reinforcement Learning over a Pokemon Battling Simulator

El objetivo principal de este proyecto fue el desarrollo de un agente de aprendizaje por refuerzo capaz de comprender la importancia de los tipos en Pokemon para optimizar sus movimientos y maximizar el daño infligido. También se buscó optimizar una estrategia a largo plazo pese a que de manera inmediata la recompensa pudiese ser negativa, como sería, por ejemplo, perder un turno de ataque para cambiar a un Pokemon con un tipo más favorable. Con este propósito implementaron un entorno de batallas Pokemon, cuyas siglas en ingles son PBE (Pokemon Batlle Environment), que preservase los elementos clave de los enfrentamientos para así poder realizar pruebas aisladas y centradas en ellos. Para ello crearon un PBE más simplificado al real donde cada equipo consta de solo 2 Pokemon, uno activo y otro inactivo. Los jugadores tienen acceso a información limitada, como la vida de su Pokemon, el tipo y los tipos de sus movimientos, pero inicialmente no conocen los movimientos del oponente. Diseñaron el entorno de manera que los movimientos tuvieran un porcentaje de precisión

distribuido entre 50 y 100 y se aplicó un aumento del 50% cuando se realizaba un ataque del mismo tipo que el Pokemon activo. Por último, cada Pokemon tiene al menos un movimiento de su tipo y tres movimientos de tipos aleatorios que no son efectivos contra el rival.

En cuanto a los algoritmos que utilizaron para el entrenamiento, optaron por algoritmos basados en políticas mixtas, que son adecuados para estrategias estocásticas donde la probabilidad de realizar una acción es relevante. Es por esto que eligieron usar GIGA-WoLF y WPL, ya que ambos están específicamente dirigidos a entornos competitivos y se ha demostrado que permiten a los agentes converger hacia estrategias óptimas en juegos simples.

Para el entrenamiento, se empleó una red neuronal con 3 capas ocultas, cada una con 150 nodos y activación ELU [11]. Se utilizó una tasa de aprendizaje (learning rate) de 10^{-4} y un factor de descuento de 0.9. Ambos algoritmos se entrenaron mediante la técnica de self-play, que consiste en tratar de mejorar el desempeño de los agentes aprendiendo a jugar contra sí mismos. Se realizaron 2 millones de episodios de entrenamiento y se realizaron pruebas en dos entornos diferentes: uno con una versión simplificada del PBE con solo 7 tipos y otro con todos los tipos disponibles. Además, fueron evaluados en situaciones iniciales de desventaja frente a un enemigo aleatorio. También tomaron como guía un modelo que simulaba la decisión de un jugador experto, el cual maximiza el daño basado en el conocimiento del oponente. Ambos algoritmos mostraron buenos resultados en el entorno PBE simplificado, siendo GIGA el más rápido en alcanzar niveles altos de desempeño. Sin embargo, en el PBE completo, el algoritmo WPL no logró aprender de manera efectiva, mientras que GIGA, pese a su alta variabilidad, logró actuaciones más cercanas al nivel experto. [12]

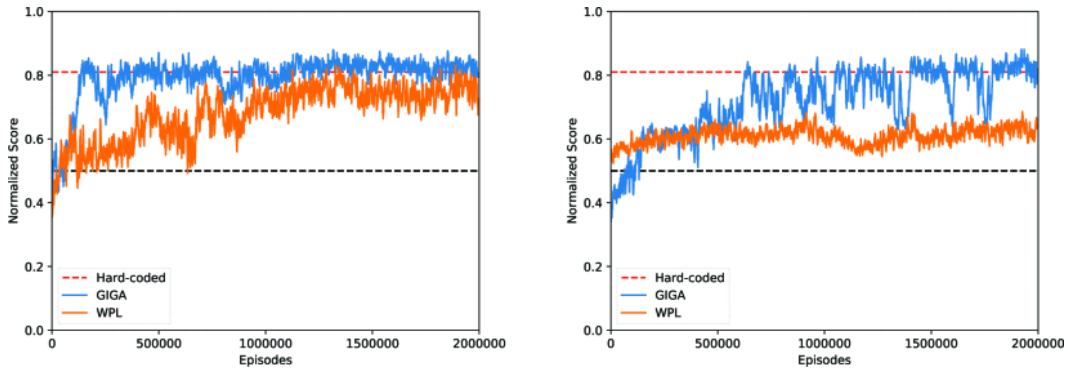


Figura 2.4: Resultados del entrenamiento de Pokemon. A la izquierda el entorno simplificado y a la derecha el entorno completo

Este proyecto demostró la viabilidad del deep reinforcement learning para comprender las ventajas de los tipos en Pokemon y establecer estrategias a largo plazo sin conocimiento previo. Sin embargo, se dejaron de lado otros aspectos importantes del juego, como los efectos de los estados, las habilidades y los ataques que suben o bajan las estadísticas de los Pokemon en combate. Por lo tanto, existe un amplio margen para mejorar y desarrollar un agente más completo en futuras investigaciones.

2.4.2 A Self-Play Policy Optimization Approach to Battling Pokemon

En este segundo proyecto se presenta otra iniciativa de aprendizaje por refuerzo enfocada en las batallas Pokemon, con la intención de afrontar la complejidad completa de estos enfrentamientos en lugar de las simplificaciones aplicadas en el proyecto anterior. Al igual que en el caso previo, se han basado en la estrategia de self-play para entrenar el modelo. La red neuronal empleada recibe como entrada el estado actual de la batalla desde el punto de vista del jugador, el cual está diseñado como una estructura compleja en forma de árbol donde la batalla está formada por: dos equipos, junto con una condición climática que también puede tener un impacto en el enfrentamiento, cada equipo consta de 6 Pokemon, y cada uno de estos Pokemon tiene una serie de características que se utilizan como entrada para la red neuronal. Estas características se detallan en la figura 2.5. Con toda esta información, la red neuronal produce dos salidas: una distribución de probabilidades sobre las posibles acciones a elegir y una estimación de la fuerza de la posición del agente en la batalla. En total, la red neuronal utilizada contiene un total de 1,327,618 parámetros.

Feature	Type	Dims	Description
species	categorical	1×1023	e.g. Pikachu
item	categorical	1×368	e.g. Leftovers, Choice Band
ability	categorical	1×238	e.g. Rough Skin, Shadow Tag
moveset	categorical	4×731	e.g. Flamethrower, Surf
lastmove	categorical	1×731	The latest move used
stats	continuous	6	HP, Atk, Def, SpA, SpD, Spe
boosts	continuous	6	Temporary boosts for stats
hp	continuous	1	Current number of hitpoints
maxhp	continuous	1	Number of HP at full health
ppUsed	continuous	4	# times a move was used
active	indicator	1	1 if Pokémon is active, else 0
fainted	indicator	1	1 if Pokémon has no HP, else 0
status	indicator	28	e.g. sleep, burn, paralysis
types	indicator	18	e.g. Bug, Fire
volatiles	indicator	23	e.g. Leech Seed, Perish Song

Figura 2.5: Lista incompleta de los parámetros de entrada a la red neuronal

Durante el entrenamiento, se llevaron a cabo m batallas utilizando la técnica de self-play, donde el agente se enfrentaba a sí mismo y cada 2m batallas se actualizaban los parámetros de la red neuronal. Asignaron recompensas de +1/-1 para representar las victorias/derrotas y, para acelerar el proceso de aprendizaje, construyeron una estructura de recompensas más densa asignando un menor porcentaje a otras acciones, como por ejemplo derrotar a un Pokemon o usar un ataque supereficaz. El entrenamiento se realizó en el formato de *gen7randombattle* de Pokemon Showdown, el cual genera equipos de Pokemon de forma aleatoria para cada batalla. Se jugaron un total de 500 episodios, donde se producían en cada uno 7680 combates, lo que implica que al final del entrenamiento realizaron casi 3840000 combates.

Para la evaluación enfrentaron al agente contra cuatro oponentes diferentes, cada uno con enfoques estratégicos distintos. El primer agente seleccionaba movimientos de forma aleatoria en cada turno. El segundo, seleccionaba el ataque con más potencia, sin tener en cuenta aspectos relevantes como la efectividad o la precisión. El tercer rival, similar al anterior, incorporaba el conocimiento de la efectividad de los ataques según el tipo de Pokemon. Y por último, se enfrentaría a un adversario basado en una heurística predefinida, el cual presentaría el nivel más alto de los agentes a enfrentarse.

Los resultados obtenidos por el agente entrenado fueron notables, como se muestra en la figura adjunta 2.6. El rendimiento del agente es excepcional al enfrentarse frente a los dos primeros agentes, que representarían un nivel inferior, logrando una tasa de victoria superior al 90% frente a ambos. Frente al tercer agente, sigue obteniendo unos muy buenos resultados superándolo en el 83% de las partidas, e incluso contra el agente final se obtienen resultados destacados por encima del 60%. Estos resultados demuestran la capacidad del agente entrenado para enfrentarse a una amplia variedad de oponentes.

Opponent	**Wins**	**Losses**
`random`	995	5
`most-damage`	929	71
`most-damage-typed`	829	171
`pmariglia`	612	388

Figura 2.6: Resultados del agente frente a los distintos rivales evaluados

3 Objetivos

Este TFG surge del profundo interés en el campo de la inteligencia artificial y tiene como objetivo principal adquirir un sólido conocimiento en una de las técnicas más populares en la actualidad: el aprendizaje por refuerzo. En este sentido, se plantearán una serie de objetivos generales que guiarán el desarrollo de este proyecto.

En primer lugar, se realizará una lectura de literatura en el campo de RL con el objetivo de obtener un sólido respaldo teórico sobre el tema. Para que, una vez adquirida una comprensión sobre sus fundamentos, aplicar esos conocimientos en la práctica mediante la creación de una IA capaz de jugar juegos clásicos o juegos de la consola de Atari. Los cuales han sido ampliamente utilizados en la comunidad de RL como entornos de prueba para evaluar el rendimiento de los algoritmos de aprendizaje automático.

En segundo lugar, el enfoque principal de este TFG se centra en la creación de una IA para jugar un videojuego. Sin embargo, la intención no es desarrollarlo en un entorno convencional, sino explorar territorios menos estudiados y más novedosos, con el objetivo de llevar a cabo un proyecto de investigación que sea relevante. Es por esto que después de un largo proceso de decisión entre varios videojuegos se tomó la decisión de entrenar un agente para jugar combates Pokemon. Esta elección se basa en diversos motivos. El primero es que es una de las franquicias de videojuegos más populares y, personalmente, una a las que mas tiempo se le ha dedicado jugando tanto en la GameBoy como en la Nintendo, por lo que existe un gran interés hacia ella.

Además, este proyecto representaría un desafío interesante, ya que tiene el potencial de ser innovador debido a la escasez de investigaciones previas en esta área específica y debido a sus características que lo hacen especialmente atractivas para el campo del RL. Más en detalle, las batallas Pokemon son un entorno parcialmente observable ya que cada entrenador solo conoce información sobre su equipo y algunas estadísticas visibles sobre los Pokemon activos del oponente. El espacio de estado es continuo y de alta dimensión, con seis Pokemon para cada entrenador, cada uno con uno o dos de diecisiete tipos diferentes, cuatro ataques por Pokemon, cada uno con su tipo también y con valores de potencia y precisión, y estadísticas para cada Pokemon. Este conjunto de propiedades requiere algoritmos que estén dirigidos a juegos complejos y que sean capaces de hacer frente a espacios de estado parcialmente observables, continuos y de alta dimensionalidad. Lo que hace este entorno ideal para probar y demostrar las propiedades de los mismos.

