Universidad Miguel Hernández de Elche

MASTER UNIVERSITARIO EN ROBÓTICA

Aprendizaje por refuerzo mediante redes neuronales convolucionales

Trabajo de Fin de Máster

2018-2019

Autor: Joaquin Carrasco Palazón
Tutor/es: Oscar Reinoso García

Agradecimientos:

A mi tutor Oscar, por su inestimable ayuda con las correcciones y su guía para llevar a buen puerto este trabajo fin de máster.

A mis padres, por haberme apoyado siempre.

A Abel Perez Morte, por su ayuda con la edición de los vídeos.

A todas las personas que me han animado o apoyado mientras realizaba este trabajo.

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de fórmulas

1. Introducción

La **inteligencia artificial** es un campo que actualmente está en **plena expansión**, el cual tendrá una infinidad de aplicaciones en un futuro próximo; la cantidad de usos de esta tecnología es abrumadora, con aplicaciones que van desde el **reconocimiento de imágenes, personas y patrones, a aplicaciones como bots conversacionales, aprendizaje máquina, creación de personajes no jugables inteligentes en videojuegos, etc.**

Figura 1: Imagen inteligencia artificial

[1]

Este trabajo se centrará en una rama concreta de la inteligencia artificial, conocida como **aprendizaje automático, combinado con redes neuronales.** Esto se engloba en lo que se conoce como **aprendizaje profundo por refuerzo (del inglés *Deep Reinforcement Learning*)**, englobado a su vez dentro del **aprendizaje máquina (*Machine Learning*)**. Todos estos conceptos quedarán explicados y definidos en los apartados siguientes de este trabajo, así como la instalación de los programas necesarios para ello, los experimentos realizados al respecto y los resultados obtenidos.

Asimismo, al final se expondrán las conclusiones obtenidas así como posibles futuros desarrollos en base a lo aprendido y desarrollado en este trabajo.

1.1 Planteamiento general del trabajo

Figura 2: Google DeepMind Human-Level control through deep reinforcement learning [2]

La idea original de este trabajo es **recrear**, en la medida de lo posible, **el agente y el algoritmo utilizado en los paper publicados por DeepMind, titulados *"Playing Atari with Deep Reinforcement Learning"*** [3] **y *"Human-level control through deep reinforcement learning"*** [4], los cuales se publicaron el 19 de Diciembre de 2013 y el 25 de Febrero de 2015 respectivamente.

En ambos paper se trata la misma **problemática: como lograr un control del entorno, suministrando únicamente información visual**, (en este caso la pantalla de varios videojuegos de la Atari 2600; la misma información que percibiría un jugador humano) **y obtener**, en algunos casos, **resultados que sobrepasan a los de un jugador humano experto.**

Para ello se utiliza **la siguiente técnica:** combinando **aprendizaje por refuerzo** (basada en la psicología humana y animal) **y *Deep Learning;*** la suma de ambas da lugar a lo que se conoce como ***Deep Reinforcement Learning*.**

El algoritmo utilizado se conoce como **Deep-Q-Network (abreviado como DQN),** está **patentado por Google** [5], y **se basa en el algoritmo** conocido como ***Q-Learning.***

El ***Q-Learning*** es un **algoritmo de aprendizaje por refuerzo,** que ante un entorno (en nuestro caso serán los diferentes juegos de Atari 2600 sobre los que realicemos los distintos tests), y recibiendo determinada información del mismo (imágenes de la pantalla, obtenidas mientras jugamos) **es capaz de determinar qué acción tiene mayores opciones de conseguir el maximizar la recompensa** (maximizar la puntuación). Este algoritmo **crea unas tablas en las que se asocia un valor (*Q-value*) a cada acción posible a realizar**: en un videojuego, serán las acciones posibles: si pensamos por ejemplo, en *SuperMario*, ante un enemigo, puede realizar distintas acciones, como correr hacia él, saltar, volver atrás, agacharse….

El algoritmo **debe de ser capaz de aprender una estrategia para cada situación que se le presente, intentando maximizar su recompensa,** la cual en los entornos probados significa hacer máxima la puntuación.

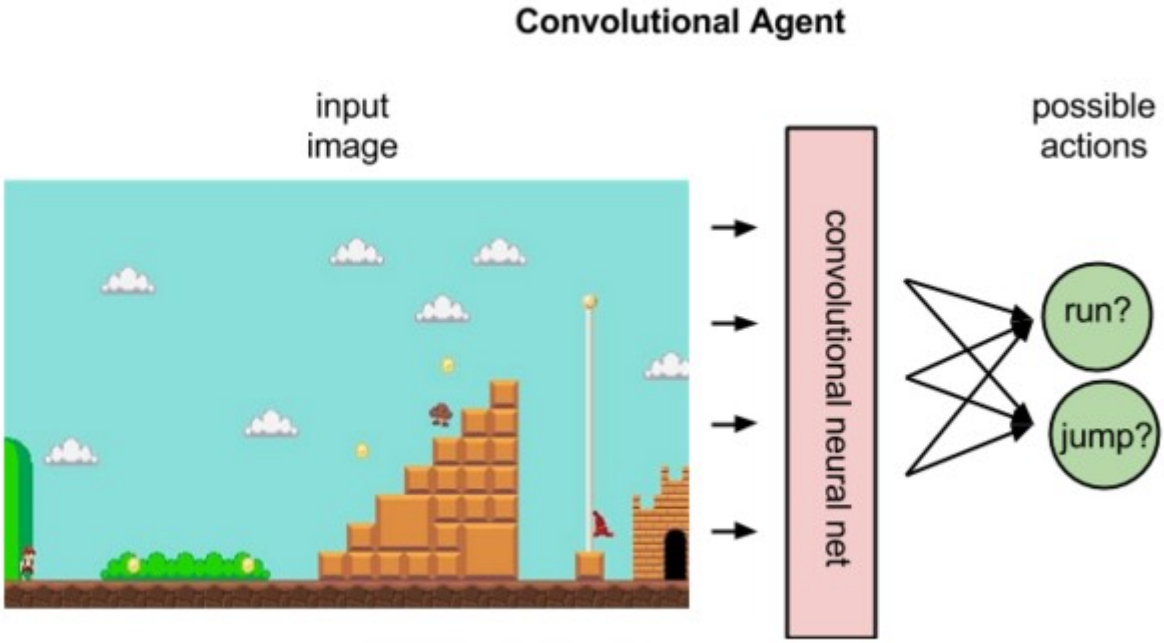

Figura 3: Ejemplo de selección de acción en SuperMario
[6]

Pero, **¿qué pasa si el entorno es muy grande o la cantidad de acciones posibles es muy alta?** Pensemos en el mismo ejemplo, *SuperMario*. El diseño de sus niveles es muy variable: encontramos obstáculos, distintos tipos de enemigos, diferentes alturas…

El ***Q-Learning*** por sí solo, **es ineficiente ante entornos con gran cantidad de situaciones distintas y múltiples acciones,** ya que el algoritmo no será capaz de aprender la estrategia correcta para cada situación.

Pensemos en que el *Q-Learning* calcula una tabla para n-estados, en la que podemos realizar m-acciones. **Cuando más grande sea el número de estados y acciones, más grande será esta tabla, lo que hará más complejo que el algoritmo encuentre una solución correcta para esa situación concreta en ese instante dado.**

¿Cómo podemos solucionar esto? **Necesitamos una función** que tome decisiones con cierta plasticidad, es decir, **que sea capaz de adaptarse conforme se le presentan las distintas situaciones.** Por tanto, **se aplican redes neuronales al *Q-Learning,* que pasa a llamarse *Deep-Q-Learning.*** Es decir, **utilizaremos las redes neuronales** (en nuestro caso redes neuronales convolucionales, ya que la información que recibe nuestro algoritmo es puramente visual) **para obtener los *Q-values* asociados a las distintas acciones posibles dentro del juego.**

El uso de redes neuronales **introduce algunas ventajas como la capacidad de adaptación a nueva información, pero también algunas desventajas como el ajuste de hiperparámetros;** todo esto quedará explicado más extensamente en los siguientes capítulos de este trabajo.

Si bien por sí solas las redes neuronales presentan un gran avance, **al usarlas como aproximadores de la función que se busca** (es decir, utilizamos las redes neuronales para encontrar la función que mejor resultado nos dé), **necesitamos introducir ciertas mejoras al algoritmo para hacerlo más eficiente.**

El aprendizaje por refuerzo se sabe que es inestable o que incluso puede ser divergente cuando un aproximador de funciones no lineal como las redes neuronales es utilizado para calcular los *Q-values*. Esta inestabilidad tiene varias **causas:** las **correlaciones presentes en la secuencia de observaciones** (es decir, las imágenes que entran a la red tienen relación entre sí, bien porque sean consecutivas o bien porque sean muy cercanas en el tiempo), el hecho de que **constantemente estemos realizando pequeñas actualizaciones a los valores de Q** [4] (los valores de Q cambian ya que para cada situación son distintos; siguiendo con el ejemplo de *SuperMario*, la acción de saltar no tendrá los mismos valores ante un enemigo que muere cuando lo aplastas que ante uno sobre el que si saltamos moriríamos nosotros).

Pensemos en los valores de Q como el punto de convergencia de nuestro algoritmo: en aprendizaje supervisado por ejemplo, este punto estaría fijo: si estamos clasificando perros y gatos, diríamos que el algoritmo ha alcanzado la convergencia cuando obtiene un porcentaje de aciertos alto, pongamos mayor del 90%. Ahora imaginemos que empezamos clasificando perros y gatos, pero vamos introduciendo más animales: vacas, pollos, tigres… **Estaríamos desplazando el punto de convergencia, y nuestro algoritmo tendría que ir adaptándose si queremos que llegue a converger.**

Este es uno de los **problemas** principales **de los algoritmos de aprendizaje por refuerzo: puede que nunca lleguen a converger,** ya que, literalmente, **estamos moviendo su punto de convergencia constantemente, al ir variando el objetivo.** Siguiendo con el ejemplo de *SuperMario*, imaginemos que en este instante se nos presenta un enemigo al cual podemos matar si saltamos sobre él; nuestra DQN debería valorar las distintas acciones y concluir que, para ese instante, la acción que debemos tomar es saltar.

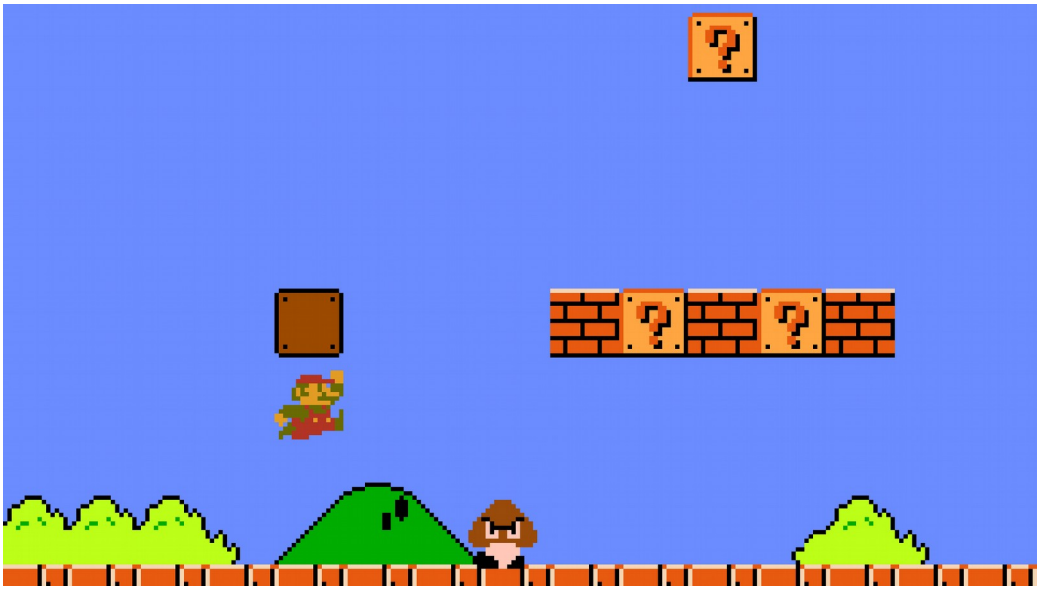

Figura 4: Super Mario

[7]

Instantes después, ante un suelo liso, ¿cuál sería la acción que nos acercaría más a la máxima puntuación? Probablemente correr. Por tanto, hemos variado el punto de convergencia del algoritmo, pero esto sucede porque el entorno (en nuestro caso el videojuego) es variable; si bien puede presentar un número de acciones determinado (saltar, correr, desplazarse a izquierda y derecha, etc), **el número de situaciones posibles que puede presentar el entorno puede ser muy alto, o incluso infinito.**

Para solucionar o paliar los problemas de convergencia que se mencionan, **se aplican dos ideas claves: un mecanismo inspirado en la biología humana que se conoce como *experience replay*** [4], el cual en nuestro algoritmo **lo que hace es almacenar experiencias pasadas, tomar un número determinado de ellas aleatoriamente e introducirlas en la red neuronal**; esto se realiza para que el algoritmo **no olvide lo aprendido anteriormente y los pesos de las neuronas tiendan a adaptarse solamente a lo que está sucediendo actualmente**. Se eligen aleatoriamente para **romper la correlatividad** de lo que aprende; nos interesa romper esta correlación para que el algoritmo no refuerce demasiado unas acciones o situaciones determinadas; buscamos que sea capaz de desenvolverse bien en todas las situaciones posibles que se le puedan presentar en el entorno en el que lo estemos probando.

El otro mecanismo que se implementa para ayudar a la convergencia es **utilizar una red neuronal objetivo (*target network)*, la cual vamos actualizando periódicamente tras un tiempo determinado:** es decir, **usamos una red para calcular los *Q-values* de las acciones en cada instante, y otra red con la que calculamos los *Q-values* de la recompensa máxima posible que podemos obtener.** De esta forma, **evitamos que los valores de Q varíen tanto;** se actualizan periódicamente para ir ajustando el objetivo de la *target network* al objetivo actual. Este sistema **ayuda** en gran medida **a garantizar la estabilidad del algoritmo** y evitar grandes variaciones en los valores de Q.

Y es aquí donde reside **la verdadera potencia de los algoritmos de aprendizaje por refuerzo:** si bien lo que estos algoritmos realizan puede realizarse por otros métodos más eficientes (tanto en términos de costo computacional como en tiempos de aprendizaje y resultados), su verdadero interés radica en su **capacidad de generalización:** son **capaces de adaptarse a múltiples situaciones y entornos, sin ningún tipo de ajuste extra** por nuestra parte o prácticamente ninguno.

En [4], encontramos una tabla que nos muestra la prueba de que estos algoritmos son capaces de encontrar una estrategia adecuada para maximizar la puntuación para muchos de los entornos (videojuegos) testeados: **de 49 juegos probados, en 43 de ellos obtiene mejores puntuaciones que las obtenidas con cualquier otro tipo de aproximación o algoritmo** hasta la fecha de la publicación del paper (2015), consiguiendo una **puntuación del 75% o más de la puntuación conseguida por un humano** en 29 juegos.

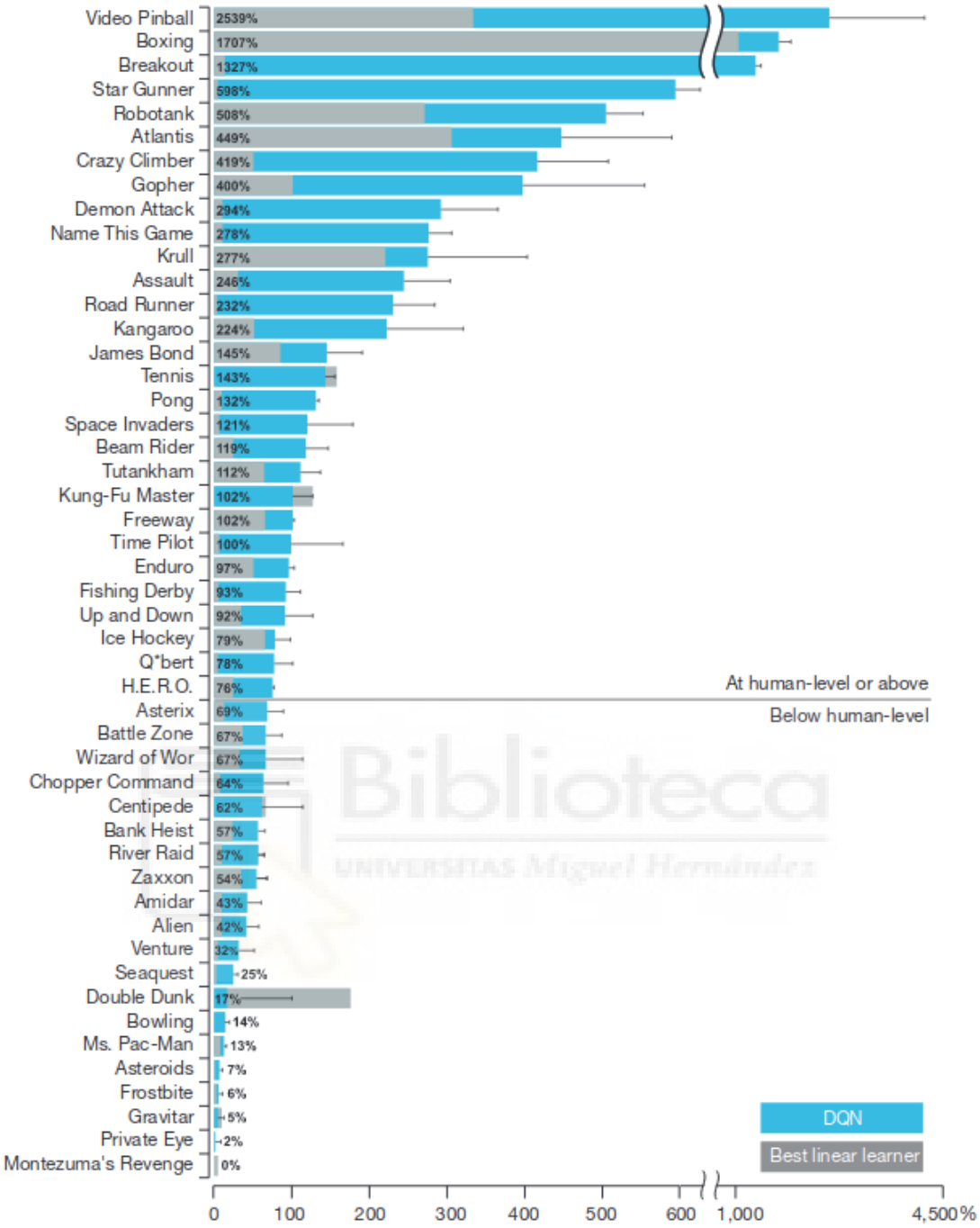

Figura 5: Comparativa resultados juegos del Paper Human-level control through deep reinforcement learning [4]

Otro problema al que nos enfrentamos con este tipo de algoritmos es que en muchos entornos, **la recompensa no es inmediata:** siguiendo con el ejemplo de *SuperMario*, imaginemos que un enemigo aparece por la parte derecha de la pantalla, mientras nosotros estamos en el centro de la misma: para matar al enemigo, debemos saltar sobre él, pero para ello, debemos estar a una distancia determinada. Por tanto, nuestro algoritmo debe intentar maximizar las recompensas a largo plazo, frente a las inmediatas.

Además, **nuestro algoritmo no tiene conocimiento *a priori* de las reglas de cada juego; esto se conoce como modelo libre,** es decir, en ningún caso se le enseñan los mecanismos que hacen que se obtengan puntos. **Es el propio algoritmo el que debe encontrarlos por sí mismo,** mediante entrenamiento; repitiendo muchas partidas (en nuestro caso concreto en el que usamos videojuegos).

Puesto que a**l principio no tenemos ningún tipo de conocimiento sobre el entorno en el que estamos, es necesario realizar acciones aleatorias durante un tiempo determinado,** para que mediante estas acciones el algoritmo vaya descubriendo las reglas del entorno, esto es: como se puntúa, como se pierde una partida, etc. **Se usa un parámetro, denominado *epsilon*, al cual se le aplica un descenso lineal en su valor** (siendo al principio su valor máximo, hasta uno mínimo fijado). **Este valor determinará la probabilidad que tendrá nuestro algoritmo de realizar una acción aleatoria,** es decir: c**uanto mayor sea epsilon, mayor probabilidad tendremos de realizar acciones aleatorias.** Mediante este sistema, **vamos dejando que el algoritmo tome progresivamente el control frente a la aleatoriedad.**

Aún así, **debemos dejar un margen a la aleatoriedad**; esto es lo que **se conoce como *explotation vs exploration*.** Se considera ***explotation*** al uso de los **conocimientos ya adquiridos,** pero para evitar que nuestro algoritmo deje de intentar realizar acciones nuevas, **hay que dejar un margen, pequeño normalmente, a la exploración.**

1.2 Estado del arte

En este apartado se realizará un **breve repaso de los usos actuales de la inteligencia artificial**, así como las distintas aplicaciones del ***Deep Learning*** y el **aprendizaje por refuerzo en situaciones prácticas.**

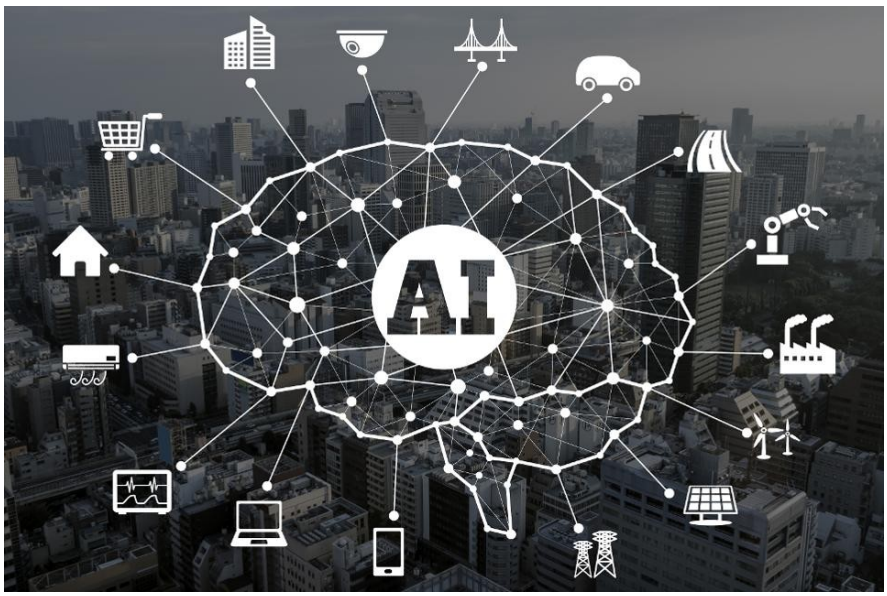

Figura 6: Diferentes usos de la inteligencia artificial a día de hoy

[8]

En la actualidad, **las aplicaciones reales de la inteligencia artificial son muchas**, incluso en la vida cotidiana. Repasamos aquí **algunos usos de la IA actuales** y bastante cotidianos, a**sí como otros** posibles usos **futuros:**

- **Asistente para fotografía:** algunos fabricantes de móviles han incorporado IA al software fotográfico; esto nos permite hacer mejores fotos en condiciones de baja luz. Otros usos en este campo son el enfoque automático de objetos con mayor precisión [9].
- **Conducción autónoma:** apoyándose en el *deep learning*, muchas marcas están experimentando con vehículos autónomos en la actualidad para que estos aprendan por dónde circular, qué objetos se encuentran a su paso, reconocimiento de las señales de tráfico, etc [9].

Figura 7: Coche autónomo

[10]

La *U.S. National Highway Traffic Safety Administration* (NHTSA) reconoce seis niveles de automatización en cuanto a conducción:

1. **Nivel cero:** conducción humana.
2. **Nivel 1: sistemas de asistencia avanzados** (*ADAS, advanced driving assistance system* en inglés), que ayudan al conductor humano con la dirección, frenado o aceleración,

pero no simultáneamente. En esta categoría se incluyen cámaras de visión trasera, y distintas alertas al conductor.

3. **Nivel 2: un sistema ADAS que puede girar y bien frenar o acelerar simultáneamente** mientras el conductor sigue al volante.
4. **Nivel 3: sistemas totalmente automáticos** (*automated driving systems, ADS* en inglés) que pueden realizar todas las tareas de conducción bajo ciertas circunstancias, como aparcar el coche.
5. **Nivel 4: ADS que es capaz de realizar todas las tareas de conducción y monitorizar todo el entorno de conducción en ciertas circunstancias.** En esas circunstancias, el ADS es lo suficientemente confiable como para que el conductor no preste atención.
6. **Nivel 5: el ADS actúa como un chófer virtual y realiza toda la conducción en cualquier circunstancia.** Los ocupantes no conducen el vehículo. [11]

- **IA en videojuegos:** algunas compañías de videojuegos ya implementan verdaderas inteligencias artificiales; no los típicos bots que responden a ciertos parámetros, sino verdaderas **inteligencias que aprenden del jugador.** Algunos ejemplos los tenemos en *Forza Motorsport 5 de* Microsoft, que ya en 2013 tomaba patrones de jugadores reales para asimilar su comportamiento a uno humano. Otro ejemplo lo tenemos en *Left 4 Dead*, de Valve, y su director de juego, el cual analiza el comportamiento de los jugadores y los premia o castiga según sus acciones [9].

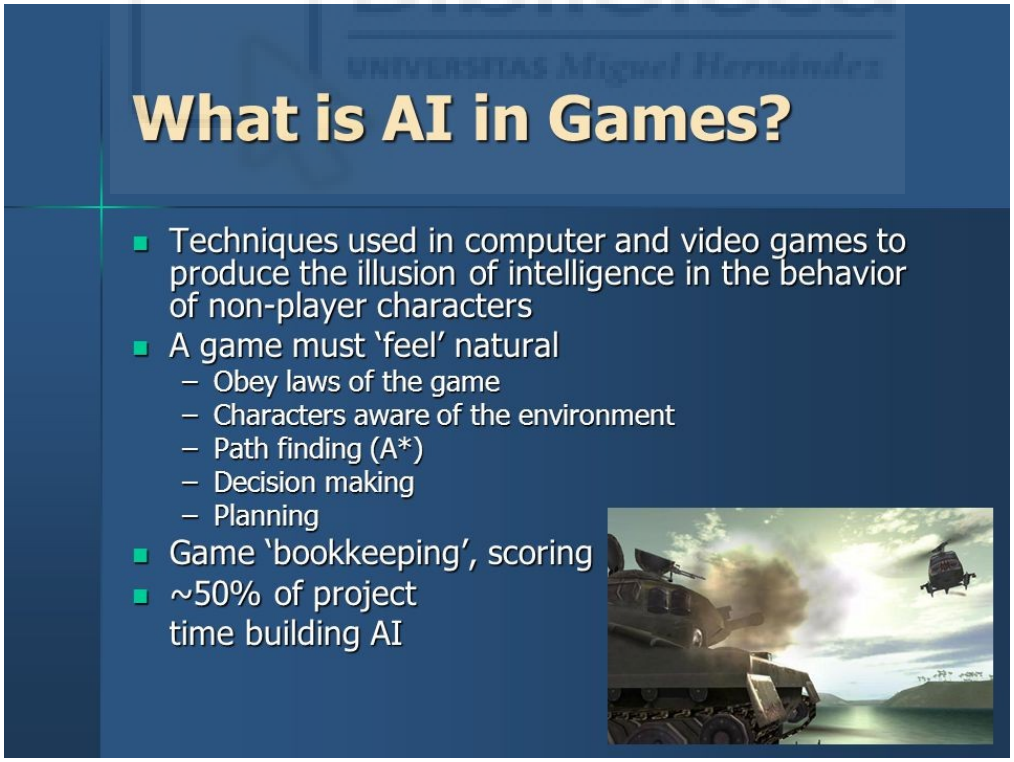

Figura 8: Inteligencia artificial en videojuegos

[12]

- **Anuncios personalizados:** grandes gigantes tecnológicos como Amazon, Microsoft, Google o Facebook utilizan inteligencia artificial para analizar nuestros gustos y recomendarnos productos que puedan ser de nuestro interés [9].
- **Colorear imágenes:** uno de los usos del *deep learning* es colorear imágenes en blanco y negro; esta aplicación es bastante importante ya que nos permite disfrutar de fotografías y vídeos, documentos históricos, etc, con una paleta de colores realista. Realizarlo mediante una IA ahorra muchísimo trabajo en comparación al tiempo que tarda una persona [9].
- **Encontrar malware:** gracias al *machine learning*, encontramos ejemplos como el caso de Google, que filtra el correo para saber cuáles son los que llevan SPAM o malware. Con cada acción que realizamos, el sistema aprende, toma decisiones y hace predicciones [9] [13].
- **Recuento de especies:** para contar los especímenes de una especie, antes era necesario hacerlo a mano. Ahora, usando cámaras, una IA (*machine learning*) puede encargarse de llevar la cuenta de la población [9].
- **Traductores automáticos:** tanto los traductores de texto como de voz utilizan IA para aprender; gracias a la interacción con los humanos, estas IA van aprendiendo [9].
- **Automatización en fábricas:** una de las áreas que más se han automatizado en los últimos años ha sido la fabricación, y, concretamente, el sector de la automoción, en el que se está incorporando la inteligencia artificial para hacer más eficiente esta automatización [9].

Figura 9: Inteligencia artificial en fábricas inteligentes
[14]

- **Reconocimiento facial en tiempo real:** encontramos dispositivos como el *iPhone X*, el cual permite desbloquear nuestro teléfono con nuestra cara. Otros ejemplos son los de la policía china y británica, las cuales utilizan cámaras para reconocer a la población y encontrar delincuentes [9] [15].

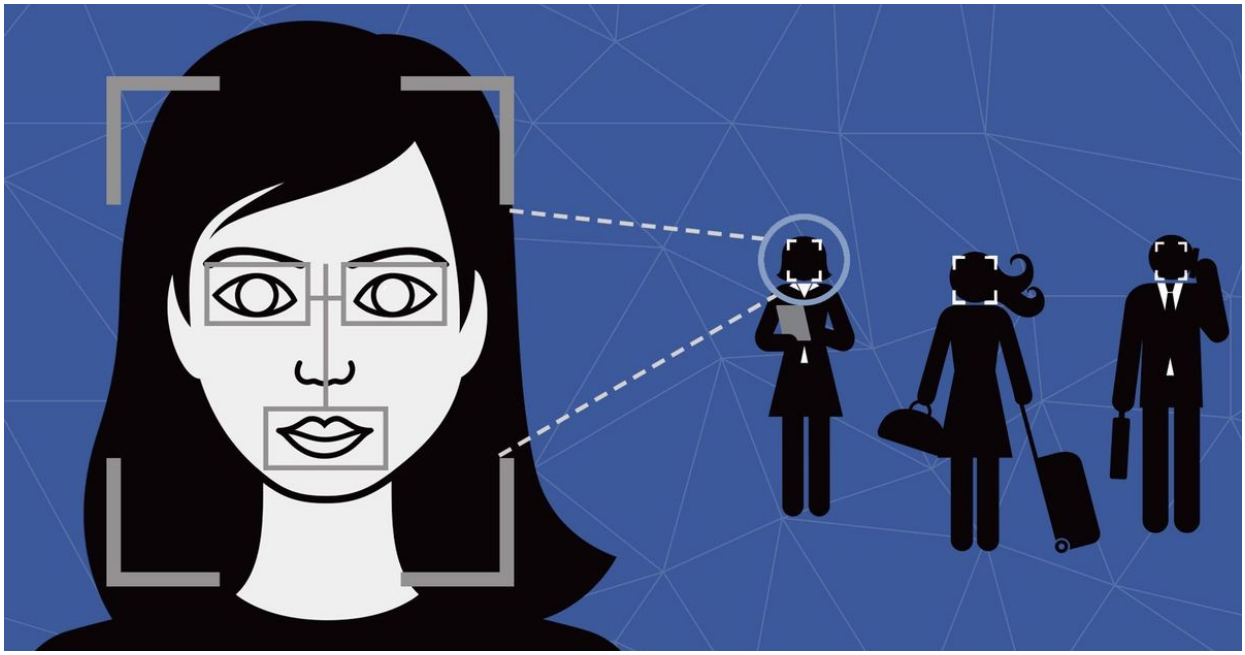

Figura 10: Reconocimiento facial

[16]

- **Finanzas:** algunas aplicaciones de la inteligencia artificial en las finanzas son prevenir fraudes o realizar pronósticos en los mercados [13] [17].

Las aplicaciones de la inteligencia artificial en el mundo real son múltiples; aún así existen muchísimas más de las que aquí se mencionan.

En el caso del **aprendizaje por refuerzo**, algunas de sus aplicaciones reales son:

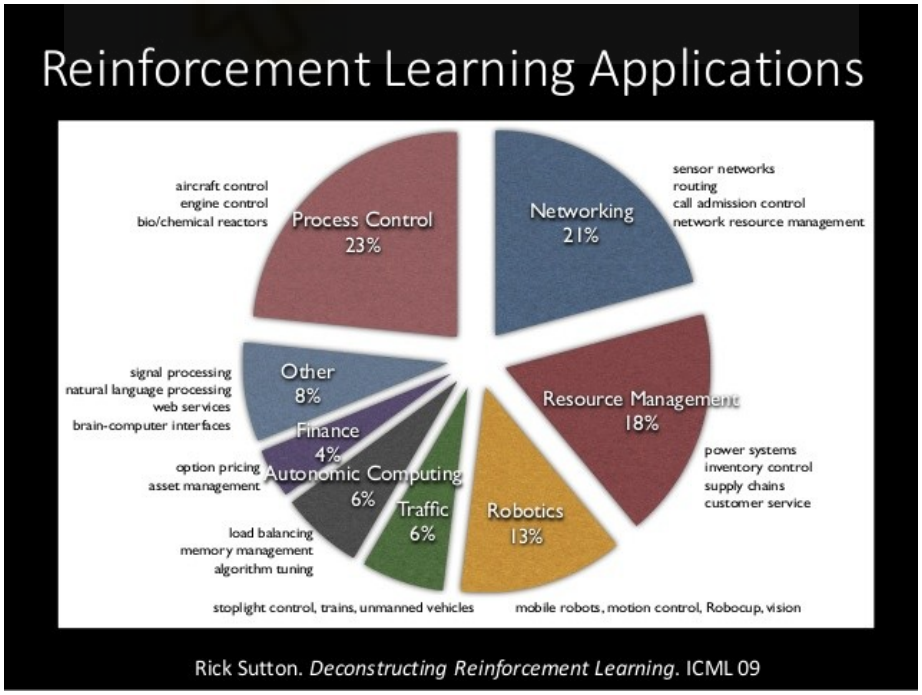

Figura 11: Aplicaciones del aprendizaje por refuerzo en la vida real

[18]

- **Manejo de recursos en clústers de ordenadores:** en el paper “Resource Management with Deep Reinforcement Learning” [19] se muestra como usar RL (“*Reinforcement Learning*”) para automáticamente asignar los recursos de los ordenadores, con el objetivo de minimizar los retrasos en el trabajo [20].
- **Control de luces de tráfico:** en el paper “Reinforcement learning-based multi-agent system for network traffic signal control” [21], los investigadores intentaron diseñar un controlador de luces de tráfico que resolviera el problema de la congestión. Aunque solo se testeó en un entorno simulado, sus métodos mostraron resultados superiores a los métodos tradicionales [20].

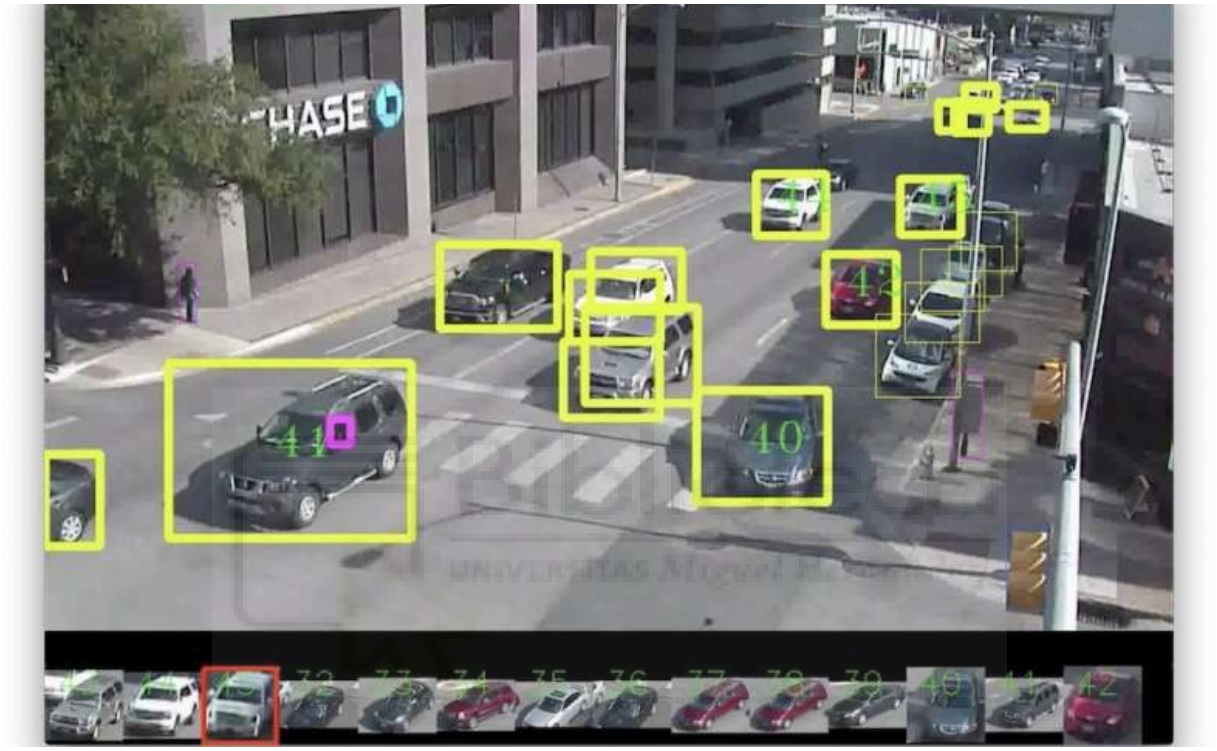

Figura 12: Control del tráfico mediante inteligencia artificial
[22]

- **Robótica:** existen multitud de aplicaciones en robótica; algunas están referenciadas en el paper “Reinforcement Learning in robotics: a survey” [23]. Por poner un ejemplo más concreto, en el paper “End-to-end training of Deep Visuomotor Policies” [24], se entrena a un robot para aprender normas para mapear imágenes de vídeo en bruto y convertirlas en acciones del robot. Las imágenes RGB se introducían en una CNN y las salidas eran los pares motor. El componente de RL era la estrategia de búsqueda guiada para generar datos de entrenamiento que vinieran de su propios estados [20].
- **Química:** el aprendizaje por refuerzo se puede aplicar en optimizar reacciones químicas. En el paper “Optimizing Chemical Reactions with Deep Reinforcement Learning” [25] se muestra que su modelo sobrepasó al algoritmo más avanzado actual. [20]
- **Juegos:** el aprendizaje por refuerzo es bastante conocido y utilizado actualmente para resolver diferentes juegos y, en algunos casos, alcanzar rendimientos superiores a los de un humano. Así por ejemplo, en los papers [3] y [4], ya citados anteriormente, encontramos que

se superan, en algunos casos, las puntuaciones que un humano podría alcanzar en varios juegos de Atari 2600 [20].

2. ¿Qué lenguaje de programación usar?

A la hora de programar, existen multitud de lenguajes de programación, enfocados a diferentes ámbitos y a diferentes aplicaciones: scripts, programación de aplicaciones, programación web…

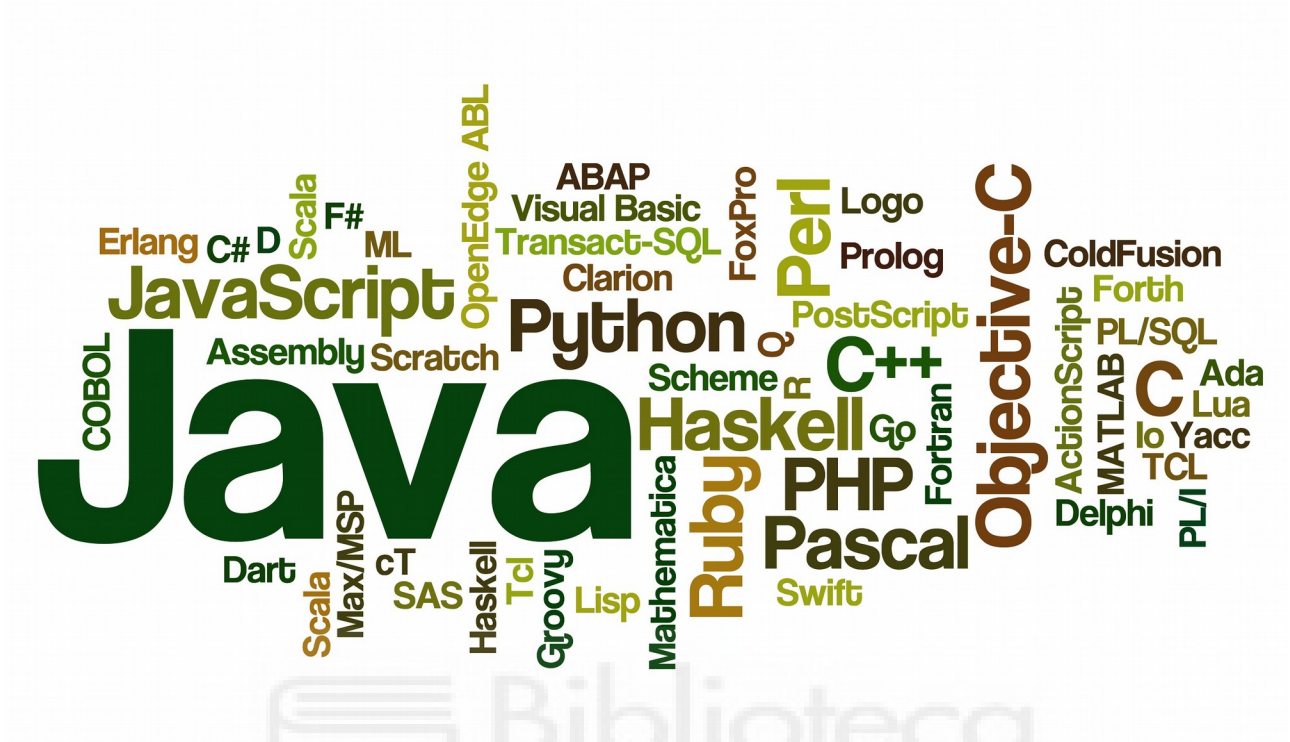

Figura 13: Lenguajes de programación

[26]

Uno de los **requisitos** a la hora de realizar este trabajo, era buscar un **lenguaje de programación que cumpliera con las siguientes premisas:**

- **Sencillez y que fuera amigable**; hay que recordar que los **conocimientos de programación** adquiridos, al no provenir de una carrera o formación puramente informática, son **limitados**, por tanto **un lenguaje sencillo de aprender e implementar reduce el tiempo de aprendizaje** en caso de que no sea conocido, y de desarrollo del proyecto.
- **Cantidad de información y recursos: es importante que para el lenguaje en cuestión haya bastante información en línea**; cursos, libros, ebooks, foros activos y una comunidad grande y bien asentada de programadores ayudan a que, en caso de que surja algún problema, podamos encontrar ayuda a la hora de resolverlo.
- **Cantidad de recursos enfocados al machine learning y/o inteligencia artificial:** de nada serviría un lenguaje que tiene una ingente cantidad de información y librerías pero ninguna enfocada a lo que buscamos, ya que esto no nos facilitaría el trabajo.
- **Coste del lenguaje:** se valoró también el hecho de que el lenguaje fuera libre, es decir, no hubiese que pagar por o bien todas o la mayoría de librerías, entornos de programación, soporte, etc.

Con estas premisas, se pasará a comentar los lenguajes más relevantes, comparándolos entre sí; finalmente se realizará una tabla comparativa y se dará una conclusión sobre el lenguaje elegido.

2.1 C

Figura 14: Lenguaje de programación C [27]

C es un lenguaje de programación **desarrollado por Dennis Ritchie entre 1969 y 1972 en los Laboratorios Bell**, como evolución del anterior lenguaje B.

Es un lenguaje **orientado en sus inicios a la implementación de sistemas operativos.** C es apreciado por su **código eficiente**; es, de hecho, el lenguaje de programación más **popular para crear software de sistemas**, aunque también se utiliza para crear aplicaciones.

Se trata de un lenguaje de tipos de datos estáticos, débilmente tipificado, de medio nivel, ya que dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a bajo nivel. Los compiladores suelen ofrecer extensiones que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o a periféricos.

Algunas de sus **características más reseñables** son:

- **Núcleo de lenguaje simple**; las funcionalidades añadidas importantes, como funciones matemáticas o de manejo de archivo, son proporcionadas por bibliotecas.
- **Lenguaje flexible**; esto facilita la programación en múltiples estilos. Normalmente se usa un estilo de lenguaje estructurado.
- **Se puede acceder a la memoria de bajo nivel** mediante el uso de punteros.
- El **número de palabras clave o reservadas** del lenguaje es **reducido.**
- **Lenguaje muy eficiente** puesto que es posible utilizar sus características de bajo nivel para realizar implementaciones óptimas.
- Ofrece **compiladores para casi todos los sistemas conocidos.** Es de hecho el lenguaje más portado actualmente.

Algunas de sus **carencias más importantes** son:

- La **velocidad de aprendizaje** de C, frente a lenguajes de datos dinámicos es mucho menor.
- **La gestión de memoria depende del usuario;** esto nos obliga a reservar y liberar la memoria explícitamente.
- **No dispone de recolector de basura nativo,** aunque existen librerías que solventan este inconveniente.

- Su **falta de soporte para programación orientada a objetos,** aunque nuevamente tenemos librerías para solucionar esto.
- **No soporta funciones anidadas,** aunque algunos compiladores, como GCC tienen esta característica como extensión.
- **No soporta programación nativa multihilo.** Se puede solucionar usando librerías externas.

  [28]

Como se ve, C tiene actualmente algunas carencias importantes, aunque muchas son salvables usando librerías externas. **La principal característica por la que este lenguaje sigue vigente en la actualidad, es su rapidez,** ya que está al nivel de velocidad de lenguaje ensamblador. Otra de sus características es la posibilidad de crear compiladores para distintas plataformas.

A nivel de inteligencia artificial y machine learning, nos encontramos que, en base a una encuesta realizada en el año 2017 a más de 2000 profesionales del ámbito [29], C/C++ (los dos lenguajes suelen ir unidos en muchos casos) e**s el segundo lenguaje más utilizado,** tanto en uso como en prioritización.

C/C++ se utiliza más por aquellos que quieren mejorar sus proyectos con machine learning, que por aquellos que quieren construir desde cero una aplicación basada en inteligencia artificial [30].

En cuanto a librerías de inteligencia artificial disponibles en este lenguaje, encontramos que tensorflow está disponible en C [31], pero caffe [32] y theano [33] no.

2.2 C++

Figura 15: Lenguaje de programación C++

[34]

C++ es un lenguaje de programación diseñado en 1970 por Bjarne Stroustrup. **Se creó para extender al lenguaje de programación C** ciertos mecanismos que permiten la manipulación de objetos.

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos.

El nombre de C++ fue propuesto por Rick Mascitti en 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico.

Algunas de sus **principales características** son:

- **Soporta programación orientada a objetos.**
- **Bajo nivel de abstracción.**
- **Manejo de memoria manual.**
- **Ligero, lenguaje compilador.**

- **Puede usarse para programar casi cualquier cosa.**
- **Principales usos:** aplicaciones de servidor, redes, juegos, drivers de dispositivos.

En cambio, algunos de sus principales **problemas** son:

- **Complejo de aprender.**
- **Ausencia de recolector de basura.**
- **Tiempos de compilación lentos.**
- **Falta de mensajes de error detallados.**

[35] [36]

Podría decirse que **C++ puede hacer todo lo que hace C, pero con funcionalidades añadidas.**

En cuanto al campo de interés de este proyecto, el del **machine learning y la inteligencia artificial,** podemos decir que al estar tensorflow disponible para C [31], es compatible con C++ también; como añadido, otra de las librerías más famosas, caffe, sí está disponible para C++ [32]; theano, otra de las librerías más utilizadas, no está disponible para este lenguaje [33].

2.3 Java

Figura 16: Icono Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos, diseñado para tener las mínimas dependencias de implementación posibles. La intención principal con la que se concibió fue permitir que **los desarrolladores escriban el programa una vez y lo ejecuten en cualquier dispositivo** (conocido en inglés como *WORA, "write once, run anywhere"*, que traducido quiere decir "*escribir una vez, ejecutar en cualquier sitio"*); es decir, la idea es que **el código que es ejecutado en una plataforma no tiene que recompilarse para correr en otra.** A partir de 2012, Java gana en popularidad en el campo de las aplicaciones cliente-servidor web, con aproximadamente unos 10 millones de usuarios.