Más concretamente y como objetivos más específicos se quiere desarrollar un agente inteligente que sea capaz de adquirir el conocimiento necesario para jugar este tipo de combates y explorar las diferentes estrategias existentes. También será necesario definir el entorno de

juego del agente, identificando los espacios de estados y acciones, para garantizar una representación adecuada del problema. Así como estudiar y seleccionar el algoritmo más adecuado para este contexto y ajustar y optimizar el modelo de la red neuronal utilizada. Por último, se quiere conseguir guiar al agente hacia el comportamiento adecuado mediante el estudio de las recompensas y la exploración de diferentes enfoques en el entrenamiento.

4 Metodología

En este apartado se presentarán las diversas herramientas que se utilizaron en el desarrollo de este trabajo. Además se proporcionará una breve descripción de cada una de ellas con la intención de proporcionar una visión general de las tecnologías empleadas.

4.1 Máquina Virtual

Para realizar todo el proceso de desarrollo y experimentación de este TFG se empleó únicamente un portátil. Tras las primeras pruebas que se realizaron probando diferentes entornos y algoritmos de RL, surgieron una serie de errores relacionados con las librerías que se estaban probando. Sin embargo, parecían estar relacionados directamente con la configuración específica del portátil utilizado y no con la implementación interna de las mismas, por lo que se buscaron otras alternativas. Ya que para realizar este TFG se requería de un entorno de desarrollo estable, se decidió utilizar una máquina virtual para garantizar que se pudiera llevar a cabo la investigación de manera adecuada. Esta máquina virtual permitió utilizar una configuración estándar y asegurar que los errores relacionados con el portátil no afectarían negativamente los resultados.

4.1.1 VMware Fusion

VMware Fusion es un software de virtualización diseñado por VMware, Inc. específicamente para ordenadores Mac con procesador Intel. Este programa permite ejecutar diversos sistemas operativos, como Windows, Linux, NetWare y Solaris, de manera simultánea junto con el sistema operativo principal de Mac OS. Esta solución proporciona un entorno aislado para cada sistema operativo invitado, lo que permite ejecutar aplicaciones y acceder a recursos específicos de cada uno de ellos de forma eficiente y segura.

4.2 Anaconda

Anaconda es un software gratuito que proporciona un conjunto de herramientas diseñadas para la investigación y la ciencia. Permite tener acceso a diferentes entornos de programación, también conocidos como entornos de desarrollo integrado (IDE), con distintas versiones de Python o R. Además, incluye el gestor de paquetes Conda que permite la instalación y gestión librerías y dependencias, lo que facilita enormemente el desarrollo de código.

Para este TFG se empleará Anaconda como herramienta para establecer un entorno virtual independiente dentro de la máquina virtual. De esta forma, se conseguirá evitar posibles conflictos al trabajar con diferentes versiones de Python o al utilizar diversas dependencias que puedan estar previamente instaladas en el sistema.

4.3 Librerías

4.3.1 OpenAI Gym

OpenAI Gym es una biblioteca de Python de código abierto desarrollada para la investigación y el desarrollo de proyectos de reinforcement learning. Ofrece una API estándar para la comunicación entre algoritmos y entornos de aprendizaje, así como un conjunto de entornos que cumplen con esa API. Además contiene una colección de problemas de referencia que exponen una interfaz común y un sitio web donde las personas pueden compartir sus resultados y comparar el rendimiento de los algoritmos. Otro punto a favor es que ofrece compatibilidad con diversas librerías que incluyen algoritmos para el entrenamiento de modelos de RL.[13]

Por estas razones, se usará OpenAI Gym como herramienta principal con el fin de realizar las primeras experimentaciones en el ámbito del aprendizaje por refuerzo, aprovechando la amplia variedad de entornos disponibles que ofrece esta plataforma.

4.3.2 SB3

Stable Baselines3 (SB3) es tambíen una biblioteca de reinforcement learning de código abierto. Está diseñada para facilitar la implementación y evaluación de algoritmos de aprendizaje por refuerzo clásicos y avanzados. Ofrece una larga selección de algoritmos eficientes para entrenar agentes en diferentes entornos. Algunos ejemplos son: DQN, PPO, A2C, SAC... Además, incluye una guía de usuario tanto de nivel básico como otras más extendidas con diversos ejemplos hechos, lo que simplifica el entrenamiento y la comparación entre distintos algoritmos.[14]

Se utilizará esa biblioteca junto con OpenAI Gym para llevar a cabo los entrenamientos de los distintos agentes desarrollados. Su elección se debe a su larga colección de algoritmos de RL los cuáles resultan interesantes para la elaboración de este proyecto.

4.3.3 Poke-env

Poke-env es una interfaz de Python para entrenar bots de Pokemon. Nació con el objetivo de proporcionar un entorno de Python que sea capaz de interactuar en batallas de Pokemon Showdown, haciendo uso de aprendizaje por refuerzo. Ofrece una API simple para manipular los diferentes objetos que aparecen en un combate como los Pokemon, las Batallas, los Movimientos, etc. También expone una interfaz abierta de OpenAI Gym para poder entrenar agentes mediante técnicas de RL. Además, en su documentación se encuentran disponibles diversas guías que serán de utilidad para configurar el servidor donde se realizarán todos los experimentos, así como para la creación inicial de agentes dentro del entorno de juego. [15]

También cabe destacar que Poke-env cuenta con la implementación de tres tipos distintos de jugadores: RandomPlayer, MaxBasePowerPlayer y SimpleHeuristicPlayer. Estos jugadores predefinidos disponen de diferentes estrategias y niveles de habilidad, lo que resultará de gran utilidad para las futuras etapas del proyecto ya que permitirán evaluar y comparar los diversos modelos de entrenamientos desarrollados.

4.3.4 Pokemon Showdown

Pokemon Showdown es una web que permite simular combates de Pokemon. Es un software gratuito de código abierto escrito en TypeScript y JavaScript, con su infraestructura de servidor diseñada para ejecutarse en Node.js. Dentro de la gran variedad de modos de juego que implementa destaca el modo "Random Battles". Éste aleatoriza los equipos de ambos jugadores, equilibrándolos según los niveles. Este tipo de combates son los ideales para entrenar al futuro agente, ya que servirá para tener un entorno mucho más diversificado donde será difícil que se repitan combates.

Asimismo, permite tener un servidor alojado de manera local en el portátil sobre el cual se podrán realizar todos entrenamientos de los modelos. Es por esto que se presenta como una herramienta indispensable para poder realizar el proyecto.

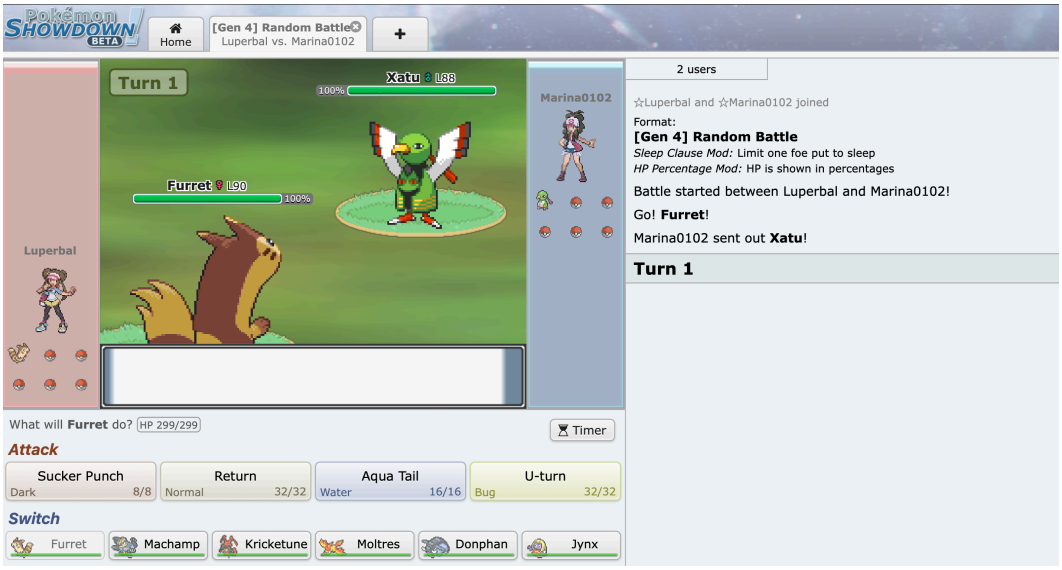


Figura 4.1: Servidor de Pokemon Showdown

5 Desarrollo

5.1 Experimentación inicial

5.1.1 Primeras pruebas con Gym y SB3

Antes de comenzar a trabajar directamente en la creación de una agente con la librería Poke-env, se decidió comenzar el proyecto explorando tanto la librería de Gym y de SB3. Como ya se ha mencionado anteriormente, ambas contienen una gran cantidad de frameworks con multitud de entornos ya configurados sobre los cuales se puede trabajar, así como diferentes ejemplos para demostrar el uso, las funciones de estas librerías y los distintos algoritmos que implementan para el entrenamiento de los modelos.

Esta decisión se eligió con el fin de familiarizarse con estas herramientas, ya que muchas no se habían tratado con anterioridad y era necesario tener una primera toma de contacto con ellas. Además, realizar este estudio permitió adquirir conocimientos fundamentales sobre cómo funciona el Reinforcement Learning y sus parámetros más importantes, como el entorno, la recompensa, el espacio de observación y de acciones, así como comprender la implementación de algoritmos de aprendizaje por refuerzo y todos sus métodos.

5.1.2 Cartpole

CartPole ha sido el primer juego que se ha utilizado como introducción en el RL ya que su funcionamiento es muy sencillo. El objetivo es simple: mantener el equilibrio de una barra vertical. Esta barra está unida por una articulación rotacional no actuada a una base móvil que se mueve a lo largo del eje horizontal sin fricción, haciendo que actúe como un péndulo invertido. Al comienzo de cada partida, el péndulo se coloca en posición vertical y el agente debe intentar equilibrar el poste moviendo la base hacia derecha o izquierda. Si no lo consigue, la barra se caerá. El espacio de acciones solo está compuesto por estas 2 opciones: el movimiento de la base hacia un lado o hacia otro. Respecto al espacio de observación, es posible conocer las siguientes 4 características: posición en el eje X y velocidad de la base, el ángulo de la barra respecto al eje Y y su velocidad angular. A la hora del proceso de aprendizaje, el agente recibe +1 punto de recompensa por cada paso que la barra mantenga el equilibrio, buscando siempre maximizar esta recompensa. Para terminar, el episodio concluirá cuando el ángulo de la barra respecto al eje Y sea mayor que +/- 12º, la posición de la base llegue al borde la pantalla o el episodio supere los 500 pasos.

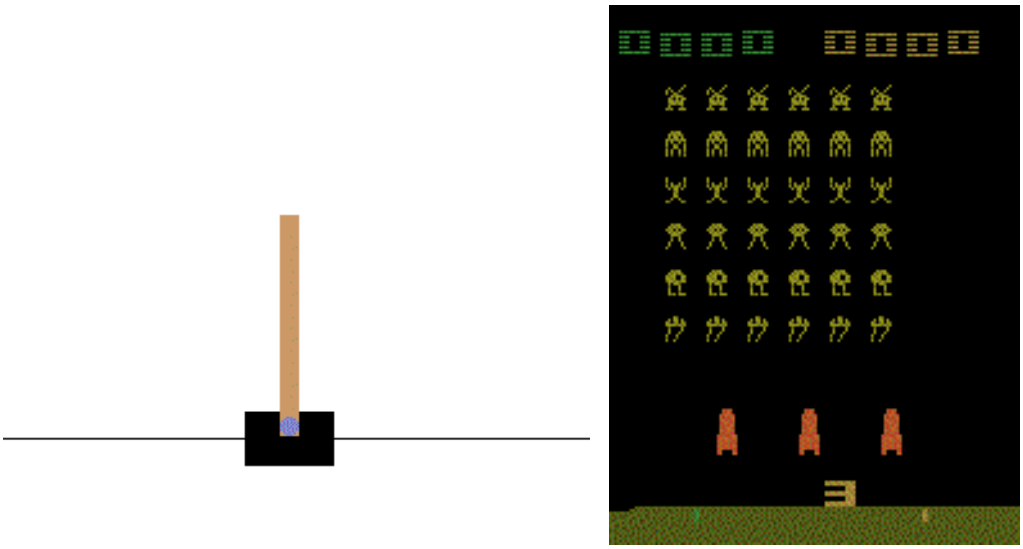
Aunque este entorno no sea desafiante, resulta muy útil para comprender los conceptos fundamentales del proyecto. Además, gracias a su espacio de acciones y observación reducido, es posible obtener resultados satisfactorios en un corto periodo de tiempo.

5.1.3 Space Invaders

Tras la experimentación previa, y una vez obtenidos los conocimientos básicos sobre estas librerías y su funcionamiento, se decidió entrenar un agente para que jugase en un juego clásico de la consola Atari como es Space Invaders. Este entorno es más complejo que el anterior y su principal objetivo es eliminar a todos los extraterrestres que descienden progresivamente mientras se evita ser alcanzado por sus proyectiles.

El espacio de observación del agente consta, por defecto, de una matriz de 210x160 píxeles en RGB que representa la pantalla del juego (adicionalmente se puede obtener también en escala de grises). En cuanto al espacio de acciones, se compone de seis acciones discretas que incluyen: mover la nave espacial a la izquierda o a la derecha, disparar haciendo una diferenciación entre hacerlo con el cañón de la derecha, de la izquierda o ambos o simplemente no hacer nada. El objetivo del agente es obtener la mayor puntuación posible eliminando a los enemigos y evitando ser alcanzado por sus ataques. El episodio termina si la nave espacial es alcanzada por un proyectil enemigo y no quedan vidas disponibles.

Este entorno es más complejo que CartPole, lo que lo convierte en una buena opción para experimentar. Durante el proceso de entrenamiento, se usaron y compararon tres algoritmos: DQN, PPO y A2C [16] [17] [18]. Después de varios entrenamientos y de ajustar los parámetros de sus redes, se observó que todos lograron alcanzar un nivel alto en pocas iteraciones.



(a) Cartpole **(b)** SpaceInvaders

Figura 5.1: Entornos de los juegos empleados para las primeras pruebas

5.2 Entorno final elegido - Pokemon

Dentro del amplio mundo de los videojuegos, en este TFG se centrarán los esfuerzos en crear un agente inteligente que sea capaz de librar combates Pokemon. En una batalla Pokemon un jugador compite con un equipo de hasta 6 Pokemon. Cada uno contiene un conjunto de estadísticas como pueden ser los puntos de vida (o en inglés Hit Points, HP), ataque, defensa o velocidad y hasta 4 movimientos (o ataques). En las batallas individuales cada jugador controla un Pokemon activo que se está enfrentando al Pokemon del rival y mantiene al resto en el banquillo. En cada turno se puede seleccionar uno de los 4 movimientos para atacar al rival o también se puede cambiar a cualquier Pokemon que se encuentre fuera del combate. Además, cuando un Pokemon se queda sin HP es obligatorio sustituirlo. Aparte de estas reglas, otro de los componentes centrales de este juego son los tipos, ya que cada Pokemon y sus movimientos poseen uno, y tienen diferente efectividad entre ellos. Por ejemplo, un movimiento de tipo Fuego es súper efectivo contra los Pokemon de tipo Planta, los de tipo Agua resisten los movimientos de tipo Fuego... La efectividad de los movimientos se traduce en una tabla de modificadores donde el multiplicador base es 1x, la debilidad tiene un 2x, la resistencia tiene un 0.5x y la inmunidad tiene un 0x.

Por todo lo comentado, este juego presenta propiedades muy interesantes que lo convierten en un entorno desafiante para el aprendizaje por refuerzo. Además, a diferencia de otros juegos, es un entorno estocástico y los ataques tienen la posibilidad de golpear y causar daño según una fórmula predeterminada. El esquema de recompensas es de suma cero, ya que las recompensas de los jugadores son simétricas y la única forma de ganar es a través de la derrota del oponente. Para lograr políticas exitosas en un periodo de tiempo razonable, los algoritmos deben generalizarse a nuevos estados, ya que la exploración de todo el espacio de estados sería demasiado costoso.

5.2.1 Primeros ejemplos de la librería Poke-env y primera clase Random Player

Gracias a las investigaciones de la sección 5.1, se adquirieron los conocimientos básicos necesarios para abordar un proyecto más complejo como entrenar una IA para competir en combates Pokemon. Ahora se tiene una comprensión completa de cómo interactuar con el entorno de un videojuego y de cómo funcionan los diferentes métodos típicos de los bucles de aprendizaje por refuerzo, como el step, reset y render. También se entiende la influencia de los diferentes parámetros de las redes neuronales, como las políticas, el número de pasos y el buffer de repetición, entre otros. Sin embargo, antes de comenzar a trabajar en la creación del agente de RL, se siguieron las guías que se encuentran en la documentación de Poke-env para aprender a crear diferentes jugadores en este entorno y enfrentarlos entre sí.

En primer lugar, se crearon varios objetos de la clase RandomPlayer cuyas decisiones son tomadas de forma aleatoria durante el transcurso de las batallas. De manera automática, estas instancias de la clase Player emplean un nombre de usuario y establecen conexión con el servidor local. Existe la posibilidad de personalizar tanto el formato del combate, que por defecto se encuentra en *gen8randombattle*, como la cantidad de combates simultáneos.

Esto sirvió para obtener una visión general de cómo funcionan los diferentes tipos de jugadores en este entorno y cómo interactúan entre sí.

5.2.2 Creación de la clase Max Damage Player

En este apartado se creó un agente desde cero a través de la herencia de la clase Player, con el objetivo de obtener los métodos base y así implementar el método "choose_move()" de manera personalizada (definido como un método abstracto en la clase base). Esta función recibe un objeto del tipo Battle, que representa el estado de la batalla y devuelve un string que indica el movimiento elegido.

El funcionamiento de dicha función es sencilla: si el Pokemon activo puede atacar lo hará utilizando el movimiento con mayor poder base disponible, sin tener en cuenta las características específicas del oponente o las condiciones de la batalla. De lo contrario, se realizará un cambio aleatorio. Estos datos se pueden obtener a través del objeto Batalla, que ofrece acceso a todos los objetos relevantes, como los Pokemon activos en el combate, los movimientos y los cambios disponibles. Cada uno de estos objetos posee sus propios parámetros, a los cuales también se pueden acceder y manipular.

Con todo esto ya se puede probar el primer agente básico y enfrentarlo contra otros jugadores. Al crear esta clase se ha podido comprobar la definición de los diferentes objetos que aparecen en el entorno de Pokemon Showdown y conocer los diferentes métodos básicos para acceder a sus diferentes características, lo que servirá para el futuro proyecto final de realizar el agente de aprendizaje por refuerzo.

5.2.3 Utilización de la clase Simple RL Player

Finalmente, se empleó el ejemplo de agente Simple RL Player, el cual hace uso de la interfaz proporcionada por OpenAI Gym, implementada en las clases EnvPlayer y OpenAI_API. La API de OpenAI Gym otorga tanto las observaciones como las recompensas para cada paso, que en este caso correspondería a cada decisión tomada en la batalla, y cada batalla completa constituye un episodio. Para llevar a cabo los primeros experimentos, se utilizaron las observaciones y recompensas predefinidas en la librería Poke-env, que si bien se planea cambiarlas y añadir nuevos parámetros para el entrenamiento del agente definitivo, estas pruebas permitieron comprender su funcionamiento para más adelante ser capaces de realizar modificaciones más específicas.

5.2.3.1 Observación

Las observaciones son una representación del estado actual de la batalla en forma de vector y pueden ser tan complejas como se desee. La clase Simple RL Player contiene un método llamado "embed_battle" para describir la observación. Además, también es necesario que ésta cumpla con las restricciones de la interfaz de Gym. Para ello existe otro método, "describe_embedding", donde se especifican los límites inferiores y superiores para cada componente del vector de observación y se devuelve como un objeto de la clase Gym.Space. El vector de observaciones descrito contiene:

- El poder base de cada movimiento disponible.
- El multiplicador de daño de cada ataque contra el Pokemon rival.
- El número de Pokemon vivos en el equipo rival.
- El número de Pokemon vivos en nuestro equipo.

5.2.3.2 Recompensa

Las recompensas son señales que el agente recibe del entorno en respuesta a sus acciones, y que se utilizan para guiar el proceso de aprendizaje. La clase EnvPlayer proporciona un método para ayudar a calcular estas recompensas, "reward_computing_helper", el cual tiene en cuenta factores como los Pokemon debilitados, los HP restantes, las condiciones de estado y la victoria o derrota. Las recompensas son simétricas, es decir, las acciones otorgan la misma cantidad de puntos, pero de manera negativa si la acción realizada es perjudicial. El sistema de recompensas funciona de la siguiente manera:

- La victoria otorga una recompensa positiva de 30 puntos.
- Derrotar a un oponente otorga 2 puntos.
- Restar un porcentaje de HP al rival corresponde a una recompensa positiva del mismo porcentaje.

5.3 Ensayos iniciales

Con todo lo explicado en la sección anterior, ya se tendría implementada la clase de nuestro agente por refuerzo. Por defecto, esta clase hereda de la clase Gen8EnvSinglePlayer, que define el espacio de acción que incluye las mecánicas de movimiento Z, Megaevolución y Dinamax. Este entorno será utilizado para entrenar el primer agente, permitiendo explorar la interacción con el servidor de Pokemon Showdown y experimentar con algunos parámetros de la red antes de pasar al entorno final.

Gracias a las pruebas realizadas en las etapas iniciales de este proyecto, se ha concluido que el algoritmo óptimo entre las diversas opciones disponibles en la biblioteca de SB3 para este entorno es el DQN (Deep Q-Network) [19]. Esta elección se fundamenta en que está pensado para entornos que involucran acciones discretas, lo cual se ajusta perfectamente a las necesidades y características particulares del entorno en cuestión.

5.3.1 Primeros entrenamientos: Generación 8

El formato de batalla utilizado en todas las pruebas realizadas en esta sección fue "[Gen 8] - Random Battle", el cual genera enfrentamientos donde los equipos de dos jugadores que participan en el combate son seleccionados de forma completamente aleatoria. Esta característica resulta muy beneficiosa ya que al ser todo al azar el agente se enfrenará a una amplia variedad de combinaciones de equipos y estrategias, lo que aumentará la diversidad de escenarios posibles durante el entrenamiento. Para estas primeras pruebas, no se realizaron

modificaciones en los hiperparámetros de la red neuronal utilizada en el entrenamiento, por lo que se entrenaron varios modelos con sus valores predeterminados y para ello, se crearon dos entornos, uno para el entrenamiento y otro para la evaluación.

La librería Poke-env implementa tres tipos diferentes de jugadores. Los dos primeros, RandomPlayer (RP) y MaxBasePowerPlayer (MBPP), tienen la misma lógica que los creados en las secciones 5.2.1 y 5.2.2. Mientras que el tercero, SimpleHeuristicPlayer (SHP), tiene un comportamiento más completo, contando con una heurísitca predefinida para tomar decisiones durante la batalla. Esta toma como referencia las estadísticas internas de los Pokemon activos en combate, teniendo en cuenta sus parámetros de ataque físico, ataque especial, defensa física, velocidad, etc, así como si tiene algún tipo de aumento o reducción en alguna de sus características. Con todos estos datos y una serie de operaciones matemáticas consigue determinar que ataque es mejor hacer en cada momento, o si por las propiedades de los Pokemon en combate sería mejor cambiar para tener un enfrentamiento más favorable. Estos agentes serán muy importantes para el proceso de entrenamiento ya que serán los empleados para la evaluación de los modelos entrenados.