[37]

Java fue desarrollado originariamente por James Gosling, de la empresa Sun Microsystems (constituida en 1982 y adquirida en 2010 por Oracle). **Java fue publicado en 1995.** Su sintaxis es derivada de C y C++, pero tiene menos utilidades de bajo nivel. Las aplicaciones de Java una vez compiladas pueden ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora en la que se ejecuta.

Java se creó con los siguientes **cinco objetivos principales:**

1. Usar **programación orientada a objetos.**
2. Permitir la **ejecución de un mismo programa en múltiples sistemas operativos.**
3. **Incluir por defecto soporte para trabajo en red.**
4. Debe estar **diseñado para ejecutar código en sistemas remotos** de forma segura.

5. Debe de ser **fácil de usar** y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

Para conseguir la ejecución de código remoto y el soporte de red, a veces los programadores de Java tienen que recurrir a extensiones externas [38].

En cuanto a la **inteligencia artificial,** encontramos que **existen algunas librerías disponibles** para Java; si bien **no es el lenguaje más utilizado** para este ámbito, sí uno de ellos [39].

Como principales librerías, encontramos que **tensorflow da soporte para Java** [40], aunque este desde la web oficial de tensorflow nos avisan que **no dan garantías sobre su estabilidad.**

Existe, por otro lado, una **librería específica para Java llamada deeplearning4j;** si bien es aún muy joven (su *release* estable es de Agosto del 2017) [41].

Consultando datos de la encuesta realizada a más de 2000 profesionales del sector, así como una entrada en la web de IBM sobre qué lenguaje de programación es el más demandado en áreas como *machine learning* o *data science* [29] [42], **Java aparece en tercer puesto en las búsquedas de empleos relacionados;** asimismo, en la encuesta se les preguntó acerca de 17 áreas de desarrollo distintas, quedando patente que el uso de Java en el sector es minoritario; el uso donde Java destaca es en aquél relacionado con redes.

2.4 Python

Figura 17: Icono Python [43]

Python es un lenguaje de programación interpretado, el cual **hace hincapié en una sintaxis de código legible y facilidad aprendizaje.**

Es un lenguaje de programación multiparadigma, ya que **soporta programación orientada a objetos, programación imperativa y programación funcional. Además, usa tipado dinámico y es multiplataforma.**

Está administrado por la Python Software Foundation, poseyendo licencia de código abierto.

Python fue creado a finales de los ochenta por Guido van Rossum, como sucesor del lenguaje de programación ABC. Su nombre proviene de la afición de su creador por el grupo humorista británico Monty Python.

Algunas de sus **características más relevantes** son:

- **Multiparadigma:** permite varios estilos de programación.
- **Tipado dinámico** y conteo de referencias para la administración de memoria.
- **Facilidad de extensión:** se pueden escribir nuevos módulos en C o C++ con relativa sencillez. Python puede incluirse en aplicaciones que necesitan una interfaz programable.

- **Multiplataforma.**
- **Sencillo de leer y sencillo de escribir.**
- **De alto nivel.**
- **Gratuito.**
- **Extensible:** podemos escribir parte del código en otros lenguajes como C++ y hacerlo compatible con Python.
- **Librería estándar muy grande:** Python viene con muchísimas funcionalidades incluidas por defecto, lo que evita que tengamos que estar buscando librerías para realizar la mayoría de funciones.
- **Gran cantidad de librerías externas,** la inmensa mayoría libres y gratuitas.

  [44] [45]

Además de todas estas características, los usuarios de Python se refieren a menudo a la **filosofía Python,** la cual es bastante análoga a la filosofía que sigue Unix. Se dice que **el código que sigue los principios de Python de legibilidad y transparencia se dice que es "pythonico"** (*"pythonic"* en inglés). El código opaco u ofuscado, se dice que es "no pythonico" (*"unpythonic"* en inglés). Estos principios fueron descritos por el desarrollador de Python Tim Peters en El Zen de Python.

Figura 18: Taza zen of Python
[46]

Aquí están los **principios propuestos:**

- **Bello es mejor que feo.**
- **Explícito es mejor que implícito.**
- **Simple es mejor que complejo.**
- **Complejo es mejor que complicado.**
- **Plano es mejor que anidado.**
- **Disperso es mejor que denso.**
- **La legibilidad cuenta.**

- **Los casos especiales no son tan especiales como para quebrantar las reglas.**
- **Lo práctico gana a lo puro.**
- **Los errores nunca deberían dejarse pasar silenciosamente.**
- **A menos que hayan sido silenciados explícitamente.**
- **Frente a la ambigüedad, rechaza la tentación de adivinar.**
- **Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.**
- **Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.**
- **Ahora es mejor que nunca.**
- **Aunque nunca es a menuda mejor que ya mismo.**
- **Si la implementación es difícil de explicar, es una mala idea.**
- **Si la implementación es fácil de explicar, puede que sea una buena idea.**
- **Los espacios de nombres (*namespaces*) son una gran idea, ¡Hagamos más de esas cosas!**

[44] [47]

Es, como se puede apreciar, un lenguaje muy **enfocado a la sencillez y facilidad** en todos sus ámbitos.

El **intérprete de Python** estándar incluye un **modo interactivo;** dicho de otro modo, **podemos introducir las expresiones una a una, pudiendo evaluarlas** inmediatamente, lo cual da la posibilidad de probar porciones de código antes de integrarlo como parte de un programa. Esto resulta muy útil, tanto para principiantes como para programadores avanzados.

Además del intérprete de Python estándar, existen otros intérpretes como IDLE, bpython o IPython, que añaden funcionalidades extra al modo interactivo como autocompletado de código y coloreado de la sintaxis del lenguaje.

Actualmente, existen dos versiones en uso de Python: la 2.x y la 3.x.

En febrero de 2009 se lanzó la primera versión de Python 3.x. Esta nueva versión incluye cambios que requieren reescribir el código de versiones anteriores. Para facilitar este proceso junto con Python 3 se ha publicado una herramienta automática llamada 2to3 [44].

En la actualidad, la versión más moderna de Python es la 3.7.2, con fecha de Junio del 2018 [48]. Es, por tanto, un lenguaje que se actualiza con frecuencia, añadiendo nuevas funcionalidades.

En cuanto al estado de **Python en los campos de machine learning e inteligencia artificial,** resulta que es el **lenguaje más utilizado.** Basándonos en la encuesta realizada a más de 2000 profesionales del sector [29], encontramos un porcentaje del **57% de personas que lo utilizan y un 33% que priorizan el desarrollo de Python.** Por realizar una comparativa, en C/C++ encontramos un 44% de personas que lo utilizan y un 19% en prioritización.

Para conocer el estado real de este lenguaje de programación, se ha utilizado un artículo publicado en la web de IBM [42], en el cual realizan una búsqueda en un portal de empleo, con los términos

machine learning o *data science;* como resultado, el lenguaje más demandado era Python, seguido por Java.

Así pues, **para Python se pueden encontrar una gran variedad de liberías y herramientas disponibles:** librerías matemáticas como NumPy [49], además de otras **específicas para *machine learning* e inteligencia artificial** como tensorflow (su soporte oficial es para Python) [50], así como theano [51], caffe [52] o keras [53].

2.5 R

Figura 19: Icono lenguaje de programación R

R es un entorno y lenguaje de programación enfocado al análisis estadístico. Nació como una reimplementación de software libre del lenguaje S. R **es uno de los lenguajes de programación más utilizados en la investigación por la comunidad estadística;** es, además muy popular en campos como minería de datos, investigación biomédica, bioinformática y matemáticas financieras. Es posible cargarle diferentes bibliotecas o paquetes con funcionalidades de cálculo y gráficas.

R está disponible para Windows, Mac, Unix y GNU/Linux.

Este lenguaje fue desarrollado inicialmente por Robert Gentleman [54] y Ross Ihaka, del departamento de estadística de la Universidad de Auckland en 1993. Su desarrollo actual es responsabilidad del R Development Core Team.

Al tratarse de un lenguaje de programación, permite que los usuarios lo extiendan definiendo sus propias funciones. La gran mayoría de funciones de R están escritas en el propio lenguaje, aunque para algoritmos computacionalmente exigentes e**s posible desarrollar bibliotecas en C, C++ o Fortran.** También existen paquetes que extienden las funcionalidades de R, muchos desarrollados por la comunidad de usuarios.

R, al igual que S, **está orientado a objetos.** Una gran ventaja de R es que **puede integrarse con distintas bases de datos;** existen también bibliotecas que facilitan su utilización desde lenguajes de programación interpretados como Perl y Python. Un uso habitual de R es el de cálculo numérico, en lo que **rivaliza con Octave y Matlab.**

Algunas de las **características más interesantes** del entorno R son:

- Un **manejo efectivo de datos y facilidad de almacenamiento.**
- Una **suite de operadores para cálculos con arrays;** en particular, matrices.
- Una **gran y coherente colección de herramientas intermedias para análisis de datos.**
- Facilidades **gráficas para análisis de datos.**
- Un **lenguaje** bien desarrollado, **simple y efectivo,** que incluye condicionales, bucles, funciones recursivas definidas por el usuario y facilidades para entrada y salida de datos.

- **Integración con software comercial** como Mathematica, Matlab o Statistica entre otros.
- **Proyecto colaborativo y abierto:** los usuarios pueden publicar paquetes para extender nuevas funcionalidades.
- R **puede ser invocado desde código desarrollado en otros lenguajes** como Python, Perl, Ruby y F#. También permite desarrollar scripts en R.

[55] [56]

¿Qué opciones encontramos para este lenguaje en el campo que nos ocupa? Encontramos varios paquetes para R, tanto para clasificación y regresión, como tratamiento de datos [57] [58]. En cuanto a **redes neuronales, encontramos un par de librerías disponibles,** siendo keras [59] la más completa de ambas [60].

A nivel profesional, consultando la encuesta llevada a cabo a más de 2000 profesionales del sector [42], encontramos que **el lenguaje R es usado en un 7% de los casos.**

Si hablamos de **deep learning, R es el quinto lenguaje preferido** para este tipo de aplicaciones [29].

2.6 Matlab/Octave

Matlab (abreviatura de Matrix Laboratory, en castellano "laboratorio de matrices") e**s un sistema algebraico computacional que ofrece un IDE con un lenguaje de programación propio** (lenguaje M). Está disponible para Unix, Windows, Mac OS X y Linux.

Figura 20: Icono MATLAB

[61]

Algunas de **sus características básicas son:**

- **Manipulación de matrices.**
- **Representación de datos y funciones.**
- **Implementación de algoritmos.**
- **Creación de interfaces de usuario (GUI).**
- **Comunicación con programas en otros lenguajes y** con otros dispositivos **hardware.**

Además de todo esto, **Matlab contiene dos herramientas fundamentales para expandir sus prestaciones: Simulink** (plataforma de simulación multidominio) **y GUIDE** (editor de interfaces de usuario(GUI)). Además Matlab incluye las llamadas *toolboxes*, y en Simulink los *blocksets*.

Matlab es **muy utilizado en el ámbito universitario,** así como en centros de investigación y desarrollo.

M es un **lenguaje de programación interpretado,** y puede ejecutarse tanto en el entorno interactivo como a través de un archivo de script (extensión de archivo *.m). El lenguaje de Matlab **permite operaciones de vectores y matrices, funciones y programación orientada a objetos entre otros.** Además, tiene funciones para **visualizar datos tanto en 2D como en 3D.**

Algunas de sus principales **limitaciones** son:

- Al ser un producto propietario de The Mathworks, **los usuarios están sujetos a las decisiones de la empresa.**
- Uno de los principales problemas de Matlab ha sido siempre que **sus programas solamente se podían ejecutar en su entorno,** es decir, necesitamos de una copia de Matlab para poder correr los programas. Recientemente se añadió una herramienta llamada Matlab Builder, bajo la sección "Application Deployment" para **utilizar funciones Matlab como archivos de biblioteca que pueden ser usados con .NET o Java.** La gran desventaja es que necesitamos el MCR (Matlab Component Runtime) para que los archivos funciones correctamente, aunque el MCR se distribuye libremente, pudiéndose generar en el momento en que se compila el programa desde Matlab.
- **Alto consumo de recursos.**

[62] [63]

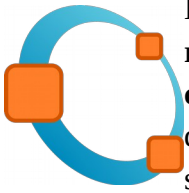

Figura 21: Icono GNU Octave

En cuanto a **Octave, es un programa y lenguaje de programación enfocado a la realización de cálculos numéricos.** Está bajo el paraguas de GNU, y está **considerado la alternativa libre a Matlab** (de ahí que se haya decidido agrupar los dos programas en este pequeño análisis). Comparten características como que ambos son lenguajes interpretados, y que permiten ejecutar órdenes en modo interactivo.

Octave fue creado alrededor de 1988, siendo s**u finalidad original un curso de diseño de reactores químicos. En 1992, se decidió extenderlo,** corriendo

[64]

su desarrollo a cargo de John W. Eaton. El nombre de Octave surge de Octave Levenspiel, profesor de uno de los autores y conocido por sus buenas aproximaciones, por medio de cálculos elementales, a problemas numéricos en ingeniería química. La **sintaxis es casi idéntica a la usada en Matlab,** otro de los motivos por los que se ha decidido agruparlos.

Sus **características más destacables** son:

- **Lenguaje interpretado.**
- **Permite generar scripts.**
- **Soporta** gran parte de las **funciones estándar de C.**
- **Posibilidad de extensión** para ofrecer compatibilidad con UNIX.
- **Pensado para trabajar con matrices.**

- **Dispone de un entorno de desarrollo integrado.**
- Al ser su licencia pública GNU, **puede ser compartido y utilizado libremente.**

[65]

En cuanto al uso de estos lenguajes para inteligencia artificial, encontramos que **Matlab incluye una toolbox de redes neuronales** (antes llamada *Neural Network Toolbox*, ahora *Deep Learning Toolbox*) [66] bastante completa, aunque de pago (las *toolbox* de Matlab se distribuyen con una licencia propia cada una, por lo que si queremos esa *toolbox* en concreto debemos pagar por la licencia de Matlab + la *toolbox*). Como posible solución a este problema, encontramos distintas *toolboxes* programadas por usuarios anónimos que se distribuyen gratuitamente [67]. Aún así, si queremos usar Matlab deberemos pagar por su licencia.

Para Octave, encontramos que algunas de las *toolboxes* de Matlab pueden funcionar en él, además de alguna propia; estas están programadas por usuarios y distribuidas libremente [68].

En cuanto al **uso profesional** que se le da a Matlab/Octave, como se observa en [29], **su prioritización a nivel de desarrollo es menor del 5%; es decir, su principal uso es académico,** pero a la hora de crear una aplicación basada en inteligencia artificial para su distribución, se recurre a otros lenguajes (como se ha visto en otros apartados).

2.7 Otros lenguajes

A modo informativo, **se mencionarán algunos otros lenguajes que se usan para inteligencia artificial o machine learning.** El uso de estos lenguajes no está tan extendido como los mencionados previamente, por eso no se valoró a la hora de elegir qué lenguaje aprender o utilizar.

Figura 22: Código de programación

[69]

- **AIML:** son las siglas de (*Artificial Intelligence Markup Language)*, y es muy utilizado en chatbots tipo ALICE.
- **Lisp:** es una notación matemática práctica para programas de ordenador basada en cálculo lambda.
- **Prolog:** es un lenguaje declarativo donde los programas se expresan en términos de relaciones. Es particularmente útil para razonamiento simbólico, bases de datos y aplicaciones de análisis de lenguaje. Su uso está bastante extendido en IA.
- **Haskell:** permite aplicar evaluación perezosa, haciendo sencillo expresar algoritmos no determinísticos, muy utilizados en IA. También permite expresar los algoritmos de manera composicional. Su único inconveniente es que trabajar con gráficos en este lenguaje es algo difícil.
- **Wolfram Language:** contiene funciones altamente automatizadas como predecir y clasificar, así como funciones basadas en métodos específicos y diagnóstico. Las funciones trabajan en muchos tipos de datos, incluyendo numéricos, categóricos, series temporales, textos e imágenes.

[70]

Aparte de los mencionados, existen muchos otros lenguajes con capacidad para inteligencia artificial. Podemos decir, de hecho, que podemos realizar aplicaciones de este tipo en cualquier lenguaje de programación; la diferencia está en la facilidad que nos brinda cada lenguaje para ello así como la cantidad de herramientas disponibles.

2.8 Conclusión y tabla comparativa

En este apartado, se pretende **justificar por qué se ha utilizado un lenguaje concreto** y no cualquiera de los analizados. **Los puntos que se han tenido en cuenta son los siguientes:**

- **Facilidad de aprendizaje:** un lenguaje sencillo reduce el tiempo de aprendizaje.
- **Conocimientos previos del lenguaje:** relacionado con el punto anterior, si se tenían conocimientos de cada lenguaje analizado previamente a empezar con este trabajo fin de máster.
- **Comunidad:** una gran cantidad de foros de programación del lenguaje, y en especial de inteligencia artificial, nos facilitará la resolución de dudas cuando surjan.
- **Cantidad de herramientas disponibles para inteligencia artificial:** de nada sirve un lenguaje fácil de aprender, si para la tarea que queremos realizar no tenemos herramientas y programas enfocados a su resolución.
- **Portabilidad:** otra premisa que tuve en cuenta a la hora de elegir un lenguaje, es que los programas realizados en él pudieran ejecutarse en multitud de dispositivos, pensando con ello en futuros proyectos como creación de robots basados en aprendizaje por refuerzo. Si solamente podemos ejecutar los programas dentro de un entorno concreto, nos limitará

mucho en ese aspecto. También se valorará la facilidad de portar el código a otros dispositivos.

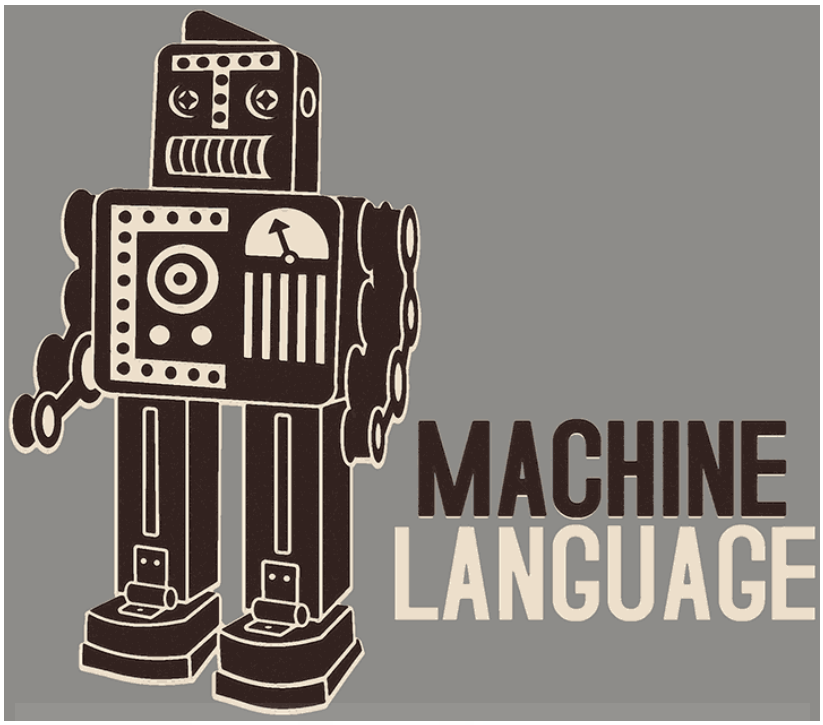

Figura 23: Robot Retro

[71]

Para tener una visión más clara de las virtudes y defectos de cada lenguaje para cada uno de los puntos expuestos, se exponen los **lenguajes valorados en la tabla 1,** valorando de 1 (peor) a 3 (mejor) cada apartado.

Lenguaje	Facilidad de aprendizaje	Conocimiento previo	Comunidad	Herramientas	Portabilidad	Total:
C	2	2	2	2	2	10
C++	1	1	2	2	2	8
Java	2	1	3	2	3	11
Python	3	1	3	3	3	13
R	2	1	1	1	2	7
Matlab/ Octave	3	3	2	3	1	12

Tabla 1: Valoración de los distintos lenguajes de programación

Análisis de los resultados: el lenguaje que, según los criterios establecidos es el más adecuado para este trabajo es **Python,** seguido en segundo lugar por Matlab/Octave; el tercer puesto corresponde a Java.

Partiendo de un conocimiento previo de Matlab, ya que se ha utilizado durante la realización del máster así como de cursos previos realizados sobre el programa, a priori, **podríamos pensar que es el lenguaje más indicado para la realización de este trabajo.** Si bien las **herramientas que presenta para la creación de redes neuronales son bastante buenas, algo que puntúa en su contra es la falta de portabilidad a otros sistemas, reduciendo su uso al entorno de Matlab.** Además su comunidad, a pesar de tener unos muy buenos foros en la web de Mathworks, no tiene una comunidad tan grande como otros lenguajes.

Comparándolo con el ganador, **Python, encontramos una gran variedad de herramientas para realizar tareas de todo tipo así como varias librerías para inteligencia artificial**; multitud de foros, cursos online, libros, tutoriales… Tanto de programación del lenguaje en general como de inteligencia artificial. Además, la **capacidad de poder funcionar en una gran variedad de dispositivos y sistemas operativos,** lo que nos permite, pensando en un futuro, realizar aplicaciones para una gran variedad de dispositivos, como Rasperri Pi e incluso Android, entre otros.

3. Herramientas, programas y hardware utilizados durante la elaboración del proyecto

Figura 24: Robot trabajando en PC

[72]

En este capítulo se mencionarán brevemente las herramientas más importantes así como el hardware utilizado durante este trabajo fin de máster.

3.1 Herramientas y programas

Figura 25: Código binario

[73]

En esta sección se tratarán tanto las herramientas como las librerías más influyentes.

3.1.1 Anaconda

Figura 26: Logo Anaconda

[74]

Anaconda es una **distribución libre y *open-source* de Python y R para computación científica** (ciencia de datos, aprendizaje máquina, etc) que intenta simplificar el manejo de paquetes, para lo cual utiliza el sistema de paquetes *conda*. Anaconda es **utilizado por más de 12 millones de usuarios,** incluyendo algunas grandes empresas como BMW, Cisco o Bloomberg.

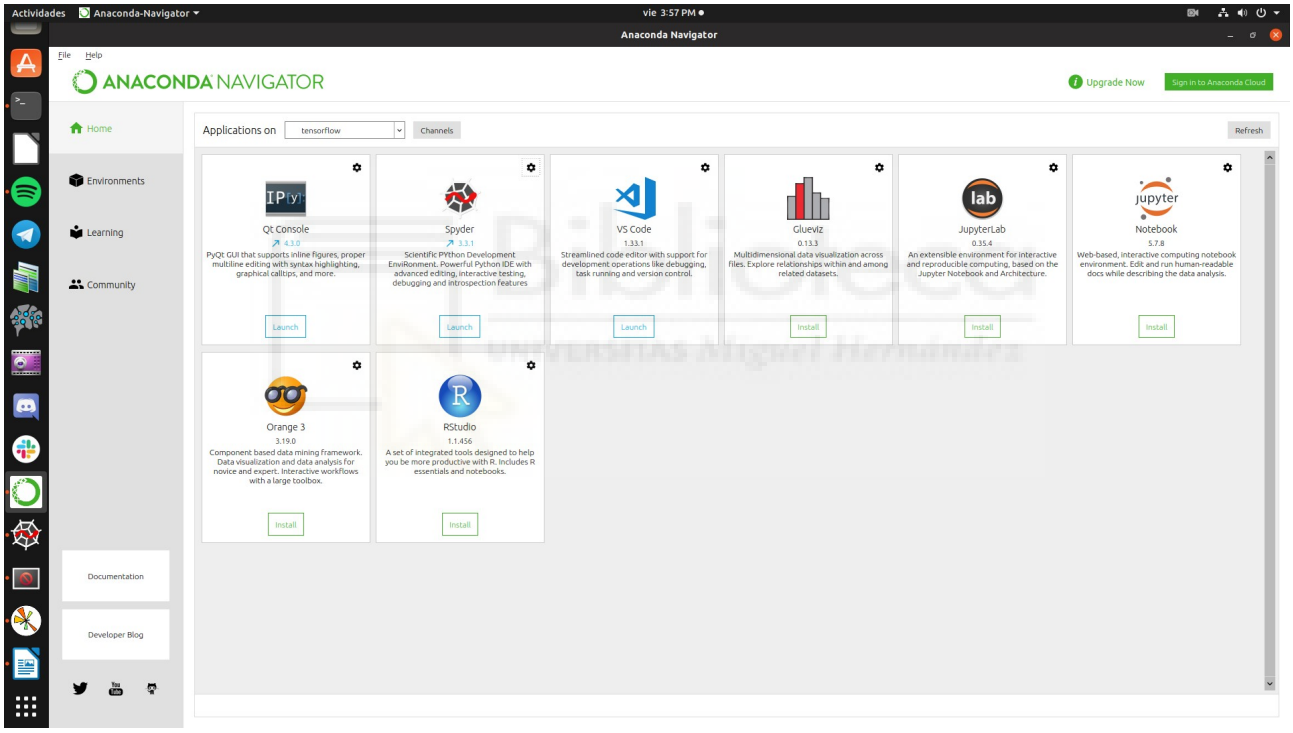

Figura 27: Captura de pantalla Anaconda Navigator

La primera versión de Anaconda data del 17 de Julio de 2012; l**a versión estable más reciente del 21 de Diciembre de 2018.** Anaconda está **disponible tanto para Linux, como Windows o MacOS,** y entre sus **características** más reseñables encontramos:

- **200 paquetes para ciencia de datos preinstalados.**
- **Más de 2000 paquetes disponibles** desde Anaconda Cloud, el repositorio donde se encuentran los paquetes para Anaconda.

- **Un gestor de entornos virtuales** (Anaconda Navigator), lo que nos permite gestionar los paquetes, entornos y aplicaciones de manera gráfica. No obstante se pueden instalar y gestionar los paquetes, entornos y aplicaciones tanto desde línea de comandos como desde el entorno gráfico.
- **Posibilidad de crear paquetes personalizados.**

Dentro de Anaconda Navigator, **las aplicaciones que encontramos por defecto instaladas son:**

- **JupyterLab**
- **Jupyter Notebook**
- **QtConsole**
- **Spyder**
- **Glueviz**
- **Orange**
- **Rstudio**
- **Visual Studio Code**

[75] [76] [77]

3.1.1.1 Spyder

Figura 28: Logo Spyder

[78]

Spyder es un IDE (Entorno de desarrollo integrado, *Integrated Development Enviroment* en inglés) **para programación científica escrito en Python para lenguaje Python.** Incluye algunos paquetes importantes para programación científica como NumPy, SciPy, Matplotlib, Pandas o Ipython. Está publicado bajo licencia del MIT.

Se publicó en 2009 por primera vez; desde 2012 se mantiene y mejora constantemente por un grupo de desarrolladores científicos de Python y la comunidad. Está disponible a través de Anaconda en Windows y Linux, y en macOS a través de MacPorts.

Algunas de **sus características más importantes son:**

- **Extensible con plugins** tanto propios como de terceros.
- **Editor con resalte de sintaxis, introspección y autocompletado de código.**
- **Posibilidad de explorar variables y editarlas a través de una GUI.**
- **Un profiler para hacer nuestro código más eficiente a nivel de tiempo.**

- **Un explorador de archivos** incorporado.
- **Un log histórico,** que recuerda cada comando introducido en cada consola.

[79] [80]

3.1.2 Tensorflow

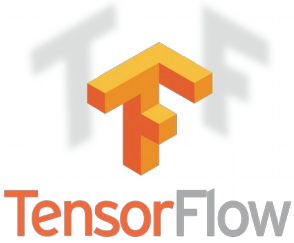

Figura 29: Logotipo Tensorflow

Tensorflow es una **librería gratuita y de código abierto muy utilizada en aprendizaje máquina.** Se usa tanto en ámbitos de investigación como de producción.

Inicialmente fue desarrollada por el equipo de Google Brain para uso interno de Google. Fue hecha pública el 9 de Noviembre de 2015. La versión 1.0.0 data del 11 de Febrero de 2017. Debido a su arquitectura flexible, **Tensoflow se puede ejecutar en múltiples CPUs y GPUs, así como en TPUs.** Está disponible en 64-bits para Linux, macOS, Windows, Android e iOS.

[81]

El nombre de Tensorflow **deriva de las operaciones que las redes neuronales realizan en arrays de datos multidimensionales, los cuales son conocidos como tensores** (*tensors* en inglés).

Para hacernos una idea del uso de Tensorflow, durante la conferencia Google I/O de Junio 2016, se estimó que habían unos 1500 repositorios en GitHub que mencionaban a Tensorflow, de los cuales solamente 5 eran de Google.

Algunas grandes empresas que utilizan Tensorflow son:

Figura 30: Compañias que usan TensorFlow

[82]

- Airbnb
- Airbus
- arm
- Cocacola
- Google
- Intel
- Lenovo
- Paypal
- Twitter

[82] [83]

3.1.3 Keras

Figura 31: Logtipo de Keras

Keras es una **librería de código abierto escrita en Python para redes neuronales, capaz de funcionar sobre Tensorflow, Theano, Microsoft Cognitive Toolkit y otros.** Keras nos **facilita la tarea de escribir redes neuronales,** ya que está diseñada para ser amigable y permitir la rápida implementación de modelos de redes neuronales.

Su principal autor y mantenedor es François Chollet, un ingeniero de Google. En 2017, el equipo de Tensorflow en **Google decidió darle soporte a Keras e implementarla conjuntamente con Tensorflow.**

Keras **está pensada** más **como una interfaz** que como un marco

[84]

de referencia. Contiene numerosas implementaciones de las estructuras usadas en la construcción de redes neuronales, tales como capas densas, convoluciones, activadores, optimizadores… Keras es capaz de funcionar tanto usando CPU como GPUs [85] [86].

3.1.4 OpenCV

Figura 32: Logotipo OpenCV

OpenCV (siglas de *Open Source Computer Vision Library*) es una **librería de visión y aprendizaje máquina de código libre.** El objetivo de OpenCV es proveer de una infraestructura común para las aplicaciones de visión por computador y acelerar el uso de visión artificial en productos comerciales.

Originariamente fue **desarrollada por Intel** y lanzada en el 2000, para después ser soportada por Willow Garage y después por Itseez (que fue adquirida de nuevo por Intel). Soporta los marcos de referencia de aprendizaje profundo de Tensorflow, Torch/Pytorch y Caffe.

[87]

La librería tiene **más de 2500 algoritmos optimizados** para usos muy variados, como detección y reconocimiento de caras y objetos, clasificación de acciones humanas en vídeos, seguimiento de objetos en movimiento, extracción de modelos en 3D de objetos, etc.

La comunidad de OpenCV cuenta con más de 47000 personas así como un número de descargas estimado de 14 millones. La librería es usada por grandes compañías, grupos de investigación y cuerpos gubernamentales.

A pesar de estar escrita originariamente en C++, la librería **tiene interfaces en C++, Python, Java y Matlab.** Soporta Windows, Linux, Android y MacOS. En la actualidad, se está desarrollando una interfaz para CUDA y OpenCL, lo cual hará esta librería más rápida para los usuarios de tarjetas gráficas Nvidia y AMD [88] [89].

3.1.5 NumPy

Figura 33: Logo de NumPy

NumPy es la **librería fundamental para computación científica en Python.** Contiene, entre otras **características:**

[90]

- **Un potente array de objetos N-dimensional.**
- **Funciones sofisticadas.**
- **Herramientas para integrar código C/C++ y Fortran.**
- **Capacidades de álgebra lineal, transformada de Fourier y números aleatorios.**

[90]

El nacimiento de NumPy es debido a que **Python no estaba diseñado originariamente para computación numérica,** pero el lenguaje pronto empezó a ser popular entre la comunidad científica y entre los ingenieros. En 1995, NumPy nace bajo el nombre de Numeric; pasó a llamarse NumPy en 2006 [91].

3.1.6 OpenAI-Gym

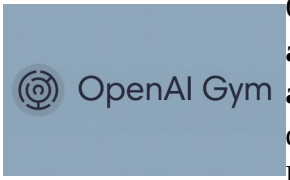

Figura 34: Logitipo de OpenAI-Gym [94]

OpenAI es una **organización de investigación de inteligencia artificial sin ánimo de lucro cuyos objetivos son desarrollar una inteligencia artificial amistosa** y asegurarse de que cuando una inteligencia artificial general sea desarrollada (sistemas autónomos que sobrepasen las capacidades humanas en la mayoría de trabajos económicamente valiosos) beneficie a toda la humanidad [92] [93].

Fue fundada en 2015 en San Francisco (algunos de sus miembros fundadores más notables son **Elon Musk** y Sam Altman). Algunas de sus **aportaciones más notables** son:

- **Gym:** la idea de gym es proveer de un **banco de pruebas general para inteligencia artificial,** con una gran variedad de entornos; la idea subyacente es similar a los sets de imágenes como ImageNet, usados como estándar en aprendizaje supervisado. Es el set que se ha utilizado en este proyecto, concretamente la parte del entorno correspondiente a Atari, la cual permite trabajar sobre varios videojuegos de la videoconsola Atari 2600.
- **Debate Game:** en 2018 OpenAI lanzó el juego de Debate, que enseñaba a las máquinas a debatir problemas ante un jurado humano. El propósito es investigar si esta aproximación puede usarse como asistente de inteligencia artificial.

Figura 35: Imagen instalaciones OpenAI [95]

- **OpenAI Five:** es el nombre designado de un equipo de cinco bots que se utilizan en el juego competitivo de 5vs5 *DOTA 2*, en el cual aprenden a jugar contra jugadores humanos de gran nivel enteramente a través de algoritmos de prueba y error.
- **Dactyl:** este proyecto usa *machine learning* para entrenar una mano robot Shadow Hand desde cero, usando el mismo algoritmo de aprendizaje por refuerzo que se utiliza en OpenAI Five.

[93]

3.1.7 Matplotlib

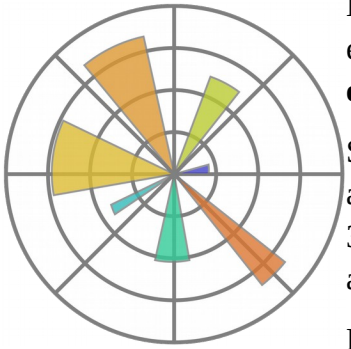

Figura 36: Logotipo Matplotlib

[98]

Matplotlib es una **librería para realizar gráficas en Python** y sus extensiones numéricas como NumPy. Está diseñada para parecerse al **estilo de gráficas de Matlab.**

Su primera versión data de 2003, y fue creada por John D. Hunter; actualmente, Matplotlib da soporte tanto a versiones de Python 2.x como 3.x. Se espera que para 2020 deje de dar soporte a las versiones anteriores a la 3.0.

La librería nos permite realizar las gráficas de la forma más simple posible: podemos generar plots, histogramas, diagramas de barras, etc, con unas pocas líneas de código [96] [97].

3.1.8 Google colaboratory

Figura 37: Logo Google Colaboratory

[99]

Colaboratory de Google es una herramienta recientemente lanzada para **educación en aprendizaje máquina.** Básicamente es una **entorno con una libreta Jupyter que no necesita instalación,** con lo cual podemos trabajar directamente en la nube y colaborando con otros usuarios si así lo deseamos. Al ser un proyecto de investigación, su uso es totalmente gratuito.

Colaboratory nos permite trabajar con nuestros proyectos de *machine learning* sin necesidad de hacer un gran desembolso de dinero en un ordenador potente, ya que pone a nuestra disposición potentes GPUs y TPUs para realizar nuestros cálculos, eso sí, con una **limitación de 12 horas por sesión** [100] [101]**.**

3.2 Hardware

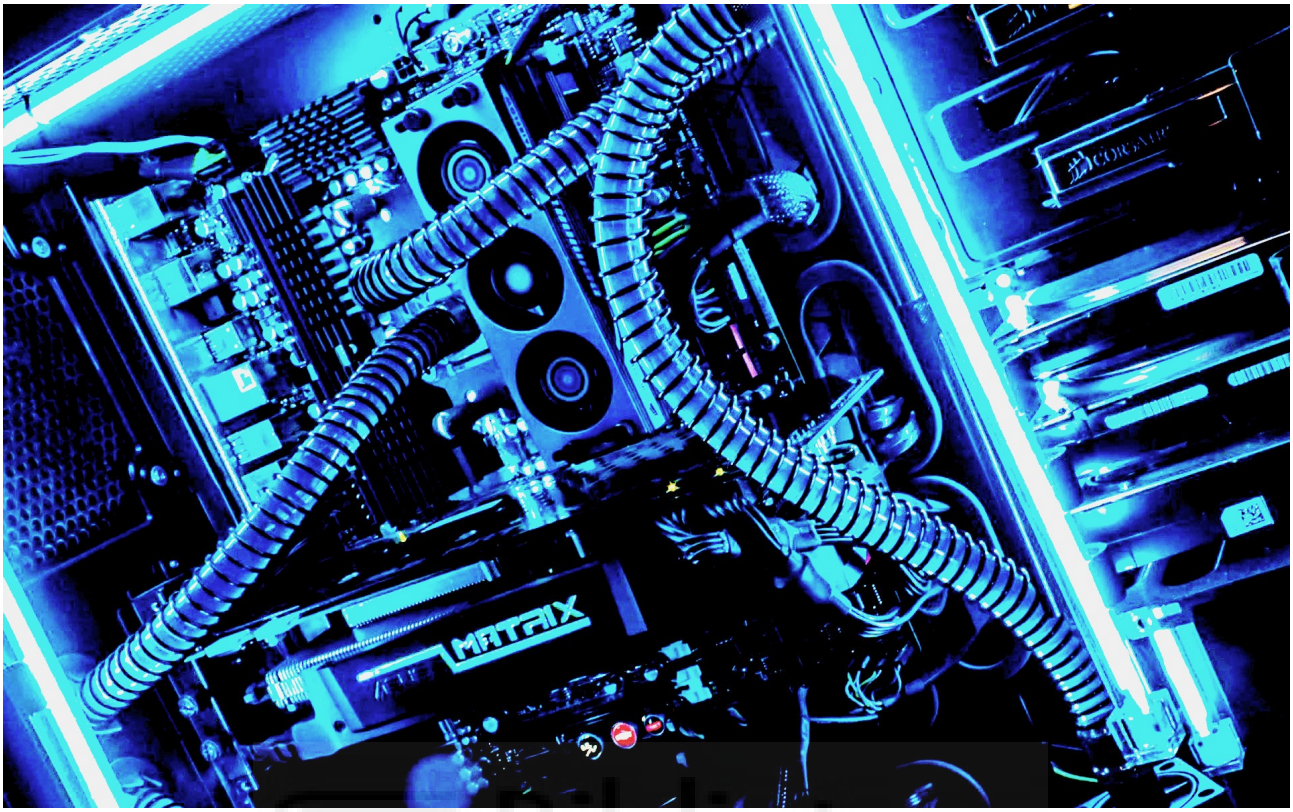

Figura 38: Hardware de ordenador

[102]

A la hora de realizar un proyecto de *machine learning*, **necesitamos tener un equipo con unas determinadas características.** En este capítulo se tratarán brevemente estas características, así como las características del equipo utilizado, para tener una referencia en caso de intentar reproducir el proyecto o emprender uno similar.

3.2.1 Placa base

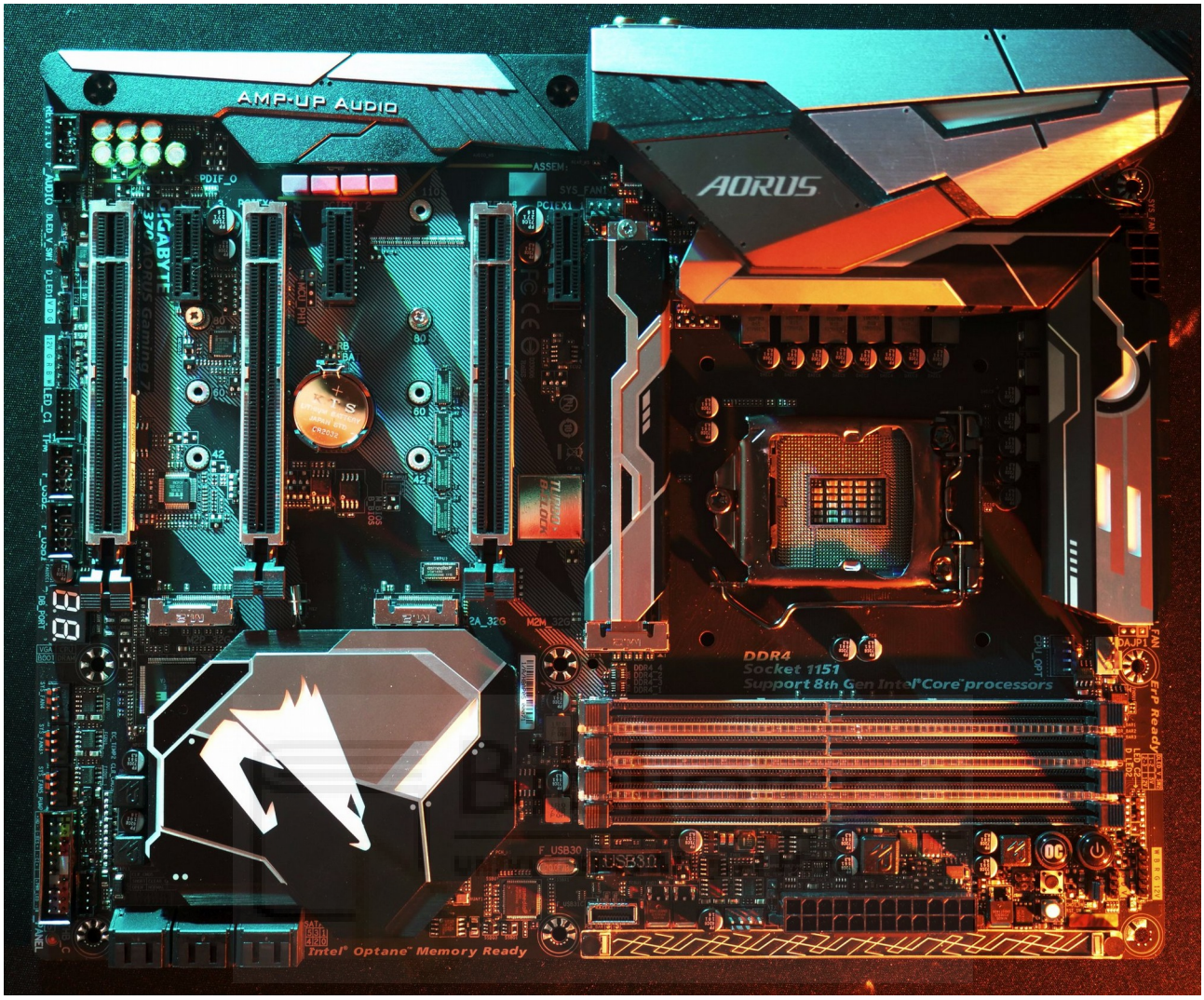

Figura 39: Placa base

[103]

La placa base se encarga de interconectar el resto de componentes del PC: CPU, RAM, GPU… Para aprendizaje máquina, una placa base debe tener:

- Suficientes bahías PCIe para todas las tarjetas gráficas que vayamos a instalar.
- Que soporte la CPU y GPU/GPUs que se vayan a montar.
- Suficientes bahías para la RAM que pensemos instalar.

Actualmente, **la mayoría de tarjetas gráficas del mercado tienen una anchura de dos slots PCI express,** por lo que es algo a tener en cuenta a la hora de elegir una placa base: la separación entre bahías debe permitir la instalación de todas las tarjetas gráficas que vayamos a utilizar en nuestro proyecto.

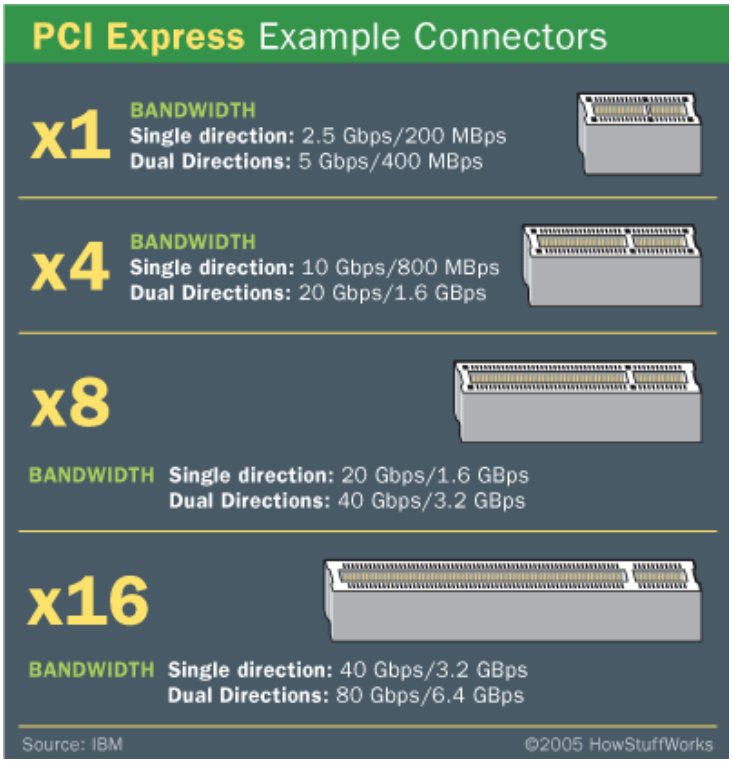

Figura 40: Tamaños y velocidades de transferencia de los distintos conectores PCI Express

[104]

En cuanto al número de carriles PCIe (*PCIe lanes*), según [105], **el número de *PCIe lanes* no es importante excepto que tengamos muchas tarjetas gráficas conectadas** (más de 4). Lo que sí debemos tener en cuenta es el formato de PCIe; como se explica en [106], es importante que sean del tamaño correcto: un carril PCIe x16 podrá acomodar tarjetas gráficas x1, x4, x8 y x16, pero una x4 no podrá acomodar x8 ni x16, solamente x1 y x4. Generalmente encontramos que las placas base actuales llevan al menos un carril x16, y generalmente uno o más carriles x8.

3.2.2 CPU

La CPU (siglas de *Central Processing Unit, unidad central de procesamiento*) se encarga de tomar las entradas de instrucción de la RAM del ordenador, las decodifica y procesa la acción, entregando finalmente una salida [107].

El punto fuerte de una CPU es ejecutar unas cuantas operaciones complejas muy frecuentemente; el aprendizaje máquina es justamente lo contrario, ya que la mayoría de procesos de computación durante el entrenamiento son multiplicaciones matriciales, el cual es un cálculo sencillo. **El problema a la hora de usar una CPU es que no es capaz de manejar la cantidad de cálculos necesarios para *machine learning*** [108].

Figura 41: AMD Ryzen 5. Nótese el tamaño de la caja con respecto al tamaño de la CPU. [109]

Comparándolas con las GPUs (se comentarán en apartados posteriores), encontramos que **los valores de velocidad de los relojes de las GPUs son mayores, además de que la cantidad de núcleos de una GPU es mayor que en una CPU** [110]. Por tanto, se puede concluir que, **para aprendizaje máquina, rara vez utilizaremos una CPU** (a no ser que carezcamos de gráfica), por lo que **no es determinante para realizar este tipo de tareas.** Debe ser lo suficientemente potente para mover los programas que instalemos y ejecutemos, pero si queremos ser eficientes, debemos utilizar una GPU.

3.2.3 Memoria RAM

La memoria RAM será la encargada de almacenar la mayoría de variables y datos del programa mientras esté ejecutándose. **Para proyectos de *machine learning*, necesitaremos una cantidad, como mínimo, equivalente a la RAM de nuestra GPU** (ya que, suponiendo que la utilicemos toda, podremos transferir todos los datos a la GPU) [105].

En cuanto a **la frecuencia de reloj de la RAM,** realmente **no es importante:** la diferencia entre una RAM rápida o una lenta está en torno al 3% como máximo [105].