Durante este primer entrenamiento, el agente se enfrentó únicamente al RandomPlayer y se entrenó hasta un máximo de medio millón de pasos. Para le evaluación, se creó una función para enfrentar al agente contra los tres jugadores explicados. Se decidió llevar a cabo un total de 500 combates, un número suficientemente alto para mitigar la influencia de la aleatoriedad en los resultados y, al mismo tiempo, evitar un exceso de tiempo en la evaluación

Los resultados obtenidos en este entrenamiento fueron muy interesantes. Como se puede apreciar en la tabla 5.1, en todos los entrenamientos por encima de los 75,000 pasos el agente logró obtener una tasa de victoria superior al 90% en las partidas contra RP. Sin embargo, los resultados contra MBPP y SHP fueron menos prometedores, con una tasa de victoria ligeramente superior al 50% frente al primero y unos resultados muy bajos frente al segundo, no llegando a superar el 15%.

Pasos	Tiempo (s)	Porcentaje de victorias en 500 batallas					
		RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
10000	90.07	32.4	-	6.0	-35.98	1.0	-42.47
50000	453.56	61.6	9.2	24.0	-20.24	2.6	-39.45
75000	793	95.2	36.08	50.4	5.34	9.2	-32.45
100000	1440.95	97.2	37.97	56.4	7.6	11.79	-31.71
150000	1774.95	95.8	37.56	55.0	7.09	11.4	-29.7
200000	2093.93	95.4	36.69	54.2	7.38	11.6	-29.05
300000	3428.30	95.8	37.38	56.8	8.09	12.5	-28.87
400000	4122.52	95.4	36.45	56.4	8.93	13.79	-27.93

Cuadro 5.1: Entrenamiento en Gen 8 frente a Random Player

Con el objetivo de analizar la importancia del oponente frente al que se entrena el agente y evaluar las diferencias en el rendimiento, se decidió realizar dos entrenamientos de 100000 y 200000 pasos, enfrentando al agente contra MBPP y contra SHP. No obstante, al examinar las tablas 5.2 y 5.3, se observa que los resultados fueron inferiores a los obtenidos en el entrenamiento inicial. Esto demuestra la necesidad de entrenar a nuestro agente frente a RP para adquirir los conocimientos básicos del juegos antes de enfrentarse contra rivales más experimentados.

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
100000	970.27	89.8	33.64	48.2	-0.52	9.6	-32.99
200000	2124.11	91.0	34.59	50.2	1.21	9.8	-32.06

Cuadro 5.2: Entrenamiento en Gen 8 frente a Max Base Power Player

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
100000	1198.74	80.8	29.34	42.8	-6.63	5.6	-26.99
200000	2220.17	83.0	30.89	44.6	-5.21	6.0	-26.06

Cuadro 5.3: Entrenamiento en Gen 8 frente a Simple Heuristic Player

5.4 Entorno definitivo: Generación 4

Después de llevar a cabo los primeros entrenamientos en la generación 8 y haberse familiarizado con el formato de batalla, así como haber entrenado y evaluado diferentes modelos, se ha conseguido una comprensión sólida acerca de la configuración de los entornos de entrenamiento. Después de este primer acercamiento, se decidió cambiar de entorno y entrenar en una generación distinta. Este cambio se fundamenta en la idea original del proyecto, que consistía en desarrollar un agente que jugase en la Generación 1 de Pokemon, la cual es reconocida por ser la primera generación con la que muchas personas comenzaron a jugar y una de las más populares. Además, en esta generación aún no habían nacido las mecánicas nuevas de las posteriores generaciones, por lo que se podrá trabajar con un espacio de acciones más reducido y obtener potencialmente mejores resultados.

Sin embargo, se encontró que la librería Poke-Env no ofrecía soporte directo para la Generación 1. Ante esta situación, se optó por utilizar la Generación 4 como alternativa, ya que cumple con las mismas especificaciones y características. Este cambio permitió continuar con el desarrollo del agente en un entorno que se asemejase lo más posible a la experiencia originalmente planeada.

Una vez establecido el entorno final para el entrenamiento de los agentes, se dio inicio al proceso de entrenamiento. Basándonos en las observaciones realizadas durante las experimentaciones anteriores, el oponente con el que el agente se enfrenta desempeña un papel crucial en el desarrollo de estrategias efectivas. Es por esto que se definieron 3 tipos diferentes de entrenamientos variando el vector de oponentes utilizado. Esta configuración tenía como objetivo permitir al agente enfrentarse a diferentes estilos de juego y estrategias dentro de un mismo bucle de entrenamiento, lo que le permitiría aumentar su adaptabilidad y capacidad de generalización. En primer lugar, se creó un vector de oponentes que incluía una instancia de cada uno de los tres jugadores disponibles en la librería. Esto ayudaría al agente a desarrollar un enfoque más equilibrado y flexible en la toma de decisiones durante los combates. En segundo lugar, se aumentó la cantidad de oponentes del vector y se ajustó la distribución de los mismos, estableciendo un entorno donde el agente se enfrentaría contra 3 RP, 2 MBPP y 1 SHP. Se pensó en este enfoque debido a que el nivel de dificultad de dos de los oponentes era mucho mayor, por lo que podría existir un desequilibrio que pudiera afectar a la efectividad del entrenamiento. Por último, se intentó aumentar la dificultad progresivamente, iniciando el entrenamiento contra RP y luego terminando contra SHP. Este enfoque permitiría al agente adaptarse gradualmente a diferentes tipos de oponentes, enfrentándose en primer lugar contra un agente aleatorio y terminar luchando contra estrategias más poderosas, lo que podría ayudarlo a ajustar el modelo de la red a medida que avanza el entrenamiento.

Al mismo tiempo, se optó por no realizar los entrenamientos de manera continua, sino hacerlo de manera iterativa. Es decir, se trató de implementar un modelo de entrenamiento gradual donde los modelos se guardaban y cargaban en intervalos regulares de pasos, lo que permitía observar de manera progresiva su evolución. Además, esto permitiría realizar análisis intermedios y efectuar ajustes si fuese necesario durante el entrenamiento. Los resultados obtenidos de los tres enfoques comentados se pueden ver en los siguientes cuadros:

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
50000	417.72	45.4	-2.15	9.8	-32.54	3.0	-40.21
100000	+465.90	45.6	-2.02	11.2	-31.97	2.4	-40.43
150000	+519.61	46.8	-1.74	6.0	-34.74	3.0	-39.76
200000	+487.98	47.2	-1.50	7.6	-33.98	4.6	-38.55

Cuadro 5.4: Entrenamiento 1a distribución

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
50000	435.77	65.6	12.13	16.0	-27.08	1.6	-39.987
100000	+452.08	70.2	17.00	15.2	-26.64	3.8	-38.47
150000	+425.85	67.6	14.14	14.4	-27.77	2.0	-39.08
200000	+446.14	66.5	13.75	16.6	-25.99	3.6	-38.24

Cuadro 5.5: Entrenamiento 2a distribución

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
50000	448.28	63.2	10.85	14.4	-27.39	2.4	-39.20
100000	+440.07	68.0	14.25	19.0	-24.29	3.2	-38.74
150000	+459.13	65.6	13.34	19.6	-23.36	1.6	-39.83
200000	+409.43	64.4	12.18	17.6	-24.78	5.0	-37.21

Cuadro 5.6: Entrenamiento 3a distribución

Como se puede observar, los entrenamientos realizados fueron relativamente cortos. Se evidenció un bajo desempeño del agente, por lo que se planteó la necesidad de realizar un estudio para identificar los posibles fallos. En un principio, se barajó la hipótesis de que la variabilidad en los oponentes podría haber influido en los resultados, ya que el hecho de enfrentarse a diferentes tipos de jugadores generaba dudas sobre la influencia de cada uno en el proceso de entrenamiento. También se planteó la posibilidad de que existieran errores en el código que afectaran al guardado adecuado del modelo de red neuronal. O que simplemente fuese necesario un proceso más largo para poder explorar y aprender de las diversas situaciones y estados del juego. Para corroborar estas ideas se decidió realizar un estudio en detalle de la documentación de la librería utilizada, donde se examinaron detenidamente los métodos y parámetros del algoritmo empleado. Gracias a este análisis, se encontró que el error se debía a que DQN hace uso de un buffer de repetición como estructura de datos para almacenar y organizar las transiciones de experiencia recolectadas durante el entrenamiento y ésste se guarda por separado de la red neuronal. Es por esto que sería necesario guardarlo después de entrenar, para que en caso de querer volver a entrenar el modelo, el agente contase con la información necesaria para continuar el entrenamiento desde el punto en el que se había dejado.

Ahora, después del proceso de cada entrenamiento se garantizó que tanto el modelo de la red neuronal como el buffer de reproducción fueran guardados y cargados conjuntamente en futuras instancias de entrenamiento. Estos nuevos resultados se pueden ver en las Tablas siguientes.

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
50000	409.24	48.2	-1.15	9.0	-33.01	3.4	—39.40
100000	+489.21	48.6	-1.06	10.2	-32.76	4.6	-37.99
150000	+504.89	47.6	-	7.4	-33.43	3.0	-40.01
200000	+506.02	-48.4	-1.10	10.8	-32.24	5.0	-37.05

Cuadro 5.7: Entrenamiento 1a distribución añadiendo el buffer de repetición

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
50000	440.80	61.0	9.82	18.8	-23.73	4.0	-37.43
100000	+449.1	58.4	7.72	17.4	-24.21	4.0	-37.48
150000	+487.79	58.6	8.72	19.4	-23.498	7.0	-35.50
200000	+434.54	62.2	10.56	22.00	-21.31	-	-

Cuadro 5.8: Entrenamiento 2a distribución añadiendo el buffer de repetición

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
50000	430.18	52.0	2.97	16.4	-26.65	3.0	-39.43
100000	+448.6	53.0	3.22	16.6	-26.18	4.0	-38.36
150000	+453.36	-	1.09	12.6	-29.06	3.0	-39.13
200000	+486.97	52.0	2.77	13.3	-28.73	3.4	-38.67

Cuadro 5.9: Entrenamiento 3a distribución añadiendo el buffer de repetición

Como se observa, se evidenció una persistente falta de mejoras pese a haber cargado el buffer de repetición durante cada entrenamiento. Este problema se atribuyó al parámetro "learning_starts" del algoritmo DQN, el cual establece la cantidad de pasos iniciales necesarios para recolectar información antes de que el modelo comience a aprender. El valor predeterminado de este parámetro es de 50000 pasos, que coincide justo con el momento en que se guarda el primer modelo. Entonces se pensaba que al cargar un modelo cuyos resultados eran bajos, el modelo no habría aprendido patrones de juego, lo que limitaría su capacidad para mejorar significativamente durante los entrenamientos posteriores. En consecuencia, se decidió realizar un entrenamiento más largo para evitar este problema, mientras que paralelamente se revisarían los hiperparámetros del algoritmo para tratar de ajustarlos lo máximo posible.

Los resultados presentados en las próximas tablas, 5.10, 5.11 y 5.12, revelan una notable mejoría en el nivel de desempeño. Sin embargo, se pueden apreciar varias aspectos reseñables. En primer lugar, destaca la consistencia de los resultados obtenidos, los cuales muestran poca variación entre los distintos modelos, y además, sorprende que el rendimiento del primer entrenamiento es muy similar al de los siguientes. Esto sugiere que la recolección de información durante un gran número de pasos, como se establece en el parámetro "learning_starts" del modelo, es demasiado alto y se puede disminuir para ajustarlo de manera más precisa. También se puede observar que a pesar del aumento en la duración del entrenamiento no se obtuvo una mejora en los resultados. Esto indica que podrían ser necesarios enfoques alternativos o un mayor tiempo de entrenamiento para lograr mejoras adicionales.