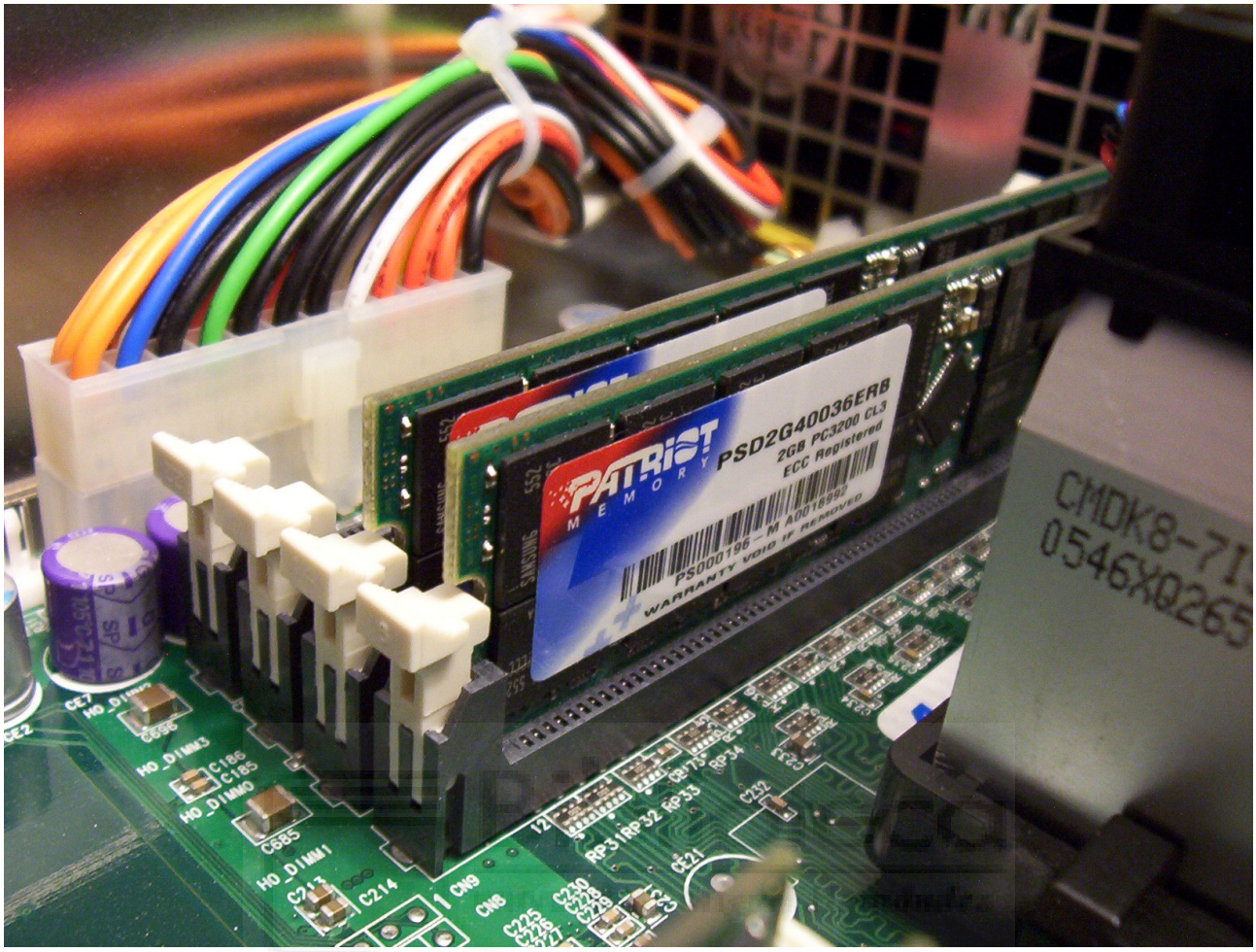

Figura 42: Memoria RAM instalada en una placa base

[111]

Para la realización de este proyecto, la RAM sí ha sido importante ya que determina el tamaño de la memoria sobre la que extraemos las experiencias para realizar el *experience replay*. Por tanto, en nuestro caso, una mayor cantidad de RAM nos permite almacenar más experiencias.

3.2.4 GPU

Las GPUs llevan usándose para videojuegos desde los años 70, siendo algo muy común a día de hoy. La gran diferencia entre una GPU y una CPU reside en como la GPU maneja los comandos: mientras que una CPU maneja unas pocas instrucciones muy complejas, **una GPU maneja muchas pero muy simples, utilizando generalmente una arquitectura paralela.**

Desde hace unos años la comunidad de aprendizaje máquina empezó a aprovechar esta arquitectura de procesamiento en paralelo, logrando a veces **velocidades de 10 veces más rápidas comparadas con las que obtendríamos en una CPU** [108].

Figura 43: MSI Nvidia GeForce RTX2070, una de las tarjetas gráficas usadas en este proyecto

[112]

Por tanto, tener una GPU rápida es importante tanto si empezamos a hacer *machine learning/deep learning* como si ya somos expertos, ya que ejecutaremos nuestros programas y agentes más rápidos, dándonos cuenta de los fallos de los mismos antes.

3.2.4.1 Uso de múltiples GPUs

Una opción (utilizada en este trabajo) es la de **utilizar varias tarjetas gráficas, para procesar paralelamente más información.** Para ello, **las librerías que utilicemos, deben ser compatibles** (tensorflow y keras contienen instrucciones para procesamiento en paralelo con varias tarjetas gráficas). Concretamente, para redes neuronales convolucionales, usando paralelización, podemos encontrar con un aumento de velocidad en torno a 1.9x/2.8x/3.5x para 2/3/4 GPUs.

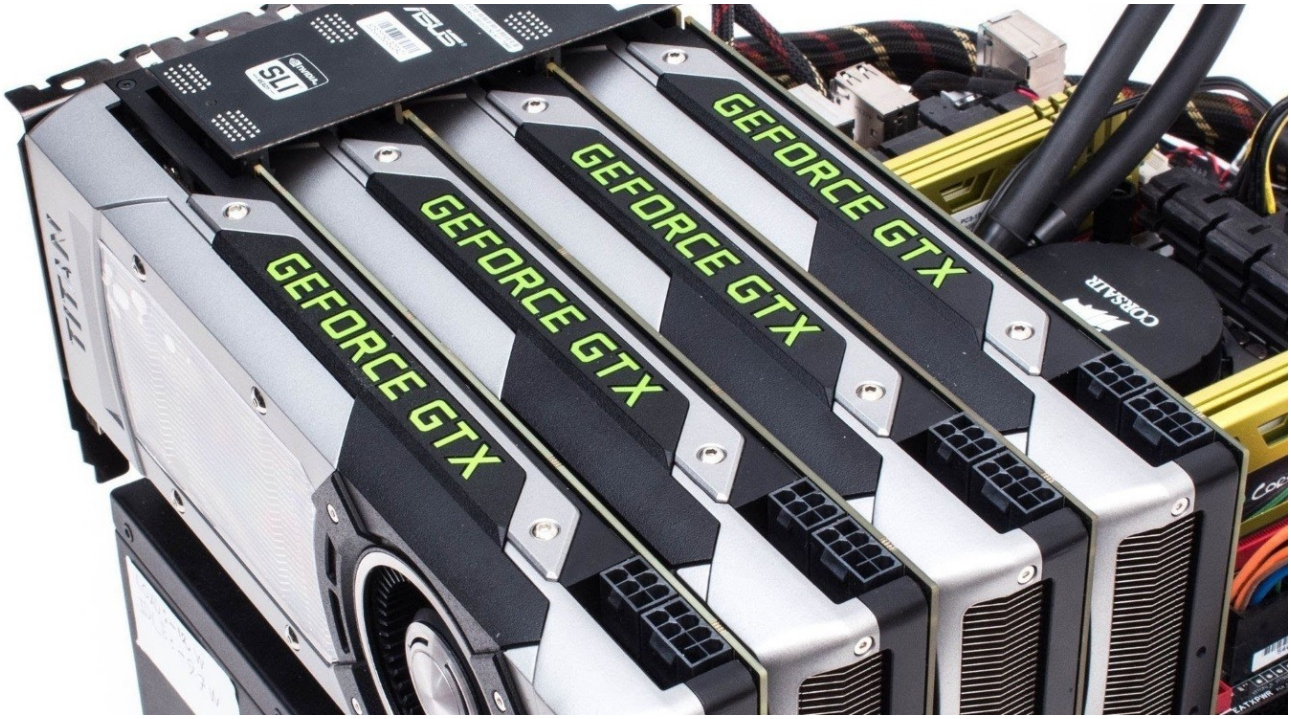

Figura 44: Configuración múltiple GPU

[113]

Otra opción al tener varias GPUs es la de que podemos ejecutar diferentes algoritmos o experimentos, uno en cada GPU, lo cual nos permite realizar diferentes pruebas en menor tiempo [114].

En cuanto al **tamaño de memoria,** podemos guiarnos por la siguiente regla, según [105]:

- **Investigación que busca resultados de estado del arte: >= 11 GB.**
- **Investigación que busca arquitecturas interesantes: >= 8 GB.**
- **Cualquier otro tipo de investigación: 8 GB.**
- **Kaggle (competiciones): 4-8 GB.**
- **Startups: 8 GB.**
- **Compañías: 8 GB para realizar prototipos, 11 GB para entrenamiento.**

3.2.4.2 ¿Qué fabricante elegir? AMD vs Nvidia

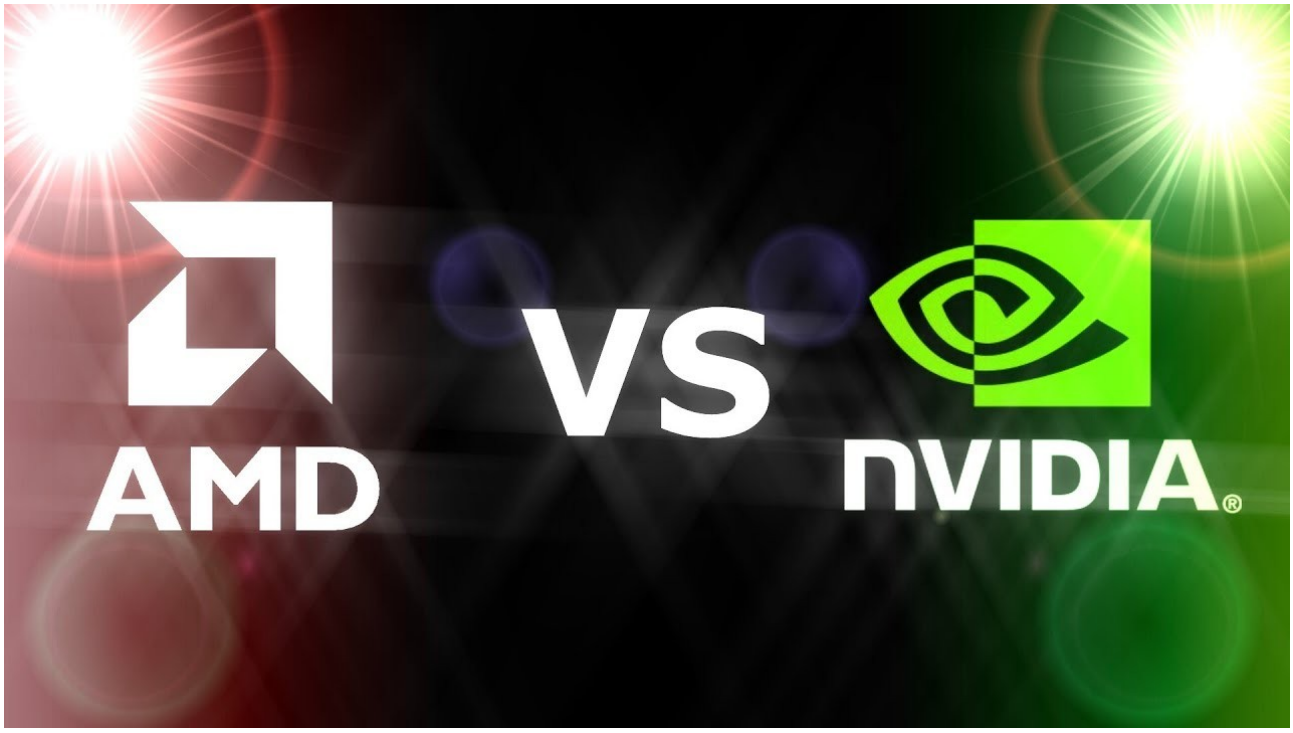

Figura 45: AMD vs Nvidia, ¿qué fabricante escoger?

[115]

Existen en la actualidad **dos grandes fabricantes de tarjetas gráficas, AMD y Nvidia.** En este apartado analizaremos qué fabricante es más conveniente elegir a la hora de realizar tareas de *machine learning*.

Nvidia se presenta como la opción más utilizada, debido a que sus **librerías CUDA,** la cual es una plataforma de computación y modelo de programación desarrollado por Nvidia para acelerar la computación en sus GPUs, están bastante estandarizadas [116].

Figura 46: Logotipo Nvidia CUDA

[117]

AMD tiene sus propias librerías para OpenCL, pero **ni su uso está tan extendido ni son tan potentes** como las CUDA de Nvidia. Además, encontramos que la comunidad de aprendizaje máquina en Nvidia es más extensa que en AMD, un punto importante a la hora de realizar consultas [114].

Por tanto, podemos concluir que **actualmente las GPUs de Nvidia son superiores a las AMD para *machine learning*.**

3.2.5 Equipo utilizado en este proyecto

Para la realización de este trabajo, se ha utilizado el siguiente equipo:

Figura 47: Foto de los componentes utilizados para este trabajo fin de máster

- **CPU:** AMD Ryzen 5 2600 3.4 GHz
- **Placa base:** Gigabyte Aorus X470 Ultra Gaming
- **RAM:** Kingston HyperX Predator RGB DDR4 2933MHz 16GB (2x8) CL15 (2 unidades, haciendo un total de 32 GB de RAM)
- **GPU:** MSI GeForce GTX 1050 Ti AERO ITX OC 4GB GDDR5 (2 unidades)
- **Posteriormente,** se cambiaron las 2 GPUs por una MSI Geforce RTX 2070 Ventus 8GB GDDR6, buscando un mayor rendimiento y velocidad.

También **se hizo uso de la plataforma Google Colaborative,** la cual cuenta con una GPU Nvidia Tesla K80.

4. ¿Qué es la inteligencia artificial?

Figura 48: Inteligencia Artificial

[118]

En este apartado del trabajo fin de máster, se intentará definir qué es la inteligencia artificial, explicando brevemente algunos de sus campos más relevantes así como sus orígenes.

La inteligencia artificial (IA abreviadamente), es, a grandes rasgos, cuando una máquina lleva a cabo un acto relacionado con la inteligencia. En el ámbito de ciencias de la computación, **una máquina considerada inteligente es un agente que percibe su entorno y lleva a cabo acciones que maximicen sus posibilidades de éxito en algún objetivo o tarea.**

Coloquialmente, **el término se aplica cuando una máquina imita las funciones cognitivas que los humanos asocian con la mente humana, tales como aprender o resolver problemas.** Otra definición, acuñada por Andreas Kaplan y Michael Haenlein, definen la **inteligencia artificial como "la capacidad de un sistema para interpretar correctamente datos externos, aprender de ellos y emplear esos conocimientos para lograr tareas y metas concretas a través de una adaptación flexible".** Curiosamente, conforme las máquinas se vuelven más capaces, la tecnología que antes se pensaba que requería de inteligencia se elimina de la definición [119].

Originariamente, **el término se desarrolló en los años 50;** nació a partir del artículo escrito en 1950 llamado *Computer Machinery and* Intelligence, en el que **Alan Turing** (matemático, científico de la computación, criptógrafo y filósofo, considerado uno de los padres de la ciencia de la computación y recursos de la informática moderna) [120], formuló la pregunta: ¿puede pensar una máquina? [119] [121].

Se considera que el término *inteligencia artificial* se definió en 1956 en una conferencia en Darthmouth, convocada por Johny McCarthy, a la cual asistieron grandes científicos como Marvin

Minsky, Allen Newell y Herbert Alexander Simon. En esta conferencia, el término *inteligencia artificial* quedó acuñado como “la ciencia e ingenio de hacer máquinas inteligentes, especialmente programas de cómputo inteligentes” [122].

Para Microsoft, la IA es cuando las máquinas o sistemas informáticos se comportan de una manera similar a la inteligencia humana [123]. La RAE define la IA como la disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico [124].

Resumiendo, **el concepto de inteligencia artificial se basa en la idea de máquinas que actúan como lo haría la mente de un humano.** En los siguientes apartados de este capítulo, definiremos los distintos tipos de inteligencia artificial que existen, profundizando en los más relevantes para el desarrollo de este trabajo.

4.1 Tipos de inteligencia artificial

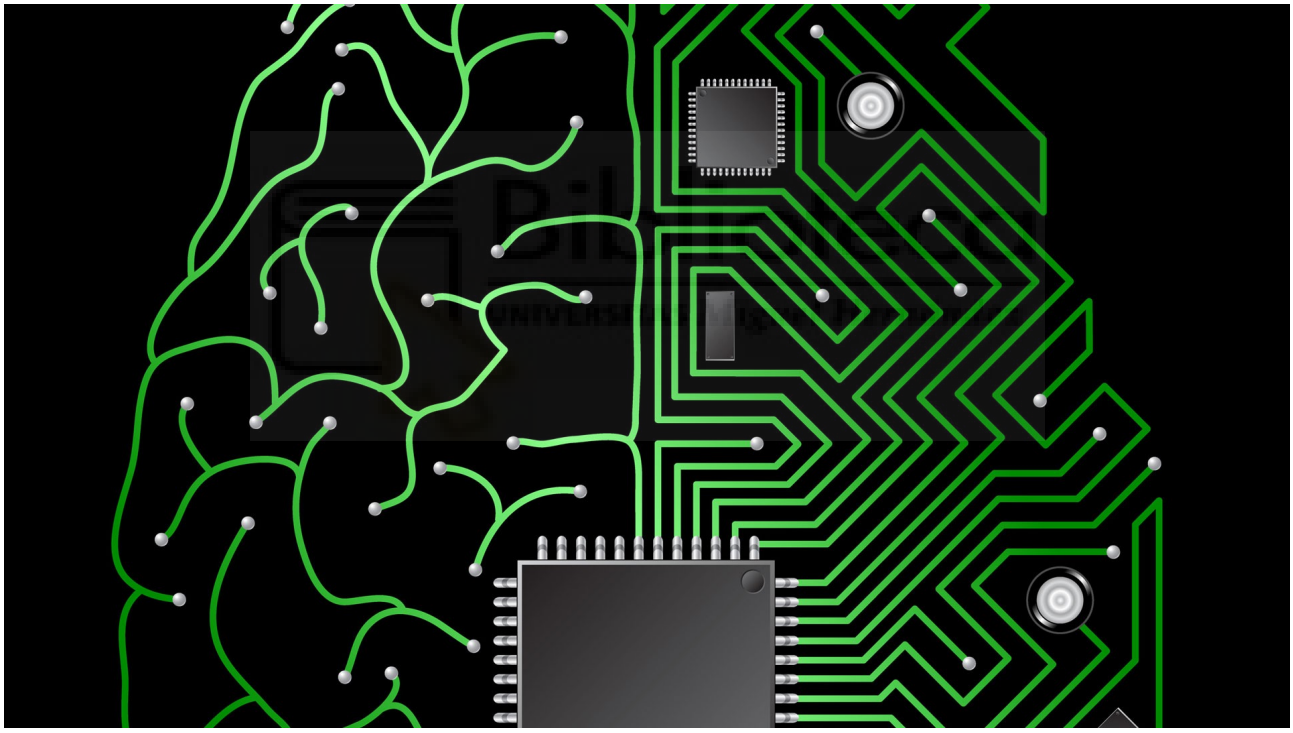

Figura 49: Cerebro-ordenador

[125]

Existen muchas clasificaciones distintas para la inteligencia artificial, ya que **no existe un consenso sobre ella.** Se ha decidido utilizar la siguiente clasificación ya que engloba todos los campos a tratar pero a la vez es bastante general.

Se clasifica en **3 grandes grupos:**

- **Inteligencia artificial débil o estrecha (del inglés *narrow artificial intelligence*).**
- **Inteligencia artificial general.**

- **Inteligencia artificial fuerte.**

La **inteligencia artificial débil** es aquella que **es capaz de resolver problemas muy bien definidos y acotados;** es la **responsable de los verdaderos avances en el campo,** y es la que de hecho **se está desarrollando actualmente.**

Algunos usos de este tipo de IA ya han sido mencionados en el primer capítulo del trabajo, tales como: reconocimiento facial, de objetos y de imágenes, reconocimiento de voz, bots de videojuegos, asistentes personales, diagnóstico médico… Las distintas aplicaciones son cada vez más extensas y afectan a todo el espectro de nuestras vidas.

Se considera **inteligencia artificial general** a **aquella que sea capaz de resolver cualquier tarea intelectual resoluble por un ser humano.** Estaríamos hablando de una **IA multitarea que podría realizar muchas acciones** con resultados satisfactorios. Se la podría considerar como un **conjunto de inteligencias artificiales débiles,** cada una enfocada en resolver un problema distinto. Se supone que sería **capaz de realizar juicios y sacar sus propias conclusiones** ante una situación incierta, en base a su aprendizaje y entrenamiento, además de comunicarse, planificar y aprender. Actualmente **no se ha logrado este tipo de inteligencia artificial.**

Por último, la **inteligencia artificial fuerte** sería aquella que poseyera **distintos estados mentales** y, además, tendría **consciencia de sí misma.** Teóricamente, **sería capaz de resolver cualquier tarea incluso superando a los humanos.** Este tipo de inteligencia artificial **plantearía grandes problemas éticos** [122] [126].

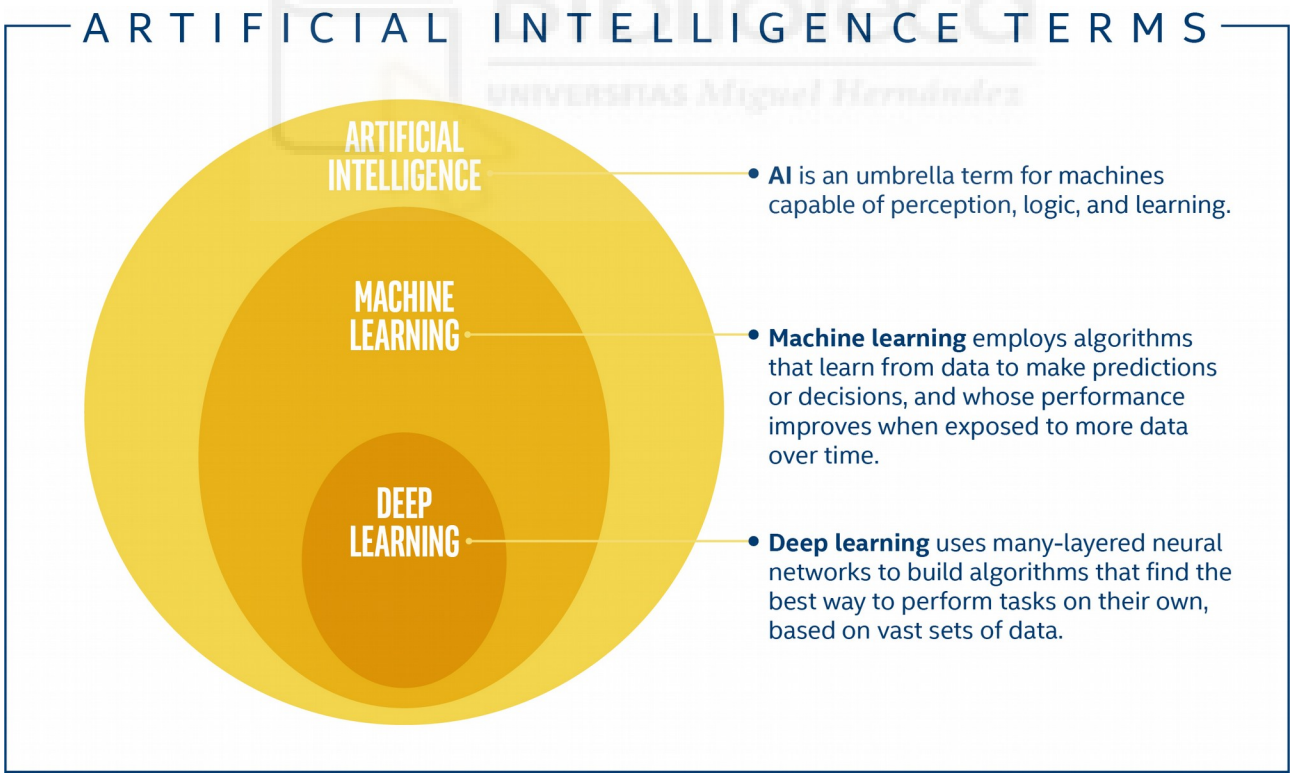

Figura 50: Clasificación interna dentro de la inteligencia artificial débil
[127]

Por tanto, **actualmente, el tipo de inteligencia artificial que se ha desarrollado es del tipo débil;** dentro de ese campo se engloba el aprendizaje máquina o ***machine learning,*** que es el campo que

en este trabajo se trata, a través del ***deep learning*** o aprendizaje profundo. Definiremos esos conceptos en los apartados siguientes del trabajo, además de los tipos de aprendizaje, haciendo hincapié en el que se ha usado para este TFM, el aprendizaje por refuerzo.

4.2 Machine learning

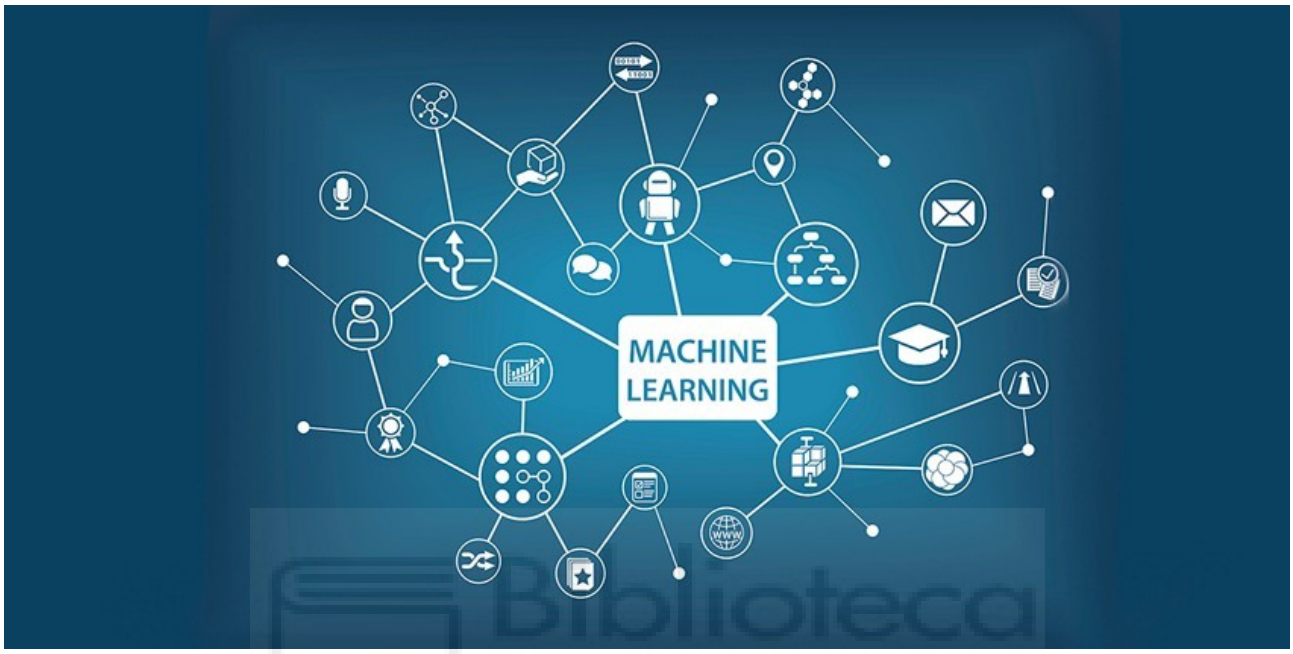

Figura 51: Machine Learning

[128]

El ***machine learning*****, también llamado aprendizaje automático o aprendizaje máquina en castellano, es una rama de la inteligencia artificial que busca desarrollar técnicas que permitan que las computadoras aprendan;** más concretamente, la creación de programas que sean capaces de generalizar comportamientos a partir de información suministrada como ejemplos. El aprendizaje automático puede ser visto como un intento de **automatizar algunas partes del método científico mediante métodos matemáticos.**

El *machine learning* usa una **gran diversidad de algoritmos** mediante los cuales se detectan patrones en los datos a analizar. Tras entrenar con miles de ejemplos, el sistema consigue realizar predicciones.

Algunos **usos cotidianos del aprendizaje máquina** son:

- **Motores de búsqueda**
- **Diagnóstico médico**
- **Reconocimiento del habla y lenguaje escrito**
- **Videojuegos**
- **Robótica**

[129]

En el siguiente apartado se expondrá una clasificación de los distintos tipos de *machine learning* más relevantes.

4.3 Tipos de machine learning

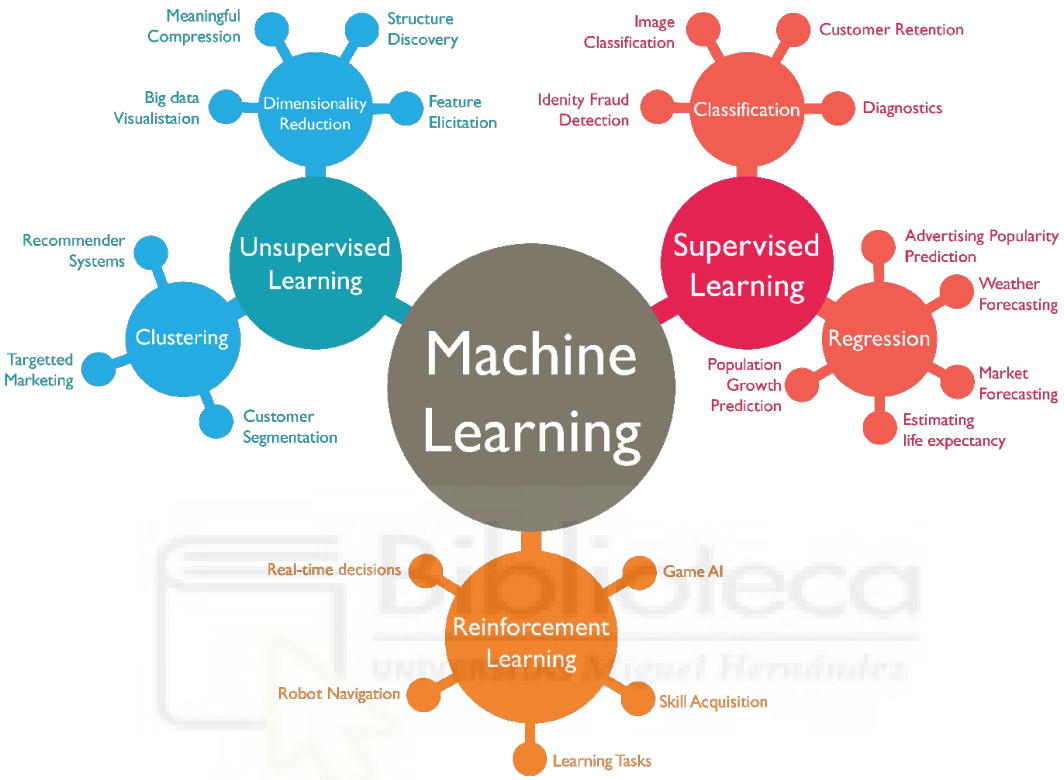

Figura 52: Clasificación y usos dentro del Machine Learning: Unsupervised Learning, Supervised Learning y Reinforcement Learning

[130]

Encontramos **tres grandes tipos de aprendizaje automático.** Si bien los tres presentan a grandes rasgos el mismo objetivo (clasificar una salida en base a unos datos de entrada, siendo estos datos el problema en sí o ejemplos con los que aprender), difieren en la manera en que lo hacen.

4.3.1 Aprendizaje supervisado

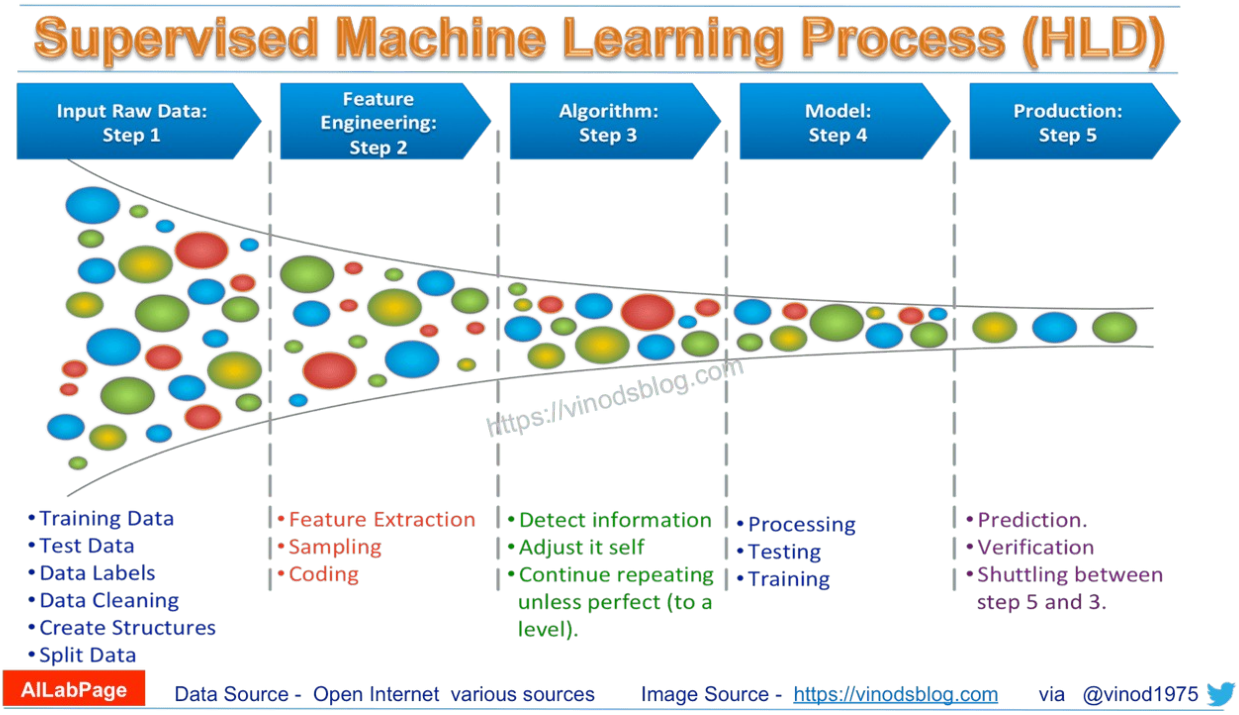

Figura 53: Esquema explicativo del funcionamiento del aprendizaje supervisado

[131]

El aprendizaje supervisado es una técnica que **deduce una función a partir de los datos de entrenamiento.** Los datos de entrenamiento consisten en **pares de objetos,** normalmente vectores; **una componente son los datos de entrada y el otro, los resultados deseados. La salida puede ser un valor numérico (problema de regresión) o una etiqueta de clase (problema de clasificación)**. Por tanto el objetivo de este tipo de aprendizaje es crear una función capaz de predecir el valor correspondiente a cualquier objeto de entrada válida después de haber visto una serie de ejemplo (datos de entrenamiento). Para conseguir esto, generaliza a partir de los datos presentados [132]. **Algunos métodos y algoritmos que se utilizan en este tipo de aprendizaje son:**

- **K vecinos más próximos (*K-nearest neighbors).***
- **Redes neuronales artificiales.**
- **Máquinas de vector soporte (*SVN, Support Vector Machines*).**
- **Clasificador Bayesiano ingenuo (*Naive Bayes classifier*).**
- **Árboles de decisión.**
- **Regresión logística.**

[133]

4.3.2 Aprendizaje no supervisado

En el aprendizaje no supervisado, l**os datos de entrada no están clasificados ni etiquetados, por lo que este tipo de algoritmos tienen que ajustar su modelo a las observaciones;** es decir, no tenemos un conocimiento a priori de los datos.

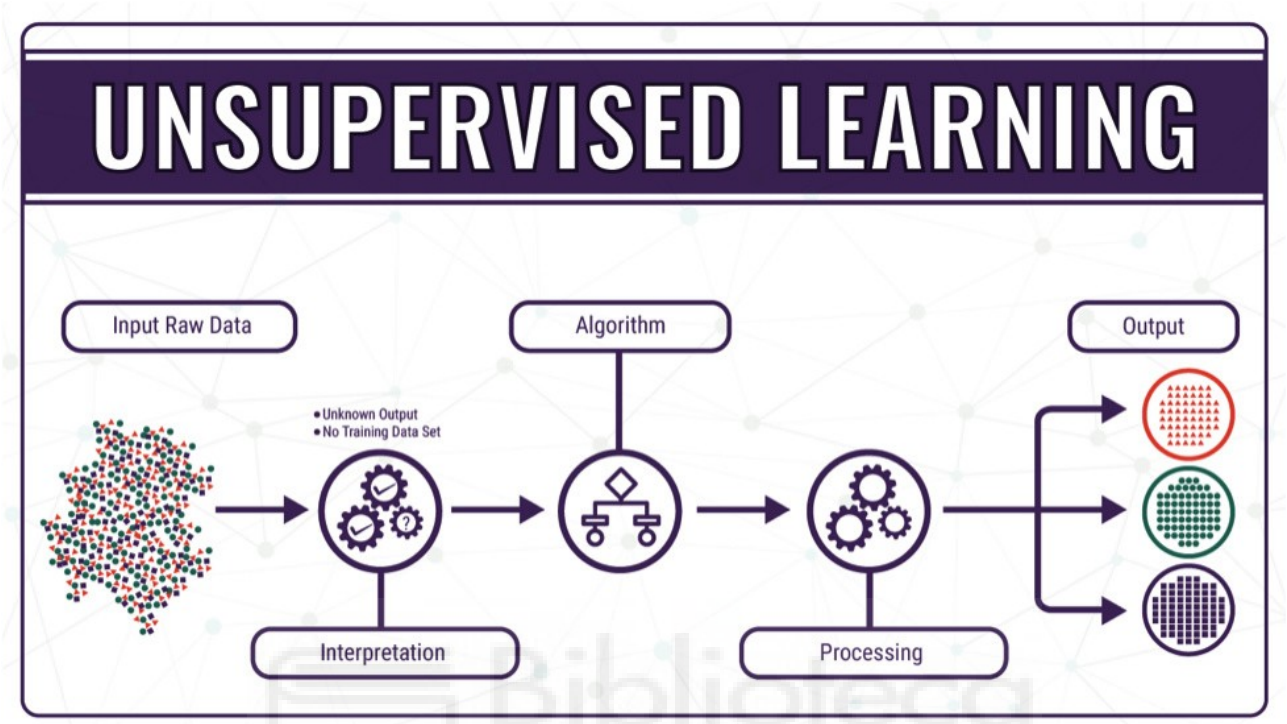

Figura 54: Esquema explicativo de aprendizaje no supervisado
[134]

Lo que hace básicamente este tipo de aprendizaje es **realizar agrupaciones entre los datos** (*clustering* en inglés), buscando características comunes entre los distintos datos.

Un uso muy común del aprendizaje no supervisado son la compresión de datos o de imágenes [135].

Los principales algoritmos del tipo no supervisado son:

- **K-medias (*K-means*).**
- **Mezcla de Gaussianas (*Gaussian mixtures*).**
- **Agrupamiento jerárquico (*Hierarchical clustering*).**
- **Mapas auto-organizados (*Self-organizing maps*).**

[133]

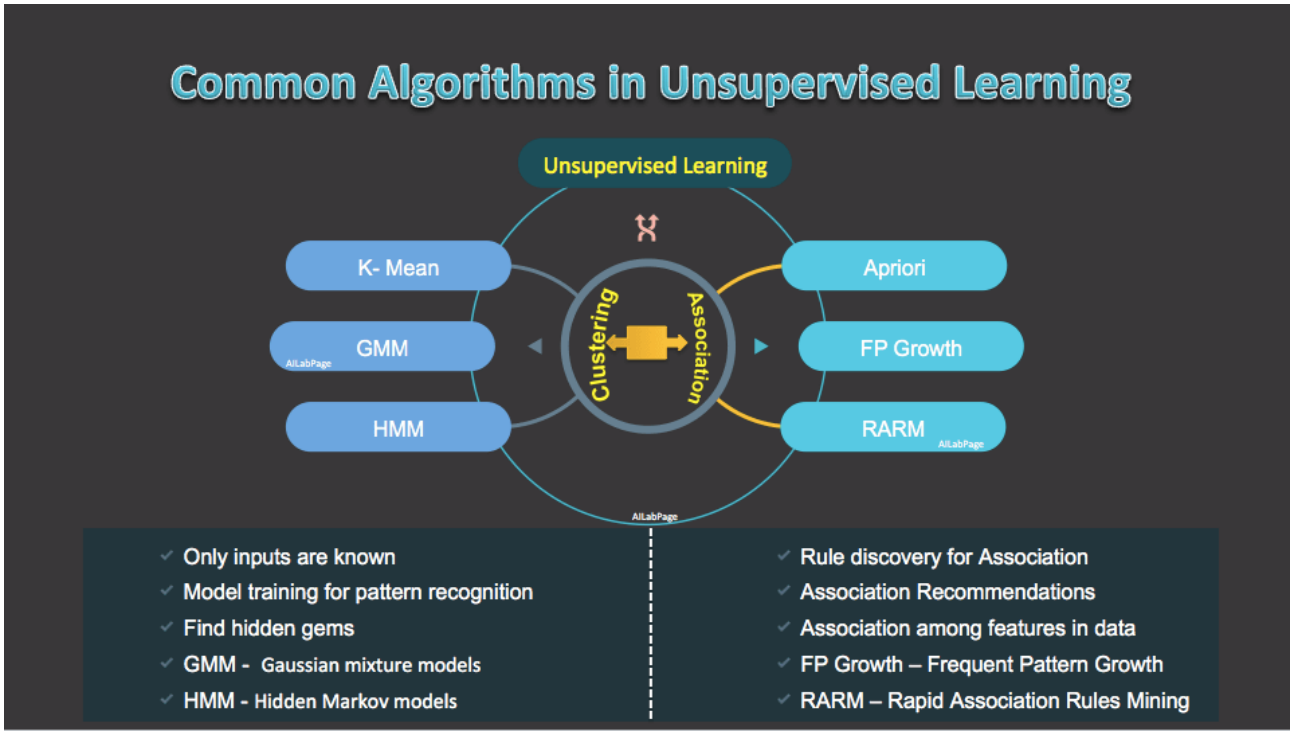

Figura 55: Algunos algoritmos comunes en aprendizaje no supervisado
[136]

4.3.3 Aprendizaje por refuerzo

El aprendizaje por refuerzo está **inspirado en el conductismo** [133] [137], el cual en psicología es la rama que **se dedica al estudio experimental objetivo y natural de la conducta** [138].

Adaptando esto al *machine learning*, nos encontramos que lo que **se intenta determinar es el comportamiento que debe presentar el agente autónomo** (así es como se suelen denominar los programas de inteligencia artificial en este ámbito) **para maximizar algún tipo de recompensa o premio acumulado mediante interacción con el medio** [139]. Este problema se estudia en otras disciplinas como la teoría del juego, teoría de control, u optimización basada en simulación, estadísticas y algoritmos genéticos. En aprendizaje automático, **el entorno se formula normalmente como un proceso de decisión de Markov** (*MDP, Markov Decision Process* en inglés) [137]. Se hablará más extensamente del proceso de decisión de Markov en el capítulo 5 de este trabajo.

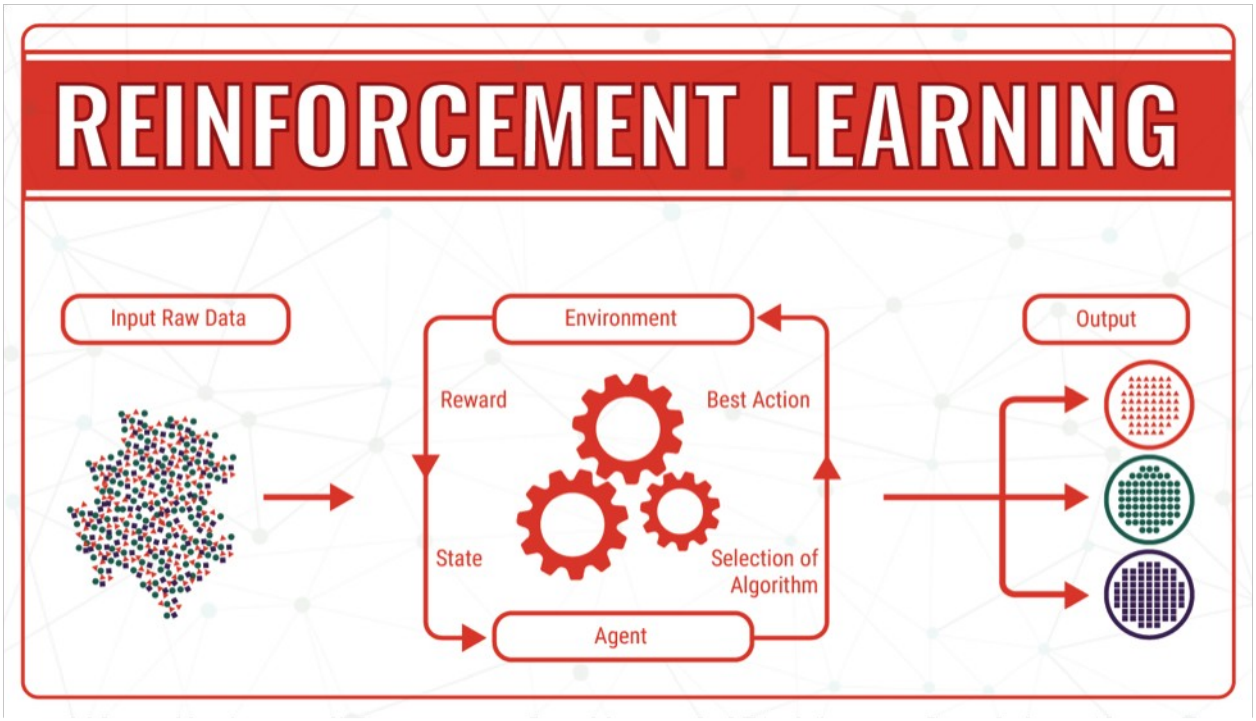

Figura 56: Esquema simplificado del funcionamiento del aprendizaje por refuerzo [140]

En el caso del aprendizaje por refuerzo, **no existen pares de entradas/salidas** como en el aprendizaje supervisado, ni **tampoco intentamos llegar a una clasificación de los datos de entrada** como en el caso del aprendizaje no supervisado; en este caso, lo que se intenta es, **dado un estado, analizarlo y extraer información del mismo;** una vez hecho esto, **se realizará una acción que el algoritmo estime oportuno, buscando siempre maximizar la recompensa final.** La dificultad de la asignación de la recompensa (*credit assignment* en inglés) en este tipo de aprendizaje se debe a que muchas veces **la recompensa no es inmediata,** sino que **el agente debe seguir un camino o realizar una serie de acciones antes de obtener una recompensa, o incluso no recibir ninguna recompensa hasta que consiga el objetivo.** Se usa además un enfoque de rendimiento en línea, buscando el equilibrio entre la exploración de lo desconocido, normalmente mediante métodos aleatorios, y la explotación de los conocimientos que hemos ido adquiriendo en base a la experiencia [137].

Por clarificar como funciona el aprendizaje por refuerzo, se expone el siguiente **ejemplo:**

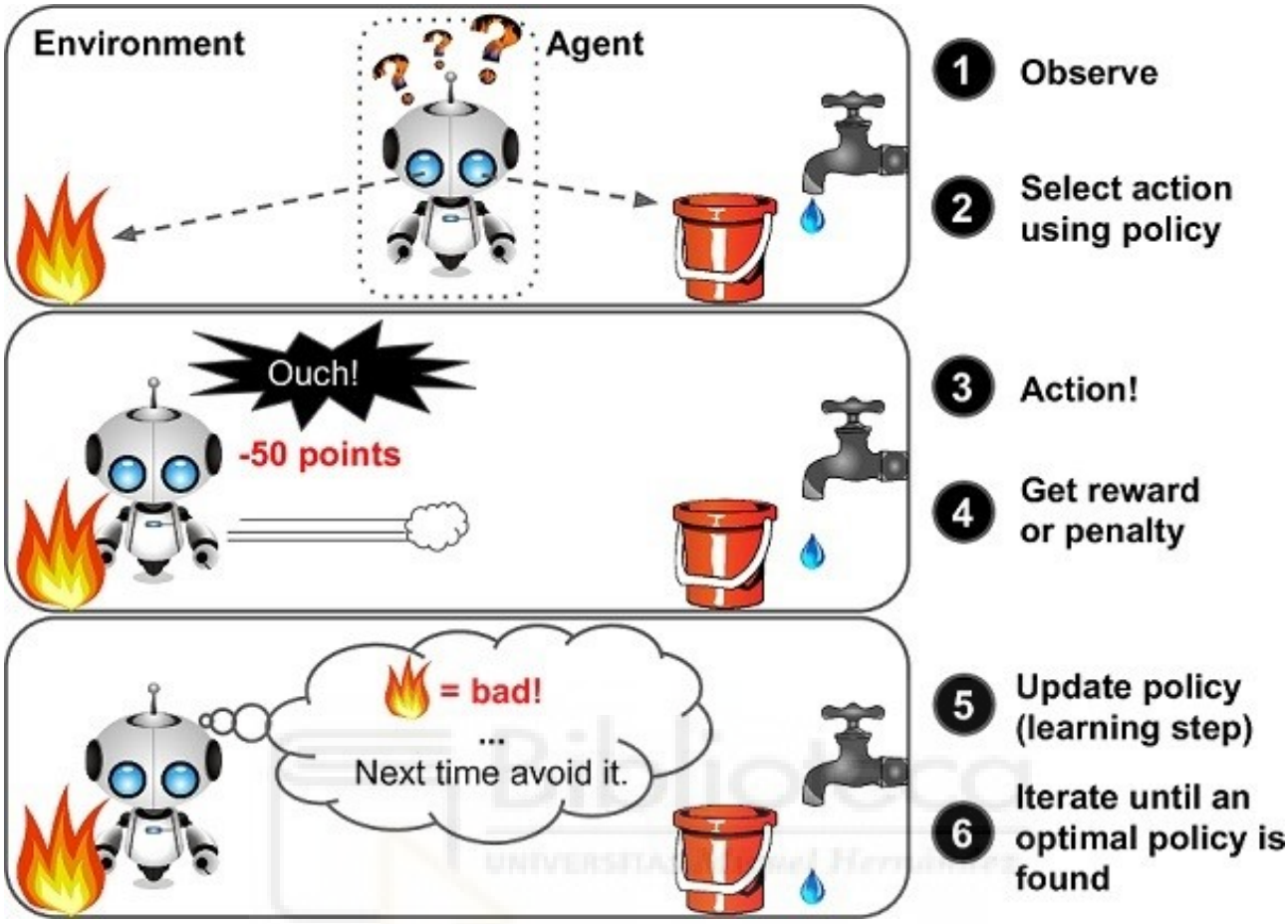

Figura 57: Ejemplo simplificado de como funciona el aprendizaje por refuerzo

[141]

Supongamos que tenemos un robot en un pasillo que forma un círculo completo, y que queremos que el robot complete 5 vueltas completas, sin chocarse, contra las paredes. Por cada paso que dé, se le recompensará con +1 puntos; cada vez que choque, con -5; y por cada vuelta completada, con +50 puntos. En un principio el robot no conoce qué acciones le darán una recompensa positiva y cuales negativas, pero sí qué acciones puede realizar (moverse hacia adelante, atrás, izquierda o derecha); lo único que sabe es que tiene que maximizar su puntuación.

El robot recibe información del entorno mediante sensores, mediante los cuales sabe la distancia que hay a las paredes y distintos obstáculos.

Mediante este tipo de algoritmos, al principio el robot realizaría acciones aleatorias (exploración), aprendiendo de los mismos cuáles le recompensan positivamente y cuáles negativamente; al cabo de un período de entrenamiento, si el algoritmo está bien implementado, el robot al final sería capaz de completar las vueltas sin chocarse o casi sin hacerlo, habiendo aprendido de su propia experiencia (explotación).

El aprendizaje por refuerzo es, probablemente, **el tipo de aprendizaje máquina que más dificultades presenta;** además **la mayoría de problemas que resuelve pueden ser resueltos mediante otros métodos de *machine learning*.**

No obstante, el aprendizaje por refuerzo es un **paradigma muy general,** y, en teoría, **un sistema de aprendizaje por refuerzo robusto podría ser eficiente en todo o en un gran número de tareas;** un sistema de aprendizaje supervisado o no supervisado será eficiente en una tarea en cambio [142]. **Combinado con *deep learning*** (este concepto será definido en el siguiente apartado de este trabajo), **conforma lo que se conoce como *Deep Reinforcement Learning* o *Deep RL.***

Los dos principales problemas del aprendizaje por refuerzo son la asignación de la recompensa a largo plazo y el balance entre exploración y explotación.

El **problema de la asignación de la recompensa a largo plazo** es que **el agente intenta maximizar** un número **(la recompensa),** el cuál es el resultado de acciones tras múltiples pasos, mezcladas con la aleatoriedad del entorno. **El problema de encontrar la acción o cadena de acciones responsables de maximizar la recompensa es lo que se conoce como *credit assignment.***

El principal inconveniente con la asignación de recompensas es debido a que **la escala a la que podemos proporcionar recompensas para acciones significativas es mucho más grande que la escala que los algoritmos actuales manejan** [143].

Existen **dos maneras de solucionar esto; una es reducir la escala a la que se dan las recompensas (*reward shaping)*;** dicho de otro modo, dar recompensas más frecuentemente. Esto puede dar lugar a un fenómeno que se conoce como ***reward hacking*;** esto es, **el algoritmo encuentra que repitiendo una acción n veces (o incluso hasta el infinito), maximiza su puntuación más que si persigue el objetivo final.** La idea detrás del ***reward shaping*** **es dar más información al algoritmo sobre lo bien o mal que está comportándose,** de manera que se le asignan puntuaciones conforme se acerca al objetivo final [142] [143] [144]. Por poner un ejemplo, en *Breakout* de Atari, se asigna una puntuación cada vez que se destruye un bloque; eso hace que el algoritmo tenga constancia de que se está acercando al objetivo que buscamos, que en este caso sería destruir todos los bloques y pasar de nivel.

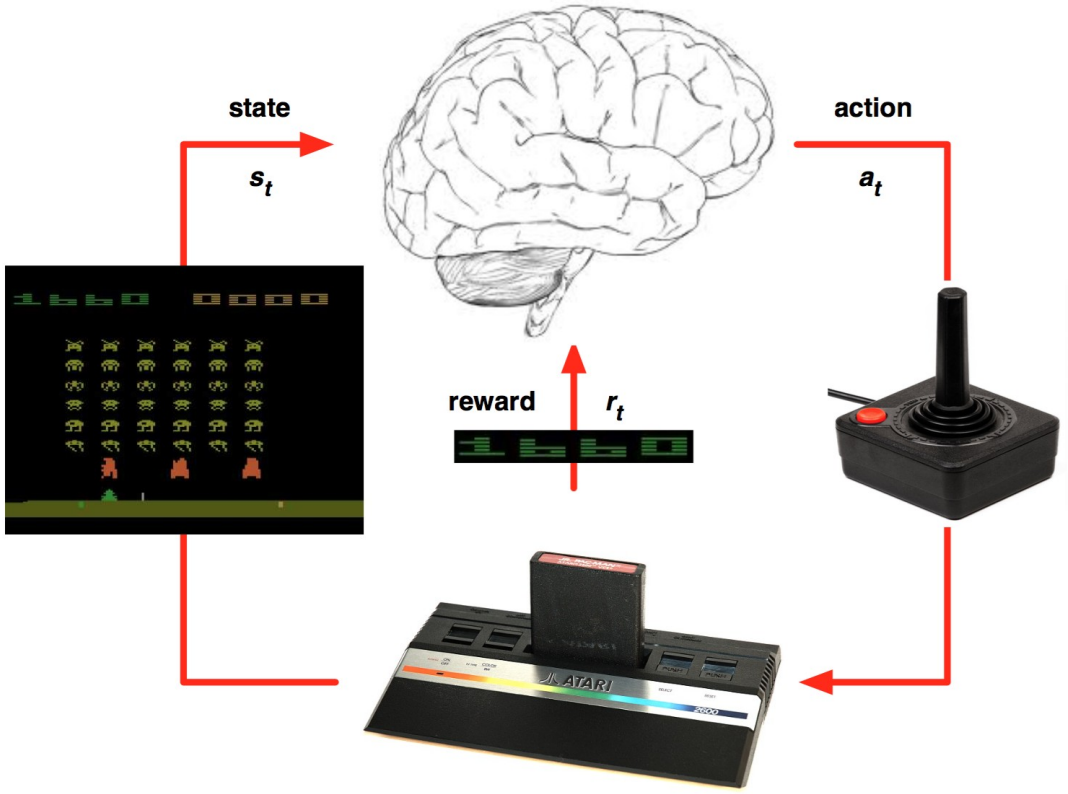

Figura 58: Funcionamiento simplificado del aprendizaje por refuerzo con Atari 2600 [145]

La **otra solución** para este problema sería lo que se conoce como ***hierarchical reinforcement learning*****;** se basa en **descomponer el problema final en una serie de objetivos y subobjetivos.** Este tipo de aprendizaje por refuerzo **está aún muy poco desarrollado** (recordemos que el aprendizaje por refuerzo, en el ámbito de la inteligencia artificial es un campo con muy pocos años de vida).

En cuanto al problema de la **exploración vs la explotación** de lo aprendido, encontramos una **gran ineficacia con las muestras, baja reproducibilidad y máximos locales.** Por lo general, **no existe una solución absoluta para estos problemas,** sino que debemos llegar a un compromiso entre ellos.

Los datos de entrenamiento se consiguen en el acto, mediante interacciones con el entorno. **Conforme exploramos y generamos nuevos estados, nuestras estimaciones del valor de las acciones van cambiando** [139] [143].

Para ilustrar esto, supongamos el siguiente ejemplo:

Imaginemos que usando una red neuronal para clasificar imágenes, vamos cambiando la etiqueta de una imagen de gato, a perro, avión, etc. La única manera de acercarse al valor verdadero de la función objetivo (en el ejemplo sería la etiqueta correcta) sería seguir explorando, es decir, probando hasta acertar.

En aprendizaje por refuerzo, no se da ninguna muestra de la función objetivo, ni siquiera en el set de entrenamiento. Y aún así, es capaz de aprender. Debido a que, como hemos dicho, los datos se van recolectando en el acto mientras se entrena, así como que la función objetivo va

cambiando, hace que **el sistema sea muy ineficiente en cuanto al número de muestras que necesita.**

Además, debido a que la exploración suele ser aleatoria, hace que el aprendizaje por refuerzo sea mucho más **sensible a los ajustes en los parámetros que en otros tipos de aprendizaje** [143]. De ahí su baja reproducibilidad.

En suma, podemos decir que el aprendizaje por refuerzo puede ser un campo dentro de la inteligencia artificial muy prometedor e interesante, si se logra perfeccionar, ya que un sistema de aprendizaje por refuerzo bien implementado es, *en teoría*, capaz de aprender cualquier cosa.

4.3.4 Deep Learning

Figura 59: Deep Learning

[146]

El *Deep Learning* o aprendizaje profundo es un **tipo de *machine learning* que utiliza redes neuronales artificiales,** las cuales imitan el funcionamiento de las neuronas del cerebro humano.

Las **ventajas** del *Deep Learning* son:

- **Hacer algoritmos de aprendizaje mucho mejores y fáciles de utilizar.**
- **Hacer avances revolucionarios en el aprendizaje máquina y la inteligencia artificial.**

[147]

Algunos usos de arquitecturas de aprendizaje profundo son: visión por computador, reconocimiento del habla, reconocimiento de señales de audio y música, proporcionando resultados excepcionales [148].

La combinación de aprendizaje profundo con aprendizaje por refuerzo da como resultado el *Deep Reinforcement Learning;* es el uso de redes neuronales en aprendizaje por refuerzo. Esto también **introduce varios problemas típicos de las redes neuronales,** tales como los ajustes de hiperparámetros, que se suman a los ya vistos anteriormente en el apartado de aprendizaje por refuerzo.

En el caso de este trabajo, **se ha usado una estructura de red neuronal convolucional** para extraer información de los distintos estados.

4.4 Redes neuronales convolucionales

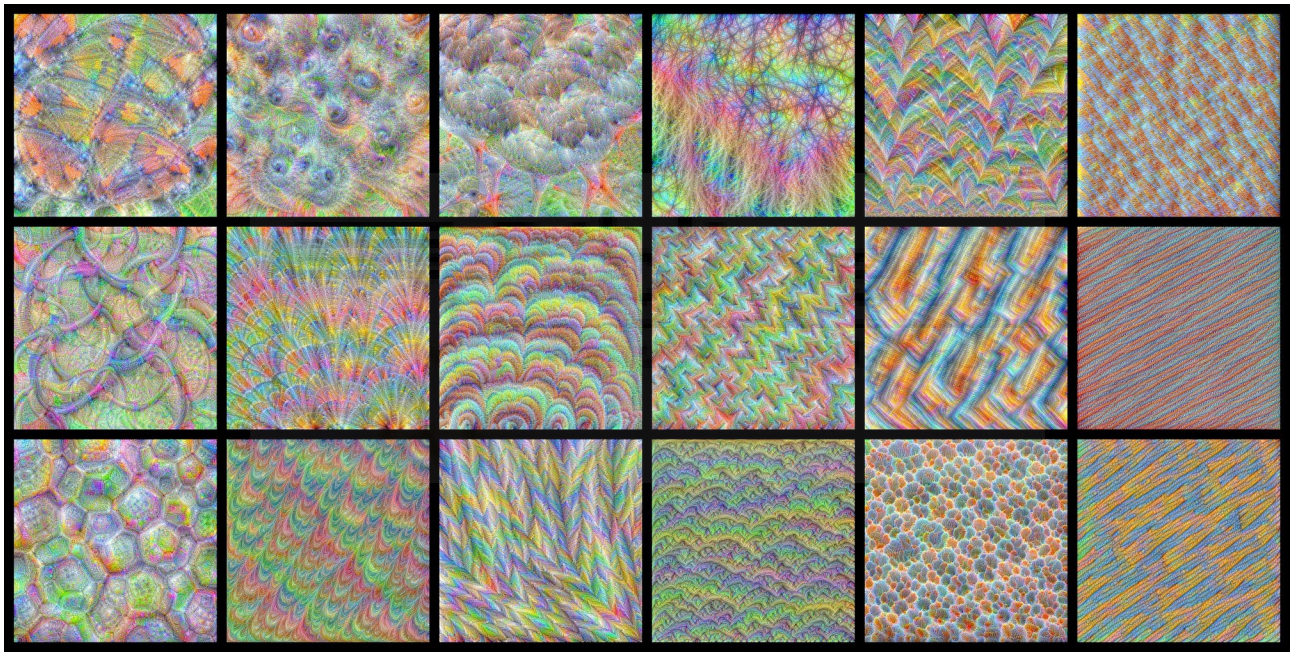

Figura 60: Características extraídas de imágenes mediante redes neuronales convolucionales. Si bien para un humano no tienen significado, a nuestro sistema de inteligencia artificial le servirán para extraer información y saber lo que está viendo.

[149]

En este apartado, se explicará como funcionan, qué son y las diferentes capas de las redes neuronales convolucionales (*CNN* en inglés, de *Convolutional Neural Network*) ya que son el tipo de redes neuronales que se han utilizado en este trabajo.

4.4.1 ¿Cómo funciona una red neuronal convolucional?

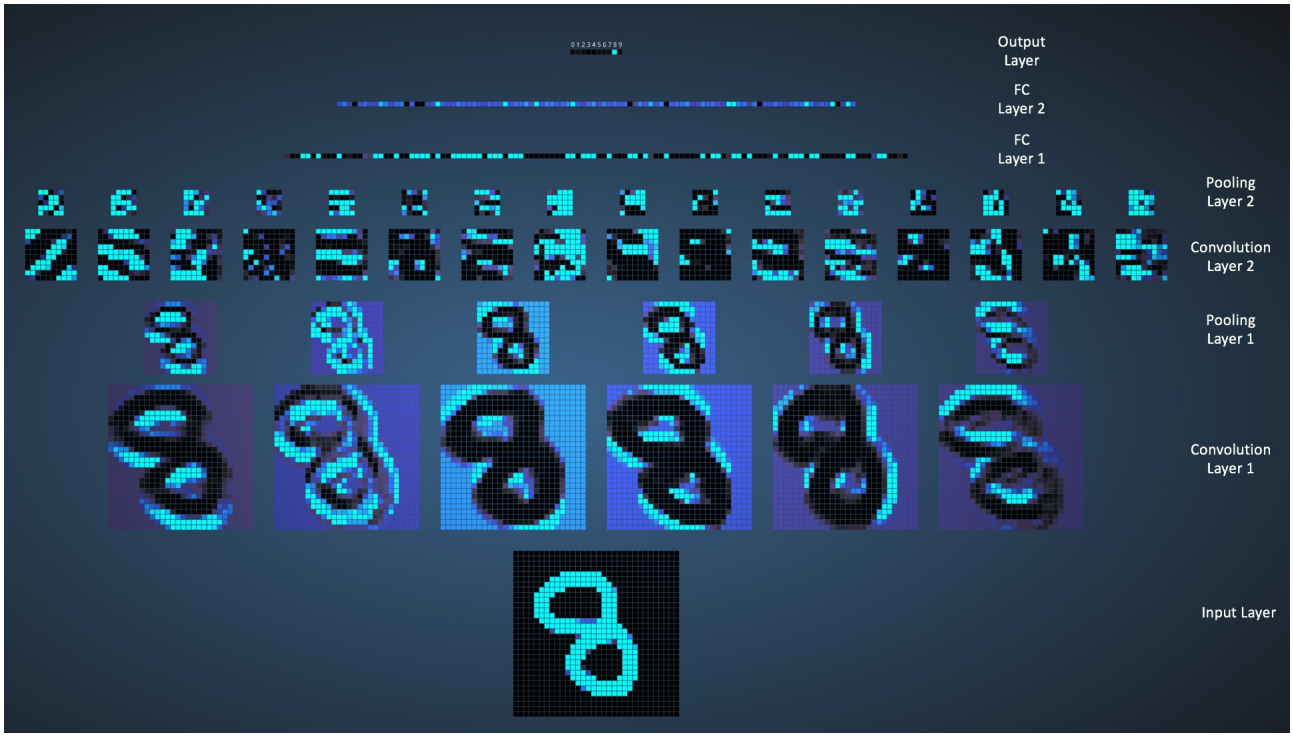

Figura 61: Imágenes de lo que sucede en el interior de cada capa de una red neuronal convolucional estándar

[150]

Una red neuronal convolucional es un tipo de red neuronal artificial donde **las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) de un cerebro biológico.** Se la considera una variación de un perceptrón multicapa. Son muy efectivas para tareas de visión artificial [151].

Consisten en **múltiples capas de filtros convolucionales,** cuya tarea es realizar la operación de convolución, la **cual consiste en una transformación lineal de dos funciones en una tercera que puede representar la magnitud, de valor real, en la que se superponen las anteriores** [152] [153].

La primera parte de una CNN trata generalmente de convoluciones y de capas de tipo *pooling* (normalmente *max-pooling)*, que actúan como extractores de características. **En el caso de aprendizaje por refuerzo, no se utilizan capas *max-pooling* ya que hacen a la red insensible a la localización de un objeto en la imagen,** cosa que en nuestro caso sería totalmente contraproducente [154]. En casos de clasificadores de imágenes sería beneficioso ya que seguiría extrayendo las características de la imagen esté donde esté el objeto en cuestión. En nuestro caso, la posición de los distintos objetos es importante.

Tras cada convolución, se coloca su activador; normalmente del tipo R*eLU* (siglas de *Rectified Linear Unit*), el cual es una función de activación que toma solamente la parte positiva de su argumento [155].

Tras las múltiples capas de convolución (convolución + activador), se encuentra una o más capas de neuronas de tipo perceptrón sencillas para realizar la clasificación final sobre las

características extraídas [156]. **La última capa,** en nuestro caso, **deberá tener el mismo tamaño que el número de acciones posibles.** Asimismo, la entrada a la CNN deberá tener el tamaño de la imagen o conjunto de imágenes que estemos analizando.

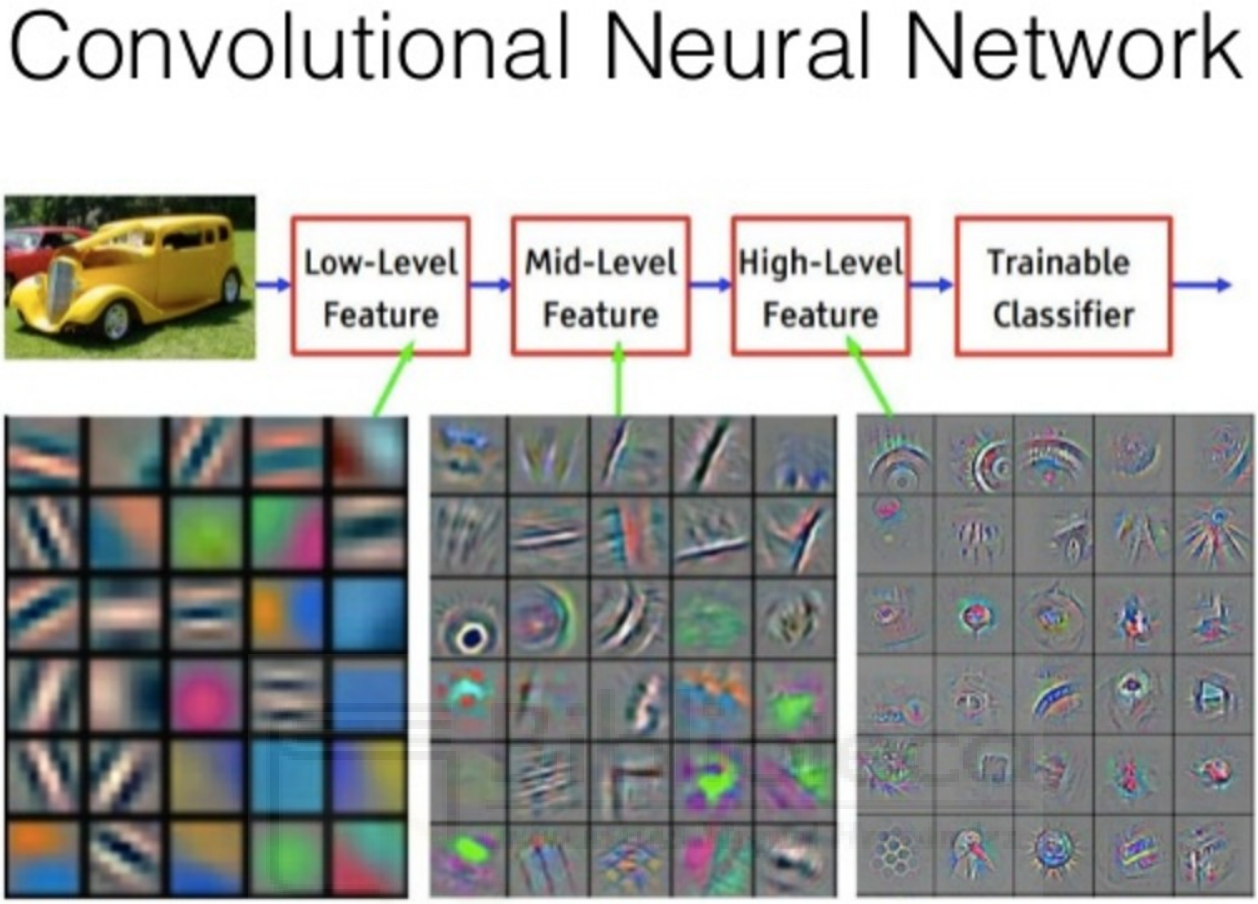

Figura 62: Extracción de características de una imagen mediante redes neuronales convolucionales

[157]

Es decir, **usamos la red neuronal convolucional como aproximador de la función,** ya que nos encontramos con gran cantidad de estados; introducimos un estado o conjuntos de estados y la red estima la mejor acción.

4.4.2 Optimizadores de redes neuronales

En este apartado **se tratarán brevemente los distintos optimizadores existentes para redes neuronales.** Se comentarán brevemente los más relevantes para este proyecto (*RMSProp* y *ADAM*).

Los algoritmos de optimización nos ayudan a minimizar o maximizar una función objetivo (también llamada función de error). Es una función matemática dependiente de los parámetros aprendibles del modelo que se usan para computar los valores objetivo. Encontramos **dos tipos de algoritmos de optimización:**

1. **Optimizadores de primer orden:** minimizan o maximizan una funcion de *loss* (o error) usando valores del gradiente con respecto a sus parámetros. El más utilizado es el descenso del gradiente. Este tipo de optimizadores nos dice si la función está decreciendo o incrementando en un punto concreto.
2. **Optimizadores de segundo orden:** usan la derivada de segundo orden (también llamada Hessiana) para maximizar o minimizar la función de error (*loss*). Son más costosos de computar, por ello no se utilizan tanto.

   [158]

4.4.3 Optimizadores más comunes

Se mencionarán los distintos optimizadores que existen, al menos en keras [159]. **La elección de uno correcto, además de su correcto ajuste, es determinante en cualquier tipo de aplicación con redes neuronales,** pero más aún en aprendizaje por refuerzo, ya que es muy sensible a los ajustes de los hiperparámetros (es el nombre por el que se conoce a los parámetros ajustables de las redes neuronales).

Los diferentes optimizadores incluidos en keras son**:**

- **SGD (*Stochastic gradient descent optimizer*, u optimizador del descenso del gradiente estocástico).**
- **RMSProp:** es el utilizado en [4]; la versión que presenta Keras es diferente, ya que en el paper de DeepMind utilizan una versión modificada, basada en [160].
- **Adagrad**
- **Adadelta**
- **Adam: son las siglas de *Adaptive Moment Estimation*.** Se puede ver como una variante de RMSProp con una versión suavizada del gradiente. Es uno de los optimizadores más modernos y sencillos de ajustar; se obtienen muy buenos resultados con este optimizador [161] [162]. Es el que se ha utilizado en este proyecto.
- **Adamax**
- **Nadam**

Como se puede apreciar, existen una gran cantidad de optimizadores, cada uno con sus parámetros a ajustar; aunque lo ideal sería poder probar cada uno de ellos, consumiría mucho tiempo.

Si bien **cada optimizador tiene sus propios hiperparámetros, todos comparten uno en común: el ratio de aprendizaje.** Pero, ¿qué ratio elegir? Un ratio muy alto podría hacer que no aprendiéramos lo suficientemente bien, y uno muy bajo que tardásemos demasiado o no fuera suficiente para extraer la información suficiente que nos permitiera aprender. Además, para cada optimizador, los ratios que funcionan son distintos, tal y como se muestra en [163].

Nótese que la estructura de las redes neuronales utilizadas en el proyecto es la misma que las utilizadas en [3] y en [4].

5. Algoritmo utilizado en este proyecto

En este capítulo **se tratará el algoritmo utilizado en el proyecto (Q-Learning)**, **sus bases matemáticas** (Proceso de decisión de Markov, ecuación de Bellman) y, finalmente, su **implementación con redes neuronales (Deep-Q-Learning)**.

5.1 Proceso de Decisión de Markov (MDP)

Un proceso de decisión de Markov (**MDP** por sus siglas en inglés, *Markov decision process*) **es un proceso de control de tiempo discreto estocástico**; de tiempo **discreto porque se trata de un proceso cuyos valores de las variables ocurren** (o se toman muestreados de un proceso continuo) **en instantes separados**, es decir, que pueden ser vistos como variables discretas. Por ejemplo, un proceso de tiempo discreto sería un reloj que marca las 3:40 durante un minuto, y al siguiente marca las 3:41, es decir, nuestra variable (en este ejemplo el tiempo) salta de un valor a otro cuando el tiempo se mueve de un período a otro. Sería de tiempo continuo si esta variable fuera cambiando progresivamente durante el tiempo [164]. **Se considera estocástico porque es un proceso determinado aleatoriamente**, es decir, no sigue ningún patrón prefijado previamente [165].

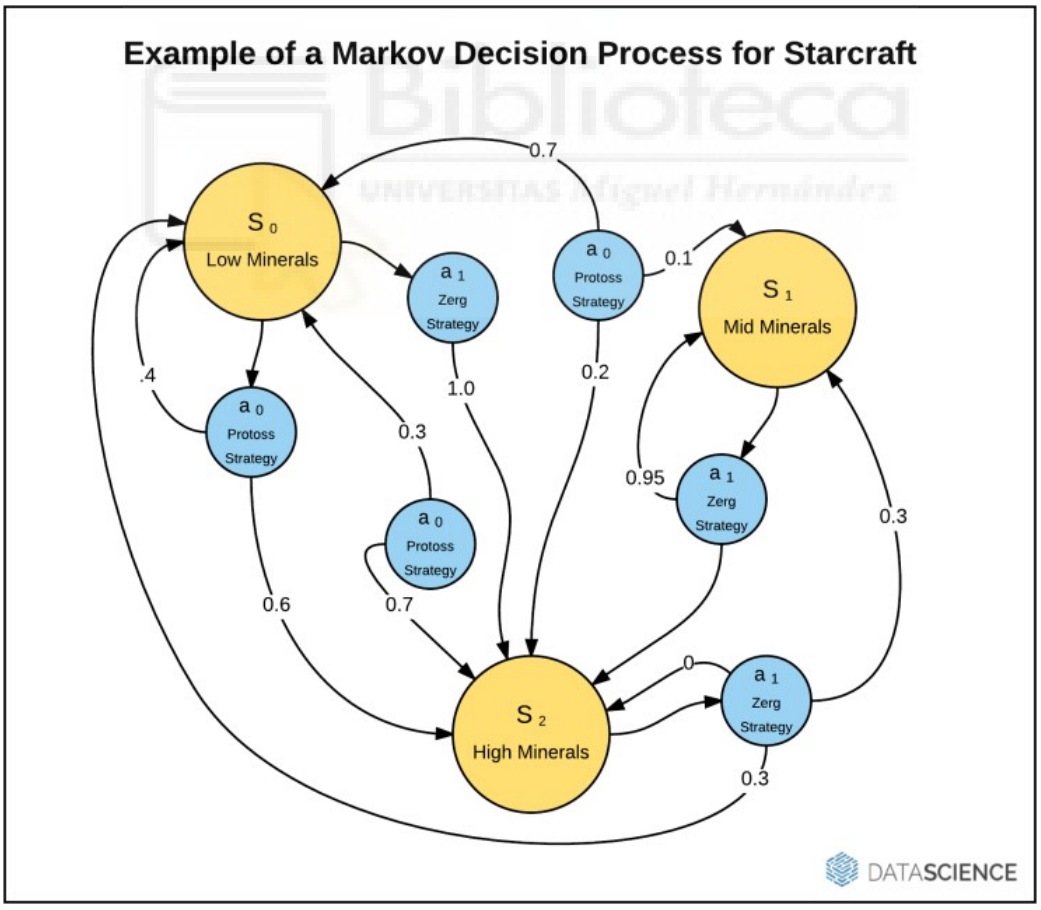

Figura 63: Ejemplo de MDP en el videojuego Starcraft
[166]

Si nos fijamos en la imagen, tenemos tres estados posibles: s0 (minerales bajos), s1 (minerales medios) y s2 (minerales altos). Existen dos posibles acciones que podemos tomar: la a0 (Protoss Strategy) y la a1 (Zerg Strategy). Dependiendo del estado en el que nos encontremos, una acción u otra nos llevará, con una probabilidad, a uno u otro estado.

Nos provee de un marco de referencia matemático para modelar la toma de decisiones en situaciones donde los resultados son parcialmente aleatorios y parcialmente bajo el control del tomador de decisiones. El MDP es útil para estudiar problemas de optimización mediante programación dinámica y aprendizaje por refuerzo. Su nombre se debe al matemático ruso Andrey Markov.

En cada instante de tiempo, el proceso se encuentra en un estado, y el tomador de decisiones (en nuestro caso el agente de inteligencia artificial) **debe elegir cualquier acción que esté disponible en ese estado. El proceso responde al siguiente estado de tiempo moviéndolo a un nuevo estado aleatorio, y dándole al tomador de decisiones su recompensa correspondiente.**

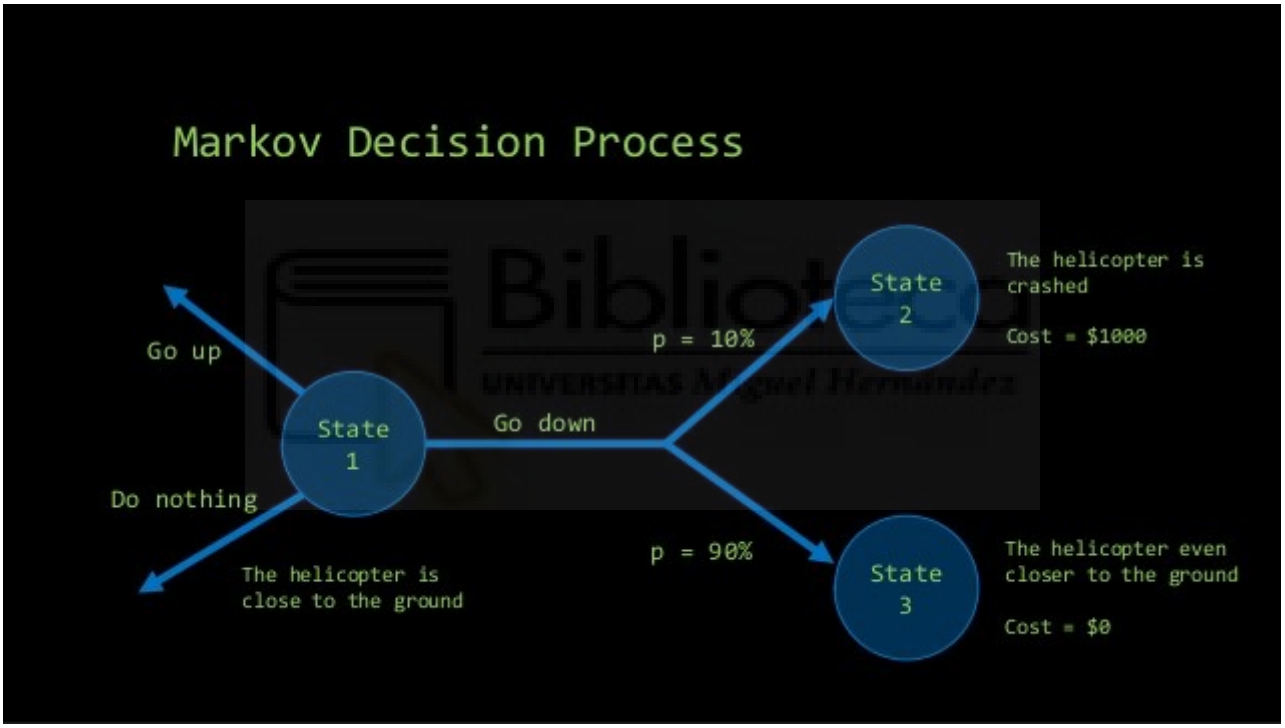

Figura 64: Ejemplo de proceso de decisión de Markov con un helicóptero
[167]

En el ejemplo presentado en la imagen, supongamos que tenemos un helicóptero en el estado 1 (cerca del suelo); podemos tomar 3 acciones distintas: ir arriba, no hacer nada o ir hacia abajo.

Supongamos que tomamos la decisión de ir hacia abajo; nos encontramos con dos posibles estados: el estado 2, en el que el helicóptero colisiona contra el suelo; la probabilidad de que suceda esto en este ejemplo es de un 10%. El otro posible estado es que nos acerquemos más al suelo (estado 3), con una probabilidad del 90%.

La probabilidad de que el proceso se mueva a este nuevo estado está influenciada por la acción elegida. Por tanto, el siguiente estado depende del estado actual y la acción del tomador de decisiones. Pero **dados el estado y la acción, es condicionalmente independiente de todos los**

estados y acciones previas: las transiciones de estado de un MDP satisfacen la propiedad de Markov [168], **la cual dice que un proceso estocástico cumple con la propiedad de Markov si la distribución de probabilidad condicional de los futuros estados del proceso dependen solamente del presente estado, no de la secuencia de eventos que lo preceden** [169].

Podemos definir el proceso de decisión de Markov como una tupla formada por 4 componentes (*S, A, P_a, R_a)* y el factor de descuento γ, los cuales quedan definidos como:

- ***S* es un conjunto finito de estados**
- ***A* es un conjunto finito de acciones**
- $P_{ss'}^{a} = Pr(s_{t+1} | s_{t+1}, a_t = a)$ (1)

  es la **probabilidad de que la acción *a* en el estado *s* en el instante *t* nos lleve al estado *s'* en el instante *t* + 1.**
- $R_{ss'}^{a} = \mathbb{E}[r_{t+1} | s_t = s, s_{t+1} = s', a_t = a]$ (2)

  que **nos indica la recompensa esperada que recibiremos cuando estando en un estado *s*, realizamos la acción *a*, y nos movemos al estado *s'*. Es la recompensa inmediata (o la recompensa inmediata esperada) recibida tras transicionar del estado *s* a *s'*, debido a una acción *a*.**
- **El factor de descuento γ: Es la preferencia por las recompensas actuales comparadas con las recompensas futuras**, es decir, con γ ajustamos la importancia que le da nuestro algoritmo a conseguir una recompensa en este instante o hacer que el algoritmo valore más el conseguir una recompensa a largo plazo. No se encuentra en la tupla, pero se define aquí ya que aparecerá posteriormente [168] [170].

5.1.1 Terminología

En este apartado se aclararán algunos términos que se han utilizado al describir el proceso de decisión de Markov y que serán importantes para apartados futuros, así como para la comprensión del algoritmo.

- **Estados finales/terminales:** son estados que no tienen acciones disponibles.
- **Episodio:** Un episodio es un conjunto completo desde uno de los estados iniciales a un estado final.

  $s_0, a_0, r_0, s_1, a_1, r_1, .., s_n$
- **Recompensa acumulada:** es la suma de las recompensas acumuladas en un episodio:

  $$R = \sum_{t=0}^{\infty} \gamma^t r_{t+1} \quad (3)$$

  El objetivo es seleccionar una estrategia que maximice la recompensa acumulada, normalmente sobre un horizonte potencialmente infinito [168].

- Política o **estrategia** (del inglés *policy)*: es la sucesión de decisiones que toma el agente para elegir una acción para cada estado. **Se denota por π.**
- Política o **estrategia óptima:** es la estrategia teórica que **maximiza la expectación de una recompensa acumulativa.** El aprendizaje por refuerzo intenta entrenar el agente para lograr esta estrategia [170].

5.1.2 Modelo libre

La distribución condicional (P_a) y la función de recompensa (R_a) constituyen el modelo del entorno. **Algunos algoritmos de aprendizaje por refuerzo pueden trabajar sin conocer el modelo;** no obstante, para aprender la mejor estrategia, **tienen que aprender o estimar el modelo durante la fase de entrenamiento.** Esto se conoce como aprendizaje por refuerzo de modelo libre (del inglés *model-free reinforcement learning)*.

Poniendo un ejemplo, en *Pong* (uno de los entornos de Atari 2600 probados), nuestro objetivo es golpear la pelota con nuestra barra y hacer pasar la pelota tras la barra del oponente. Si previamente introducimos este conocimiento, así como el sistema de puntuaciones, y el estado que seguirá tras realizar una determinada acción, estaremos utilizando lo que se conoce como *model-based*, es decir, nuestro agente tiene conocimiento *a priori* del entorno.

En nuestro caso, es modelo libre porque nuestro agente ni siquiera conoce como funcionan las reglas de puntuación en un principio (es decir, no sabe qué acciones en qué estados determinados le premiarán o penalizarán; por ejemplo no conoce que si la bola va hacia su barra y la aparta, el contrario se anotará un punto). Nuestro algoritmo deberá aprender todo esto en base a la experiencia.

Este tipo de algoritmos son muy importantes debido a que una gran cantidad de problemas de la vida real son de este tipo. Es decir, un algoritmo de modelo libre no conoce de antemano ni los estados ni las recompensas por las distintas acciones frente a un estado en un tiempo dado; tiene que aprenderlo sobre la marcha [170].

5.2 Ecuaciones de Bellman

En este apartado se explicarán las ecuaciones de Bellman, las cuales son fundamentales para entender como funcionan los algoritmos de aprendizaje por refuerzo. Estas ecuaciones fueron propuestas por Richard Bellman, un matemático norteamericano; gracias a sus ecuaciones, podemos resolver los MDPs.

Con este tipo de algoritmos, **se espera que el agente aprenda a maximizar la futura recompensa acumulativa, es decir, a hacer más alta la puntuación.** Como se ha establecido antes, a la futura recompensa acumulativa se la llamará retorno o recompensa, y se denotará con una *R.* Utilizaremos el subíndice *t* para saber la recompensa en determinado instante de tiempo. En notación matemática quedaría definido como:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + ... = \sum_{k=0}^{\infty} r_{t+k+1} \quad (4)$$

Teóricamente, con esta serie acabaríamos con una recompensa infinita, lo cual para nuestro problema no tendría mucho sentido; esta ecuación solo toma sentido si se espera que la serie de recompensas tenga un final. Las tareas que siempre tienen un final se conocen como episodicas. En nuestro caso, un episodio consistiría en una partida completa del juego en cuestión; en Pong, uno de los entornos de la Atari 2600 probados, una partida termina cuando uno de los dos jugadores llega a 21 puntos; en *Breakout*, otro de los juegos utilizados para este trabajo fin de máster, cuando al jugador se le acaban las vidas, al igual que en *Space Invaders*, otro de los entornos disponibles en OpenAI-Gym, por mencionar unos ejemplos.

Más que utilizar la futura recompensa acumulativa como retorno o recompensa, **se utilizará la futura recompensa descontada acumulativa, que se define como:**

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (5)$$

donde $0 < \gamma < 1$. Existen dos motivos por los que se define de esta manera a la recompensa, y son los siguientes:

1. **El retorno o recompensa queda bien definido para una serie infinita.**
2. **Podemos ajustar la importancia de las recompensas más cercanas frente a las recompensas más lejanas; cuanto más pequeño sea γ, más importancia daremos a las recompensas más inmediatas.** Al contrario, **si hacemos γ muy cercano a 1, daremos más importancia a las recompensas futuras. Si γ es 0, haremos que a nuestro algoritmo solamente le importe la recompensa más inmediata**, es decir, tomará la mejor acción para ese instante sin tener en cuenta las posibles consecuencias de esa acción en un futuro.

Para aprender la estrategia óptima, hacemos uso de funciones de valor. Existen dos tipos de funciones de valor que se utilizan en aprendizaje por refuerzo: la función de valor estado, que se denota como *V(s)*, y la de valor de la acción, que se denota como *Q(s,a)*.

La diferencia entre la función valor estado y la de valor de la acción, radica en que **la función de valor estado asigna un valor a los estados, y la función valor de la acción, a la acción que nos lleva a ese estado.** Imaginemos por ejemplo, un robot que debe llegar a una meta. Si usamos la función de valor estado, el algoritmo asignará valores a los distintos estados por los que se encuentre nuestro robot; si utilizamos la función de valor de la acción, asignará unos valores a las posibles acciones que podamos realizar en los distintos estados por los que vaya pasando nuestro robot.

La función de valor estado describe el valor de un estado cuando se sigue una determinada estrategia, es decir, es lo bueno o malo que es un estado en función de la estrategia que estemos usando. Siendo el retorno o recompensa esperada cuando se empieza desde un estado *s* y se actúa de acuerdo a nuestra estrategia π:

$$V^{\pi}(s)=\mathbb{E}_{\pi}[R_t|s_t=s] \quad (6)$$

Para el mismo entorno, **la función de valor cambia dependiendo de la estrategia. Esto es debido a que el valor de los estados cambia dependiendo de qué acciones se toman, ya que que depende de las acciones tomadas en un estado en particular, afecta a la recompensa que se espera obtener.**

Poniendo un ejemplo, imaginemos un robot en una cuadrícula que debe llegar a la casilla final. Cuando realice un movimiento, estará en una u otra casilla distinta, es decir, habrá cambiado de estado; si evaluamos el valor de ese estado (casilla) como la distancia a la que se encuentra de la meta, podrá estar más cerca (mayor valor) o más lejos (menor valor). Por tanto, la sucesión de estados, dependerá de las acciones realizadas, que nos acercarán más o menos a nuestro objetivo. En este ejemplo estaríamos hablando de la función valor estado.

Si estudiamos el mismo caso con la función valor de la acción, evaluaríamos no el estado en el que nos encontramos, sino la acción que tomamos, es decir: ¿nos acerca más esta acción a la meta, o nos aleja de ella?. En este caso, la misma acción puede tener valores distintos para distintos estados.

Siguiendo con el ejemplo, si imaginamos que la salida no está en una esquina de la cuadrícula, y nos encontramos a dos casillas por debajo y dos casillas a la izquierda de la meta. Si avanzamos a la derecha, nos encontraríamos más cerca de la meta (a dos casillas por debajo, pero a una a la izquierda). Si nuevamente realizamos la acción, nos acercaría aún más; pero si nos movemos una vez más hacia la derecha, nos alejaría de la meta; por tanto, en ese estado (casilla), la misma acción que antes tendría un valor alto (ya que nos acerca a nuestro objetivo), en este caso tendría un valor bajo, ya que nos aleja del mismo.

La función del valor de la acción indica el valor de tomar una acción en un estado determinado cuando se sigue una determinada estrategia. Siendo el retorno esperado dado el estado y la acción bajo π:

$$Q^{\pi}(s,a)=\mathbb{E}_{\pi}[R_t|s_t=s,a_t=a] \quad (7)$$

$$R_{ss'}^{a}=\mathbb{E}[r_{t+1}|s_t=s,s_{t+1}=s',a_t=a] \quad (8)$$

que nos indica la recompensa esperada que recibiremos cuando estando en un estado *s*, realizamos la acción *a*, y nos movemos al estado *s'*.

Derivamos las ecuaciones de Bellman, empezando por la ecuación de función de valor estado. Usando la definición de recompensa, reescribimos la ecuación (6):

$$V^{\pi}(s)=\mathbb{E}_{\pi}[r_{t+1}+\gamma r_{t+2}+\gamma^2 r_{t+3}+...|s_t=s]=\mathbb{E}_{\pi}[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t=s] \quad (9)$$

Sacando la primera recompensa de la suma, reescribimos como:

$$V^{\pi}(s)=\mathbb{E}_{\pi}[r_{t+1}+\gamma\sum_{k=0}^{\infty}\gamma^{k}r_{t+k+2}|s_t=s] \tag{10}$$

La expectación aquí describe la recompensa esperada si continuamos desde un estado *s* siguiendo la estrategia π. La expectación puede escribirse como la suma de todas las posibles acciones y todos los posibles estados devueltos por estas acciones.

$$\mathbb{E}_{\pi}[r_{t+1}|s_t=s]=\sum_{a}\pi(s,a)\sum_{s'}P^{a}_{ss'}R^{a}_{ss'} \tag{11}$$

$$\mathbb{E}_{\pi}[\sum_{k=0}^{\infty}\gamma^{k}r_{t+k+1}|s_t=s]=\sum_{a}\pi(s,a)\sum_{s'}P^{a}_{ss'}\gamma\mathbb{E}_{\pi}[\sum_{k=0}^{\infty}\gamma^{k}r_{t+k+2}|s_{t+1}=s'] \tag{12}$$

Distribuyendo la expectación entre estas dos partes, podemos manipular la ecuación de la siguiente forma:

$$V^{\pi}(s)=\sum_{a}\pi(s,a)\sum_{s'}P^{a}_{ss'}[R^{a}_{ss'}+\gamma\mathbb{E}_{\pi}[\sum_{k=0}^{\infty}\gamma^{k}r_{t+k+2}|s_{t+1}=s']] \tag{13}$$

La ecuación (6) está de la misma forma que el final de esta última ecuación. Podemos sustituirla en ella, quedando:

$$V^{\pi}(s)=\sum_{a}\pi(s,a)\sum_{s'}P^{a}_{ss'}[R^{a}_{ss'}+\gamma V^{\pi}(s')] \tag{14}$$

La ecuación de Bellman para la función del valor de la acción (también se le conoce como función de acción-valor o valor-Q; *Q-value* en inglés) puede derivarse de una forma similar, partiendo de la ecuación (7):

$$Q^{\pi}(s,a)=\mathbb{E}_{\pi}[r_{t+1}+\gamma r_{t+2}+\gamma^{2}r_{t+3}+...|s_t=s,a_t=a]=\mathbb{E}_{\pi}[\sum_{k=0}^{\infty}\gamma^{k}r_{t+k+1}|s_t=s,a_t=a] \tag{15}$$

$$Q^{\pi}(s,a)=\mathbb{E}_{\pi}[r_{t+1}+\gamma\sum_{k=0}^{\infty}\gamma^{k}r_{t+k+2}|s_t=s,a_t=a] \tag{16}$$

$$Q^{\pi}(s,a)=\sum_{s'}P^{a}_{ss'}[R^{a}_{ss'}+\gamma\mathbb{E}_{\pi}[\sum_{k=0}^{\infty}\gamma^{k}r_{t+k+2}|s_{t+1}=s']] \tag{17}$$

$$Q^{\pi}(s,a)=\sum_{s'} P^{a}_{ss'}[R^{a}_{ss'}+\gamma\sum_{a}\mathbb{E}_{\pi}[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2}|s_{t+1}=s', a_{t+1}=a']] \quad (18)$$

$$Q^{\pi}(s,a)=\sum_{s'} P^{a}_{ss'}[R^{a}_{ss'}+\gamma\sum_{a'}\pi(s',a')Q^{\pi}(s',a')] \quad (19)$$

La importancia de las ecuaciones de Bellman es que nos permiten expresar valores de estado como valores de otros estados. Esto quiere decir que si conocemos el valor de s_{t+1}, fácilmente podremos calcular el valor de s_t [171].

5.3 Q-Learning

El Q-Learning es un algoritmo de aprendizaje por refuerzo de modelo libre (no requiere modelo; el agente lo aprende mediante el entrenamiento). **Su objetivo es aprender una estrategia que le enseñe al agente que acción realizar bajo cada circunstancia, maximizando su puntuación.**

Para cualquier proceso de decisión de Markov finito (**FMDP** por sus siglas en inglés, de *finite Markov decision process*), **el Q-Learning encuentra una estrategia que es óptima, maximizando el valor esperado de la recompensa total durante los pasos sucesivos desde el estado actual.** La "Q" de Q-Learning nombra la función que devuelve la recompensa que se utiliza para el refuerzo y se puede decir que nos da la "calidad" de una acción para un estado determinado [172].