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
75000	723.36	94.2	36.87	68.6	15.15	27.2	-15.99
150000	+698.88	96.4	37.94	70.0	15.71	30.0	-15.04
225000	+737.69	94.6	36.93	69.6	15.88	31.6	-15.98
300000	+934.26	94.0	36.20	70.8	16.07	29.0	-15.98

Cuadro 5.10: Entrenamiento 1a distribución aumentado la duración de los entrenamientos

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
75000	779.51	97.8	38.57	69.0	14.93	30.0	-14.49
150000	+761.33	98.0	39.17	69.4	14.91	27.2	-16.08
225000	+802.30	95.0	36.86	70.2	16.12	29.0	-15.05
300000	+629.13	96.43	38.08	73.0	17.84	29.0	-15.08

Cuadro 5.11: Entrenamiento 2a distribución aumentado la duración de los entrenamientos

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
75000	683.28	95.6	37.53	66.0	13.025	26.6	-16.86
150000	+669.51	96.0	37.98	67.6	14.61	29.8	-14.8
225000	+658.06	95.2	37.12	70.6	16.14	27.6	-16.64
300000	+669.48	94.8	36.93	65.2	12.19	28.2	-16.36

Cuadro 5.12: Entrenamiento 3a distribución aumentado la duración de los entrenamientos

Como se mencionó con anterioridad, durante el desarrollo de los entrenamientos previos se llevó a cabo una investigación paralela para identificar posibles causas de los malos resultados en la segunda serie de entrenamientos. A raíz de este análisis, se descubrió cual era el motivo. El método utilizado por DQN para el entrenamiento incluye un parámetro booleano que indica si se debe reiniciar o no el contador interno de pasos del modelo, que por defecto está en True. Este parámetro no se le estaba pasando a la función por lo que cada vez que se iniciaba un nuevo entrenamiento el contador de pasos volvía a 0. En particular, en los experimentos mencionados, no se especificó ningún valor de "learning_starts" por lo se inicializaba automáticamente a 50000, y se cargaban y reentrenaban modelos en esa misma cantidad de pasos. Sin embargo, debido a que el contador de pasos se reiniciaba en cada entrenamiento y al no pasarle el parámetro mencionado, los modelos nunca superaban los 50000 pasos de entrenamiento. Como resultado el agente solo actuaba de manera aleatoria ya que se estaba recolectando de nuevo información del entorno durante todo el tiempo del entrenamiento. Esto explicaba la falta de progresión en los resultados y proporciona una solución clara de como solventar el problema.

Gracias a esta comprensión y a las conclusiones obtenidas tras la tercera serie de entrenamientos, se tomó la decisión de llevar a cabo un último entrenamiento con el objetivo de cerrar esta sección, donde se solucionarían todos los errores previamente identificados, al tiempo que se ajustarán cuidadosamente los hiperparámetros de la red para lograr una mayor optimización en el modelo final. Además, se decidió extender la duración de los entrenamientos para permitir un aprendizaje más profundo y completo. Los resultados finales de este último entrenamiento se presentan en las tablas 5.13, 5.14 y 5.15, donde se puede apreciar el rendimiento del agente tras la implementación de las mejoras.

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
10000	110.92	89.6	32.02	55.0	4.01	18.2	-24.27
30000	+201.76	98.2	39.42	67.0	13.39	28.0	-16.10
50000	+216.71	98.0	39.82	71.0	16.35	32.2	-12.59
100000	+508.06	97.4	38.65	70.0	15.57	31.6	-12.88
200000	+973.30	96.8	39.00	70.4	16.34	30.6	-14.86
500000	3019	95.0	-	70.0	15.91	29.8	-15.48

Cuadro 5.13: Entrenamiento final 1a distribución

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
10000	86.01	94.6	35.75	61.2	8.53	24.4	-19.83
30000	+234.31	95.6	37.71	71.0	17.99	29.6	-15.01
50000	+237.94	95.6	37.18	68.0	15.94	27.4	-16.22
100000	+628.91	95.2	37.04	67.2	13.89	30.8	-14.06
200000	+1251.32	94.0	36.65	64.8	11.95	26.6	-17.78
500000	+3611.38	95.8	37.52	67.8	16.26	29.0	-15.95

Cuadro 5.14: Entrenamiento final 2a distribución

Pasos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia	SHP (%)	Recompensa Promedia
10000	103.16	85.2	29.49	45.0	-3.25	12.2	-29.55
30000	+206.15	98.0	39.15	67.8	13.84	30.6	-14.46
50000	+196.18	93.6	36.28	75.2	19.26	31.8	-13.37
100000	+499.47	95.2	37.72	68.0	14.12	36.0	-10.27
200000	+964.48	92.8	35.46	64.0	11.38	33.2	-12.14
500000	+2917.29	83.4	27.51	50.6	0,812	18.0	-24.64

Cuadro 5.15: Entrenamiento final 3a distribución

Para llevar a cabo estas pruebas, se estableció un valor fijo de learning_starts=5000 y como se puede apreciar en los resultados, se mantiene un alto nivel de rendimiento en todas las distribuciones evaluadas. Aunque los resultados son muy similares entre sí, es posible extraer conclusiones importantes de los diferentes enfoques utilizados. En cuanto a la primera distribución, destaca por su mayor estabilidad y consistencia a lo largo del tiempo de entrenamiento. Esto se atribuye a la estrategia adoptada de entrenar al agente contra tres tipos diferentes de jugadores, asegurando que todos tengan igual probabilidad de aparecer, lo que contribuye a una mayor robustez en los resultados obtenidos. Por otro lado, la segunda tabla muestra un rápido progreso hacia niveles de rendimiento aceptables, aunque no logra superar los resultados más altos en estas comparativas. Este fenómeno es comprensible, ya que en este caso la distribución no era equitativa y existían mayores probabilidades de enfrentar al agente contra jugadores más débiles durante el entrenamiento. Esto facilitaría la adquisición de estrategias básicas en las etapas iniciales pero a largo plazo esta distribución desigual podría limitar la exploración de tácticas más complejas debido a la relativa falta de desafío por parte de los oponentes. Por último, en el enfoque final se pueden apreciar los valores más altos de rendimiento, como se evidencia en el modelo de 50000 pasos con un 75% de victorias contra el agente MaxBasePowerPlayer y un 36% frente al jugador que utiliza la heurística en el siguiente modelo. Sin embargo, también se observa que a largo plazo, entrenar el modelo contra un único agente conlleva el riesgo de especializarse en una estrategia particular. Lo que provocaría que no obtuviese resultados más altos contra otros jugadores y se traduciría en un problema de falta de generalización, como se puede ver en la última fila de la tercera tabla.

En resumen, los resultados obtenidos han sido altamente satisfactorios. En estas se ha demostrado un desempeño más que adecuado frente a la mayoría de los agentes evaluados. No obstante, se quieren seguir mejorando los resultados obtenidos, especialmente en lo que respecta al rendimiento contra el jugador SHP. A través de la recopilación y análisis de todas estas experiencias se pueden diseñar estrategias más sofisticadas y explorar nuevas técnicas de entrenamiento con el objetivo de alcanzar resultados aún más completos.

5.5 Vectorización del entorno

Tras alcanzar resultados aceptables, surgió el interés de mejorar aún más el proceso de entrenamiento. Es por esto que después de mucha investigación se consideró la implementación de la vectorización de entornos como una estrategia para potenciarlos. Ésta es una técnica muy utilizada poder ejecutar múltiples instancias de un entorno de forma simultánea y/o paralela. En lugar de ejecutar un solo entorno de manera secuencial, la vectorización permite ejecutar varios entornos durante un mismo proceso de entrenamiento. Esto se logra mediante la creación de un entorno *Wrapper* que administra múltiples instancias del entorno de manera eficiente y se encarga de distribuir las acciones a cada instancia del entorno y recopilar las observaciones, recompensas y estados resultantes de cada una. Este enfoque aporta una gran serie de ventajas como pueden ser la mayor diversidad en la experiencia al tener múltiples instancias de entornos, ya que cada una puede encontrarse en un estado diferente, lo que enriquece la variedad de muestras. Además, si se trabaja de manera paralela se mejoraría la eficiencia computacional y permitiría recolección de muestras más rápida, lo que podría acelerar la convergencia del modelo.

Stable Baselines 3 proporciona dos métodos principales para vectorizar entornos: DummyVecEnv y SubprocVecEnv. La diferencia entre ambos radica en cómo manejan la paralelización del entorno. Mientras que DummyVecEnv es una implementación de un *wrapper* que ejecuta los entornos de forma secuencial en un solo proceso, SubprocVecEnv aprovecha la paralelización al ejecutar múltiples instancias de un entorno en procesos separados y paralelos. Cada uno se encarga de ejecutar un entorno de manera independiente y se comunican entre sí para recopilar y sincronizar información. Ambos métodos son útiles en diferentes contextos, pero en este caso particular, se trataría de trabajar con el segundo de ellos para aprovechar las ventajas de la paralelización y conseguir así de acelerar el proceso de entrenamiento.

Sin embargo, a pesar de los múltiples intentos, fue imposible utilizar SubprocVecEnv debido a un error recurrente al ejecutar el código de entrenamiento. Tras investigarlo se encontró que se debía a que este método crea varios procesos para ejecutar los entornos en paralelo, por lo que se necesitaba serializar estos objetos antes de compartirlos entre procesos. La serialización consiste en convertir un objeto en una secuencia de bytes para que pueda ser almacenado o transmitido a través de la red. Este mecanismo es el que daba el problema ya que trataba de convertir a bytes un objeto que contenía un bloqueo. Bloquear un hilo es una práctica habitual para sincronizar el acceso a recursos compartidos en entornos de programación multihilo, para así evitar problemas de concurrencia garantizando una ejecución coherente y predecible del programa. Por tanto, el error se producía al intentar serializar un bloqueo el cual no tiene una representación directa en bytes, por lo que python no podía realizar la operación y generaba el problema mencionado. Para solucionar este error, se decidió cambiar a DummyVecEnv.

Pese a que con este nuevo método no se pudiese aprovechar el paralelismo, su uso seguía siendo útil en este proyecto ya que aunque se ejecuten los entornos de manera secuencial, sería posible entrenar diferentes entornos de forma simultánea aplicando enfoques distintos o variando los oponentes a los que se enfrenta el agente. Al tener múltiples entornos en funcionamiento, cada uno con su propia configuración, estrategias y oponentes, se podría obtener una mayor cantidad de muestras y experiencias en un periodo de tiempo determinado. Esto permitiría explorar un rango más amplio de situaciones y adaptar mejor las políticas de acción del agente a diferentes escenarios, lo cual ayudaría a reducir la variabilidad en el aprendizaje.

Inicialmente, se comenzó trabajando con 50 y 100 entornos en el entrenamiento, sin embargo, los resultados obtenidos fueron bastante decepcionantes. Para abordar este problema, se realizó un análisis para determinar si era necesario ajustar diversos hiperparámetros del algoritmo. Se consideraron aspectos como la frecuencia de actualización del modelo, el intervalo de tiempo para actualizar la función objetivo y el tamaño del buffer de repetición, con el fin de mejorar la estabilidad y los resultados del entrenamiento.

Pese a realizar estas pruebas y dedicar a ellas una gran cantidad de entrenamientos de entre medio millón y un millón de pasos, los resultados no cumplieron las expectativas. Los mejores resultados que se obtuvieron fueron solo un 80% de victorias frente al RandomPlayer y sin superar el 45% frente al MaxBasePowerPlayer.

Con el fin de solucionar este problema, se llevaron a cabo una serie de experimentos utilizando un número reducido de entornos, que osciló entre 2 y 6. El propósito era evaluar el desempeño de la técnica de vectorización a una escala más pequeña y observar si existía algún indicio de efectividad, para que en caso afirmativo, ir incrementando gradualmente el número de entornos hasta alcanzar una cantidad significativa. Los resultados obtenidos tras este estudio mostraron una tendencia preocupante. Como se puede apreciar en las tablas 5.16 y 5.17, a medida que se aumentaba el número de entornos, el porcentaje de victorias frente a los jugadores evaluados mostraba una tendencia descendente de manera gradual. El motivo de este problema era que solo se contaba con un único servidor de Pokemon Showdown así que todos los entornos actuaban sobre el mismo combate e interferían entre ellos. Frente a estos resultados se decidió no seguir adelante con el uso de esta técnica y explorar alternativas distintas.

learning_starts=30k - total_timesteps=100k					
Número de Entornos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia
1	1005.72	97.33	42.81	52.33	2.61
1	950.44	94.33	40.30	62.0	11.48
2	827.83	93.66	38.74	60.66	9.21
2	856.40	94.33	38.9	56.99	6.66
3	766.25	93.33	38.61	52.33	2.57
3	758.04	90.66	36.54	50.0	0.53
4	-	88.33	34.36	47.0	-0.92
4	-	91.0	35.7	40.66	-6.74
5	759.25	87.33	32.48	41.33	-5.92
5	726.88	91.66	36.50	44.66	-4.05
6	738.75	71.66	20.76	27.6	-17.81
6	735.41	73.91	22.65	30.3	-15.46

Cuadro 5.16: Entrenamiento aplicando la vectorización de los entornos realizando un entrenamiento de 100k pasos

learning_starts=5k - total_timesteps=50k					
Número de Entornos	Tiempo (s)	RP (%)	Recompensa Promedia	MBPP (%)	Recompensa Promedia
1	489.08	95.0	41.18	69.66	16.44
2	401.99	93.66	38.90	56.99	6.70
3	455.84	89.66	34.99	44.33	-3.99
4	410.14	85.66	31.34	42.66	-5.75

Cuadro 5.17: Entrenamiento aplicando la vectorización de los entornos realizando un entrenamiento de 50k pasos

5.6 Subida gradual de los oponentes

Ante la falta de éxito en la vectorización del entorno se exploraron diversas estrategias para continuar con el proyecto. Entre las opciones consideradas, destacaron dos enfoques distintos. El primero consistía en aumentar gradualmente la dificultad de los oponentes, evitando repetir los problemas experimentados en la sección 5.4. Mientras que el segundo consistía en en emplear el método de self-play para el entrenamiento. Finalmente, se optó por la primera opción, ya que parecía una solución más sencilla de implementar y se tenía la expectativa de obtener resultados satisfactorios. En caso de que esto no fuera así, se podría valorar la opción de cambiar la estrategia.

Este planteamiento seleccionado se basa en los resultados y conclusiones obtenidos en los entrenamientos previos, con el objetivo de elevar gradualmente el nivel de los enfrentamientos sin reincidir en los errores del pasado. La idea consiste en entrenar al agente contra oponentes de menor nivel y, de manera cuidadosa, ir aumentando progresivamente la dificultad. Sin embargo, en lugar de establecer un número fijo de pasos para cambiar de rival se consideraron otros aspectos como el total de victorias, el porcentaje de victorias en las últimas partidas o la recompensa promedio obtenida. De esta manera, al comenzar el entrenamiento contra el oponente más sencillo el agente adquirirá los conocimientos básicos necesarios para jugar, y se iría cambiando de oponente evitando la especialización excesiva frente a un único rival, como sucedió en pruebas anteriores. Para ello se llevaron a cabo pruebas iniciales utilizando episodios de entrenamiento de 50000 pasos y posteriormente se evaluaría el rendimiento del agente frente al mismo rival contra el que se había estado entrenando, realizando un total de 300 partidas. Si los resultados obtenidos superaban un umbral de porcentaje predefinido, se procedía a cambiar de oponente y se iniciaba un nuevo proceso de entrenamiento contra el siguiente agente. Este proceso se repetiría hasta lograr alcanzar un rendimiento satisfactorio frente al último oponente, SimpleHeuristicPlayer.

Tras las primeras pruebas los resultados obtenidos no cumplieron las expectativas esperadas. Se establecieron porcentajes objetivo del 90% para RP, 80% para MBPP y 50% para SHP. Pero tras ejecutar numerosos episodios y evaluar al agente muchas veces, el modelo no lograba superar al agente MBPP. Además, debido a que el agente se entrenaba en un gran número de partidas contra MBPP, se estaba volviendo demasiado especializado y acabó perdiendo la capacidad de generalizar, lo cual era contrario al objetivo deseado. Con el fin de solventar este problema se implementaron dos cambios significativos para la siguiente tanda de entrenamientos. En primer lugar, se decidió crear un agente intermedio entre RP y MBPP con el fin de suavizar la transición entre ambos y fomentar la adopción de estrategias más sólidas antes de enfrentarse a este último. Este nuevo oponente se construiría utilizando la clase Player de Poke-env y modificando el método que elegiría la acción a realizar. En lugar de seleccionar el movimiento con mayor potencia de daño, el agente elegiría entre los cuatro ataques disponibles aquel que tuviera la menor potencia, excluyendo aquellos con potencia 0. Además, se introdujo un factor *aplha* para permitir que la acción escogida pudiese ser aleatoria en ciertos casos. Esto no siempre implicaría una opción peor, ya que existe la posibilidad de que la elección aleatoria resulte en un ataque con mayor potencia, lo que mejoraría la elección base.

Con la incorporación de este nuevo agente intermedio, se introdujo el segundo cambio pensado. En lugar de seguir con el enfoque anterior de entrenar contra un oponente específico hasta alcanzar el porcentaje requerido, lo cual había demostrado dar resultados insatisfactorios, se implementó una planteamiento más dinámico que tuviera en cuenta la cantidad de partidas ganadas frente a un rival antes de cambiar de oponente, sin la necesidad de interrumpir el proceso de entrenamiento. Más específicamente, se estableció un umbral fijo y si el agente lograba ganar un porcentaje de partidas superior a ese umbral, cambiaría de oponente y se reiniciarían los contadores de batallas y victorias. Pero si el porcentaje obtenido estaba por debajo de un umbral inferior, el agente volvería a enfrentarse al oponente anterior. De esta manera, se evitaría la pérdida de generalización al estar constantemente cambiando de oponente durante el entrenamiento.

Con esta nueva práctica se pensaba que se permitiría al agente adaptarse de forma más flexible y progresiva a diferentes niveles de dificultad y evitando la especialización frente a un único oponente. También se buscaba encontrar un equilibrio entre la exploración de diferentes estrategias para poder hacer frente a los rivales mas complejos. Sin embargo, a pesar de las mejoras implementadas los resultados obtenidos no cumplieron las expectativas. Aunque se observaron mejoras respecto a los intentos anteriores, el agente seguía encontrándose con dificultades para superar el umbral establecido frente al oponente SHP. Como resultado, el agente se encontraba atrapado en un ciclo constante entre MBPP y SHP, sin lograr progresar más allá de esa etapa. En esta situación el problema radicaba en la determinación adecuada de los umbrales para el cambio de oponente. Si el umbral superior se fijaba en un valor demasiado alto, existía el riesgo de que el agente nunca alcanzara ese nivel requerido, lo que lo mantendría estancado en su progreso. Por otro lado, establecerlo en un valor demasiado bajo provocaría un cambio constante entre oponentes, sin permitir al agente consolidar las estrategias necesarias antes de avanzar hacia desafíos más complejos. Estos mismos problemas surgían al tratar de seleccionar el umbral inferior. Si se establecía muy bajo, existía la posibilidad de que el agente no se enfrentara nuevamente a oponentes de menor nivel, lo cual podría ser necesario para mejorar sus estrategias antes de enfrentar a los oponentes más exigentes. Por otro lado, un umbral inferior demasiado alto impediría al agente aprender nuevas estrategias contra oponentes de mayor nivel, ya que cualquier bajo desempeño resultaría en un descenso en la calidad del oponente.

Ante los obstáculos encontrados a la hora de seguir esta estrategia de entrenamiento, los cuales requerirían un análisis detallado y extensas pruebas para determinar los umbrales adecuados y posiblemente la creación de un nuevo oponente intermedio entre los dos oponentes de nivel más alto, se tomó la decisión de explorar la técnica del self-play como una alternativa prometedora. Esta técnica se basa en enfrentar al agente contra diferentes versiones de sí mismo, lo que permitiría un entrenamiento escalonado y gradual. De esta manera se eliminaría la dependencia de encontrar umbrales específicos y crear oponentes intermedios, simplificando el proceso de entrenamiento. Además, aportaría la ventaja de conseguir una adaptación continua y progresiva a medida que avanzaba el entrenamiento.

5.7 Self-Play

Más en detalle, la técnica de self-play es una estrategia ampliamente utilizada en el aprendizaje por refuerzo, donde un agente se enfrenta y aprende a jugar contra versiones anteriores de sí mismo. Es decir, en lugar de enfrentare contra oponentes previamente definidos o contra jugadores humanos, el agente juega contra copias de sí mismo que tienen diferentes políticas de toma de decisiones. Esta técnica es la misma que se empleó para el entrenamiento de los proyectos mencionados en el marco teórico (2.4.1 y 2.4.2) donde sus resultados obtenidos fueron satisfactorios, por lo que se espera conseguir lo mismo en esta sección final. Entonces, en este trabajo se implementó esta técnica de la siguiente manera: inicialmente el agente se enfrentará contra el jugador RandomPlayer, que realiza movimientos aleatorios, y se llevan a cabo m batallas donde se acumulan experiencias y se desarrollan nuevas estrategias. Con esta base, el agente actualiza su política de toma de decisiones y en el siguiente episodio se enfrentaría a dos oponentes, otro agente RandomPlayer y una versión actualizada de sí mismo. En el próximo episodio el rival de menor nivel, en este caso el RandomPlayer, se sustituirá por una versión más reciente del modelo, y así sucesivamente, por lo que siempre habrán dos oponentes durante el bucle de aprendizaje, uno de un nivel relativamente mayor a otro. Durante cada episodio el agente aprende y mejora continuamente, lo que le permite aprender de sus errores y mostrar una tendencia al alza en la mejora de su rendimiento. De esta manera se conseguiría un entrenamiento verdaderamente gradual, no como en los intentos anteriores, ya que a medida que avanza el entrenamiento se iría enfrentando a oponentes de mejor nivel, pero sin grandes diferencias de habilidad. Además, al enfrentarse a sí mismo el agente no depende de jugadores externos y puede generar una cantidad ilimitada de datos de entrenamiento que le servirán para su aprendizaje.

Este proceso de entrenamiento se realizará durante n iteraciones, tras las cuales se evaluará el desempeño del agente frente a los tres agentes establecidos. Pero antes de iniciar este proceso se realizarán ajustes en las observaciones recibidas por el modelo y se afinarán las recompensas con el objetivo de lograr que el agente desarrolle una estrategia más efectiva.

5.7.1 Cambios en la observación y en la recompensa

Como se explicó en la sección 5.2.3, la clase utilizada para entrenar al agente definía unas observaciones y unas recompensas iniciales, Con el fin de guiar al agente hacia el desarrollo de estrategias más complejas, se decidió modificarlas.

En relación a la observación, se agregaron dos componentes al vector que representan el porcentaje de vida restante tanto del Pokemon activo del agente como del oponente, para proporcionarle información sobre el estado de salud de los Pokemon involucrados en el combate. Adicionalmente se añadieron catorce posiciones extra, de las cuales doce corresponden a los tipos de los Pokemon en el equipo del agente (el doble de la cantidad de Pokemon en un equipo debido a que cada uno puede tener hasta dos tipos), mientras que las dos restantes indican el tipo del Pokemon activo del rival. Esta información permitiría al agente tener conocimiento sobre los tipos de Pokemon presentes en la batalla para así seleccionar los movimientos más efectivos. Cabe destacar que internamente los tipos de Pokemon se encuentran definidos como una enumeración, en la cual cada tipo está asociado a un número del 1 al 18.