En el Q-Learning, se construye una tabla de memoria Q[s,a] para guardar todos los valores-Q para todas las posibles combinaciones de s y a. Se busca la recompensa R (si la hay) y el nuevo estado s'. Desde la tabla de memoria, se determina la acción siguiente a' que tiene el máximo Q(s',a'), es decir, la acción con la máxima recompensa.

Se presenta a continuación el algoritmo Q. Si γ es más pequeño que 1, es probable que Q consiga converger:

```
Inicializamos con Q₀(s,a) para todos los s,a, siendo el Q inicial Q₀
Tomamos el estado inicial s
Para k =1,2… hasta la convergencia, siendo k el número de iteraciones:
      Muestrea una acción a, toma el siguiente estado s'
      Si s' es terminal:
```

$$target = R(s,a,s')$$

```
            Muestreamos un nuevo estado inicial s'
      si no:
```

$$target = R(s,a,s') + \gamma \max Q_k(s',a')$$

$$Q_{k+1}(s,a) \leftarrow (1-\alpha) Q_k(s,a) + \alpha[target]$$

$$s \leftarrow s'$$

Supongamos el siguiente ejemplo: tenemos un robot que debe encontrar la salida en una mazmorra dividida en una cuadrícula. En la mazmorra hay minas; el robot solo puede moverse de casilla en casilla. Si el robot pisa una mina, estaría muerto; el robot tiene que alcanzar la casilla final en el tiempo más corto posible. En las casillas podemos encontrar un rayo, minas, o nada.

El sistema de recompensas es el siguiente:

1. El robot pierde 1 punto a cada paso. Esto se constituye así para que el robot tome el camino más corto y alcance el objetivo lo más rápidamente posible.
2. Si el robot pisa una mina, pierde 100 puntos y el juego termina.
3. Si el robot pisa el símbolo del rayo (⚡), gana 1 punto.
4. Si el robot alcanza el objetivo final, consigue 100 puntos.

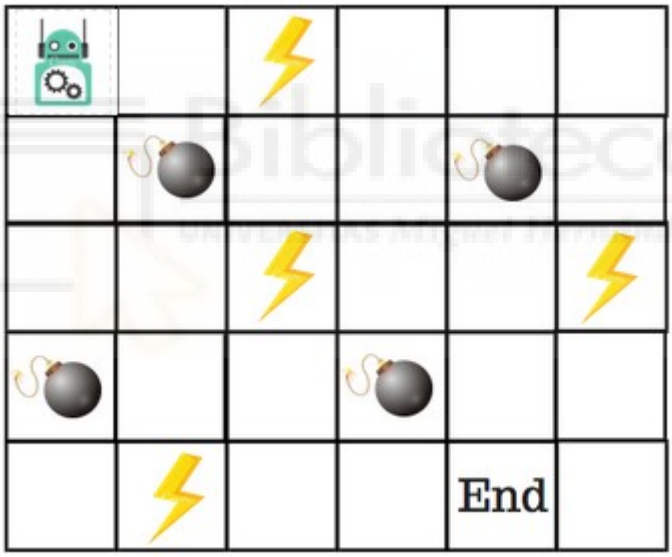

Figura 65: Entorno donde se desenvolverá nuestro robot para el ejemplo propuesto [173]

Existen cuatro posibles acciones para cada cuadrícula que no sea borde (izquierda, derecha, arriba y abajo).

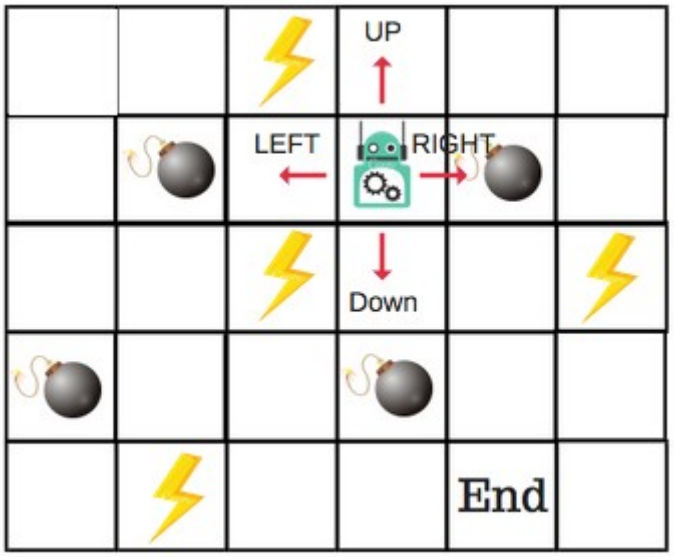

Figura 66: Robot en movimiento dentro del entorno [173]

Creamos ahora nuestra Tabla-Q. En la tabla, las columnas son las acciones y las filas los estados.

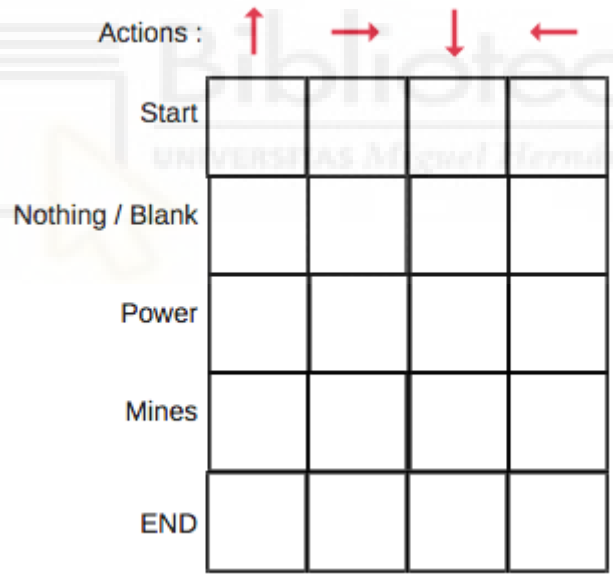

Figura 67: Presentación Tabla-Q [173]

Cada puntuación en la tabla-Q será la recompensa futura máxima esperada si toma esa acción en ese estado. Al ser un proceso iterativo, se actualiza la tabla en cada iteración. La función Q nos dará cada vez mejores aproximaciones conforme realicemos más iteraciones.

Se describe a continuación el algoritmo paso por paso:

1. **Inicializar la tabla-Q:** Primero se construye la tabla-Q, donde **habrán tantas columnas como acciones y tantas filas como estados. Se inicializan los valores a 0.** Para este

ejemplo, tenemos cuatro acciones (arriba, abajo, izquierda y derecha, a = 4) y cinco posibles estados (comienzo, nada, energía, mina y final, s = 5). Así que nuestra tabla tendrá cuatro columnas y cinco filas.

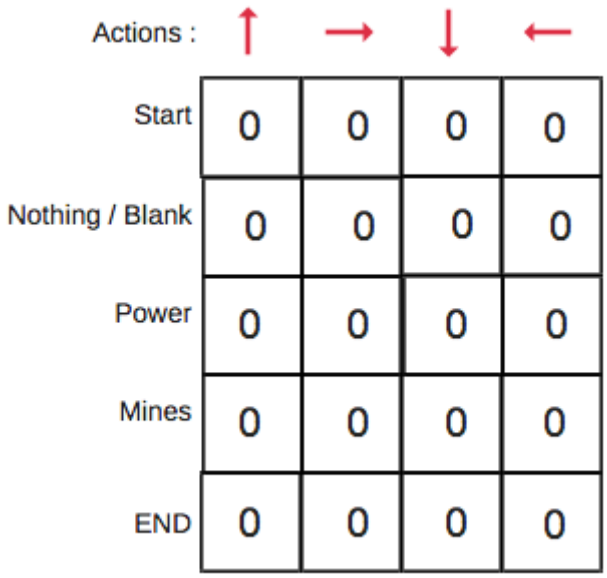

Figura 68: Inicialización Tabla-Q [173]

2. **Elegir y realizar una acción:** Esta combinación de pasos se realiza durante una cantidad de tiempo indefinida, es decir, que se estará realizando mientras no paremos el entrenamiento. Aquí entra en juego el concepto de **exploración frente a explotación** (del inglés *exploration vs explotation)*, explicado anteriormente. Al principio, **se utiliza** lo que se conoce como **estrategia de epsilon codiciosa** (del inglés *epsilon greedy strategy)*. Esta estrategia **realiza un descenso lineal del valor de epsilon, desde un valor máximo** (normalmente 1, en el que todas las acciones serán tomadas aleatorias) **hasta un valor mínimo, cercano a cero pero nunca cero**, ya que si epsilon llegase a cero, no dejaríamos margen a la exploración de nuevas acciones.

   Al principio, los ratios de epsilon serán altos; esto quiere decir que el robot explorará el entorno y elegirá acciones aleatorias. El por qué se realiza esto es porque no tenemos conocimiento del entorno (recordemos que es un algoritmo de tipo *model-free*, así que aprenderemos sobre el entorno mientras entrenamos).

   Durante este proceso de exploración, el robot progresivamente hace mejores estimaciones de los valores-Q. Para nuestro ejemplo, el robot selecciona una acción aleatoria; digamos que ir hacia la derecha. Actualizamos los valores-Q con la acción realizada desde la casilla de salida hacia la casilla derecha usando la ecuación de Bellman.

   Imaginemos por ejemplo, que nuestro robot acaba de empezar nuestro ciclo de entrenamiento (epsilon 1 o muy cercano a 1). A la hora de transicionar de una casilla a otra, deberemos elegir si realizamos una acción aleatoria (elegimos un movimiento aleatoriamente) o usamos el conocimiento adquirido previamente.

Puesto que nuestro conocimiento actual del entorno no es muy grande (acabamos de empezar el entrenamiento), es muy probable que la acción seleccionada sea aleatoria; conforme avancemos en el entrenamiento, epsilon irá disminuyendo, y en teoría deberíamos tener un mejor conocimiento del entorno y qué acciones debemos realizar y en qué situaciones concretas, para maximizar la puntuación (en nuestro caso llegar a la salida). Por tanto, **la estrategia de epsilon greedy lo que nos permite es al principio de los ciclos de entrenamiento realizar una exploración aleatoria, para progresivamente ir descubriendo qué acciones son más eficientes en las distintas situaciones que se nos presenten**, para progresivamente ir utilizando ese conocimiento adquirido mediante exploración, es decir, **explotar la experiencia adquirida mediante exploración aleatoria para realizar mejores acciones conforme disminuimos cantidad de acciones aleatorias que realizamos** (al disminuir epsilon, reducimos la probabilidad de realizar una acción de exploración, es decir, una acción aleatoria y que no se base en el conocimiento adquirido previamente).

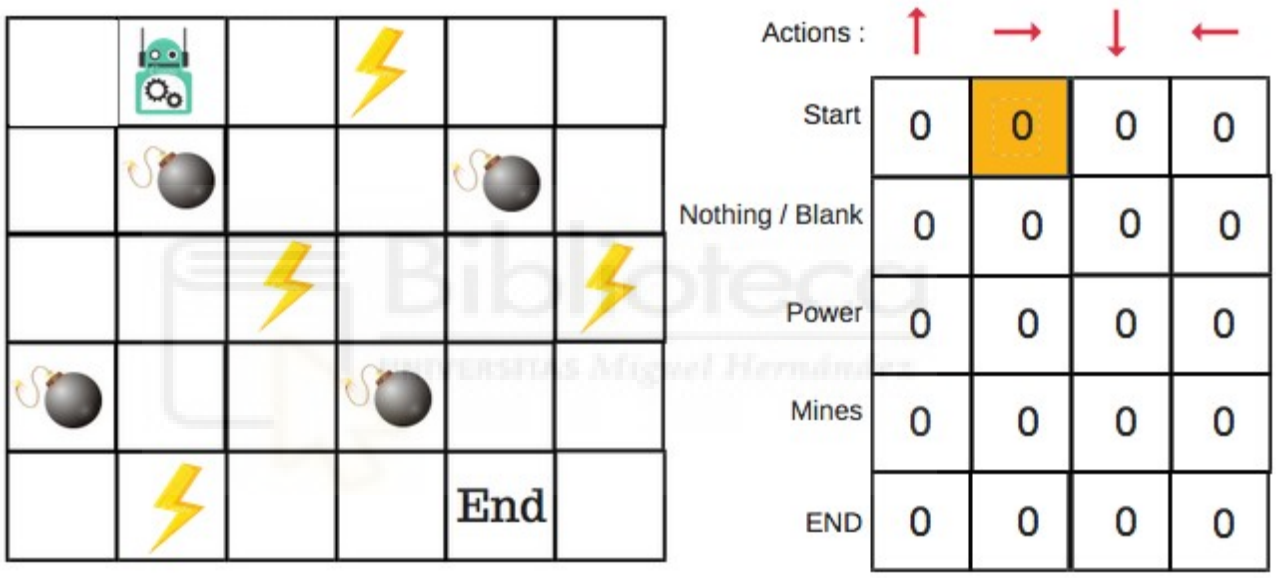

Figura 69: Entorno y Tabla-Q [173]

3. **Evaluar:** Una vez realizada la acción y observado su resultado y recompensa si la hay, necesitamos **actualizar la función Q(s,a). Repetiremos esto hasta que el entrenamiento termine.**

$$\underbrace{NuevoQ(s,a)}_{a}=\underbrace{Q(s,a)}_{b}+\underbrace{\alpha}_{c}[\underbrace{R(s,a)}_{d}+\underbrace{\gamma}_{e}\underbrace{maxQ'(s',a')}_{f}-\underbrace{Q(s,a)}_{b}] \quad (20)$$

a) Nuevo valor Q para ese estado y acción.

b) Valores actuales de Q.

c) Ratio de aprendizaje.

d) Recompensa por realizar esa acción en ese estado.

e) Gamma: ratio de descuento (el valor que le damos a las recompensas futuras).

f) Recompensa máxima futura esperada dado el nuevo estado (*s'*) y todas las posibles acciones en ese estado.

[173]

Este método es efectivo siempre y cuando el número de acciones y estados a manejar no sea muy alto; si es muy grande, usaremos el método conocido como Deep-Q-Learning.

5.4 Deep-Q-Learning

Como se ha visto en el apartado anterior, **el Q-Learning es un algoritmo que intenta maximizar la recompensa dado un estado y un conjunto de acciones determinadas, creando una tabla que nos indica la calidad de cada acción en un estado determinado. El principal problema de esta aproximación es que si el número de estado o acciones es muy grande, necesitamos una manera de extraer la información relevante del entorno y de manejar toda esa cantidad de información.** Ahí es donde entran en juego las redes neuronales, dando lugar a lo que se conoce como Deep-Q-Learning o abreviadamente DQN. Además, se comentarán algunos pequeños añadidos que se utilizan en las DQN comúnmente y que han sido utilizadas en la elaboración del agente de inteligencia artificial de este proyecto.

Aunque el Q-Learning es un algoritmo muy potente, su principal debilidad es su falta de generalización. Para estados que el agente no ha visitado previamente, no tiene forma de saber qué acción realizar, es decir, no tiene forma de estimar valores para estados no visitados previamente. **Para ocuparse de este problema, se introducen las redes neuronales.**

Imaginemos el caso expuesto anteriormente: una rejilla de tamaño 6x5 en la que buscamos que un robot llegue a la casilla final, sin pisar ninguna bomba. En este caso el Q-Learning, al ser un número reducido de posibles estados (30) será capaz de seleccionar la mejor acción (unas 120 posibles acciones en este entorno). Pero, ¿qué pasaría si en vez de 6x5, el tamaño fuera de 100x100? Tendríamos unos 10000 estados posibles; si pensamos que para cada estado podemos realizar 4 acciones posibles (arriba, abajo, izquierda o derecha) nos daría, para ese caso, una cantidad total de 40000 acciones posibles en nuestro entorno. ¿Cómo elegir la mejor?

Pensemos, por ejemplo, en *Pong*, uno de los entornos probados. Nuestro objetivo consiste en colar la bola tras la barra del contrario (situada a la izquierda) con nuestra barra (situada a la derecha). ¿Cuántos posibles estados hay? Si atendemos a las posibles variables del entorno, tendríamos:

- La posición de nuestra barra
- La posición de la barra del contrario
- La posición de la bola

El número de posibles estados, si bien no es infinito, resultaría muy difícil de calcular; aún así se puede deducir que el número de posibles estados es muy alto.

Es en este punto donde entran en juego las redes neuronales. **Debido a la capacidad de predecir de las redes neuronales artificiales, son capaces de, a pesar de no haberse enfrentado a una situación concreta, predecir con éxito en base a lo experimentado previamente.**

En una DQN, usamos la red neuronal como estimador de los valores de la función Q. La entrada de la red es el estado actual, mientras que la salida es el valor-Q asociado a cada acción; así, podemos seleccionar la que nos dé el valor más alto ya que la red neuronal estima que esa acción, para ese estado dado, es la mejor.

Como se ha mencionado antes, en 2013 DeepMind aplicó no una red neuronal convencional, sino una red neuronal convolucional para conseguir que su agente jugara a juegos de la videoconsola Atari [3] tan sólo recibiendo como entrada de la red las imágenes del juego, obteniendo como salida los valores-Q asociados a cada acción para cada estado dado. Para la construcción del agente utilizado en este proyecto se han utilizado el mismo tipo de redes neuronales.

El proceso de entrenamiento de la red se basa en la ecuación de actualización del Q-Learning. Recordemos que el objetivo del valor-Q se puede escribir como:

$target = r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta^-)$ [4], donde ϕ es el equivalente al estado s, y θ son los pesos de la red neuronal. La función de pérdida o *loss* se calculará por tanto como la diferencia al cuadrado entre los valores-Q objetivos y los valores-Q obtenidos por la red [174].

Algunas técnicas que se añaden a la DQN que no aparecen en el Q-Learning tradicional son:

- ***Experience Replay:*** **Los datos de entrenamiento en aprendizaje por refuerzo suelen estar altamente correlados** (es decir, tienen mucha relación entre ellos al ser consecutivos; este es uno de los motivos que hacen que el aprendizaje por refuerzo sea muy poco eficiente a la hora de necesitar datos), por lo que para solucionar este problema, **guardamos las transiciones, de las cuales un grupo serán seleccionadas aleatoriamente** (para romper la correlación) **y se utilizarán para actualizar los conocimientos que adquiere nuestro agente** [3] [174] [175].
- **Red objetivo separada:** se crea una red objetivo, con la misma estructura a la original (mismo número y tipo de capas, filtros, etc). Cada C pasos, se copian los pesos de la red original a la red objetivo [174]. **El porqué de usar dos redes neuronales idénticas, usando una red para calcular el *target*, es debido a que si usamos la misma red, estaremos variando el objetivo constantemente, generando inestabilidades y haciendo la convergencia del algoritmo más difícil** [175] [176] .

De los dos añadidos, ***experience replay*** **tiene mayor impacto en la mejora del rendimiento de la DQN** [176] [174].

El algoritmo Q por tanto con las nuevas modificaciones:

Algoritmo 1: deep Q-Learning con *experience replay:*

Inicializamos la memoria D con capacidad N

Inicializamos la función acción-valor Q con pesos aleatorios θ

Inicializamos el objetivo acción-valor ^Q con pesos $\theta^- = \theta$

Para el episodio 1, M haz

Inicializa la secuencia $s_1 = \{x_1\}$ y preprocesa la secuencia $\Phi_1 = \Phi(s_1)$

Para t = 1, T haz

Con probabilidad ε selecciona una acción aleatoria a_t;

Si no, selecciona $a_t = \text{argmax}_a\ Q(\Phi(s_t), a; \theta)$

Ejecuta la acción a_t, observa la recompensa r_t y el siguiente estado x_{t+1}

Actualizamos $s_{t+1} = s, a_t, x_{t+1}$ y preprocesamos $\Phi_{t+1} = \Phi(s_{t+1})$

Guardamos la transición $(\Phi_t, a_t, r_t, \Phi_{j+1})$ en D

Tomamos un número de transiciones $(\Phi_t, a_t, r_t, \Phi_{j+1})$ de D

Actualizamos $y_j =$

- r_j si el episodio termina en el paso j+1
- $r_j + \gamma \max_{a'} \hat{Q}(\Phi_{j+1}, a'; \theta^-)$, si no

Realizamos un paso de descenso del gradiente en $(y_j - Q(\Phi_j, a_j; \theta))^2$ con respecto a los parametros de la red θ

Cada C pasos, reseteamos ^Q = Q

fin

fin

[4] [176]

donde ϕ es en nuestro caso las 4 últimas imágenes; es lo que llamaremos estado. Para capturar el movimiento, usaremos cuatro cuadros (cuatro imágenes) [175] [176].

6. Construcción del agente de inteligencia artificial

En este apartado se explicará todo lo referente a los diferentes programas realizados para este trabajo fin de máster, explicando las distintas partes y funciones de los mismos.

Los recursos utilizados a la hora de realizar el código de este agente son:

Como **recursos teóricos**, se ha consultado [170] y [177]; como **tutoriales para la construcción del agente a nivel de programación**, [175] se utilizó como consulta sobre como construir una *DQN* con *tensorflow;* [178] para la realización de puntos de control en *Keras*; en [179] se explica como guardar modelos de *Keras;* y en [180] como comprobar la memoria disponible en Python. También **se ha consultado código** de diferentes autores, en los cuales se pueden ver diferentes estructuras para la construcción del agente de inteligencia artificial. Los recursos consultados en este caso son [181], la cual es una implementación completa de un agente para jugar a *Space Invaders;* [182], donde se expone el código de como hacer el *experience replay*; [183], donde se explica como construir la red e implementar propiamente dicho el algoritmo; [184], donde se presenta una implementación para el videojuego *Flappy Bird*; [185], el cual es una implementación completa sobre *OpenAI;* [186] corresponde a las *baselines* de *OpenAI*, las cuales son pequeñas funciones de código que nos sirven como pequeñas ayudas para el uso de *OpenAI-Gym*; [187], donde aparecen diferentes agentes de aprendizaje por refuerzo; [188], otra implementación para *Atari;* [189], el cual contiene ejemplos de implementaciones de agentes de aprendizaje por refuerzo con *Keras* y *Atari*; [190], una implementación de *Breakout* (sobre *Universe*, en vez de sobre *OpenAI-Gym*). Por último, se consultan diferentes **tutoriales con código**, que ayudan a clarificar aún más la construcción de nuestro agente. Los recursos en este caso consultados son [191], el cual es un tutorial de aprendizaje por refuerzo con *Keras* y *OpenAI-Gym*; [192], en la que se explica como ganar en los juegos de *Atari* con *Deep Reinforcement Learning*; [193], el cual es un tutorial básico de *Keras* para aprendizaje por refuerzo; [194], un curso con varias implementaciones de varios agentes de *reinforcement learning;* [195], del mismo curso, la implementación concreta de una DQN para *Space Invaders* de la *Atari*; [196], otro tutorial sobre como realizar *Deep-Q-Learning* con *Tensorflow* y *Space Invaders;* en [197] encontramos otro tutorial sobre como utilizar *Keras* y la construcción de una *DQN* para jugar a *FlappyBird*. Por último, en [198] se presentan ciertas mejoras sobre una *DQN* básica.

6.1 Premisas

Las premisas sobre las que se ha construido este agente son las siguientes:

- **Simplicidad de código:** se ha intentado, siempre que se ha podido, realizar un **programa lo más accesible posible** (dentro de la complejidad del mismo), **usando el mínimo número de funciones e instrucciones.**
- **Comprensible:** las variables y funciones se han intentado llamar con nombres coherentes, que expliquen su funcionamiento o den a entender para qué se usan. Además **el código ha sido comentado para clarificar lo que realiza cada función.**

- **Modular: se han separado**, siempre que no ha afectado al funcionamiento del agente, **las funciones propias del agente de las de procesamiento**, así como del bucle del programa y las funciones accesorias.
- **Métrica: se han intentado recoger datos suficientes durante las sesiones de entrenamiento, tanto gráficos como numéricos**, para comprobar la evolución del agente durante el entrenamiento, tanto en el momento como para su posterior análisis.

6.2 Esquema y funcionamiento general del programa

Se presenta un esquema del programa para clarificar su funcionamiento. En los siguientes apartados se explicará cada parte del programa y cual es su función.

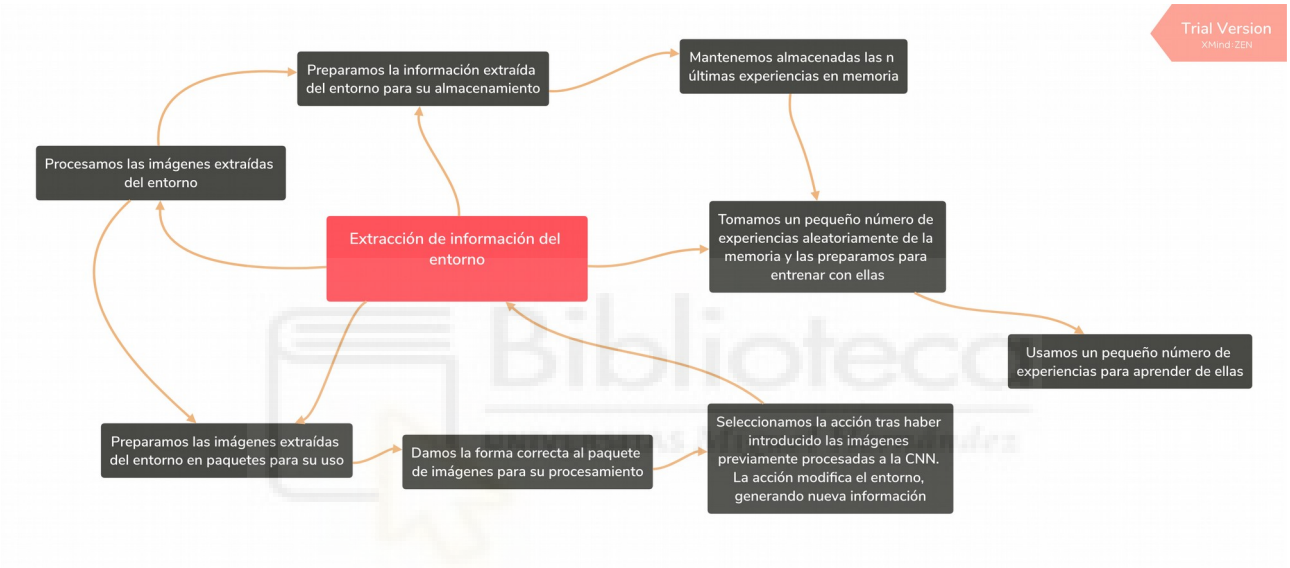

Figura 70: Esquema general de funcionamiento del programa

Se describe el esquema general presentado:

1. **Extracción de información del entorno:** del juego (entorno) extraemos la siguiente información; esto se realiza en la parte descrita como bucle de programa, explicada en apartados posteriores de este capítulo.
    - **Estado (state):** corresponde a la imagen anterior (frame) a la que genera la acción que realizamos.
    - **Acción (action):** es la acción determinada por la red neuronal.
    - **Estado siguiente (next_state):** corresponde a la imagen que genera la acción ejecutada; es decir, partiendo de un estado (state), la red neuronal selecciona la acción que mejores Q-values tiene; al realizar esta acción en nuestro entorno, genera un nuevo estado, y así sucesivamente.

- **Recompensa (reward):** este valor simplemente almacena los puntos acumulados durante la partida; nos indica lo acertadas que son nuestra acciones.
- **Finalizado (done):** este valor indica si hemos llegado a un estado terminal; es decir, si hemos terminado o no la partida; simplemente devuelve un valor verdadero (true, si la partida ha acabado) o falso (false), si por el contrario la partida aún no ha acabado.
- **Vidas (lives):** devuelve el número de vidas que tenemos en ese instante dado; en algunos casos este valor determina si el juego ha terminado o no.

Al conjunto de estado, acción, nuevo estado, recompensa, finalizado y vidas se le conoce como experiencia, ya que define en un instante dado en qué condiciones se encuentra el agente.

2. **Procesamiento de imagen:** debido a que si introdujéramos la imagen en color y con el tamaño que presenta tal y como la recibimos del entorno en la red neuronal sería muy costoso computacionalmente hablando procesarla, debemos **pasarla a escala de grises** (ya que en este caso, el color no aporta información extra); asimismo, **reducimos el tamaño de la imagen** por este motivo. **La función que realiza todo esto se llama *image_processing***, y queda explicada en detalle en los apartados posteriores de este capítulo.
3. **Preparación de las experiencias para su almacenamiento en memoria:** puesto que debemos almacenar las experiencias para su posterior uso de una manera determinada, necesitamos una función que se encargue de ello; de esto se encarga la función *four_memory*, la cual está contenida dentro de nuestro agente.
4. **Almacenamiento en memoria de las n últimas experiencias:** puesto que necesitaremos experiencias para realizar el *experience replay* (esto es, extraer información de las experiencias pasadas para aprender de ellas), almacenaremos un número determinado de las últimas experiencias en memoria; cuando se llegue al máximo, se irán eliminando las más antiguas para almacenar las nuevas. Esto se realiza con la función *memory*, integrada dentro de nuestro agente.
5. **Tomar un pequeño número de experiencias aleatoriamente de la memoria y las preparamos para entrenar con ellas:** puesto que computacionalmente sería muy costoso entrenar con todas las experiencias almacenadas, seleccionamos un pequeño número aleatorio de paquetes de cuatro experiencias (previamente procesadas por *four_memory* y almacenadas en *memory*) con las que entrenaremos nuestro agente para extraer información. De esto se encarga la función *replay_sampler*, la cual se encuentra dentro de nuestro agente.
6. **Usamos un pequeño número de experiencias para entrenar con ellas:** en la función *training*, contenida en nuestro agente, utilizamos los paquetes de experiencias creados previamente (y seleccionados aleatoriamente), introduciéndolas en nuestro algoritmo para extraer información de las mismas. Esto se realiza para evitar que el agente acabe solamente aprendiendo de las últimas experiencias, y olvide lo aprendido anteriormente.
7. **Preparamos las imágenes extraídas del entorno en paquetes para su uso:** antes de poder introducir las imágenes extraídas de las experiencias del entorno, debemos prepararlas; de ello se encargará la función *four_batcher*, construida dentro del agente. Esta función se

tomará las cuatro últimas imágenes (los cuatro últimos *next_states*) y creará un paquete con los mismos. Cuando se produce un nuevo estado, se elimina el más antiguo, creando así un nuevo paquete.

8. **Damos la forma correcta al paquete de imágenes para su procesamiento:** puesto que el paquete creado en *four_batcher* debe tener una forma determinada antes de poder utilizarlo, la función *batcher*, creada dentro de nuestro agente, se encarga de realizar esto.
9. **Selección de la acción:** tomamos el paquete de imágenes creado en *batcher* y lo introducimos en la función *action_selection* (la cual se encuentra dentro de nuestro agente). Esta función utilizará la red neuronal convolucional para extraer información del paquete creado por *batcher*; la salida de la red neuronal serán una serie de valores (Q-values) que nos indicarán qué acción es la que se considera mejor a realizar en ese momento (cuanto más alto el valor-Q, mejor se considera la acción para ese momento).

6.3 Partes del programa

En este apartado se detallarán las distintas partes del agente, explicando lo más claramente posible el código y como funciona el mismo.

El código ha sido dividido en tres archivos distintos: el central (***Deimos_v3_x_agent.py****, siendo la x la versión del programa; cuando más alto el número, más reciente será)*, en el que se recogen el agente y el bucle de programa desde donde se ejecuta el mismo; otro (***Deimos_v3_aux.py)***, donde se han recogido las funciones accesorias que se han podido extraer sin problemas del agente o del archivo central; y por último, un archivo (***Deimos_v3_metrics.py)*** donde se recogen las métricas (valores a medir) del programa. Esto se ha realizado para que quede todo más limpio y ordenado.

También se ha incluido una versión multi-gpu del archivo central (*Deimos_v3_x_agent_multi.py*), para que en caso de que el equipo en el que se vaya a ejecutar disponga de múltiples GPUs podamos aprovecharlas; asimismo, se incluye el archivo *Deimos_v3_google_colab.py*, el cual es una versión integrada del programa con todas las funciones del mismo contenidas en un mismo archivo y preparada para ejecutar en la nube mediante la plataforma Google Colab. El archivo *Deimos_v3_google_colab_graph.py* se encarga de, una vez recopilados los datos correspondientes de las sesiones con Google Colab, recrear la gráfica para su posterior uso, ya que no podemos realizar la gráfica dinámicamente como sí se realiza cuando el agente se ejecuta de manera local.

Se incluyen además varios programas accesorios para el testeo del agente: *Deimos_v3_agent_test.py*, el cual se encarga de realizar un testeo durante 200 episodios, cargando los datos de entrenamiento y devolviendo una gráfica con los resultados, y *Deimos_v3_agent_random.py*, que realiza la misma función pero seleccionando acciones de manera aleatoria.

6.3.1 Agente

En este apartado **se explicarán las funciones referentes al agente**, es decir, las encargadas del cálculo del algoritmo, construcción de redes neuronales, selección de acción, etc. Las funciones correspondientes al agente se encuentran dentro de *class AI*.

6.3.1.1 Construcción de las redes neuronales: funciones CNN y target_CNN

Se presentan las **funciones *CNN* y *target_CNN***, ya que **se utilizan dos redes: con *CNN* calculamos los *Q-values* de las acciones y de las experiencias pasadas, y con *target_CNN* calculamos la recompensa máxima posible.** Esto se realiza así para, como se ha visto anteriormente, dotar al agente de más estabilidad.

Los pesos de las neuronas se copian de la red principal (*CNN*) a la red objetivo (*target_CNN*) tras un número determinado de pasos. Para ello, la arquitectura de las redes debe ser idéntica (mismas capas, mismo orden, etc). **Se describe a continuación la construcción de las mismas**, la cual es la misma que la presentada en [4]:

- **Una capa convolucional**, formada por 32 filtros de tamaño 8x8, stride (píxeles que da cada filtro realizando la operación de la convolución) de 4x4; como tamaño de entrada tenemos a la variable *AI.cnn_shape*, cuyo valor es (105, 80, 40), esto es, cuatro imágenes de 105x80. Por último encontramos el activador de la capa, en este caso *relu*.
- **Otra capa convolucional** de 64 filtros, de tamaño 4x4, stride 2x2, y activador *relu*.
- **Capa convolucional** de 64 filtros de tamaño 3x3, stride 1x1 y activador *relu*.
- **Una capa *flatten***, cuyo cometido es convertir los vectores multidimensionales de las capas convolucionales en uno unidimensional, de forma que sea entendible para las capas de tipo *Dense* posteriores [199].
- **Una capa de tipo *Dense*** de tamaño 512, es decir, una capa de neuronas totalmente conectada, y cuyo activador es de tipo *relu*.
- **Una última capa de tipo *Dense***, de tamaño *AI.action_size*; esta variable contiene el número de acciones del entorno en cuestión; se almacena en una variable para que automáticamente se seleccione el número de acciones, independientemente del entorno, lo cual nos ahorra tener que averiguar cuantas acciones posibles tiene cada entorno.
- ***model.compile*** se encarga de compilar la red; para el *loss* utilizamos *mse*, que corresponde a *mean squared error*, es decir, el error cuadrático medio; como optimizador, *Adam*; y en cuanto al ratio de aprendizaje, se usará un ratio fijo, almacenado en la variable *AI.learning_rate*. Los ratios utilizados han sido de 0.00001 y 0.00025.
- La instrucción ***model.summary*** nos muestra por consola la arquitectura de la red neuronal. Se utiliza para comprobar que se ha construido correctamente.

En cuanto a la construcción de la red multi-GPU, la estructura es la misma; simplemente se añade la línea `parallel_model = multi_gpu_model(model, gpus=gpu_number)` justo después de la última capa densa. En vez de llamar a `model.compile`, llamaremos a `parallel_model.compile`. La variable *gpu_number* contiene el número de GPUs disponibles en el sistema que estemos utilizando.

A continuación **se presenta un esquema de la estructura de la red neuronal convolucional**, para clarificar mejor su arquitectura, así como el código que se ha utilizado para implementarlo en nuestro agente:

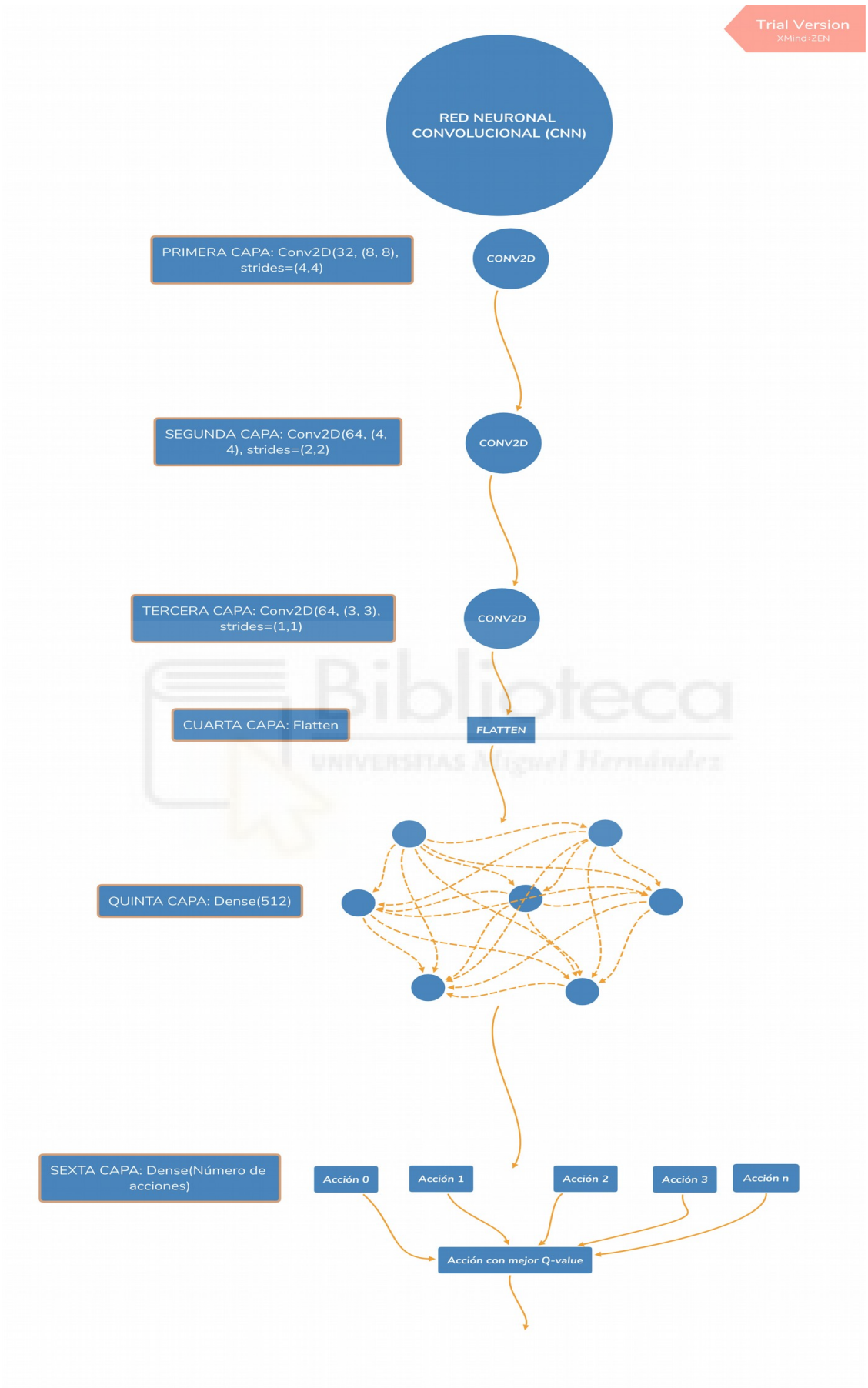

Figura 71: Esquema red neuronal convolucional utilizado en este proyecto

```
    def CNN(self):
        model = Sequential()
        model.add(Conv2D(32, (8, 8), strides=(4,4), input_shape = AI.cnn_shape,
activation='relu'))
        model.add(Conv2D(64, (4, 4), strides=(2,2), activation='relu'))
        model.add(Conv2D(64, (3, 3), strides=(1,1), activation='relu'))
        model.add(Flatten())
        model.add(Dense(512, activation='relu'))
        model.add(Dense(AI.action_size, activation='linear'))
        model.compile(loss="mse", optimizer=Adam(lr=AI.learning_rate))
        model.summary()
        print("Model CNN construction complete")
        return model


    def target_CNN(self):
        target_model = Sequential()
        target_model.add(Conv2D(32, (8, 8), strides=(4,4), input_shape =
AI.cnn_shape, activation='relu'))
        target_model.add(Conv2D(64, (4, 4), strides=(2,2), activation='relu'))
        target_model.add(Conv2D(64, (3, 3), strides=(1,1), activation='relu'))
        target_model.add(Flatten())
        target_model.add(Dense(512, activation='relu'))
        target_model.add(Dense(AI.action_size, activation='linear'))
        target_model.compile(loss="mse", optimizer=Adam(lr=AI.learning_rate))
        target_model.summary()
        print("Target CNN construction complete")
        print("\033[1;32;38mCNN construction completed sucessfully")
        return target_model
```

6.3.1.2 Update net

Esta función se encarga de tomar los pesos de la red neuronal principal (*CNN*) y transferirlos cada determinado número de pasos a la red neuronal objetivo (*target_CNN*). Se ha utilizado un número de pasos de 10000, tal y como se hizo en [4].

```
    def update_net(self):
        model = self.model
```

```
model_weights = model.get_weights()
target_model = self.target_model
target_model.set_weights(model_weights)
print("Target model sucessfully updated")
```

6.3.1.3 Memory

La función *memory* es la encargada de almacenar las experiencias, previamente procesadas. El tamaño que se ha estipulado inicialmente es de 100000, mientras que el utilizado en [4] es de 1 millón. La función, cuando está llena la variable, elimina automáticamente la experiencia más antigua.

```
def memory(self):
    if len(AI.mem_batch) < AI.mem_size:
        AI.mem_batch.append(AI.four_memory_batch)
    else:
        AI.mem_batch.popleft()
        AI.mem_batch.append(AI.four_memory_batch)
```

6.3.1.4 Replay sampler

Esta función se encarga de seleccionar un número determinado de experiencias para realizar el *experience replay*. Las experiencias se seleccionan aleatoriamente, para romper la correlación existente entre las mismas, ya que si no rompemos esa correlación, reforzaríamos determinados comportamientos, lo que evitaría que el algoritmo generalizase correctamente, tal y como se explica en [200].

La función opera de la siguiente manera:

- Comprueba el tamaño de la memoria `len(AI.mem_batch)`. Si es menor que el tamaño de la muestra que se quiere utilizar `(AI.replay_sample_size)`, almacena en `AI.sample_batch` un número de experiencias equivalentes al tamaño de la memoria, ya que sería inferior a `AI.replay_sample_size.`
- Si el tamaño de la memoria es mayor al del número de experiencias que se quieren utilizar para *experience replay*, selecciona aleatoriamente un número de experiencias igual a `AI.replay_sample_size` y se almacenan en `AI.sample_batch`.
- Despúes, se transforma `AI.sample_batch` en un array (`np.asarray(AI.sample_batch)`); por último, le damos las siguientes dimensiones al array: número de experiencias, longitud del array, número de frames por experiencia. Como resultado, y teniendo en cuenta que se utiliza un `AI.replay_sample_size` de 32 en la mayoría de los casos, como el utilizado en [4], tenemos un tamaño de 32x6x4, es decir, 32 experiencias con 6 variables internas

(estado, acción, nuevo estado, recompensa, terminal y vidas), la cual contiene cada uno de ellos 4 frames (recordemos que se introducen 4 imágenes consecutivas a la red).

```
    def replay_sampler(self, mem_batch):
        AI.sample_batch = []

        if len(AI.mem_batch) < AI.replay_sample_size:
            AI.sample_batch.append(random.sample(AI.mem_batch,
len(AI.mem_batch)))
        else:
            AI.sample_batch.append(random.sample(AI.mem_batch,
AI.replay_sample_size))

        AI.sample_batch = np.asarray(AI.sample_batch)
        AI.sample_batch.shape = (AI.replay_sample_size, 6, 4)
```

6.3.1.5 Action selection

Esta función se encarga de seleccionar la acción a realizar por la red neuronal; el procedimiento que realiza es el siguiente:

- **Calculamos un número aleatorio;** por defecto la función `np.random.rand()` de la librería *numpy* nos dá un valor aleatorio entre 0 y 1. **Si ese valor está por debajo del valor de `AI.epsilon` o el episodio en ejecución es menor que `observation_episodes`** (esto último se hace para generar un número de experiencias aleatorias y guardarlas en la memoria, tal y como se hace en [4]), **realizamos una acción aleatoria.**
- **En caso contrario, utilizamos el paquete de cuatro cuadros actuales**, ya procesados `(s_batch)` **y predecimos el *Q-value* de cada acción para el estado en el que nos encontramos.**
- **De esos valores, seleccionamos el de mayor valor**, que corresponderá a la acción que la CNN ha estimado que es más probable que consiga un punto.
- Tras esto, y siempre que el episodio en el que estemos no sea de observación, **comprobamos si el valor de epsilon está por encima del valor final que hemos estimado; si no es así, al valor actual le restamos un valor** `(AI.epsilon_decay).` La fórmula de *epsilon decay* utilizada es la siguiente, y queda declarada al inicio del agente: `epsilon_decay = (initial_epsilon-final_epsilon)/epsilon_frames`, siendo `initial_epsilon = 1`, `final_epsilon = 0.1` y `epsilon_frames = 300000`. Es decir, empezamos con un epsilon de 1 (acciones totalmente aleatorias) y vamos descendiendo, durante un número determinado de cuadros, hasta 0.1 (posibilidad de un realizar una acción aleatoria del 10%). El valor de `epsilon_frames` se varió durante los distintos experimentos, para comprobar su incidencia en el aprendizaje, siendo el valor inicial

utilizado de 300000. `epsilon_frames` no se hace cero para garantizar que existe un margen de exploración siempre. En [4] el descenso de epsilon se realiza durante 1 millón de cuadros.

```
def action_selection(self, s_batch):
    random_value = np.random.rand()
    if random_value < AI.epsilon or i < observation_episodes:
        best_action = env.action_space.sample()
        best_action_value = 0
    else:
        action_values = self.model.predict(s_batch)
        best_action = np.argmax(action_values[0])
        best_action_value = action_values[0, best_action]
    if i > observation_episodes:
        if AI.epsilon > AI.final_epsilon:
            AI.epsilon -= AI.epsilon_decay
    return best_action, best_action_value
```

6.3.1.6 Training

Esta función es la encargada de realizar el *experience replay*, es decir, de extraer información de experiencias pasadas. A continuación se explicará la función en detalle:

- **Inicializamos los *batches* que vamos a utilizar vacíos**; de esta forma cada vez que se llame a la función tenemos la certeza de que no se producen errores y que estamos utilizando las experiencias que se han seleccionado.
- **Se crean los *batches* que se van a utilizar.** Recordemos que `sample_batch` contiene 32 experiencias, cada una de las cuales está formada por 4 estados, 4 acciones, 4 acciones siguientes, 4 recompensas, 4 estados terminales y 4 estado de las vidas (`state, action, next_state, reward, done, lives`). Para ello, recorremos `sample_batch` a lo largo de su longitud (`len(AI.sample_batch)`), extrayendo cada elemento correspondiente `state, action, next_state, reward, done`) (la variable `lives` no es necesaria para el funcionamiento del algoritmo). Por tanto, por ejemplo, en `AI.training_s_batch = np.asarray(list(AI.sample_batch[m,0])).reshape(AI.batch_shape)` encontramos que recorremos `AI.sample_batch` con `m` (utilizado en el `for` en el que se encuentran contenidos todos los `AI.training`), siendo el segundo número (en este ejemplo el 0) el correspondiente al índice que determina qué estamos extrayendo, si `state, action, next_state, reward, done` o `lives`. Puesto que Python indexa los arrays empezando por el 0, nuestro array tendrá de índices del 0 al 5, siendo 0 el correspondiente a `state`, 1 a `action`, 2 a `next_state`, 3 a `reward`, 4 a `done`, y por último, 5 a `lives`, es

decir, en el orden en el que las hemos introducido previamente. En la misma línea de código se aprovecha para cambiar de formato la variable que se esté almacenando en el `AI.training` correspondiente y convertirla en array; asimismo se le da la forma adecuada utilizando la instrucción `reshape`. La forma con la que debe introducirse está almacenada en la variable `AI.batch_shape`, la cual queda definida al inicio del agente como `batch_shape = (1,105,80,4)`, es decir, 1 paquete de 105x80 formado por 4 *frames*, acciones, etc.