Tras llevar a cabo varias pruebas para evaluar la relevancia de los cambios introducidos, se tomó la decisión de mantener la información acerca de la cantidad de vida restante en los agentes activos en la batalla pero se eliminaría la inclusión de los tipos de los Pokemon en el vector de la observación. Aunque inicialmente se esperaba que esta información fuera beneficiosa para el rendimiento del agente, los resultados obtenidos mostraron lo contrario, debido a que la representación numérica de los tipos no proporcionaba una comprensión clara de su significado, lo que acabó provocando una disminución del desempeño del agente en torno al 10% en comparación con las pruebas realizadas sin esta inclusión.

En cuanto a la recompensa, ésta ha sido objeto de una atención especial durante el proceso de mejora. Inicialmente, se consideraban factores muy simples, por lo que se requería la creación de una función de recompensa más densa, sobre la cual se han añadido diferentes parámetros que aportan recompensas relativamente pequeñas para guiar al agente hacia una estrategia más efectiva. Se añadieron varios parámetros:

- Un primer parámetro para evaluar si el tipo del Pokemon activo del agente es favorable o desfavorable en comparación con el el tipo del Pokemon rival. Esto se basa en el hecho de que normalmente los Pokemon suelen tener entre sus movimientos disponibles un ataque del mismo tipo que el del Pokemon, por lo que puede resultar una referencia valiosa para el agente al otorgarle una puntuación positiva si tiene a un Pokemon que a priori tenga ventaja de tipo.
- Un segundo parámetro para regular si el tipo del Pokemon cambiado por el agente es favorable o no ante al tipo del rival. Ya que el cambiar, ya sea o bien porque han debilitado al Pokemon activo o bien porque el agente ha decidido cambiar de Pokemon en su turno, es una acción muy determinante en el desenlace de la batalla, por lo que hacer este cambio si es negativo debe afectar seriamente a la recompensa y si es positivo también recompensarle porque será una buena acción a futuro.
- Un tercer indicador para fomentar el uso de ataques eficaces o supereficaces contra el Pokemon del oponente. Se otorgaría una recompensa positiva siempre que el agente reconociera y aprovechase las debilidades del rival a la hora de atacar. Del mismo modo se le daría una recompensa negativa si el ataque realizado no es eficaz.
- Por último, se añadió un parámetro adicional que tiene en cuenta si el Pokemon activo del agente ha experimentado algún tipo de aumento o disminución en sus atributos. Por ejemplo en el ataque, la defensa, la velocidad, etc. Se le sumaría un valor positivo en caso de ser una modificación favorable y viceversa.

Con estos cambios tanto en la configuración de la observación como en la de la recompensa se tiene como objetivo proporcionar una visión más completa del combate y brindar al agente información mas relevante para tomar decisiones durante la batalla, fomentando la adopción de estrategias más efectivas durante el entrenamientos.

Para evaluar de manera individual los efectos de los cambios implementados se llevaron a cabo una serie de experimentos mediante entrenamientos muy cortos de 30000 pasos. Todos

los parámetros que se han mencionados fueron analizados con la intención de medir su influencia en el rendimiento del agente y su capacidad para mejorar el porcentaje de victorias. En general, los resultados obtenidos fueron prometedores ya que demostraron tener un impacto positivo en la estrategia aprendida por el agente, como se puede observar en la tabla posterior. Sin embargo, el único parámetro que no mostraba signos claros de mejoría para el agente era el del cambio de Pokemon en combate. Pese a este hecho se decidió mantenerlo en el proceso de entrenamiento debido a su potencial para enriquecer el aprendizaje y su capacidad para comprender mejor el juego en su conjunto.

	Prueba 1		Prueba 2		Prueba 3	
Observación Añadida	RP (%)	SHP (%)	RP (%)	SHP (%)	RP (%)	SHP (%)
-	93.2	21.2	94.4	22.0	91.0	20.8
Tipo favorable del Pokemon	97.0	28.6	95,0	28.0	-	-
Tipo Favorable en el cambio	92.8	20.8	95.2	23.2	92.0	21.2
Ataque eficaz	95.6	26.4	94.8	26.0	96.4	28.4
Aumento de atributos	94.2	24.2	95.6	27.0	-	-
Todos juntos	94.4	31.0	96.0	30.2	93.2	28.0

Cuadro 5.18: Resultados de diferentes modelos al añadir los parámetros de recompensa

En resumen, se ha podido demostrar que la introducción de estos nuevos cambios pueden mejorar significativamente las políticas de decisiones del agente y guiarlo hacia estrategias que maximicen las probabilidades de éxito.

5.7.2 Entrenamiento realizado

Tras la implementación de las mejoras mencionadas anteriormente y la explicación del funcionamiento del self-play, se dio inicio al proceso de entrenamiento con el fin de evaluar los resultados obtenidos mediante esta estrategia. Como se ha comentado, durante el entrenamiento el agente se enfrentó siempre a dos oponentes, que se eligen se manera aleatoria. Inicialmente, sus rivales fueron dos RandomPlayer y a medida que avanzase el entrenamiento, se irían sustituyendo gradualmente por diferentes versiones del modelo como oponentes. Esta transición se realizará cada 12500 pasos, un valor elegido de forma completamente arbitraria sin un criterio específico. Además, se llevaron a cabo una serie de entrenamientos iniciales de corta duración para verificar el correcto funcionamiento de esta nueva técnica, asegurando que los oponentes se intercambiaban adecuadamente y que cargaban correctamente el modelo de red. Posteriormente, se procedió a realizar entrenamientos de mayor duración, llegando hasta 1 millón de pasos, con la intención de que el agente mejorase su rendimiento al aumentar la cantidad de tiempo del entrenamiento. Los resultados obtenidos se resumen a continuación:

Como se puede observar, y de manera sorprendente, los resultados obtenidos no cumplieron las expectativas. Se evidenció una disminución de los resultados a medida que se aumentaban los pasos del entrenamiento, lo cual contrasta con la idea general de que mayor tiempo de

Cantidad de Pasos	RP (%)	Recompensa promedia	SHP (%)	Recompensa promedia
75k	95.0	32.00	30.0	-13.12
100k	93.0	30.98	26.0	-16.45
300k	93.6	31.45	18.6	-18.89
1M	92.2	30.60	16.6	-20.04

Cuadro 5.19: Resultados de la primeras pruebas aplicando Self-Play

entrenamiento suele traducirse en una mejora del desempeño del modelo. Ante esta situación, se consideraron diferentes posibilidades y se planteó la hipótesis de que el intervalo de cambio de modelos podría ser demasiado pequeño. Esto limitaría la capacidad del agente para aprender ya que no tendría suficiente tiempo para explorar y desarrollar estrategias efectivas frente a diferentes agentes. Por lo que se decidió aumentar este intervalo para permitir jugar al agente más tiempo contra cada nuevo rival y así que éste pudiera ajustarse mejor a sus estrategias. Sin embargo, tras realizar pruebas adicionales se demostró que esta hipótesis no era correcta, de hecho, los resultados obtenidos fueron aún peores. Esta situación planteó nuevas interrogantes sobre la causa de este comportamiento inesperado. Se consideraron diversos factores como por ejemplo posibles problemas en la implementación de esta técnica o que la cantidad de oponentes a la que se enfrentaba no proporcionaba suficiente diversidad de entornos. Como medida para abordar este último aspecto, se decidió aumentar la cantidad de oponentes de 2 a 5. El objetivo de esta modificación era proporcionar al agente una mayor variedad de oponentes y, por ende, muestras de aprendizaje más representativas.

Con la intención de determinar si esta modificación podría solucionar el problema se llevaron a cabo entrenamientos de 100000 y 300000 pasos. Estos períodos de entrenamiento más cortos permitieron evaluar si dicho cambio había tenido algún efecto en el rendimiento. Sin embargo, los resultados mostrados en la tabla 5.20 fueron muy similares a los anteriores, lo que llevó a pensar que los entrenamientos estaban estancados en un ciclo en el que todas las versiones del agente tenían niveles de habilidad muy parecidos. Esto implicaba que el agente podía ganar o perder contra cualquier modelo sin lograr progresar más allá de ese punto.

Cantidad de Pasos	RP (%)	Recompensa promedia	SHP (%)	Recompensa promedia
100k	93.66	31.35	26.2	-14.29
	95.33	32.78	26.0	-14.65
300k	92.33	-	18.2	-19.24
	95.66	32.53	21.6	-17.62

Cuadro 5.20: Resultados tras cambiar a 5 oponentes

En este momento y tras los nulos resultados obtenidos, se decidió adoptar un enfoque más flexible y experimental. En primer lugar, se introdujo al jugador de la heurística como un rival más para el agente, asegurando así la presencia de un oponente con un alto nivel de habilidad y tratando de evitar que el agente se quedase estancado jugando contra sus propias copias. Además, se realizaron ajustes en la frecuencia del cambio de modelos entre episodios y se exploraron diferentes hiperparámetros de la red que aún no habían sido considerados, como la tasa de actualización de los pesos de la red neuronal. La intención era disminuir este valor para evitar que los cambios en los pesos del modelo fueran muy bruscos y que por este motivo el agente se quedase atrapado en una estrategia subóptima. De esta manera, la convergencia de la red puede ser más precisa y estable, aunque por ello se aumentase el tiempo que requiere el agente para aprender.

También se tomó la decisión de cambiar la estructura de la red neuronal, la cual en este caso concreto estaba formada únicamente por una capa de entrada, una capa oculta y una capa de salida. Inicialmente, en el modelo la capa oculta contaba únicamente con 64 neuronas. Añadir más capas permite a la red obtener relaciones más complejas con el coste de aumentar el tiempo de entrenamiento, por lo que se decidió agregar más, así como aumentar también el número de neuronas por capa para conseguir una representación más profunda.

Estos cambios tuvieron como objetivo buscar nuevas configuraciones y estrategias que pudieran mejorar el rendimiento del agente y su capacidad de predicción. Los resultados de todas las pruebas y experimentos realizados se resumen en la Tabla final. En éstas se puede ver como, a pesar de los esfuerzos y las diferentes estrategias aplicadas, los entrenamientos no lograron alcanzar los resultados esperados y no se han podido conseguir mejoras significativas en comparación con las secciones anteriores. Se puede ver en el Anexo un resumen final de toda la información relevante con respecto a la topología de la red neuronal, así como los diferentes parámetros que se han ido modificando a lo largo del proyecto sobre la misma. También están añadidos los espacios de acciones y observaciones utilizados, al igual que las variables finales de la función de recompensa y sus valores.

Cantidad de Pasos	Información Extra	RP (%)	Recompensa promedia	SHP (%)	Recompensa promedia
300k	Cambios de modelo =10k Estructura de la red = [128,128]	95.66	32.53	24.0	-15.6
	Cambios de modelo=15k Estructura de la red = [128,128]	92.66	-	18.66	-
	Cambios de modelo=10k Estructura de la red = [128,128] Learning_rate=0.00005	94.66	31.86	24.66	-15.50
	Cambios de modelo=15k Estructura de la red = [128,128] Learning_rate=0.00005	89.33	28.78	22.3	-16.81
	Cambios de modelo=10k Estructura de la red = [128,128] Learning_rate=0.00005 Recompensa por victoria = 30	92.0	38.36	25.33	-19.71
1M	Cambios de modelo=10k Estructura de la red = [128,128] Learning_rate=0.00005 Recompensa por victoria = 30	91.33	37.76	20.66	-22.97
	Cambios de modelo=10k Estructura de la red = [256,128] Learning_rate=0.00005 Recompensa por victoria = 30 Añadido MBPP como rival	93.33	39.91	20.0	-24.92
	Cambios de modelo=10k Estructura de la red = [64,128] Learning_rate=0.00005 Recompensa por victoria = 30	93.333	39.77	18.66	-25.92
	Cambios de modelo=10k Estructura de la red = [256,128] Learning_rate=0.00005 Recompensa por victoria = 30	90.8	36.38	13.66	-29.80

Cuadro 5.21: Resultados Finales del entrenamiento

6 Resultados

A pesar de ser la primera incursión en el campo de la inteligencia artificial y, más específicamente, en el aprendizaje por refuerzo, se han superado los obstáculos y los problemas encontrados en el camino y, aunque no se haya logrado alcanzar la excelencia deseada en algunos aspectos, los resultados obtenidos en este proyecto han sido satisfactorios.