- **Simultáneamente, aprovechamos el `for` para ir ejecutando el algoritmo**, el cual es de la siguiente forma: `Q_values[m] = self.model.predict(AI.training_s_batch.reshape(AI.batch_shape))`

  `Q_values[m, AI.training_a_batch] = AI.training_r_batch`

  En estas dos líneas de código **se utiliza `model.predict` para predecir los *Q-values* mejores para el *batch* de estados correspondientes al índice `m` en ese instante**; en la siguiente línea, se asocian estos valores con su acción correspondiente y la recompensa que ha obtenido. Esta parte del algoritmo, como se puede observar, se realiza con la red neuronal principal.

- Siguiendo con el algoritmo, tal y como se describe en [4], **calculamos la recompensa máxima que se puede obtener**; si el juego es distinto de *Pong*, utilizamos el número de vidas restantes: si ha perdido una vida, se le penaliza evitando que maximice su recompensa, es decir, a pesar de que el episodio continúe, se toma a efectos del algoritmo como estado terminal. Con *Pong* no se puede realizar debido a que no existen vidas en el juego, ya que la partida se gana cuando uno de los jugadores llega a 21 puntos. Esta predicción se realiza con la red neuronal objetivo.

- Por último, **calculamos el loss**; para ello comparamos todos los *batches* de estado con los *Q-values* calculados.

```
def training(self, sample_batch):
        AI.training_s_batch = []
        AI.training_a_batch = []
        AI.training_ns_batch = []
        AI.training_r_batch = []
        AI.training_d_batch = []
        Q_values = []
        loss_batch = []
        batch_size = len(AI.sample_batch)
        Q_values = np.zeros((batch_size, AI.action_size))
        new_Q = np.zeros((batch_size, AI.action_size))

        for m in range(batch_size):
```

```
            AI.training_s_batch =
np.asarray(list(AI.sample_batch[m,0])).reshape(AI.batch_shape)
            AI.training_a_batch = np.asarray(list(AI.sample_batch[m,1]))
            AI.training_a_batch = AI.training_a_batch[:,0].astype(int)
            AI.training_ns_batch =
np.asarray(list(AI.sample_batch[m,2])).reshape(AI.batch_shape)
            AI.training_r_batch = np.asarray(list(AI.sample_batch[m,3]))
            AI.training_d_batch = np.asarray(list(AI.sample_batch[m,4]))


            loss_batch.append(AI.training_s_batch)


            Q_values[m] =
self.model.predict(AI.training_s_batch.reshape(AI.batch_shape))
            Q_values[m, AI.training_a_batch] = AI.training_r_batch


            if 'Pong' in game:
                    new_Q[m] =
self.target_model.predict(AI.training_ns_batch.reshape(AI.batch_shape))
                    Q_values[m, AI.training_a_batch] += (AI.gamma *
np.max(new_Q))
            else:
                if np.all(AI.training_d_batch) == False:
                    new_Q[m] =
self.target_model.predict(AI.training_ns_batch.reshape(AI.batch_shape))
                    Q_values[m, AI.training_a_batch] += (AI.gamma *
np.max(new_Q))


        loss =
self.model.fit(np.asarray(loss_batch).reshape(batch_size,105,80,4), Q_values,
batch_size=batch_size, verbose=0, shuffle=False)
        loss_history = loss.history['loss'][0]
        AI.loss_list.append(loss_history)


        if 'Pong' in game:
            if done == True:
                AI.loss_plt = np.mean(AI.loss_list)
                AI.loss_list = []
        else:
```

```
if lives == 0:
    AI.loss_plt = np.mean(AI.loss_list)
    AI.loss_list = []
```

6.3.2 Bucle de programa

El bucle de programa es de donde extraemos la información que luego procesaremos y utilizaremos para el algoritmo. Es el encargado de gestionar los episodios y los pasos que se dan dentro de los mismos.

El bucle está formado a su vez por dos `for` anidados: el primer `for` se encarga de los episodios, y el segundo del número de pasos en cada episodio.

En el bucle de episodios, encontramos que se reinicia el entorno cada vez que comienza un nuevo episodio (`env.reset()`), así como que se reinician las recompensas a 0 (`total_reward = 0`); si hemos pasado el número de episodios de observación, guardamos cada 10 episodios (`agent.save(i_update, AI.epsilon)`) y mostramos la información referente al uso de memoria y la CPU.

Si estamos ante el inicio del primer episodio (`if i == 0 and steps == 0`), **creamos una acción aleatoria** (`random_action = env.action_space.sample()`), **damos un paso en el entorno que estemos ejecutando, extrayendo la información del mismo** (`state1, reward1, done1, lives1 = env.step(random_action)`). Generamos un `exp_batch` con los datos extraídos y llamamos cuatro veces a `agent.four_batcher`; esto se hace para inicializar el agente y que siempre tenga al menos una experiencia completa. Posteriormente borramos las variables generadas; si bien el gasto de memoria no es significativo, no los volveremos a utilizar.

Por otra parte, también inicializamos `no_op_counter` en un número aleatorio entre 4 y 30, siguiendo el funcionamiento propuesto en [4], en el que al comienzo del episodio se realiza la acción *no-op* (no hacer nada) durante 30 pasos como máximo.

Pasamos a explicar el bucle de pasos, done se realizan la mayoría de acciones referentes a la toma de información para el agente. El número de pasos máximo se ha establecido en 10000, un número que se ha considerado más que suficiente para la finalización de cualquier episodio en cualquier juego.

Lo primero que se comprueba al inicio del bucle, es si se han realizando el número de acciones *no-op*; si es así, realizamos una llamada a la función dentro del agente para que realice una acción (`action = agent.action_selection(AI.s_batch)`). Una vez obtenida la acción, la aplicamos al entorno que estemos ejecutando; en este punto, extraeremos la información de este paso con (`next_state, reward, done, lives = env.step(action[0])`), de la cual extraeremos, como se indica, el estado siguiente, la recompensa obtenida, si ha terminado o no, y el número de vidas.

En cuanto a la recompensa, **se utiliza la función de *numpy np.sign* para que internamente todas las recompensas valgan -1, 0 o 1**; esto se realiza siguiendo los motivos expuestos en [4], es decir, limitar el error y poder utilizar el mismo ratio de aprendizaje en múltiples juegos. También puede

afectar al rendimiento del agente ya que no puede diferenciar entre recompensas de diferente magnitud. **Este procedimiento de "limitar" las recompensas se conoce como *reward clipping*.** Siguiendo con lo expuesto en [4], **se utiliza el número de vidas (en los juegos en los que existan), para que cuando nuestro agente pierda una vida durante el episodio, el estado cambie a terminado** (`done == True`). De esta manera, **forzamos al agente a que en cierto modo tenga en cuenta su propia integridad, ya que si el estado se da como terminado, aunque se le permita terminar el episodio normalmente, internamente en el algoritmo no podrá maximizar su recompensa.**

Seguidamente, **almacenamos toda la información relevante en un paquete**; procesamos a la vez las imágenes (`image_processing(state)) e image_processing(next_state)`. El `exp_batch` será enviado para su procesamiento y almacenaje en memoria (`agent.four_batcher(exp_batch)` y `agent.four_memory(exp_batch)`). Como hemos dado por terminado el paso, **el nuevo estado pasa a ser el estado actual** (`state = next_state`). Vaciamos el paquete de experiencias, añadimos un paso al contador de pasos, y mostramos el entorno con el resultado actual.

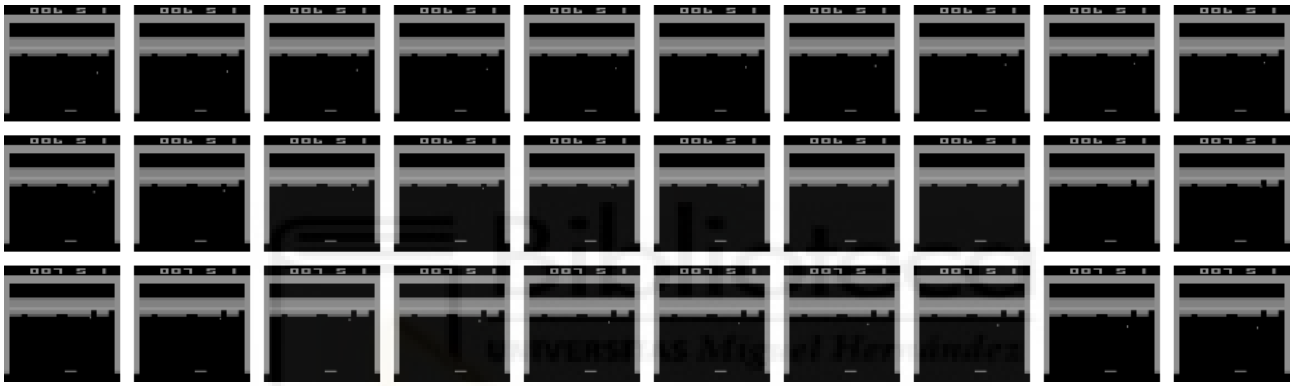

Figura 72: Capturas de pantalla de Breakout tomadas una vez procesada la imagen [201]

Conviene comentar aquí el ***frame skipping***. Se podría pensar que los cuatro *frames* que se introducen son consecutivos (y estaríamos en lo cierto en cualquier versión del entorno que llevase la extensión `NoFrameSkip`).

Tanto en [3] como en [4] **lo que se aplica es saltarse un número de cuadros determinado** (viene determinado por la versión del entorno que estemos utilizando; más información al respecto en el anexo correspondiente a *OpenAI-Gym*). **En nuestro caso, estamos tomando de cada cuatro cuadros, el último.**

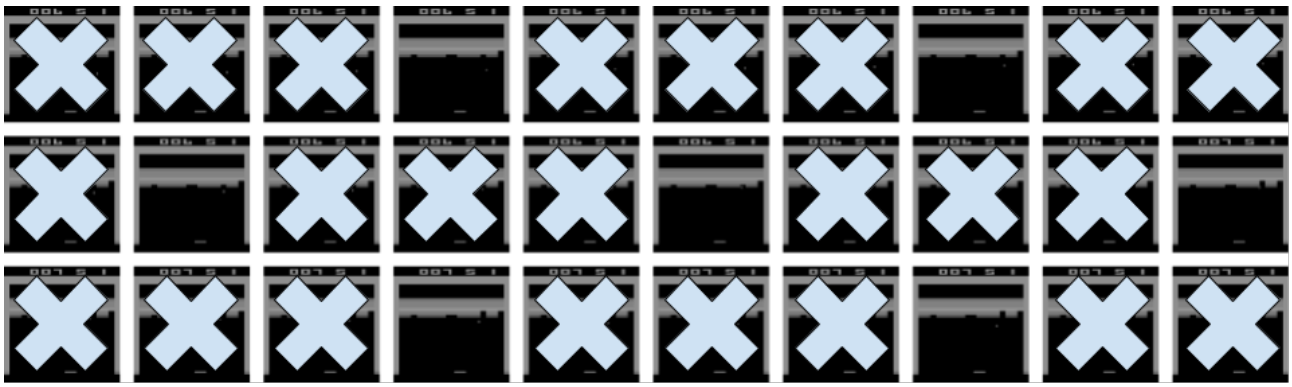

Figura 73: De las imágenes anteriores, tomamos una de cada cuatro [201]

Nombremos cada cuadro que no está marcado con una X en la imagen como x1, x2… hasta x7. Lo que estamos introduciendo a nuestro algoritmo sería `s1 = (x1,x2,x3,x4)`, después `s2 = (x2,x3,x4,x5)` y así sucesivamente. **Se podría pensar que de esta manera el agente está perdiendo mucha información; pero la realidad es que si no realizamos este subsampleo, cuatro cuadros consecutivos no dan a nuestro algoritmo suficiente información para distinguir el movimiento** [201].

Si hemos terminado con los episodios de observación, llamamos a `agent.replay_sampler(AI.mem_batch)` y `agent.training(AI.sample_batch)` para realizar el *experience replay*. También se realiza la llamada a la función que copia los pesos de la red principal a la red objetivo, siempre que el número de pasos sea el estipulado. El resto del bucle del programa muestra como se almacenan los diferentes valores que luego se utilizan como métricas para mostrar por pantalla, así como las llamadas a las funciones correspondientes.

```
for i in range (episodes):
    state = env.reset()
    total_reward = 0
    if i > observation_episodes:
        if i % 10 == 0:
            agent.save(i_update, AI.epsilon)
            memory_usage()
            print("\033[1;37;38mMemory length: ", len(agent.mem_batch))
    if i == 0 and steps == 0:
        random_action = env.action_space.sample()
        state1, reward1, done1, lives1 = env.step(random_action)
        done1 = False
        action1 = [random_action, 0.0]
        lives1 = lives1['ale.lives']
        next_state1 = state1
```

```
        exp_batch = (image_processing(state1), action1,
image_processing(next_state1), reward1, done1, lives1)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        reward1 = np.sign(reward1)
        total_reward += reward1
        del state1, action1, next_state1, reward1, done1, exp_batch,
random_action
    no_op_counter = 0
    no_op_max = np.random.randint(4,30, dtype=int)
    for steps in range (10000):
        if no_op_counter < no_op_max:
            action = [0, 0.0]
            no_op_counter += 1
        else:
            action = agent.action_selection(AI.s_batch)
        Q_values.append(action[1])
        next_state, reward, done, lives = env.step(action[0])
        lives = lives['ale.lives']
        reward = np.sign(reward)
        if (lives < lives1):
            done == True
        total_reward += reward
        exp_batch = (image_processing(state), action,
image_processing(next_state), reward, done, lives)
        agent.four_batcher(exp_batch)
        agent.four_memory(exp_batch)
        state = next_state
        exp_batch = []
        step_counter += 1
        env.render()
        if i >= observation_episodes:
            agent.replay_sampler(AI.mem_batch)
            agent.training(AI.sample_batch)
```

```
            if step_counter % update_network == 0:
                agent.update_net()
            if 'Pong' in game:
                if done == True:
                  i_update += 1
                  episodes_sum += total_reward
                  episode_time = time.time()
                  if len(l10) < 10:
                      l10.append(total_reward)
                  if total_reward > max_reward:
                    max_reward = total_reward
                  else:
                    l10.popleft()
                    l10.append(total_reward)
                  l10_mean = np.mean(l10)
                  Q_values_mean = np.mean(Q_values)
                  Q_values = []
                  med_episodes = round((episodes_sum/((i+1)-
observation_episodes)),2)
                  h,m,s = elapsed_time(start_time, episode_time)
                  if med_episodes > med_sup:
                      med_sup = med_episodes
                  epsilon = AI.epsilon
                  loss_plt = AI.loss_plt
                  print_episode_final(i, episodes, steps, total_reward, epsilon,
loss_plt, h,m,s, max_reward, med_sup, med_episodes, l10_mean, Q_values_mean)
                  graph_plot(i, loss_plt, total_reward, med_episodes, l10_mean,
Q_values_mean, observation_episodes)
                  break
            else:
                if lives == 0:
                    i_update += 1
                    episodes_sum += total_reward
                    episode_time = time.time()
                    if len(l10) < 10:
                        l10.append(total_reward)
```

```
                    if total_reward > max_reward:
                        max_reward = total_reward
                    else:
                        l10.popleft()
                        l10.append(total_reward)
                    l10_mean = np.mean(l10)
                    Q_values_mean = np.mean(Q_values)
                    Q_values = []
                    med_episodes = round((episodes_sum/((i+1)-
observation_episodes)),2)
                    h,m,s = elapsed_time(start_time, episode_time)
                    if med_episodes > med_sup:
                        med_sup = med_episodes
                    print_episode_final(i, episodes, steps, total_reward,
epsilon, loss_plt, h,m,s, max_reward, med_sup, med_episodes, l10_mean,
Q_values_mean)
                    graph_plot(i, loss_plt, total_reward, med_episodes,
l10_mean, Q_values_mean, observation_episodes)
        if steps % 100 == 0:
            print_progress(i, episodes, steps, step_counter, total_reward)
        if 'Pong' in game:
            if done == True:
                break
        else:
            if lives == 0:
                break
```

6.3.3 Funciones accesorias

Describiremos a continuación las funciones accesorias que se han utilizado en el programa.

Se considerarán accesorias las funciones que no sean parte del agente (a pesar de que puedan estar contenidas en él) **pero sean necesarias o bien para su funcionamiento o para realizar ciertas funciones que nos ayuden a la correcta ejecución de nuestro agente.**

6.3.3.1 Image processing

Esta función procesa cada cuadro (*frame*) que genera el bucle de programa; el procedimiento es el siguiente:

1. **Realizamos un cambio de tamaño de 210x160 a 105x80**; esto se debe a que aunque el tamaño no es excesivamente grande, la cantidad de información a procesar sí lo es, y por tanto ocuparía muchísima memoria (recordemos que tenemos una variable llena de experiencias en la RAM de la que luego extraemos un número determinado para el *experience replay).* Mientras que en [4], así como en otras implementaciones ([183], [189]) se reduce su tamaño a 84x84 (en muchos casos debido a un recorte de la imagen, ya que en algunos entornos por ejemplo se muestra la puntuación, dato que al agente no le aporta información), se ha decidido mantener a la mitad exacta para mantener la proporcionalidad de la imagen, así como tener un mayor número de píxeles con los que contar a la hora de proporcionar información a la red neuronal.
2. **Color:** la información extraída del entorno está en color; **ya que no nos aporta información extra, se pasa a blanco y negro**; esto implica pasar de tener tres canales (105x80x3) a uno solo (105x80x1), reduciendo así su dimensión y cantidad de memoria necesaria para su almacenamiento, así como el tiempo de procesamiento en las redes neuronales convolucionales.
3. **Normalización:** por último, **se dividen los píxeles de la imagen entre 255.0**; con esta acción **normalizamos los valores de los píxeles entre 0 y 1**. Eliminar esta normalización provoca que los valores-Q se hagan muy grandes, pero no viene acompañado de una mejora en la puntuación.

Para realizar esta tarea, se ha utilizado la librería *OpenCV.* A continuación se muestra la función:

```
def image_processing(image):
    processed_image = cv2.resize(image, (105,80))
    processed_image = cv2.cvtColor(processed_image, cv2.COLOR_RGB2GRAY)
    processed_image = processed_image/255.0
    return processed_image
```

image es la variable que se le introduce a la función; esto es, una imagen extraída directamente del bucle de entrenamiento tras realizar un paso. Después se aplican los diferentes procesamientos (cambio de tamaño, cambio de color a blanco y negro y normalización).

6.3.3.2 Elapsed time

Esta función simplemente **calcula el tiempo de ejecución en horas, minutos y segundos**. Nos es útil para tener una referencia del tiempo que tarda el programa por episodio así como del tiempo de ejecución del programa. El código es el siguiente:

```
def elapsed_time(start_time, episode_time):
    execution_time = episode_time-start_time
```

```
    m, s = divmod(execution_time, 60)
    h, m = divmod(m, 60)
    h = int(h)
    m = int(m)
    s = int(s)
    return h, m, s
```

Se le introduce a la función las variables *start_time y episode_time*, que son la hora de inicio del programa y el tiempo de ejecución por episodio respectivamente. En la función, usamos *divmod* para retornar el tiempo en minutos y segundos primeramente; volviendo a dividir calculamos las horas.

6.3.3.3 Memory usage

Esta función nos muestra por pantalla el uso de la CPU y la memoria en uso; es útil para detectar problemas. Si bien el uso de la CPU debe de ser bajo (ya que procesamos con la GPU), el uso de la RAM si es algo a tener en cuenta: si se llena, el ordenador simplemente se parará. Se han utilizado las funcionalidades de la librería *psutil* de Python para comprobar el uso de CPU y memoria RAM.

```
def memory_usage():
    print("CPU usage: ", psutil.cpu_percent())
    print(psutil.virtual_memory())
```

6.3.3.4 Directory creation

Esta función se encarga de comprobar si existen las rutas de archivos necesarias para realizar los guardados correspondientes. Nuestro agente guardará cada 10 episodios los valores de los pesos de la red neuronal, el número de pasos, episodios y valor de epsilon. El motivo de este guardado es poder seguir entrenando en cualquier momento, ya que los entrenamientos son muy largos.

Para su correcto funcionamiento, se requiere de la librería *os*, incluida en el sistema.

```
def directory_creation(save_path, game):
    try:
        os.makedirs(save_path+game)
        print("Directory", save_path+game, "created.")
    except FileExistsError:
        print("Directory ", save_path+game, "already exists. Variables would be
loaded from there.")
```

La estructura *try except* en Python funciona de la siguiente manera: prueba a realizar la función que existe dentro de *try*; si falla, realiza lo siguiente (*except*). Se puede implementar la función solamente con *try* en caso de ser necesario.

6.3.3.5 Path check

Esta función se encarga de comprobar si existe fichero de guardado para las variables que creamos. Si no es así, lo crea. El formato utilizado es *hdf5*, para ello deberemos importar la librería correspondiente.

```
def path_check(save_filepath):
    if os.path.isfile(save_filepath) is False:
        savedata = h5py.File(save_filepath, 'w')
        savedata.close()
        print("Restoration point created.")
    else:
        print("Restoration point found.")
```

6.3.3.6 Load episodes

Esta función se encarga de comprobar si existe fichero con los episodios guardados; si es así, resta el numero de episodios ya realizados al total para continuar con los mismos.

Si no encuentra fichero, empieza por el numero de episodios estipulados.

```
def load_episodes(save_filepath, epidoses, i_update, episodes_start):
    episodes = 100010
    try:
        savedata = h5py.File(save_filepath, 'r')
        saved_i = savedata['i']
        i_update = saved_i[()]
        savedata.close()
        episodes = episodes-i_update
        print("\033[1;32;38mEpisodes sucessfully loaded")
    except:
        episodes = episodes_start
        savedata.close()
        print("\033[1;31;38mEpisodes could not be loaded. Starting at episode
0")
```

```
    return episodes
```

6.3.3.7 Load steps

Esta función se encarga de comprobar si se encuentra guardado en el fichero *hdf5* el número de pasos totales dados. Si es así, los carga; si no, comienza desde cero.

```
def load_steps(save_filepath):
    try:
        savedata = h5py.File(save_filepath, 'r')
        saved_steps = savedata['steps']
        step_counter = saved_steps[()]
        savedata.close()
        print("\033[1;32;38mSteps sucessfully loaded")
    except:
        step_counter = 0
        savedata.close()
        print("\033[1;31;38mSteps could not be loaded. Starting at step 0")
    return step_counter
```


6.3.3.8 Load

Esta función, contenida dentro del agente de inteligencia artificial, **nos permite cargar el archivo que contiene los pesos de las redes neuronales.** Lo cargaremos justo antes de iniciar el bucle de programa. Los pesos de las redes se guardan en un archivo de formato *hdf5*. Si no puede cargarlos, devuelve el mensaje de que no pueden cargarse esos pesos.

```
    def load(self):
        model = self.model
        try:
            model.load_weights(filepath)
            print("\033[1;32;38mModel weights sucessfully loaded")
            print("\033[1;31;38mALL SYSTEMS NOMINAL")
        except:
            print("\033[1;31;38mError: Model values cant be loaded\
033[1;37;38m")
```

6.3.3.9 Save

Esta función, también contenida dentro del agente, **se encarga del guardado de los pesos de la red neuronal, así como del guardado del valor de epsilon.** Epsilon se guarda con el resto de variables, mientras que para los pesos de las neuronas se crea otro archivo aparte, ambos en formato *hdf5*.

```
  def save(self, i, steps):
        model = self.model
        exploration_rate = AI.epsilon
        if episodes % 10 == 0:
            model.save(filepath)
            print("\033[1;37;38mModel saved")
            savedata = h5py.File(save_filepath, 'w')
            saved_epsilon = savedata.create_dataset('epsilon', data =
AI.epsilon)
            saved_i = savedata.create_dataset('i', data = i_update)
            saved_steps = savedata.create_dataset('steps', data = step_counter)
            savedata.close()
        return exploration_rate, saved_epsilon, saved_i, saved_steps
```

6.3.3.10 Four batcher

Esta función toma una experiencia tras otra, extrae el estado siguiente (*next_state*) y forma un paquete de 4, el cual manda a la función *batcher*.

Una experiencia está formada por el siguiente conjunto: estado, acción correspondiente a ese estado, estado que genera esa acción (estado siguiente), recompensa en ese instante, si ha terminado o no la partida (*done*), y el número de vidas. En el programa aparece como:

```
exp_batch = (state, action, next_state, reward, done, lives)
```

Cuando la variable *four_batch_state* está llena, la siguiente experiencia que entra elimina automáticamente la experiencia más antigua, de forma que el paquete que se le pasa a la función *batcher* siempre contiene las cuatro últimas experiencias consecutivas.

```
    def four_batcher(self, exp_batch):
        AI.four_batch_state.append(exp_batch[2])
        if len(AI.four_batch_state) == AI.four_batch_size:
            AI.batcher(AI.four_batch_state)
```

6.3.3.11 Batcher

Esta función, escrita dentro del agente de inteligencia artificial, simplemente **transforma el paquete de cuatro experiencias creado por *four batcher***, dándole el formato adecuado para que nuestra red neuronal pueda procesarlo.

```
def batcher(four_batch_state):
    AI.s_batch = []
    AI.s_batch = np.asarray(AI.four_batch_state)
    AI.s_batch.shape = AI.batch_shape
```

6.3.3.12 Four memory

Esta función, contenida dentro del agente de IA, **recoge experiencias, separa sus elementos y los agrupa de cuatro en cuatro en un paquete** (*four_memory_batch)*. Es decir, tendríamos cuatro estados, cuatro acciones, cuatro estados próximos, etc, todos consecutivos. Cuando se agrupan cuatro, se llama a la función de memoria (*AI.memory)*.

```
def four_memory(self, exp_batch):
    AI.four_memory_state.append(exp_batch[0])
    AI.four_memory_action.append(exp_batch[1])
    AI.four_memory_next_state.append(exp_batch[2])
    AI.four_memory_reward.append(exp_batch[3])
    AI.four_memory_done.append(exp_batch[4])
    AI.four_memory_lives.append(exp_batch[5])
    if len(AI.four_memory_state) == AI.four_batch_size:
        AI.four_memory_batch.append((AI.four_memory_state,
AI.four_memory_action,AI.four_memory_next_state,AI.four_memory_reward,AI.four_me
mory_done,AI.four_memory_lives))
        AI.memory(AI.four_memory_batch)
```

6.3.4 Funciones métricas

En este apartado **se expondrán las funciones relativas a la toma de medidas utilizadas para evaluar el rendimiento y progreso del agente** de inteligencia artificial creado para este trabajo fin de máster.

6.3.4.1 Print episode final

Esta función se muestra por pantalla cuando termina el episodio, y nos da la siguiente información:

- Episodio en el que estamos/Episodios restantes
- Pasos totales durante todo el entrenamiento
- Puntuación total del episodio
- Valor de epsilon al finalizar el episodio
- Tiempo total de ejecución del programa
- Puntuación máxima durante la ejecución de esta sesión de entrenamiento
- Media máxima alcanzada durante esta sesión de entrenamiento
- Media actual
- Media de las últimas 10 partidas
- Valores medios-Q de este episodio

```
def print_episode_final(i, episodes, steps, total_reward, epsilon, loss_plt, h,m,s, max_reward, med_sup, med_episodes, l10_mean, Q_values_mean):
    print("\033[1;32;38mEpisode: {}/{}, Total steps: {}, Total Score: {}, e: {:.4f}, loss: {:.6f}, Execution time: {}:{}:{}"
                        .format(i, episodes, steps, total_reward, epsilon, loss_plt, h,m,s))
    print("\033[1;32;38mMax score: {:.2f}, Maximum mean: {:.2f}, Actual mean: {:.2f}, L10 mean: {:.2f}, Q-values mean: {:.6f}" .format(max_reward, med_sup, med_episodes, l10_mean, Q_values_mean))
```

6.3.4.2 Print progress

Esta función es llamada cada 100 pasos (cada 100 estados), **proporcionando información sobre como está progresando ese episodio. La información que proporciona es la siguiente:**

- Episodio en el que estamos/episodios restantes
- Estados vistos durante este episodio
- Pasos totales
- Puntuación conseguida hasta ese momento

```
def print_progress(i, episodes, steps, step_counter, total_reward):
    print("\033[1;37;38mEpisode: {}/{}, Steps: {}, Total steps: {}, Score: {}".format(i, episodes, steps, step_counter, total_reward))
```

6.3.4.3 Graph plot

Esta función es la encargada de mostrar las gráficas, las cuales se realizan utilizando la librería matplotlib. Encontramos tres gráficas:

1. **La superior, muestra el *loss***, el cual nos sirve para comprobar como está funcionando nuestra red neuronal.
2. **En la segunda encontramos tres valores:** en rojo, **la puntuación obtenida durante los episodios**; en verde, **la media total acumulada**, y en amarillo **la media de las diez últimas partidas**.
3. **En la gráfica inferior, encontramos los valores-Q medios**, los cuales nos sirven para hacernos una idea de lo bien está funcionando nuestro algoritmo.

```
def graph_plot(i, loss_plt, total_reward, med_episodes, l10_mean, Q_values_mean, observation_episodes):
    plt.ion()
    plt.subplot(3,1,1)
    plt.plot(i, loss_plt, 'b*')
    plt.title('Loss, reward and Q-values vs episodes')
    plt.ylabel('Loss', fontsize=12)
    plt.subplot(3,1,2)
    plt.plot(i, total_reward, 'ro', label = 'Reward')
    plt.plot(i, med_episodes, 'g.', label = 'Mean')
    plt.plot(i, l10_mean, 'y.', label = 'L10 Mean')
    if i == observation_episodes:
        plt.legend(loc='upper right', bbox_to_anchor=(1.14,0.30), prop={'size':5.5})
    plt.xlabel('Episodes', fontsize=12)
    plt.ylabel('Reward', fontsize=12)
    plt.subplot(3,1,3)
    plt.plot(i, Q_values_mean, '.m')
    plt.xlabel('Episodes', fontsize=12)
    plt.ylabel('Q-values', fontsize=12)
    plt.pause(0.05)
    plt.show()
```

6.3.4.4 Deimos_v3_x_google_colab.py

Este código contiene las mismas funciones básicas que el programa principal, con las salvedades de que todas sus funciones están contenidas en un fichero, ya que este programa es el que utilizamos para ejecutar en la nube.

Este programa **no genera por sí mismo gráficas**, ya que no podemos obtener gráficas dinámicas en tiempo real cuando ejecutamos en la nube; asimismo, tampoco podemos mostrar la partida, por lo que estas funcionalidades se eliminan.

Puesto que aún necesitamos los datos para generar las gráficas a posteriori, **se crea la función *save_graph*, la cual es la encargada de guardar las diferentes variables que necesitamos para generar las gráficas.** Se muestra la función a continuación:

```
def save_graph(graph_filepath, l10_mean, max_reward, med_sup, med_episodes,
q_mean_list, reward_list, i, loss_list):
    savefile = h5py.File(graph_filepath, 'w')
    saved_l10 = savefile.create_dataset('l10', data = l10_list)
    saved_max_score = savefile.create_dataset('max_score', data = max_reward)
    saved_max_mean = savefile.create_dataset('max_mean', data = med_sup)
    saved_actual_mean = savefile.create_dataset('med_episodes', data =
mean_list)
    saved_q_mean = savefile.create_dataset('q_mean', data = q_mean_list)
    saved_reward = savefile.create_dataset('reward', data = reward_list)
    saved_episodes = savefile.create_dataset('i', data = i)
    saved_loss = savefile.create_dataset('loss_list', data = loss_list)
    savefile.close()
    return saved_l10, saved_max_score, saved_max_mean, saved_actual_mean,
saved_q_mean, saved_reward, saved_episodes, saved_loss
```

6.3.4.4 Deimos_v3_x_google_colab_graph.py

Esta función, la cual compone en su integridad el archivo *Deimos_v3_x_google_colab_graph.py* **se encarga de crear los gráficos** usando los datos obtenidos por *Deimos_v3_x_google_colab.py* a**l ejecutarse una sesión en Google Colaboratory. Es importante recordar que debemos utilizar este programa tras cada sesión de entrenamiento.**

```
import matplotlib.pyplot as plt
import h5py


game = "BreakoutDeterministic-v4"
save_path = "Save/"
graph_savefile = "/Backup_google_colab/graph_save.hdf5"
```

```
graph_filepath = save_path+game+graph_savefile
saved_l10 = []
saved_max_score = 0.0
saved_max_mean = 0
saved_actual_mean = 0
saved_q_mean = 0
saved_reward = 0
saved_episodes = 0
saved_loss = []
i = []
try:
    savefile = h5py.File(graph_filepath, 'r')
    saved_l10 = savefile['l10']
    saved_max_score = savefile['max_score']
    saved_max_mean = savefile['max_mean']
    saved_actual_mean = savefile['med_episodes']
    saved_q_mean = savefile['q_mean']
    saved_reward = savefile['reward']
    saved_episodes = savefile['i']
    saved_loss = savefile['loss_list']
except:
    print("Data cannot be loaded")

max_score = saved_max_score[()]
max_mean = saved_max_mean[()]
Q_values_mean = list(saved_actual_mean[()])
total_reward = list(saved_reward[()])
episodes = saved_episodes[()]-49
loss_plt = list(saved_loss[()])
med_episodes = list(saved_actual_mean[()])
l10_mean = list(saved_l10[()])
savefile.close()
for x in range(episodes):
    i.append(x)
```

```
plt.ion()
plt.subplot(3,1,1)
plt.plot(i, loss_plt, 'b*')
plt.title('Loss, reward and Q-values vs episodes')
plt.ylabel('Loss', fontsize=12)
plt.subplot(3,1,2)
plt.plot(i, total_reward, 'ro', label = 'Reward')
plt.plot(i, med_episodes, 'g.', label = 'Mean')
plt.plot(i, l10_mean, 'y.', label = 'L10 Mean')
plt.legend(loc='upper right', bbox_to_anchor=(1.14,0.30), prop={'size':5.5})
plt.xlabel('Episodes', fontsize=12)
plt.ylabel('Reward', fontsize=12)
plt.subplot(3,1,3)
plt.plot(i, Q_values_mean, '.m')
plt.xlabel('Episodes', fontsize=12)
plt.ylabel('Q-values', fontsize=12)
plt.show()
```


6.3.4.5 Deimos_v3_x_random_test.py

Este programa se encarga de realizar los tests aleatorios correspondientes, que luego utilizaremos para comparar con los resultados obtenidos por el agente. El código no se comenta aquí ya que es muy similar a otras partes del código principal, pero se adjunta en los anexos.

6.3.4.6 Deimos_v3_x_agent_test.py

Este programa se encarga de realizar los tests para la evaluación del rendimiento del agente. Para ello toma la parte del programa principal correspondiente a la toma de decisiones, y evalúa durante 200 episodios su rendimiento, generando las gráficas correspondientes. Puesto que el código es muy similar al del programa principal ya que deriva del mismo, no se adjunta aquí; el código completo puede encontrarse en los anexos.

6.3.5 Métricas utilizadas

Para la medición del rendimiento de nuestro agente, **se han tomado como referencia las siguientes medidas:**

- Puntuación obtenida por cada episodio o partida.
- Media total de todas las partidas por sesión de entrenamiento.
- Media de las últimas diez partidas.
- *Q-values.*
- *Loss.*
- Número de pasos por episodio.
- Número de pasos por sesión de entrenamiento.
- Número de pasos visualizados totales.
- Tiempo de entrenamiento por sesión.
- Tiempo total de entrenamiento.
- Puntuación máxima por sesión de entrenamiento.
- Puntuación media máxima por sesión de entrenamiento.
- Episodios por sesión de entrenamiento.
- Episodios totales.

Pasaremos ahora a comentar todos estos puntos, uno por uno, así como su relación entre ellos en los casos en la que las haya.

La **puntuación obtenida por cada episodio o partida** nos **indica la evolución real de nuestro agente**; es, probablemente, el **parámetro más importante**, ya que, **mediante la gráfica podemos ver lo bien o mal que se desenvuelve nuestro agente**, y si va obteniendo mejores resultados o no. A pesar de que la evolución no siempre es constante (encontramos que las puntuaciones a veces son fluctuantes), sí se puede apreciar una evolución o estancamiento en la mayoría de los casos.

La **media total de todas las partidas por sesión de entrenamiento** nos da una medida relativa de si realmente está mejorando o no nuestro agente; ya que lo graficamos, podemos ver la tendencia de la media, si es ascendente, descendente o tiende a estancarse.

El principal problema de la media es que los valores extremos le afectan mucho cuando la cantidad de partidas es pequeña, y muy poco cuando la cantidad de partidas es grande. Por ejemplo:

Si llevamos 10 partidas con una media de 150 puntos en *Space Invaders* y en la partida número 11 hacemos una puntuación de 30, la puntuación descenderá mucho más que si, con la misma media y la misma puntuación, el número de partidas es de 300.

Debido a esto, se añade la **media de las diez últimas partidas**; **con esta medida podemos tener una visión más clara de la tendencia de nuestro agente, ya que mantenemos esa sensibilidad**

de la media al ser un número reducido de partidas. Lo que se hace es comparar las dos curvas: si la media de las diez últimas partidas está por encima de la media, indica que nuestro agente está mejorando; si está por debajo, que está empeorando. Normalmente la media L10 (así es como se ha llamado, por *last 10*) oscila entre la media cuando el agente está en un período estable.

Con los ***Q-values*** tenemos una idea de la estimación que está realizando el algoritmo; esta medida **recoge la media del valor de todos los valores-Q tomados por episodio**, con lo cual nos podemos hacer una idea de la esperanza de obtener una puntuación que tiene nuestro agente. **El valor-Q es relativo, es decir, no refleja la puntuación que espera obtener**; es, si recordamos de apartados anteriores de este trabajo, un valor que indica la calidad de una acción respecto al resto de acciones en un momento dado. ¿Para qué podemos usarlo entonces? **Lo usaremos para comparar entre episodios como es esta estimación; si el *Q-value* aumenta**, (a pesar de que en el Q-Learning el agente tiende a sobreestimar), **quiere decir que el agente tiene una mayor confianza en obtener una buena puntuación.**

El ***loss*** nos **indica lo bien o mal que está funcionando la red neuronal, ya que mide la diferencia entre las experiencias que le introducimos y los *Q-values* calculados.** Un *loss* bajo indica un buen funcionamiento del algoritmo, aunque puede fluctuar.

El **numero de pasos por episodio** nos indica lo largo o corto que ha sido ese episodio. **Un episodio largo normalmente indica que el agente está aprendiendo o que está realizando buenas acciones en ese episodio.** Cada paso se considera como una tupla (s, a, s+1, r).

El **número de pasos por sesión de entrenamiento nos da una medida relativa, al compararlo con el número episodios y con el tiempo de entrenamiento por sesión, de lo eficiente que está siendo nuestro agente y de la velocidad de ejecución del mismo**: si realiza pocos episodios pero en cambio estos son más largos, es un indicador de que probablemente nuestro agente esté aprendiendo. Asimismo, si lo comparamos con el tiempo, podremos saber lo rápido o lento que se ejecuta el agente en nuestro ordenador.

En cuanto al **número de pasos visualizados totales**, es un indicador, **comparándolo con la puntuación, de lo bien o mal que evoluciona nuestro agente**: si la puntuación no evoluciona conforme vamos avanzando, es un indicador de que algo no va bien. Se estima que un agente necesita más de 4 millones de pasos para empezar a aprender en muchos casos.

El **tiempo de entrenamiento por sesión nos da**, comparándolo con el número de episodios, **una medida de lo rápido o lento que se ejecuta el agente.**

El **tiempo total de entrenamiento es la suma de todos los tiempos de entrenamiento por sesión, y nos indica cuanto tiempo está tardando el agente en aprender.** Depende en gran medida de la potencia de procesamiento (GPU) que estemos utilizando.

La **puntuación máxima por sesión de entrenamiento nos indica cuál ha sido la mejor partida realizada en esa sesión**; en algunos casos puede servir para ver si evoluciona el agente correctamente, ya que podemos comparar la puntuación actual con la máxima, o incluso las medias, y hacernos una idea de lo bien o mal que está evolucionando nuestro agente.

La **puntuación media máxima por sesión de entrenamiento nos indica cuál ha sido la media máxima alcanzada en esa sesión.** Puede servirnos para ver si la media está aumentando, disminuyendo, o si se mantiene en unos valores estables.

El número de **episodios por sesión de entrenamiento** nos **indica cuantos episodios ha visto el agente en esa sesión.** Indica lo rápido o lento que está yendo nuestro agente, al compararlo con el tiempo de entrenamiento por sesión.

Por último, el número de **episodios totales** nos **indica el total de episodios de entrenamiento que nuestro agente ha visto.**

7 Experimentos y resultados obtenidos

En esta sección se explicarán los experimentos realizados, así como los resultados que se han obtenido de los mismos, comentando los mismos.

7.1 Construcción del experimento

El experimento **se ha realizado en varias sesiones**, ya que los tiempos de aprendizaje de nuestro agente son muy largos; en la duración influye en gran medida, como ya se ha comentado previamente, el hardware utilizado, concretamente la GPU.

Para realizar experimentos ajustando distintos hiperparámetros, **se ha utilizado la plataforma *google colaboratory*, de esta forma se ha podido entrenar el mismo agente en el mismo entorno con ajustes distintos**, para comprobar como afecta la variación de distintos hiperparámetros al mismo.

Al comienzo del entrenamiento, tal y como se explica en [4], las acciones que toma nuestro agente son aleatorias; tras un período de observación y durante un número determinado de pasos, la posibilidad de realizar una acción aleatoria desciende linealmente hasta un valor previamente establecido. A partir de este valor, se entiende que el agente tiene el control total.

El objetivo del experimento es alcanzar la máxima puntuación posible en diferentes entornos de *OpenAI-Gym*, concretamente en varios videojuegos de la videoconsola Atari 2600.

El experimento **constará de dos partes:**

- **Una parte de entrenamiento**, en la que aproximadamente se entrena el agente durante unos 10 millones de pasos;
- **Una parte de test**, en la que se evalúa lo aprendido en el entrenamiento, comparando lo aprendido en el entrenamiento durante 200 episodios, comparándolo con 200 episodios realizados aleatoriamente.

7.2 Entornos, hiperparámetros utilizados y resultados

En este apartado se comentarán los distintos entornos y el objetivo del mismo, así como los hiperparámetros que se han utilizado en los experimentos. Finalmente, se comentarán los resultados obtenidos.

7.2.1 Pong

El objetivo de Pong es hacer pasar la bola tras la barra del contrario 21 veces; nuestro agente controlará la barra de la derecha (de color verde).

Figura 74: Captura de pantalla de Pong durante una prueba del agente aleatorio

7.2.1.1 Experimento 1

Los hiperparámetros utilizados son los siguientes:

- Episodios máximos de entrenamiento: 100010
- Episodios de observación: 50
- Pasos para actualizar la red objetivo: 10000
- Tamaño de memoria: 100000
- Número de experiencias para *experience replay*: 32
- Ratio de aprendizaje: 0.00001
- Cuadros hasta epsilon mínimo: 300000

- Epsilon final: 0.1
- Gamma: 0.99

Los **resultados del entrenamiento** se adjuntan en la tabla siguiente:

Fecha	Puntuación máx	Media máx	Media a L10	Media final	Loss	Media Q-Values	Tiempo sesión	Tiempo total	N.º episodios sesión	Episodios totales	Experiencias sesión	Experiencias totales
12/04/19	0	No recogido	-20,5	-20,59	0,07526	0,084928	21:11:16	21h 11min 16s	500	500	440988	440988
13/04/19	0	0	-20,7	-20,76	0,008157	0,002920	17:14:50	38h 26min 06s	420	920	363441	804429
16/04/19	0	0	-20,7	-20,71	0,007191	0,370592	21:18:14	59h 44min 47s	500	1420	437371	1241800
16/04/19	0	0	-20,9	-20,63	0,006045	0,168321	18:56:04	78h 40min 53s	440	1860	385191	1626991
17/04/19	0	0	-20,9	-20,69	0,006396	0,265422	17:36:02	96h 26min 55s	420	2280	367069	1994060
17/04/19	0	0	-20,6	-20,65	0,007242	0,045825	10:33:16	106h 59min 12s	270	2550	237874	2231934
13/04/19	0	0	-20,9	-20,67	0,004894	0,036527	10:05:55	117h 05min 07s	260	2810	228122	2460056
20/04/19	0	0	-21	-20,72	0,004896	0,194362	16:42:38	133h 47min 45s	400	3210	348391	2808447
22/04/19	0	0	-20,3	-20,74	0,006604	0,074004	08:55:14	142h 42min 59s	240	3450	208682	3017129
22/04/19	0	0	-20,6	-20,68	0,006510	0,017185	02:23:13	145h 06min 12s	100	3550	88893	3106022
23/04/19	0	0	-21	-20,87	0,006242	0,233599	15:48:07	169h 54min 19s	770	4320	629344	3735366

25/04/19	-17	-20,82	-20,6	-20,85	0,006386	0,145043	16:28:24	186h 12min 43s	820	5140	668793	4404159
26/04/19	-17	-20,73	-20,9	-20,84	0,005729	0,231674	17:13:29	203h 26min 12s	860	6000	700316	5104475
27/04/19	-16	-20,5	-21	-20,85	0,005754	0,157297	15:47:54	219h 14min 06s	780	6780	636674	5741189
30/04/19	-17	-20,75	-20,6	-20,83	0,007229	0,147130	21:04:10	240h 18min 16s	1020	7800	836213	6577362
30/04/19	-17	-19,75	-20,8	-20,68	0,006241	0,140304	19:25:30	259h 43min 46s	900	8700	774107	7351469
01/05/19	-17	-20,48	-20,7	-20,71	0,005646	0,168276	16:29:07	276h 12min 53s	770	9470	669907	8021376
02/05/19	-17	-20	-20,5	-20,65	0,005628	0,241871	18:26:51	294h 39min 44s	840	10310	730547	8751923
03/05/19	-18	-20,29	-20,8	-20,61	0,005903	0,319593	16:23:53	311h 03min 37s	750	11060	657621	9409544
05/05/19	-17	-20,61	-20,8	-20,64	0,010000	0,230000	20:00:40	331h 04min 17s	910	11970	805811	10215355

Tabla 2: Resultados sesiones entrenamiento Pong

Análisis de los resultados del entrenamiento: antes de empezar el análisis como tal, mencionar que las casillas de puntuación máxima y media máxima donde aparece el comentario "No recogido" es debido a que el programa en un principio inicializaba esos valores a 0; puesto que los valores obtenidos no superaban ese valor (cada vez que el oponente nos marcaba un punto, internamente lo contabilizaba como -1), el valor seguía siendo 0; posteriormente se modificó esa parte del programa para adecuarlo a esta peculiaridad de *Pong*. Esta modificación no afecta al funcionamiento interno del programa, solamente a la recogida de datos.

Vemos que **la puntuación máxima en ningún momento supera los valores positivos; lo más cercano es -16**, esto es, 5 puntos marcados al adversario (recordemos que la puntuación máxima es 21; si el adversario gana la partida, son -21 puntos para nosotros; sumando 5 a este resultado, obtenemos -16).

En cuanto a la **media máxima**, a pesar de no ser un valor especialmente significativo, encontramos que es bastante baja; la media máxima **apenas supera los 20** (-19,75).

La media final, valor el cual sí es importante, **es bastante pobre**: se mantiene entre -21 (puntuación mínima) y -20, siendo su máximo -20,59, obtenido en la primera sesión.

Los valores del *loss* se mantienen cercanos a cero, lo cual indica el buen funcionamiento interno de las redes neuronales.

Los *Q-values* son bajos, lo que indica que el agente no es capaz de encontrar una estrategia adecuada y adaptarse correctamente a este entorno.

El total del entrenamiento ha constado de 331 horas, 4 minutos y 17 segundos, repartido en 20 sesiones de entrenamiento; el máximo de pasos realizados ha sido de 10215355.

Este experimento fue realizado íntegramente con 2 gráficas Nvidia GTX1050Ti OC hasta la sesión del 22/04/19; a partir de la siguiente sesión (23/04/19) se realizó con una Nvidia RTX2070. Se aprecia el aumento de rendimiento en el número de pasos. Si comparamos por ejemplo la sesión del 20/04/19, de una duración de 16 horas, 42 minutos y 38 segundos, obtenemos un número de pasos dados durante ese tiempo de 348391; en la primera sesión con la nueva gráfica, de 15 horas, 48 minutos y 7 segundos, se realizaron un total de 629344 pasos. Siendo la diferencia de una hora aproximadamente entre las dos sesiones, se observa un incremento de casi el doble de pasos dados; esto implica un menor número de sesiones para alcanzar el objetivo previsto.

Se adjuntan las gráficas de las sesiones de entrenamiento; en la gráfica superior, en azul, aparece el *loss*; en la gráfica intermedia, en rojo, la puntuación máxima por episodio; en amarillo la media L10; en verde, la media total. En la gráfica inferior, los *Q-values* medios por episodio. En las primeras gráficas no existe leyenda debido a que se implementó después. Esto tampoco afecta al funcionamiento de nuestro agente, solamente a la visualización de los datos recogidos en las gráficas.

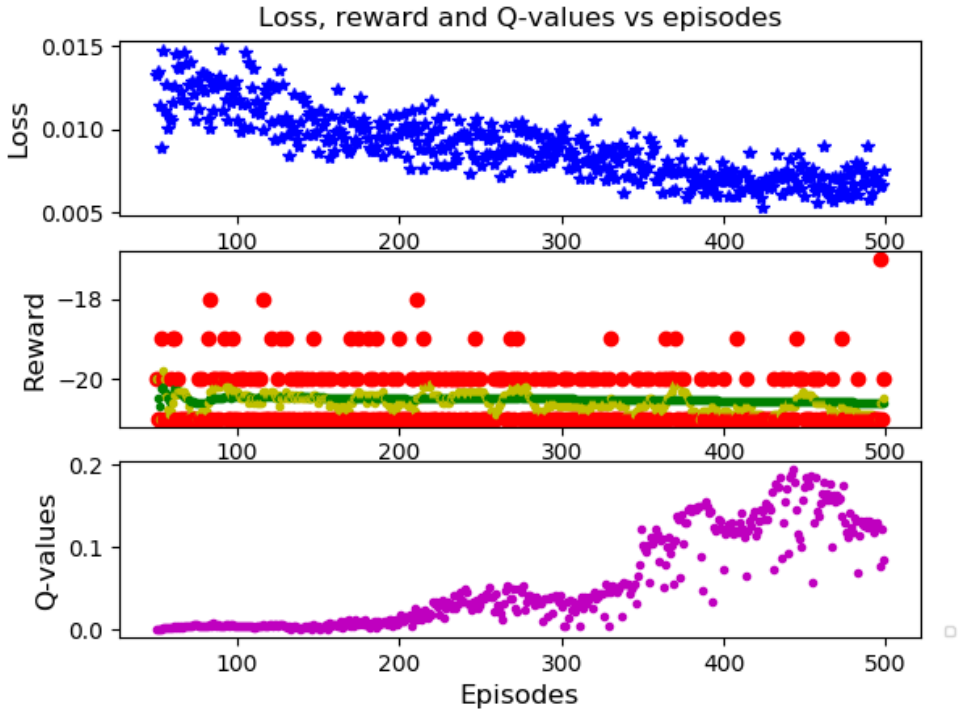

Figura 75: Gráfica PongDeterministic-v4, sesión 1

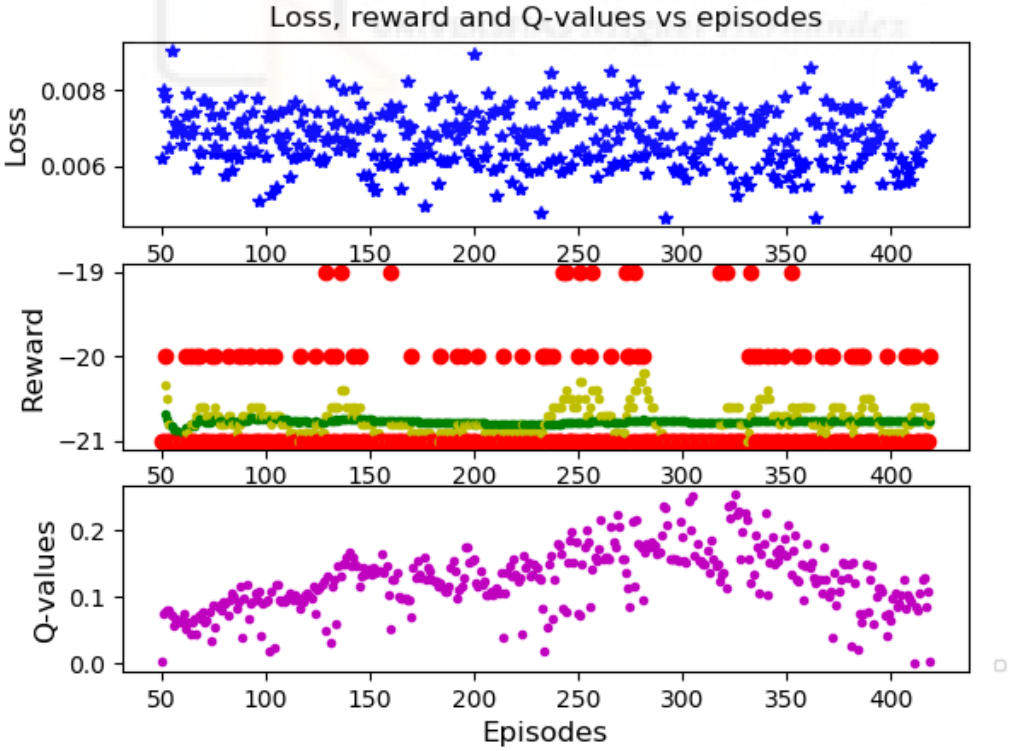

Figura 76: Gráfica PongDeterministic-v4, sesión 2

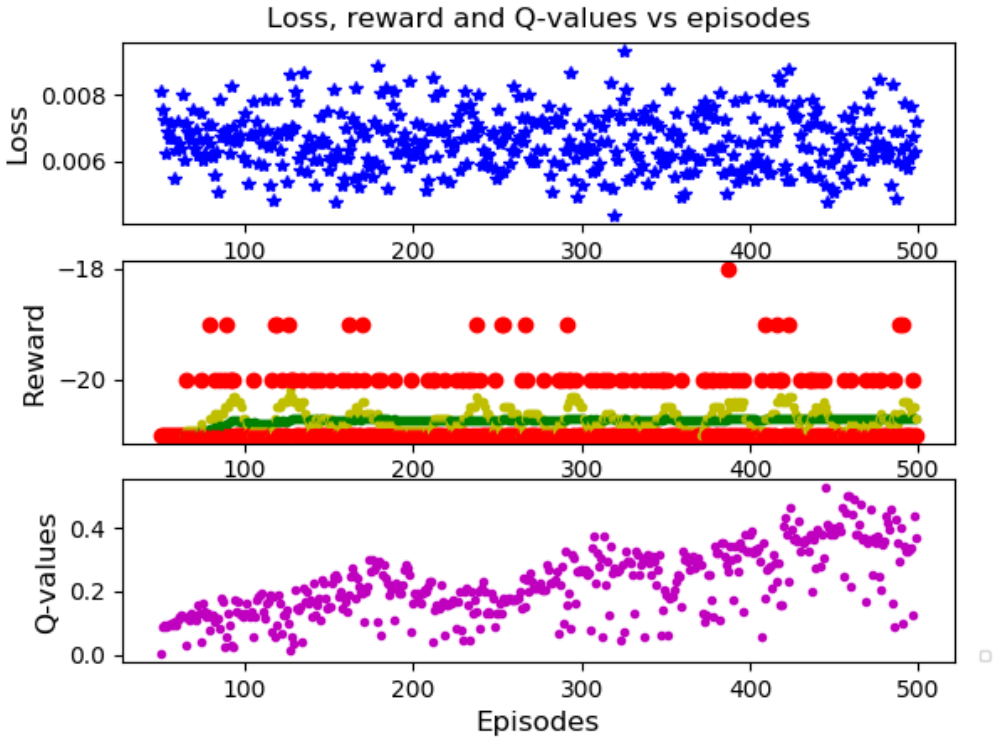

Figura 77: Gráfica PongDeterministic-v4, sesión 3

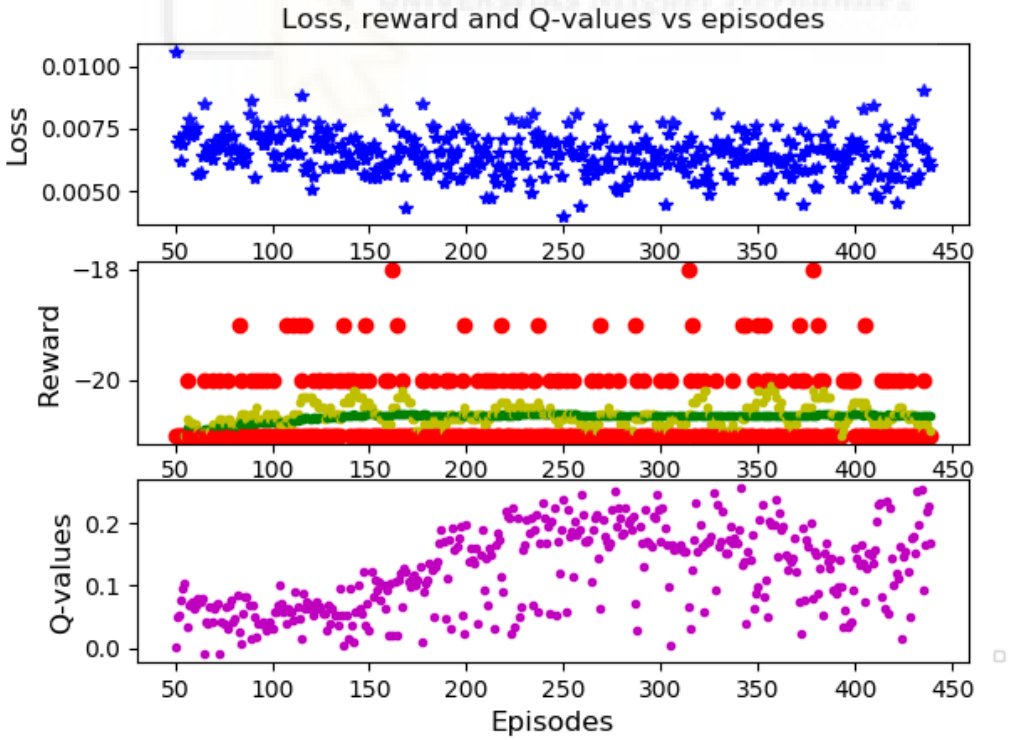

Figura 78: Gráfica PongDeterministic-v4, sesión 4

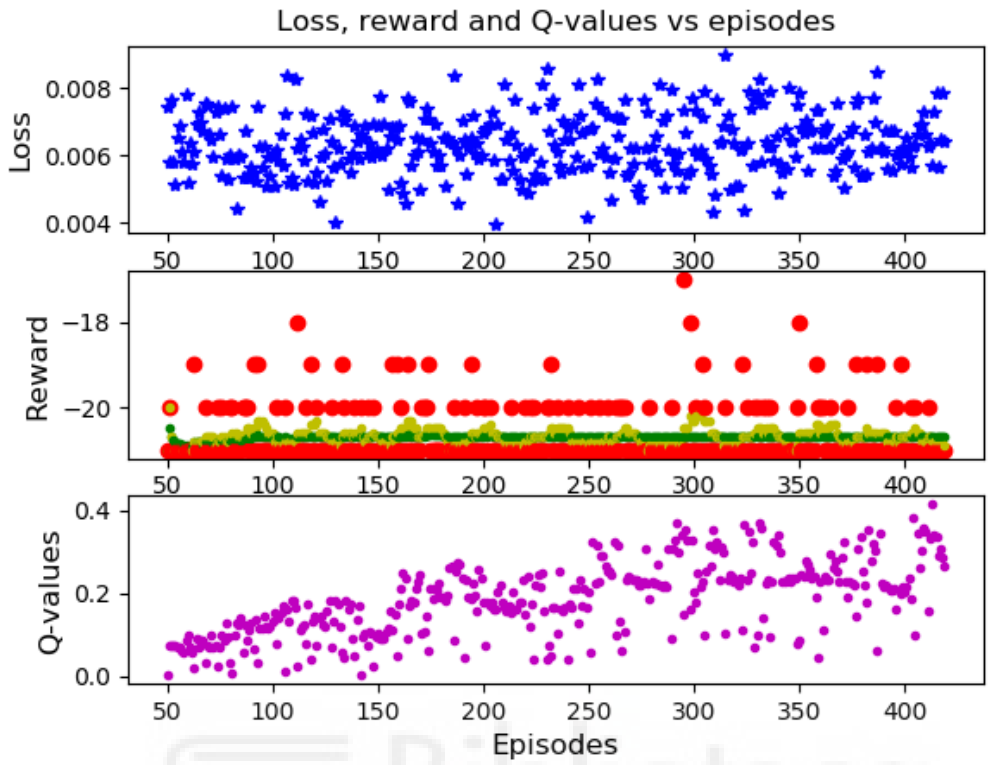

Figura 79: Gráfica PongDeterministic-v4, sesión 5

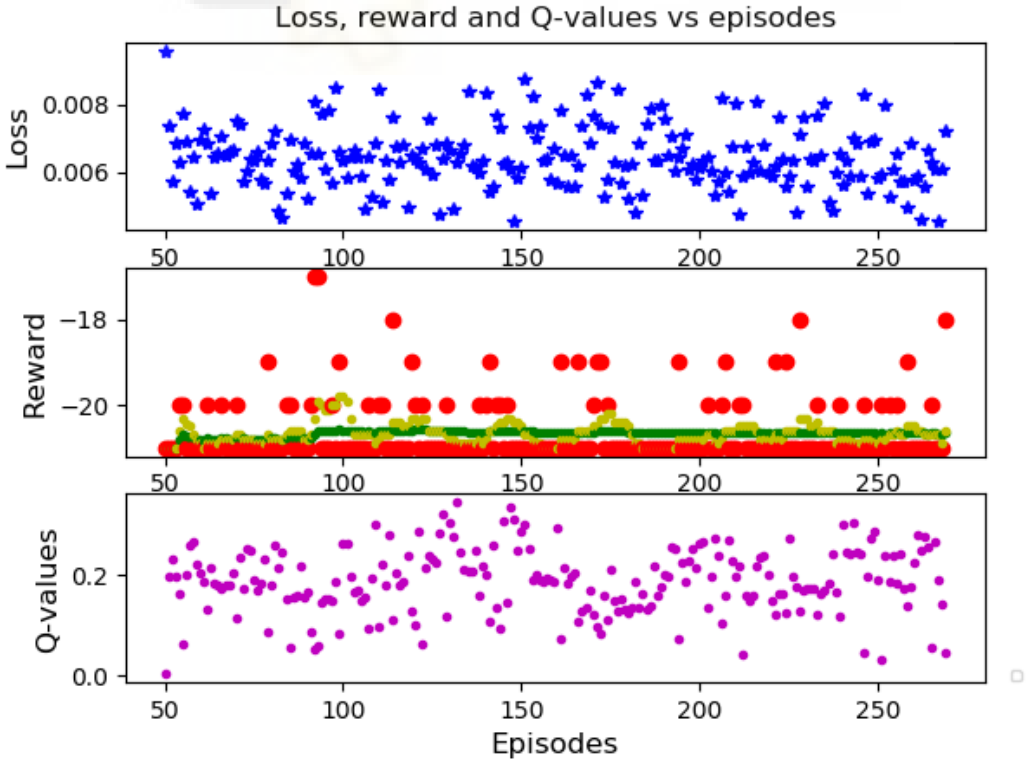

Figura 80: Gráfica PongDeterministic-v4, sesión 6

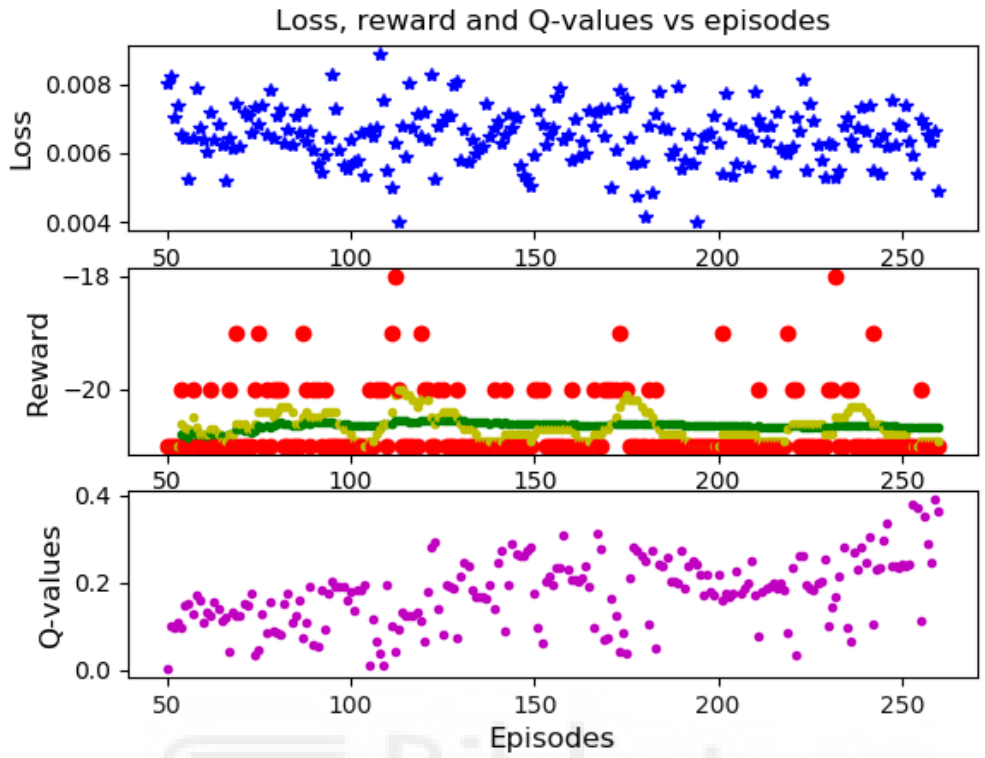

Figura 81: Gráfica PongDeterministic-v4, sesión 7

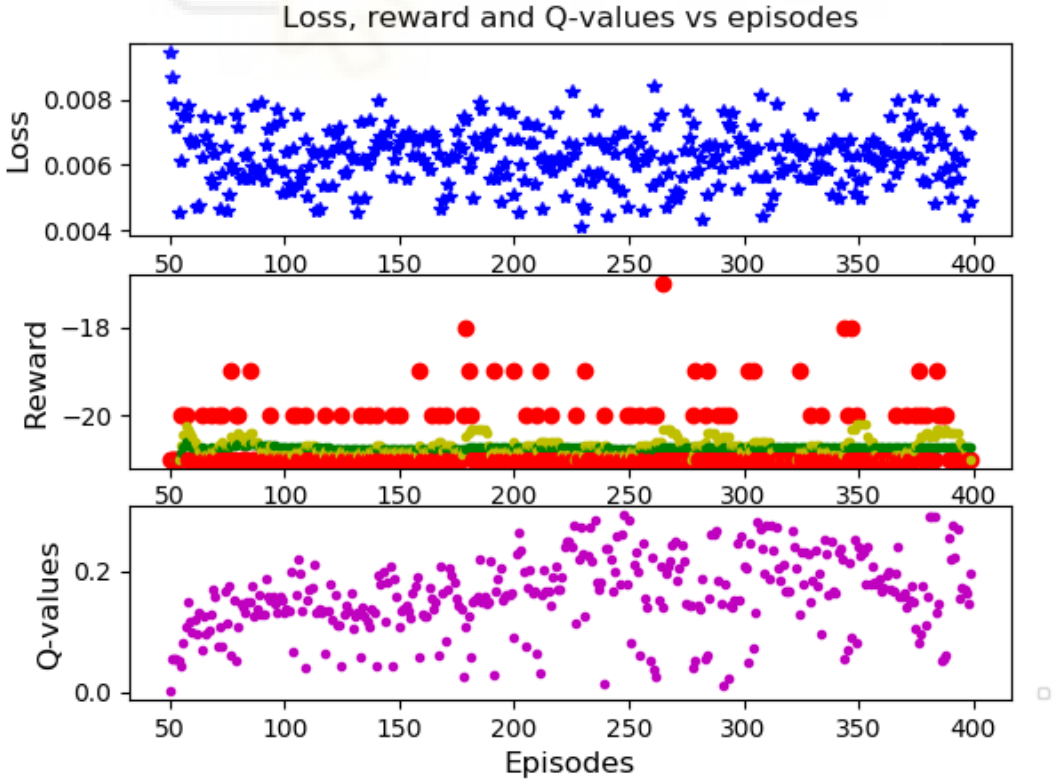

Figura 82: Gráfica PongDeterministic-v4, sesión 8

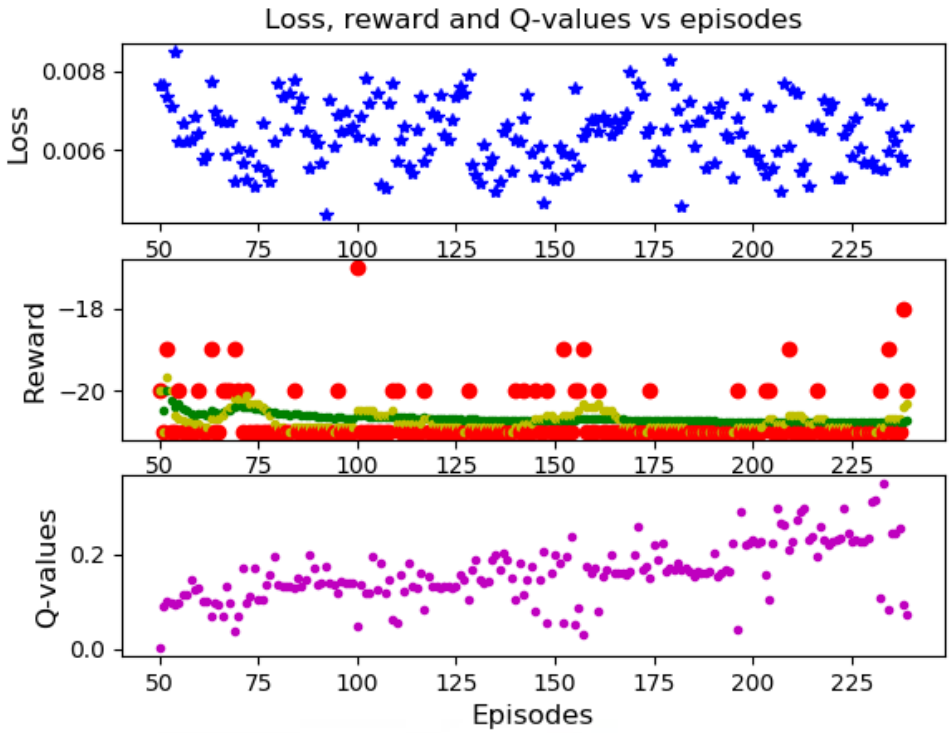

Figura 83: Gráfica PongDeterministic-v4, sesión 9

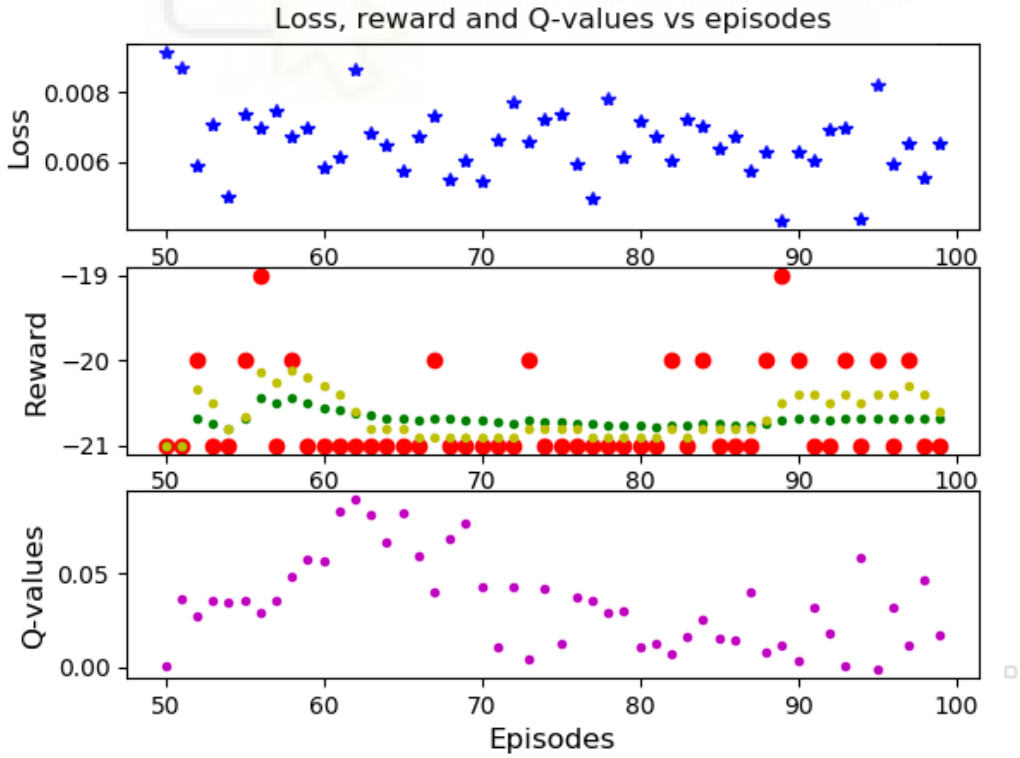

Figura 84: Gráfica PongDeterministic-v4, sesión 10

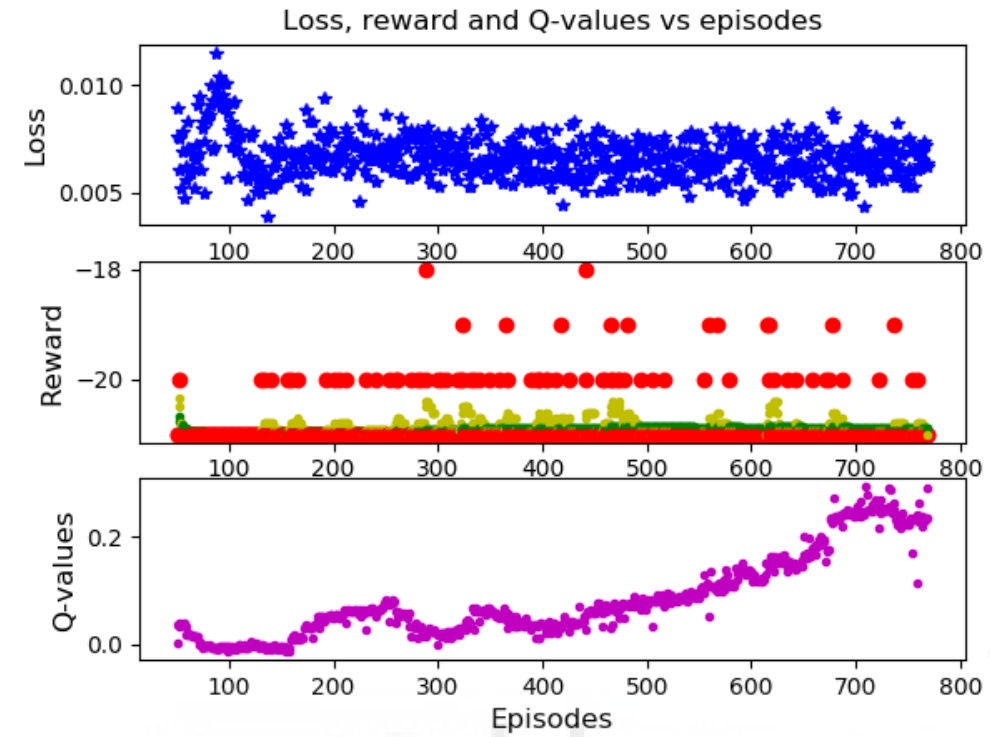

Figura 85: Gráfica PongDeterministic-v4, sesión 11

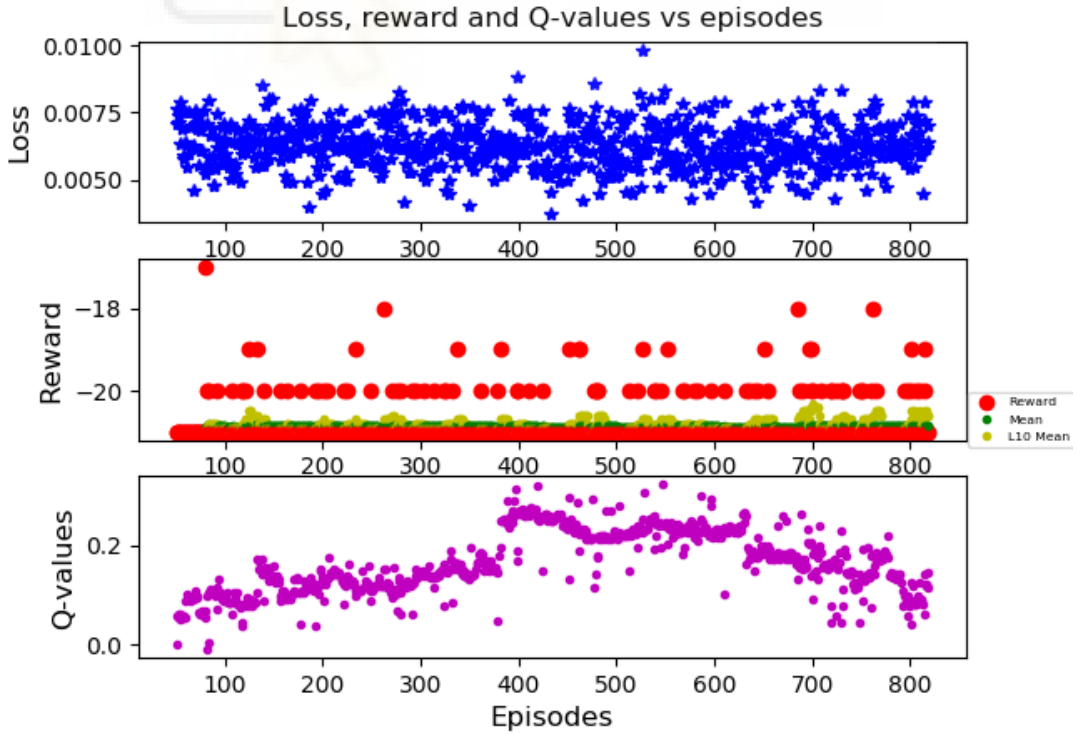

Figura 86: Gráfica PongDeterministic-v4, sesión 12

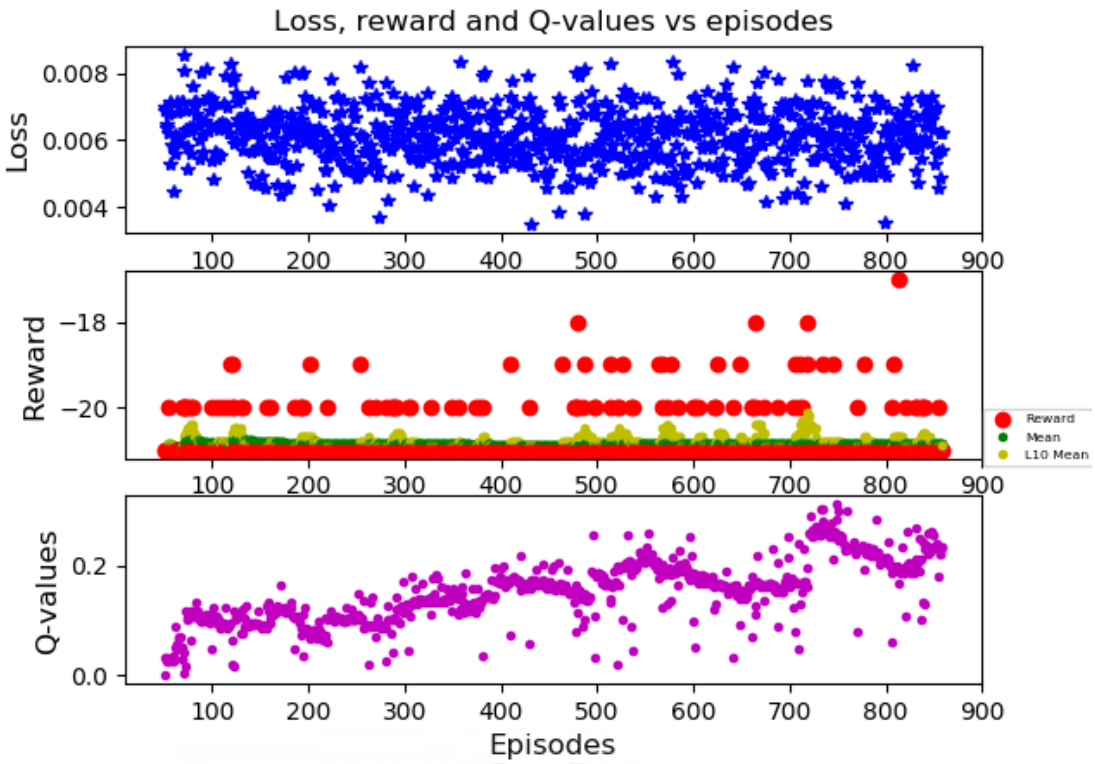

Figura 87: Gráfica PongDeterministic-v4, sesión 13

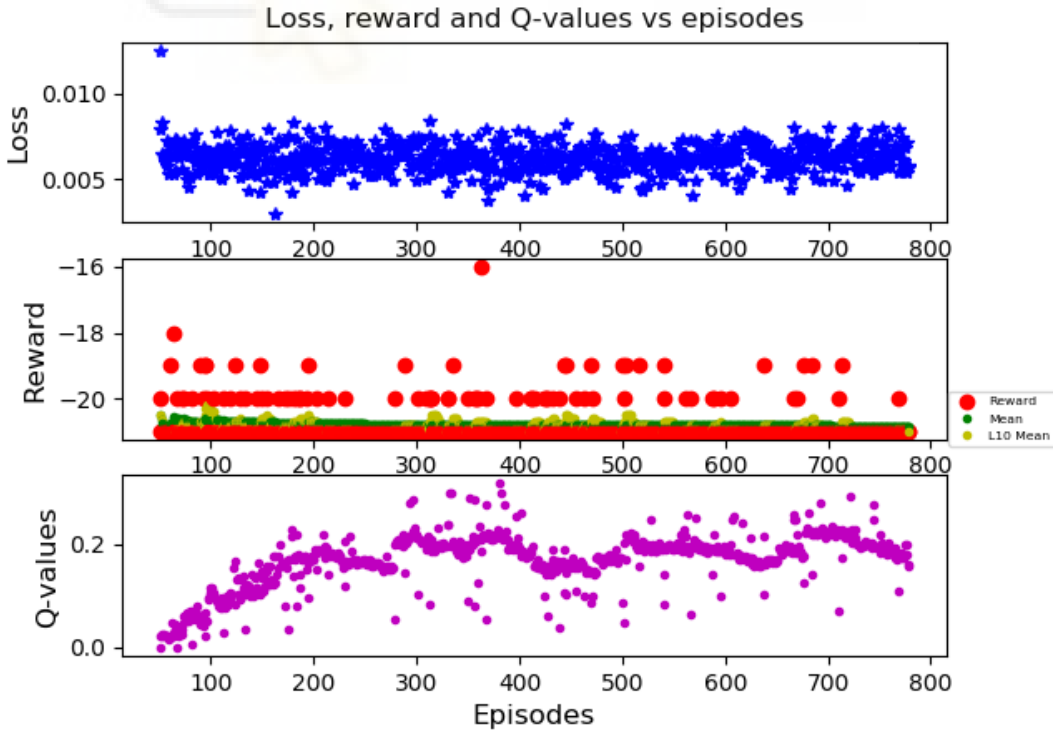

Figura 88: Gráfica PongDeterministic-v4, sesión 14

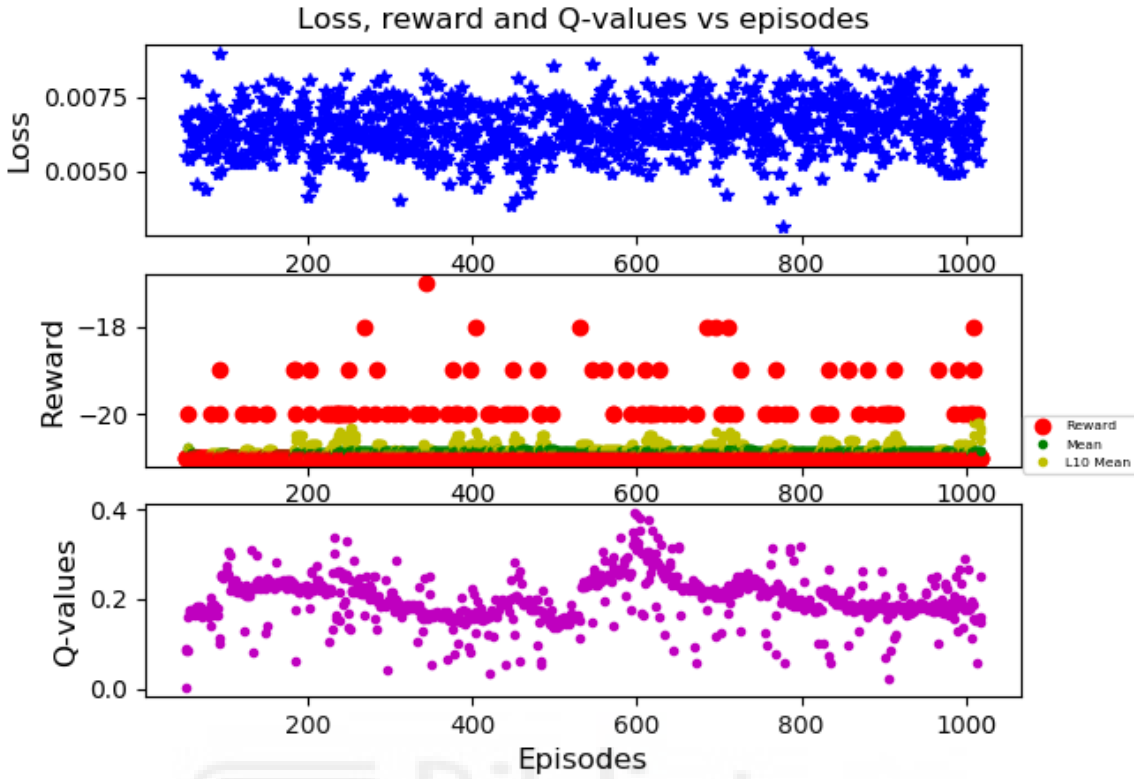

Figura 89: Gráfica PongDeterministic-v4, sesión 15

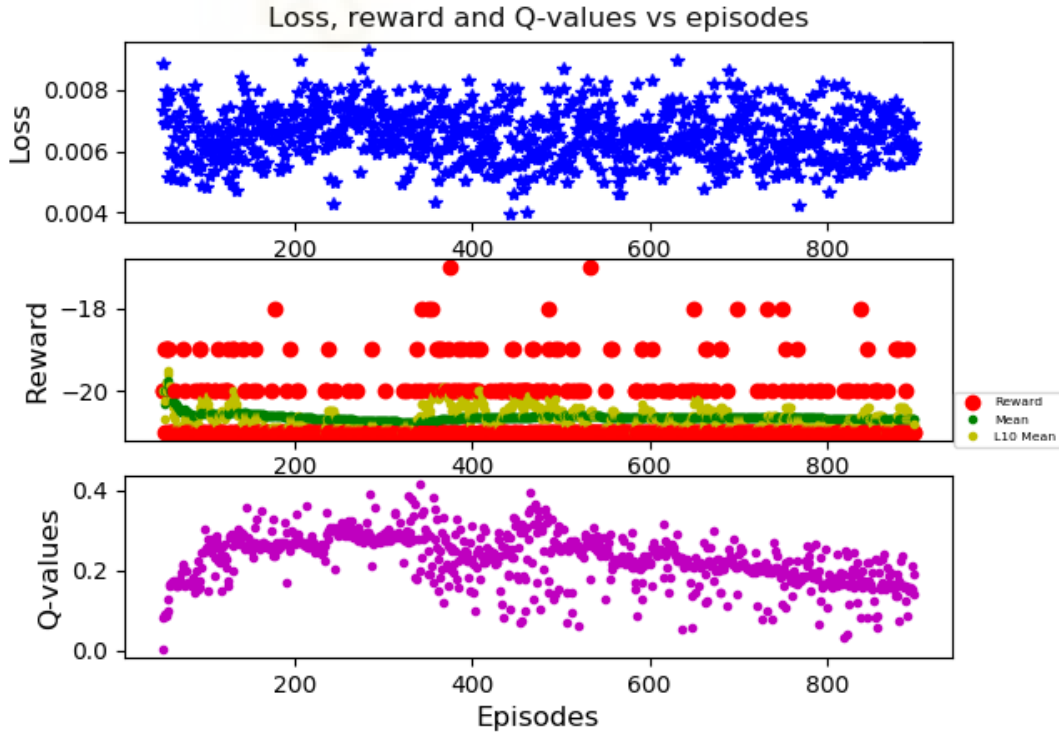

Figura 90: Gráfica PongDeterministic-v4, sesión 16

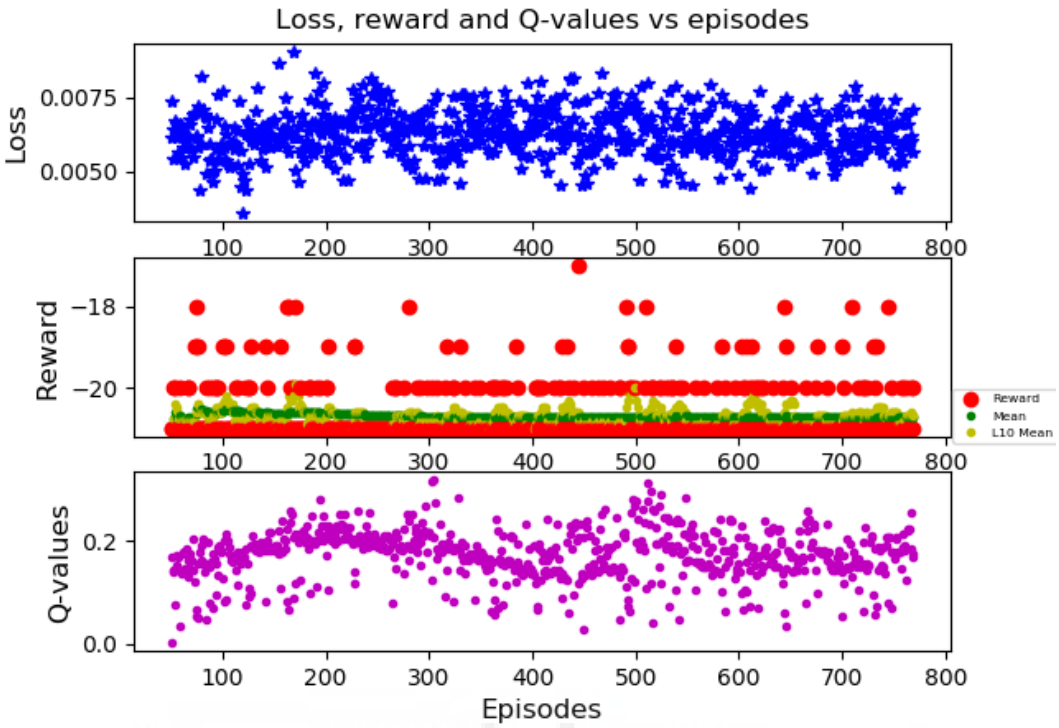

Figura 91: Gráfica PongDeterministic-v4, sesión 17

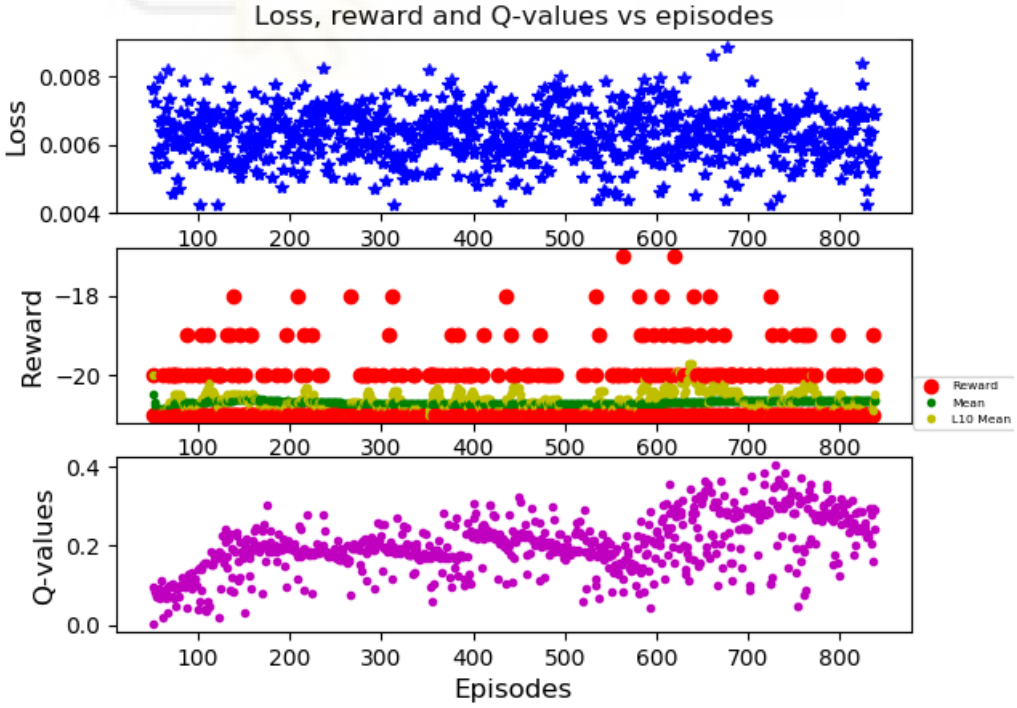

Figura 92: Gráfica PongDeterministic-v4, sesión 18

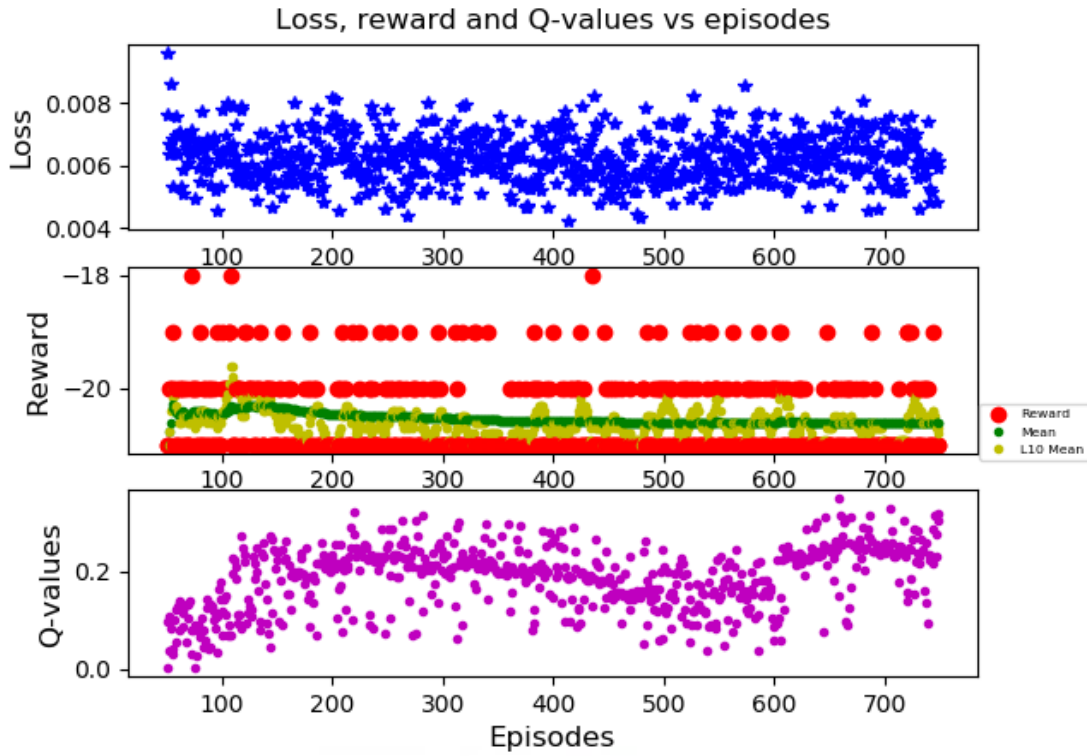

Figura 93: Gráfica PongDeterministic-v4, sesión 19

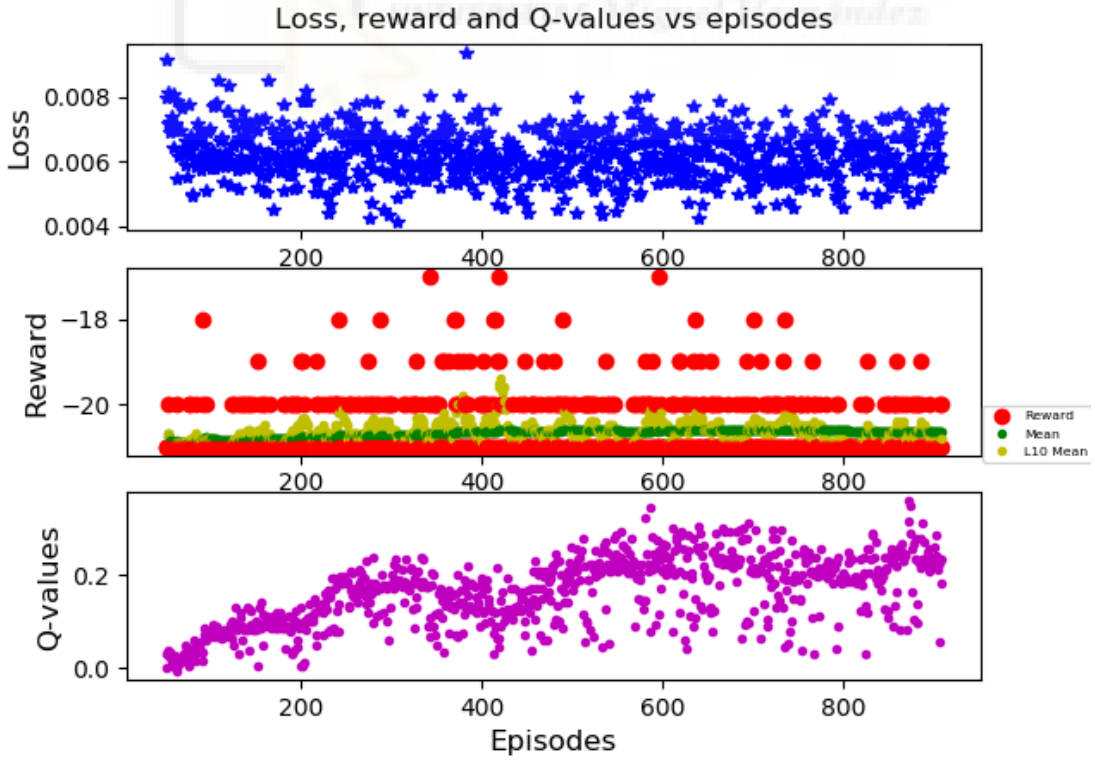

Figura 94: Gráfica PongDeterministic-v4, sesión 20

Análisis de las gráficas de las sesiones de entrenamiento: encontramos que **el *loss* se mantiene en valores aceptables**, cercanos al cero; en cuanto a las puntuaciones, encontramos una **media bastante estable**, sin grandes altibajos en general; esto indica que **el agente no logra encontrar una estrategia que le haga evolucionar durante las sesiones**. En cambio, **si observamos la media L10, encontramos grandes picos en algunas sesiones; esto indica que el agente alterna períodos en los que es capaz de encontrar una mejor estrategia**, si bien, no son la norma.