Las primeras pruebas realizadas en el entorno final de la generación 4 arrojaron resultados prometedores. La diversidad de estrategias implementadas reveló información valiosa sobre el proceso de entrenamiento y, pese a los fallos que se cometieron, se lograron superar y llevar a cabo un entrenamiento sólido que produjo un agente altamente competente. Se obtuvo una tasa de victoria del 95% contra el agente aleatorio, lo cual era un resultado esperable dado que este agente representa un nivel muy bajo. Sin embargo, los resultados frente a los dos otros rivales, los cuales representan un nivel intermedio y elevado respectivamente, también fueron aceptables. El agente logró vencer al jugador que seleccionaba los ataques de mayor poder base en aproximadamente el 70% de los enfrentamientos, una cifra considerablemente elevada para tratarse de un rival de categoría media. Y ante el oponente más desafiante, se obtuvo una tasa de victoria del 30%, lo cual no es un mal resultado teniendo en cuenta el nivel del rival y fueron los primeros entrenamientos.

Con el fin de mejorar los resultados anteriores se realizaron varios intentos utilizando diferentes técnicas, pero ninguna de ellas tuvo éxito . Se consideró la idea de vectorizar el entorno como una estrategia prometedora para permitir al agente recopilar información de múltiples entornos y enriquecer así el buffer de repetición con muestras más variadas del espacio de estados. No obstante, su implementación produjo problemas inesperados y los resultados obtenidos no fueron los esperados, obteniendo tasas de victorias inferiores a los de las secciones anteriores. A pesar de este contratiempo, se mantuvo el enfoque de búsqueda de soluciones alternativas para mejorar el rendimiento del agente, lo que llevó al planteamiento de otras dos alternativas. Una de ellas consistía en realizar transiciones más suaves al cambiar de oponentes, utilizando agentes con niveles de habilidad menos distantes entre sí. Como la librería utilizada solo implementaba tres agentes distintos, se creó un nuevo jugador para abordar este problema. Sin embargo, los resultados obtenidos seguían siendo similares a los anteriores, lo que indicaba que era necesario contar con más tipos distintos de jugadores para conseguir lograr una progresión gradual de nivel. Ante esta situación, se tomó la decisión de cambiar de estrategia y tratar de implementar la técnica del self-play. Ésta consistía en permitir que el agente se enfrentara progresivamente a copias de sí mismo durante el entrenamiento, lo que además podría suponer una solución ante el problema anterior. Con esta aproximación se pensó que se garantizaría que en ningún momento se encontrarían desnivelación con un oponente, consiguiendo así un equilibrio en el proceso de aprendizaje del agente. Además, para orientar al agente hacia estrategias más óptimas se llevaron a cabo modificaciones en

el espacio de observaciones, así como en la estructura de las recompensas con la intención de definir una función más densa y con más información para el agente. Se observó como con todos estos cambios, junto con la nueva técnica del self-play, el agente obtuvo resultados similares a los anteriores en entrenamientos relativamente cortos, aunque al alargarlos en el tiempo su rendimiento descendía.

Como último recurso, se implementaron cambios adicionales sobre la red neuronal utilizada, con la finalidad de conseguir superar los resultados anteriores. Con estas modificaciones se logró mitigar la bajada del rendimiento del agente a lo largo del tiempo, aunque no fueron suficientes para alcanzar mejoras significativas en su desempeño. Por lo que finalmente y a pesar de los esfuerzos realizados los resultados obtenidos no alcanzaron los niveles deseados.

Al combinar todas las experiencias y estrategias previas, se observó que los mejores resultados se alcanzaron durante los entrenamientos de corta duración, en los cuales se implementó un enfoque gradual de enfrentamiento contra oponentes de mayor nivel. En este contexto, el agente más destacado logró obtener una tasa de éxito cercana al 70-75% en partidas contra el agente MBPP y entorno al 30-35% frente a SHP. Por lo que para lograr mejoras adicionales, se requería un proceso de investigación más profundo.

7 Conclusiones

El presente trabajo se inició con una extensa búsqueda de información sobre la inteligencia artificial con el objetivo de adquirir un conocimiento sólido en la materia. Además, se realizó una larga revisión bibliográfica sobre el funcionamiento del aprendizaje gracias a la cual se ha logrado adquirir un sólido conocimiento teórico sobre el tema. Durante el transcurso del proyecto, se ha profundizado en los conceptos clave y se ha alcanzado una completa comprensión sobre sus principios fundamentales y se han adquirido competencias prácticas relevantes mediante el desarrollo de agentes inteligentes capaces de jugar a juegos simples como Cart-Pole o Spacae Invaders. En éstos, los resultados obtenidos fueron excelsos, demostrando la importancia de la investigación previa realizada para alcanzar este éxito.

En cuanto al objetivo final de este TFG, que fue desarrollar una inteligencia artificial capaz de aprender a jugar en un entorno más complejo, como es el de las batallas Pokemon, las cuales cuentan con un espacio de estados mucho más amplio que en los juegos anteriores, se puede concluir que se lograron resultados satisfactorios. Tras dedicar numerosos meses a esta fase del proyecto, se lograron desarrollar varios modelos de calidad donde el agente obtuvo una tasa de victorias considerable. Hasta llegar a estos resultados se aplicaron y estudiaron diversas estrategias con la intención de obtener el entrenamiento óptimo. No obstante, a medida que se avanzaba en su ejecución, las expectativas se elevaron y se buscaron resultados aún más destacados, pero a pesar de la diversidad de enfoques estudiados y aplicados no fue posible alcanzarlos. Sin embargo, este hecho no debe restar importancia a los logros obtenidos ya que estas investigaciones sientan las bases para futuros proyectos donde se mejore el rendimiento alcanzado.

Se contempla la posibilidad de retomar este proyecto en el futuro con el propósito de mejorarlo, ya sea como Trabajo Final de Máster o como una investigación independiente. Se deja la puerta abierta la exploración de nuevas técnicas y métodos para el proceso de entrenamiento, así como la revisión y mejora de las propuestas planteadas en este TFG. Además, se plantean diversos trabajos futuros que implican la ampliación de los modelos a generaciones posteriores, las cuales presentan espacios de acciones más complejos, con la incorporación de nuevos ataques y mecánicas que pueden dar lugar a nuevas estrategias de combate. También se plantea la posibilidad de entrenar al agente en un entorno competitivo más desafiante alejado de las batallas aleatorias, ya que en este entorno se presenten combinaciones de equipos de Pokemon menos comunes, apartadas de las representaciones del entorno competitivo real. Sería interesante enfocarse en combates de tipo META (Most Efficient Tactic Available), donde la diversidad de Pokemon es más reducida y los jugadores suelen emplear estrategias más sofisticadas y complejas. Este nuevo desafío se vislumbra como un objetivo más ambicioso, pues implica la necesidad de desarrollar métodos completamente nuevos, adaptados a formatos y condiciones de juego distintas.

En conclusión, el proyecto representa un importante punto de partida en el mundo de la inteligencia artificial y el aprendizaje por refuerzo, siendo una experiencia en la que se han enfrentado a nuevos desafíos y se ha explorado un terreno desconocido. Los resultado aportados brindan una base sólida para futuras investigación y mejoras en el campo del RL y las batallas Pokemon.

ANEXO

Red neuronal - Arquitectura de la red
Número de capas = 4
Número de capas ocultas = 2
Número de Neuronas en las capas ocultas = [128, 128]

Cuadro 7.1: Arquitectura empleada para los últimos experimentos

Entorno final de las batallas Pokemon	
Espacio de observación	
[agent_HP, opponet_HP]	Los puntos de salud de los Pokemon activos en combate
moves_base_power	La potencia de ataques disponibles
moves_dmg_multiplier	El multiplicados de cada ataque en función del tipo
[fainted_mon_team, fainted_mon_opponent]	La cantidad de Pokemon derrotados tanto del agente como del rival
Espacio de acciones	
Acción = -1	Retirarse de la batalla
Acción = [0,3]	Selección de un movimiento disponible
Acción = [4,8]	Selección de un cambio de Pokemon disponible
Función de recompensa	
fainted_value = 2.0	El peso de la recompensa por derrotar a un Pokemon
hp_value = 2.0	El peso de la recompensa por la fracción de vida disminuida
status_value = 0.2	El peso de la recompensa por tener al Pokmeon afectado por algún tipo de estado Ej: envenenamiento, quemadura, paralizado …
victory_value = 30.0	El peso de la recompensa por ganar una batalla
dmg_value = 0.45	El peso de la recompensa por realizar un ataque eficaz
type_value = 0.35	El peso de la recompensa tener a un Pokemon con un tipo favorable al del rival También se utiliza para el cambio de Pokemon
boost_value = 0.2	El peso de la recompensa por tener alguna estadística del Pokemon aumentada

Cuadro 7.2: Parámetros del entorno final

Bibliografía

[1] A. M. Turing, *Computing Machinery and Intelligence*, pp. 23–65. Dordrecht: Springer Netherlands, 2009.

[2] A. Rockwell, "The history of artificial intelligence," *Harvard University*, 2017.

[3] A. GALIPIENSO, M. ISABEL, M. A. Cazorla Quevedo, O. Colomina Pardo, F. Escolano Ruiz, and M. A. LOZANO ORTEGA, *Inteligencia artificial: modelos, técnicas y áreas de aplicación*. Ediciones Paraninfo, SA, 2003.

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[5] A. G. Barto and R. S. Sutton, "Chapter 19 - reinforcement learning in artificial intelligence," in *Neural-Network Models of Cognition* (J. W. Donahoe and V. Packard Dorsel, eds.), vol. 121 of *Advances in Psychology*, pp. 358–386, North-Holland, 1997.

[6] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A survey of deep reinforcement learning in video games," *arXiv preprint arXiv:1912.10944*, 2019.

[7] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with lstm," in *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2, pp. 850–855 vol.2, 1999.

[8] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[9] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[10] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8, 2016.

[11] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," 2016.

[12] D. Simões, S. Reis, N. Lau, and L. P. Reis, "Competitive deep reinforcement learning over a pokémon battling simulator," in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 40–45, 2020.

[13] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[14] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[15] H. Sahovic, "Poke-env: pokemon ai in python."

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[20] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén, "Augmenting automated game testing with deep reinforcement learning," in *2020 IEEE Conference on Games (CoG)*, pp. 600–603, 2020.

[21] D. Huang and S. Lee, "A self-play policy optimization approach to battling pokémon," in *2019 IEEE Conference on Games (CoG)*, pp. 1–4, 2019.

Lista de Acrónimos y Abreviaturas

A2C	Advantage Actor Critic.
API	Interfaz de Programación de Aplicaciones.
DQN	Deep Q-Network.
HP	Hit Points.
IA	Inteligencia Artificial.
MBPP	MaxBasePowerPlayer.
PPO	Proximal Policy Optimization.
RL	Reinforcement Learning.
RP	RandomPlayer.
SAC	Soft Actor-Critic.
SB3	Stable Baselines 3.
SHP	SimpleHeuristicPlayer.
TFG	Trabajo Final de Grado.