En cuanto a los *Q-values*, al principio oscilan entre 0 y 0.2; a partir de la sesión 15 de entrenamiento, se acercan más al 0.4, lo que indica en teoría una mejor selección de acciones; en cambio, en la última sesión vemos que no se acerca a esos valores y se mantiene en valores superiores al 0.2 hacia el final de la sesión.

Se presenta a continuación la tabla con los resultados de los experimentos finales:

Fecha	*Experimento*	*Puntuación max*	*Media Max*	*Media L10*	*Media Final*	*N.º Episodios*
28/05/19	Aleatorio	-17	-20,27	-20,7	-20,34	200
28/05/19	Experimento 1	-19	-20,8	-21	-20,88	200

Tabla 3: Tabla resultados finales Pong

Análisis resultados prueba final: el experimento totalmente **aleatorio**, a rasgos generales, sorprendentemente **consigue mejores resultados** (por poco) **que nuestro agente**. En puntuación máxima, supera por dos puntos (-17 frente a -19); asimismo, la media máxima es ligeramente superior (-20,27 frente a -20,8); pasa lo mismo con la media L10 (-20,7 frente a -21) y la media final (-20,34 en el caso aleatorio, -20,88 usando nuestro agente).

En este caso, **solo se puede concluir que el experimento no demuestra un rendimiento mejor que ante un entorno puramente aleatorio**; bien porque los ajustes no son los adecuados, bien porque el entorno presenta dificultades que hacen que el aprendizaje no sea correcto.

Se adjuntan las gráficas de las sesiones de prueba final, tanto en el experimento aleatorio como en el realizado con el agente.

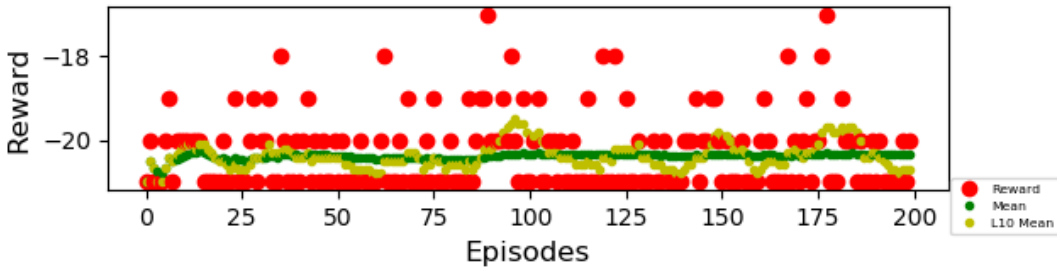

Figura 95: Gráfica puntuaciones experimento aleatorio PongDeterministic-v4

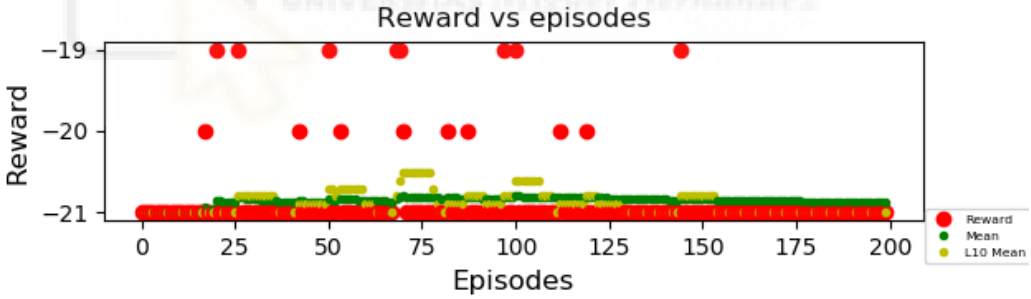

Figura 96: Gráfica puntuaciones agente PongDeterministic-v4

7.2.2 Breakout

El objetivo de Breakout es destruir todos los cuadros posibles, para ello utilizaremos una bola que haremos rebotar utilizando para ello la barra inferior que aparece en pantalla.

Figura 97: Captura de pantalla de Breakout

7.2.2.1 Experimento 1

Los hiperparámetros utilizados son los siguientes:

- Episodios máximos de entrenamiento: 100010
- Episodios de observación: 50
- Pasos para actualizar la red objetivo: 10000
- Tamaño de memoria: 100000
- Número de experiencias para *experience replay*: 32
- Ratio de aprendizaje: 0.00001
- Cuadros hasta epsilon mínimo: 300000
- Epsilon final: 0.1
- Gamma: 0.99

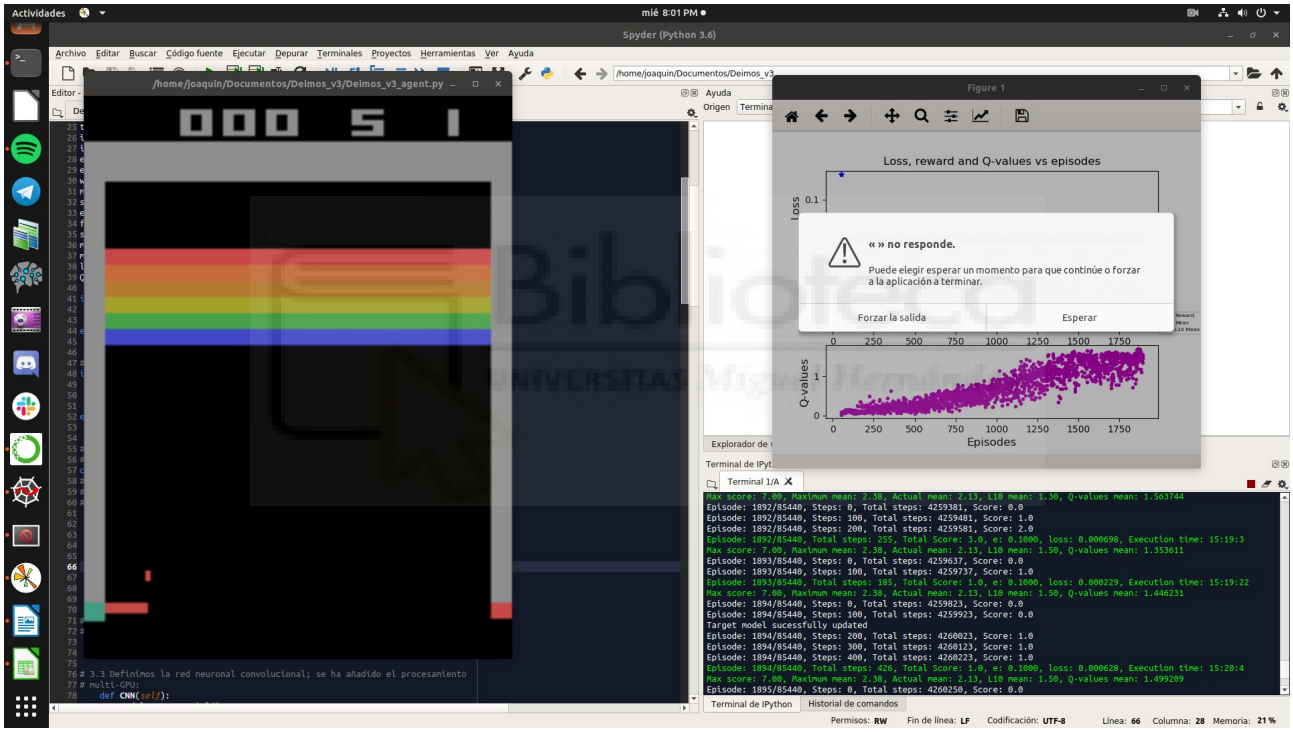

Figura 98: Captura de pantalla del entorno de trabajo durante una sesión de entrenamiento con Breakout

Durante las sesiones de entrenamiento, se elabora la siguiente tabla:

Fecha	Puntuación máx	Media máx	Media L10	Media final	Loss	Media Q-Values	Tiempo sesión	Tiempo total	N.º episodios sesión	Episodios totales	Experiencias sesión	Experiencias totales
06/05/19	8	1,62	2,1	1,59	0,001781	0,642608	10:37:19	10h 37min 19s	1740	1740	413429	413429
07/05/19	8	2,3	2,5	1,56	0,000156	2,039893	16:29:46	27h 07min 05s	2040	3780	621343	1034772
08/05/19	9	1,88	0,1	1,83	0,000369	1,98474	19:35:08	46h 42min 13s	2300	6080	724004	1758776
09/05/19	8	2,2	2,3	1,94	0,00097	1,449573	17:17:10	63h 59min 23s	2110	8190	653073	2411849
10/05/19	9	2,48	4,4	2,05	0,000796	1,299886	16:38:01	80h 37min 24s	2020	10210	623609	3035458
11/05/19	8	3,5	1,8	2,15	0,000158	1,697094	14:59:49	95h 37min 13s	1830	12040	572273	3607731
13/05/19	10	2,4	1,8	2,28	0,002524	0,782492	19:44:35	115h 21min 48s	2430	14470	738841	4346572
14/05/19	8	4	2	2,47	0,002292	0,984005	19:56:25	135h 18min 13s	2420	16890	738371	5084943
15/05/19	8	4	2,3	2,61	0,000818	1,068405	17:41:46	152h 59min 59s	2190	19080	666020	5750963
16/05/19	9	4	3,2	2,69	0,002089	0,804413	16:40:08	169h 40min 07s	2000	21080	620517	6371480
18/05/19	12	3	4,3	2,8	0,001677	1,516954	40:04:06	209h 44min 13s	4700	25780	1497379	7868859
20/05/19	9	4,33	2,9	2,69	0,000772	1,430053	17:01:02	226h 45min	1990	27770	636910	8505769

								15s				
21/05/19	9	4	3,1	2,66	0,0000695	1,680908	16:51:06	243h 36min 21s	1940	29710	636196	9141965
22/05/19	8	2,58	3,7	2,58	0,000421	1,727162	17:07:50	260h 44min 11s	1980	31690	643166	9785131

Tabla 4: Resultados entrenamiento Breakout, experimento 1

Análisis de los resultados obtenidos durante las sesiones de entrenamiento: primero, destacar que en este caso, **el número de sesiones es menor que en las utilizadas con *PongDeterministic-v4*** (20 con *Pong* frente a 14 con *Breakout*); esto es debido, en gran medida, al uso de la nueva gráfica.

Las puntuaciones máximas oscilan por lo general entre 8 y 9, encontrando que en dos sesiones se consiguieron 10 y 12 puntos de máxima; curiosamente, la puntuación máxima corresponde a la sesión de más duración (40 horas aproximadamente).

En cuanto a la máxima media obtenida, encontramos que en la sesión del 20/05/19 **se obtiene una puntuación media máxima de 4,33**, muy probablemente obtenida al principio, ya que no se corresponde con la media final (2,69).

El caso interesante en cuanto a **la máxima media se da en la última sesión**: coincide con la media final (2,58). Esto nos indica que, **en el momento de parar el entrenamiento, estaba en ascenso.** Un vistazo a la media L10 de esa sesión (3,70) nos lo confirma.

Los valores de las medias L10 oscilan entre el 0,1 y el 4,4, lo que indica que dependiendo de la sesión y el instante en el que nos encontremos, los resultados varían, es decir, no existe un incremento lineal del aprendizaje.

En cuanto a las medias finales, se aprecia una mejoría durante la mayoría de las sesiones: en casi todas las sesiones, se mejora la media anterior. La media más alta obtenida es de 2,8 puntos, siendo la más baja de 1,56. La media de la última sesión se sitúa en 2,58.

El *loss* se aprecia bastante bajo, pero nunca cero, lo cual indica que las redes neuronales funcionan correctamente.

Los *Q-values* se sitúan, en su gran mayoría entre 1 y 2, siendo su máximo 2,039893, correspondiente a la segunda sesión; el mínimo corresponde a la primera sesión, con 0,642608.

Un dato significativo es que **durante la sesión de 40 horas, no hubo una mejoría excesiva en las puntuaciones medias** (2,69 en la anterior sesión de 16 horas, y 2,69 en la siguiente, de 17 horas) lo que indica que un tiempo de sesión mayor no implica una mejoría excesiva en los resultados. Asimismo su *Q-value* tampoco es el más alto (1,516954).

Se presentan a continuación **las gráficas recogidas durante las sesiones de entrenamiento:**

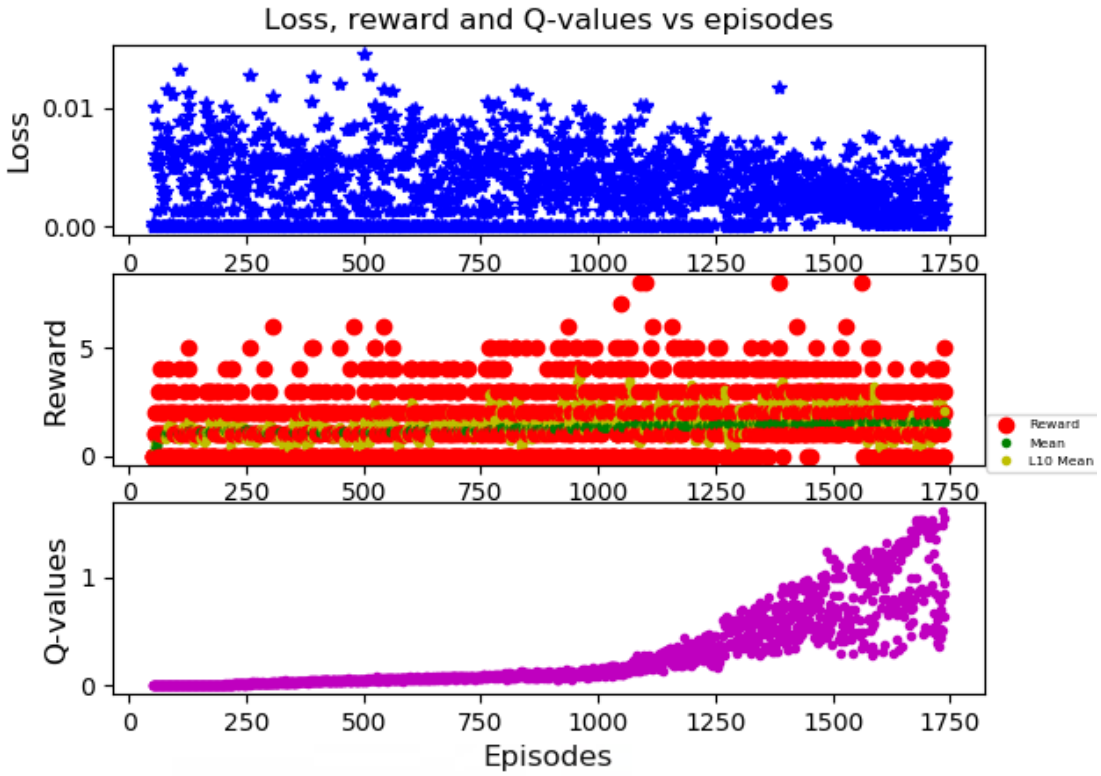

Figura 99: Grafica BreakoutDeterministic-v4, experimento 1, sesión 1

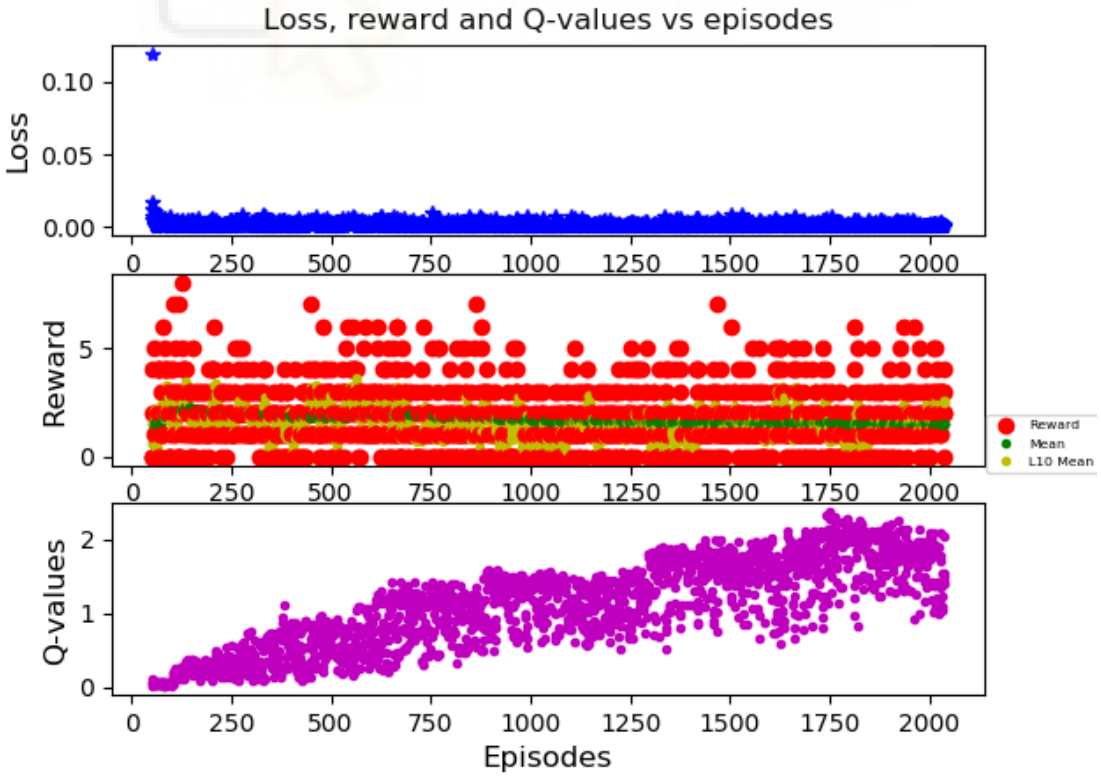

Figura 100: Grafica BreakoutDeterministic-v4, experimento 1, sesión 2

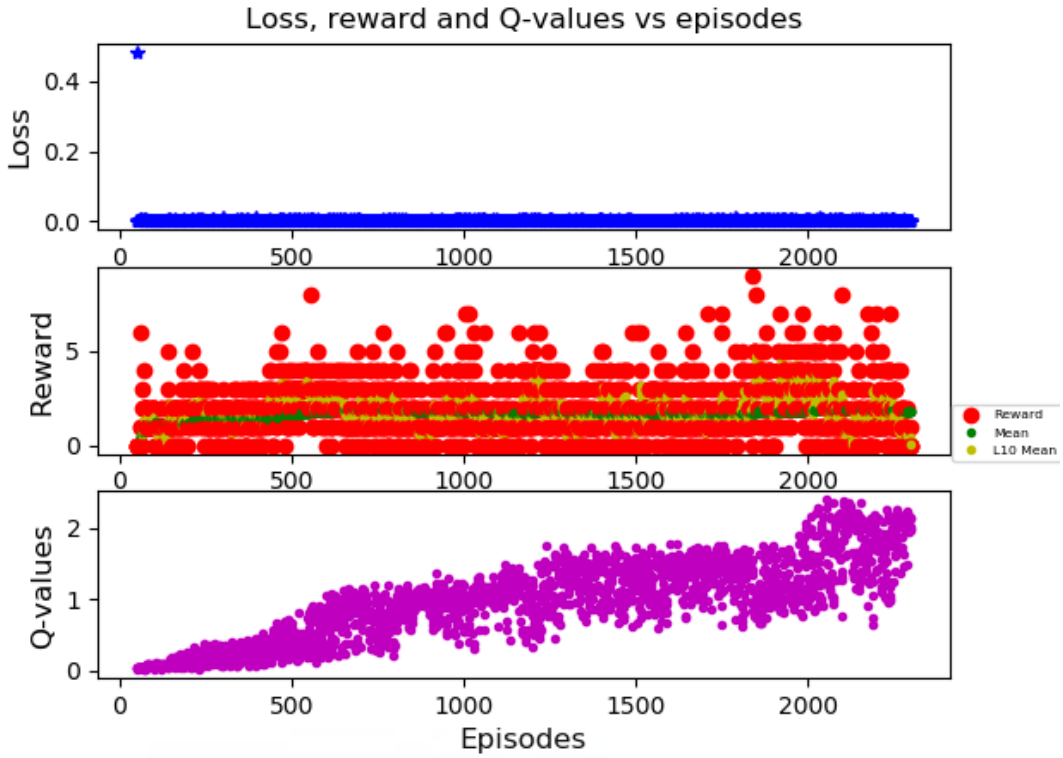

Figura 101: Grafica BreakoutDeterministic-v4, experimento 1, sesión 3

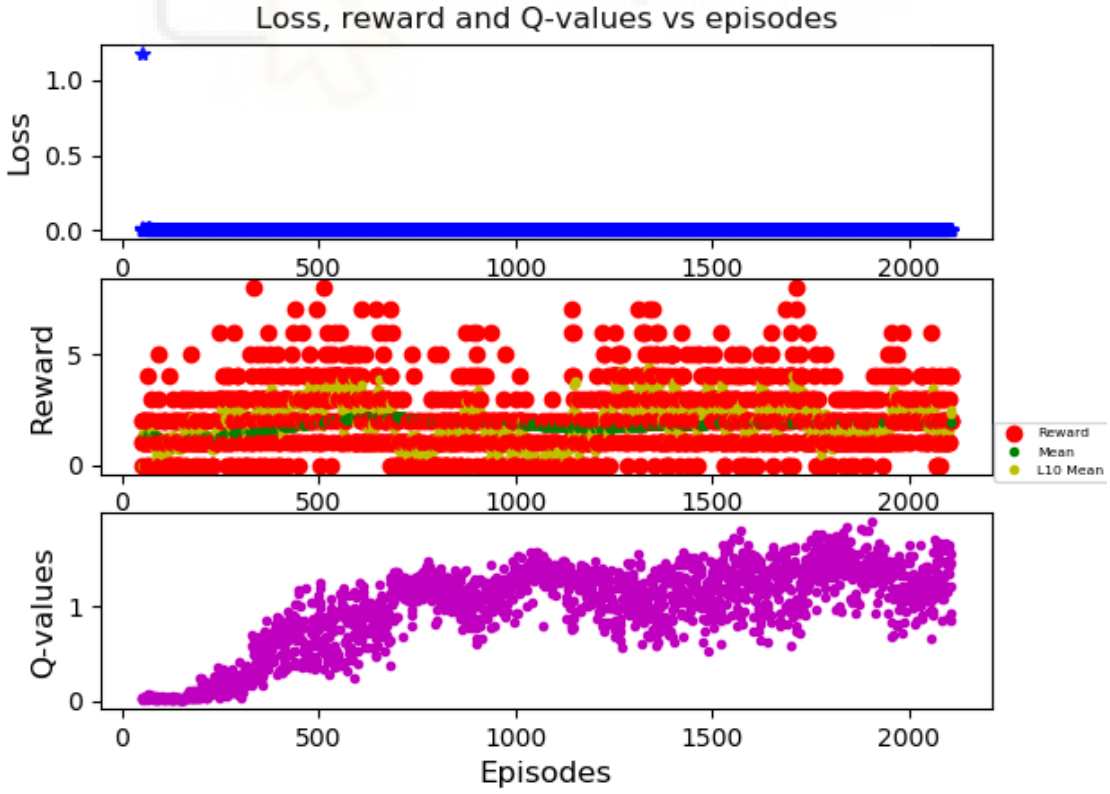

Figura 102: Grafica BreakoutDeterministic-v4, experimento 1, sesión 4

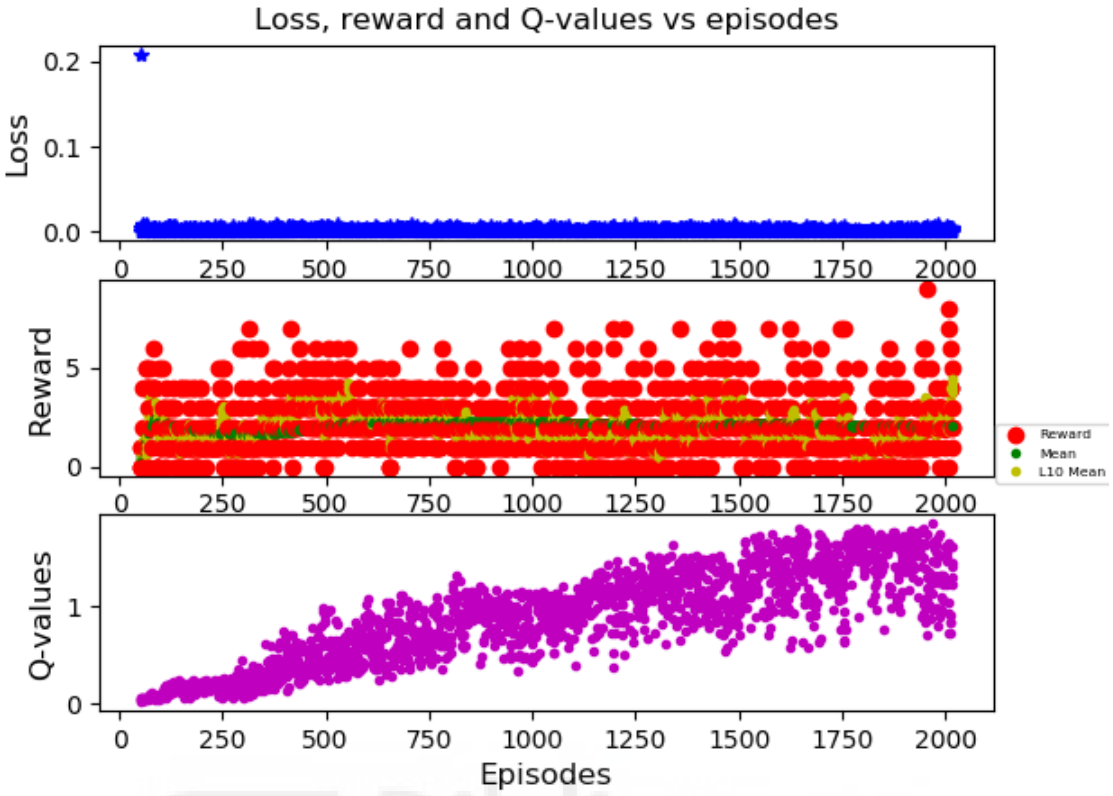

Figura 103: Grafica BreakoutDeterministic-v4, experimento 1, sesión 5

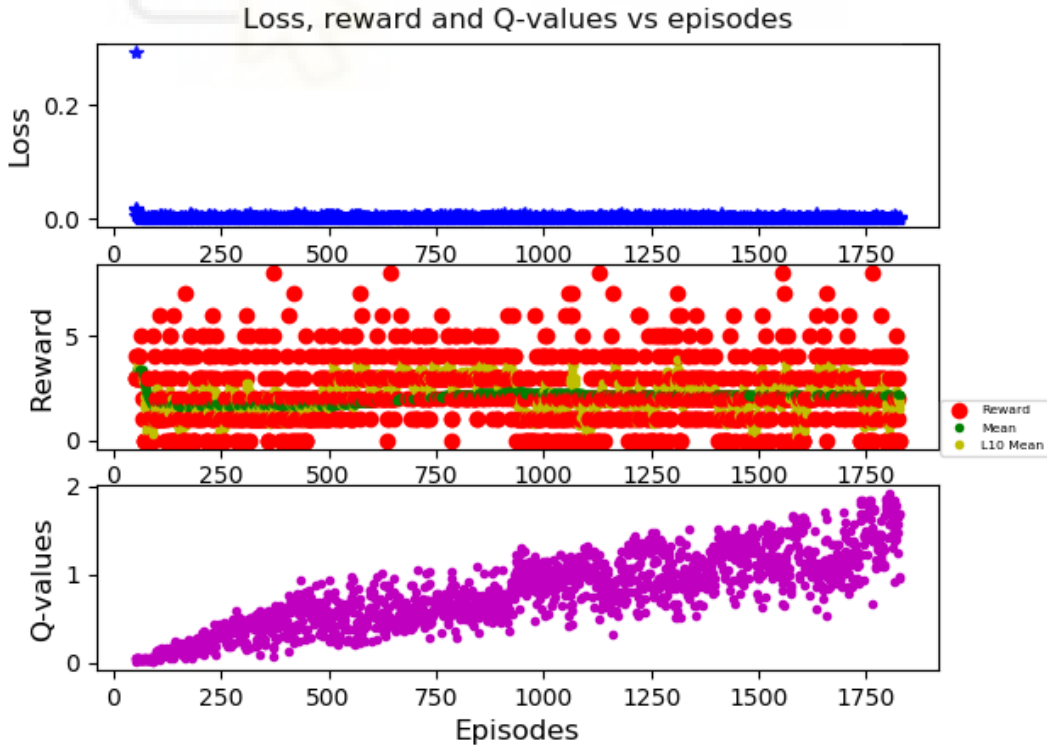

Figura 104: Grafica BreakoutDeterministic-v4, experimento 1, sesión 6

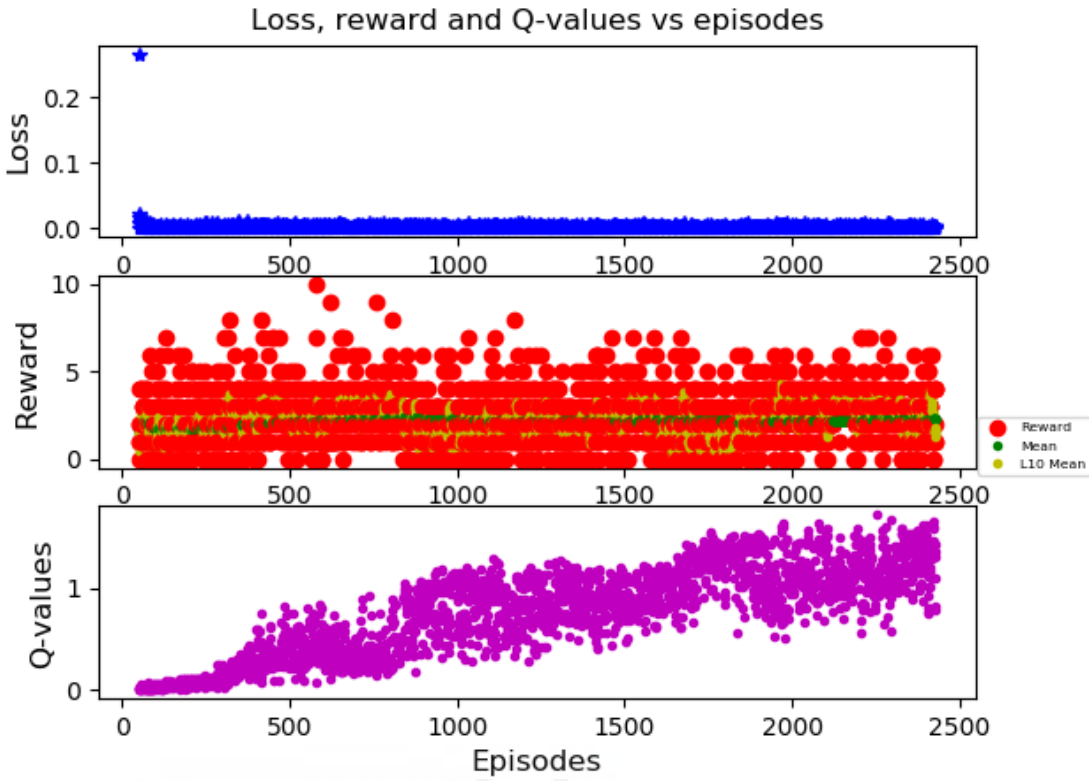

Figura 105: Grafica BreakoutDeterministic-v4, experimento 1, sesión 7

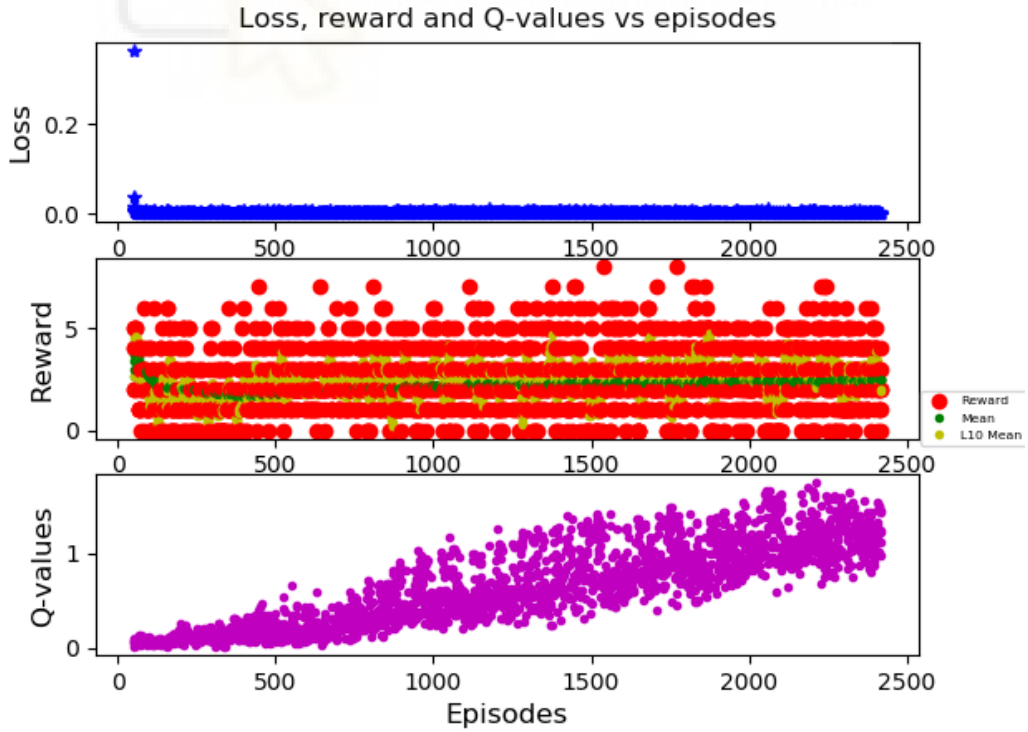

Figura 106: Grafica BreakoutDeterministic-v4, experimento 1, sesión 8

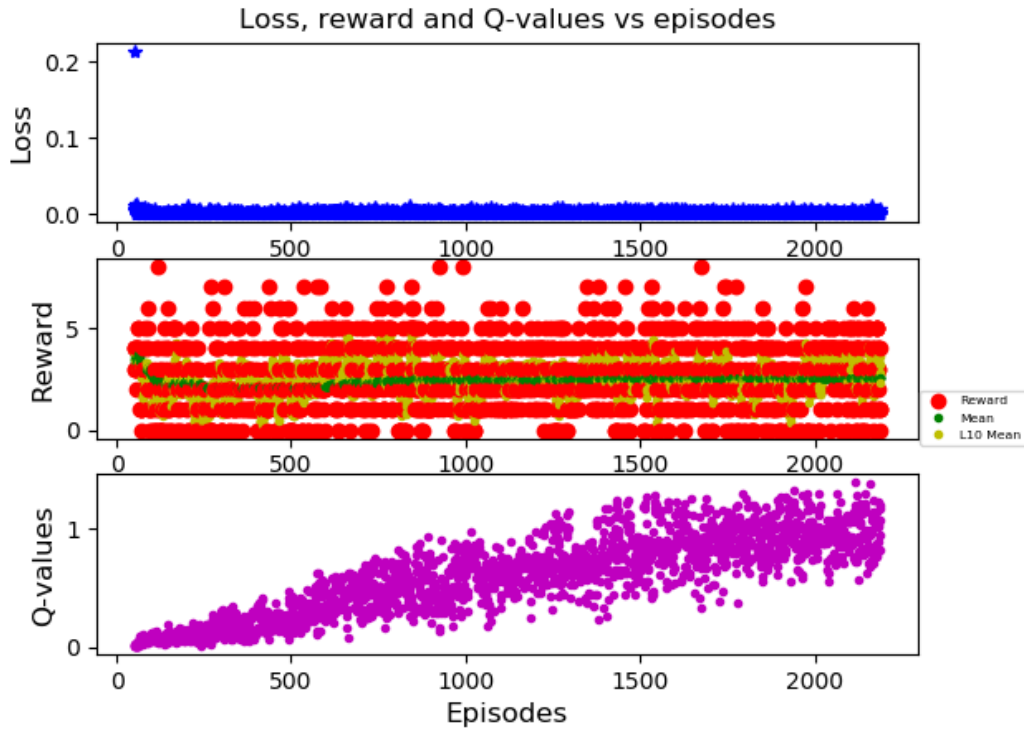

Figura 107: Grafica BreakoutDeterministic-v4, experimento 1, sesión 9

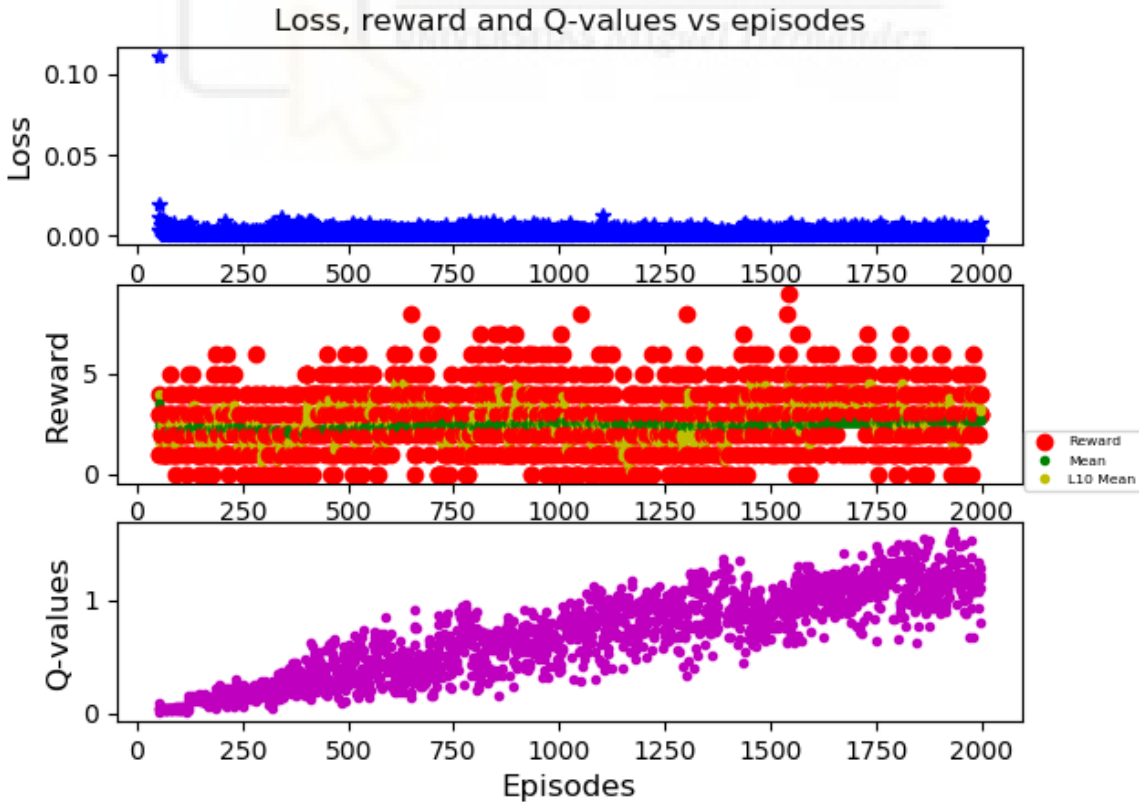

Figura 108: Grafica BreakoutDeterministic-v4, experimento 1, sesión 10

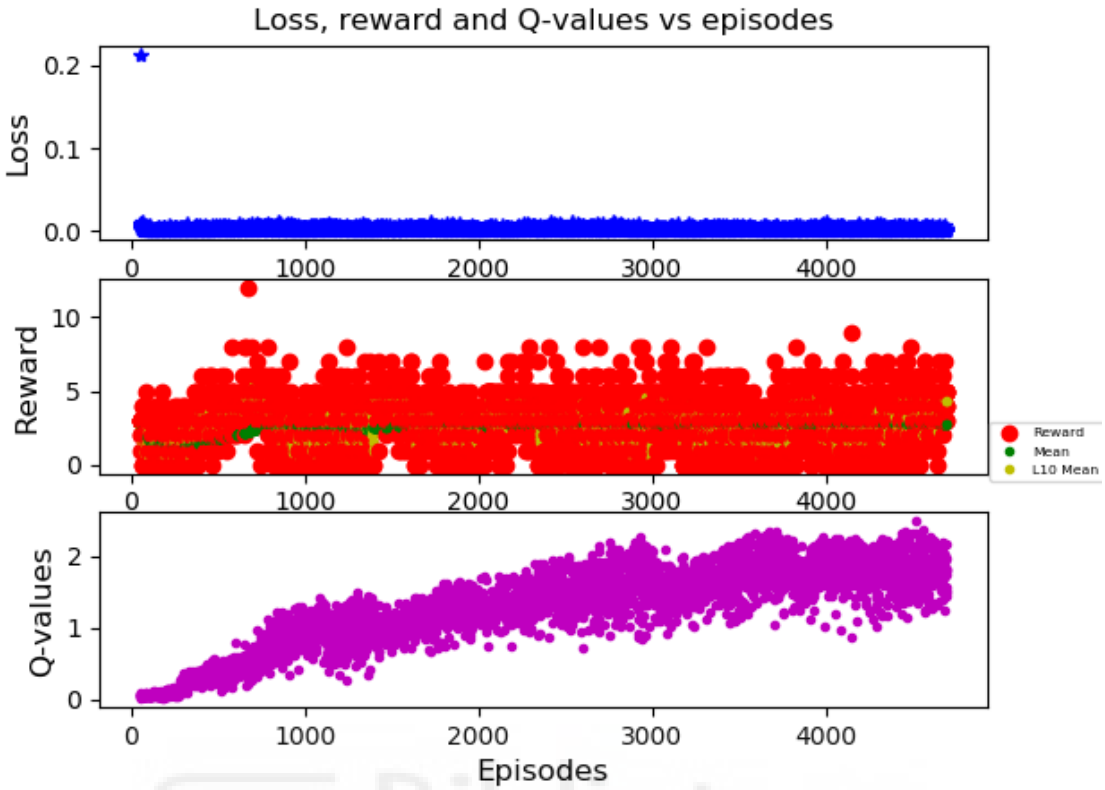

Figura 109: Grafica BreakoutDeterministic-v4, experimento 1, sesión 11

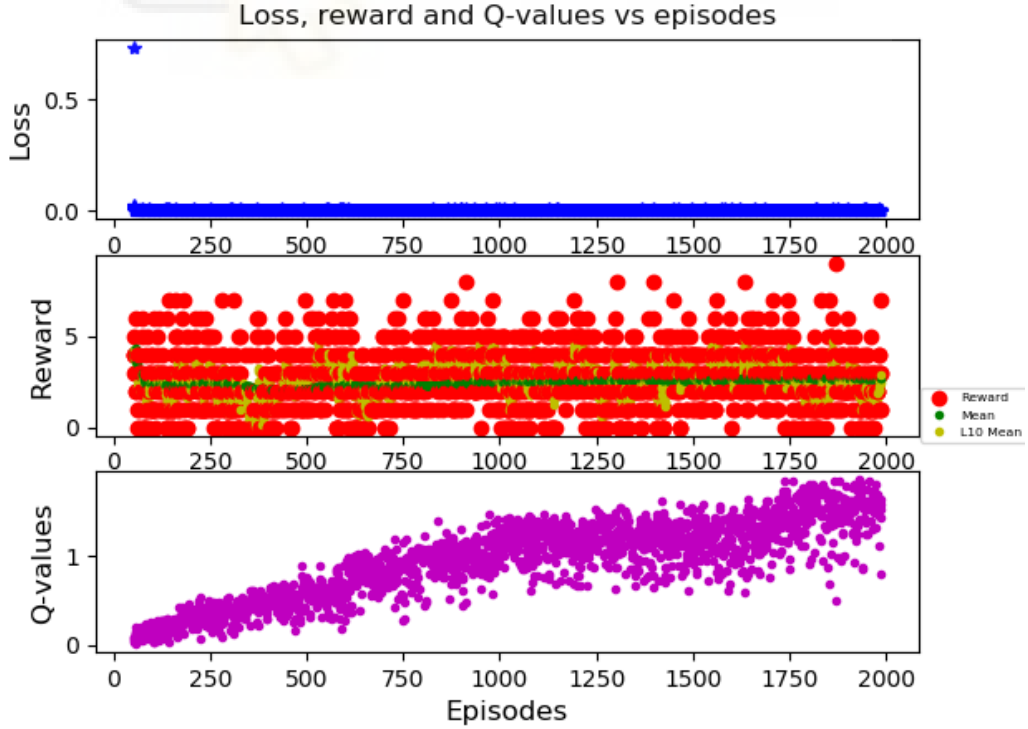

Figura 110: Grafica BreakoutDeterministic-v4, experimento 1, sesión 12

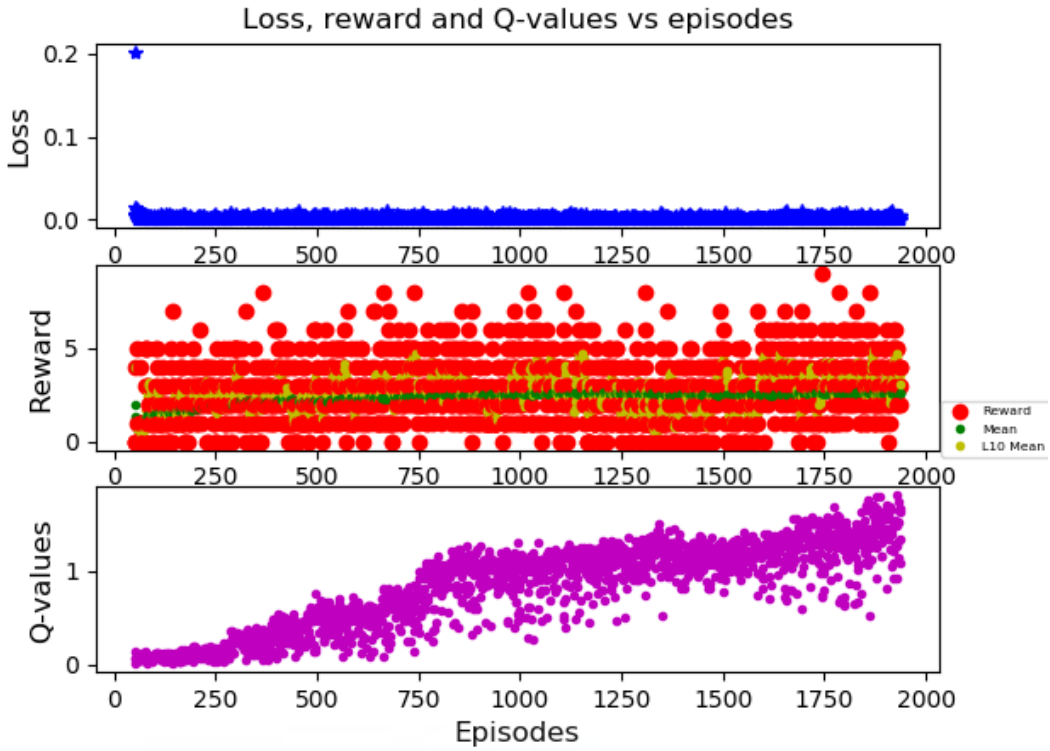

Figura 111: Grafica BreakoutDeterministic-v4, experimento 1, sesión 13

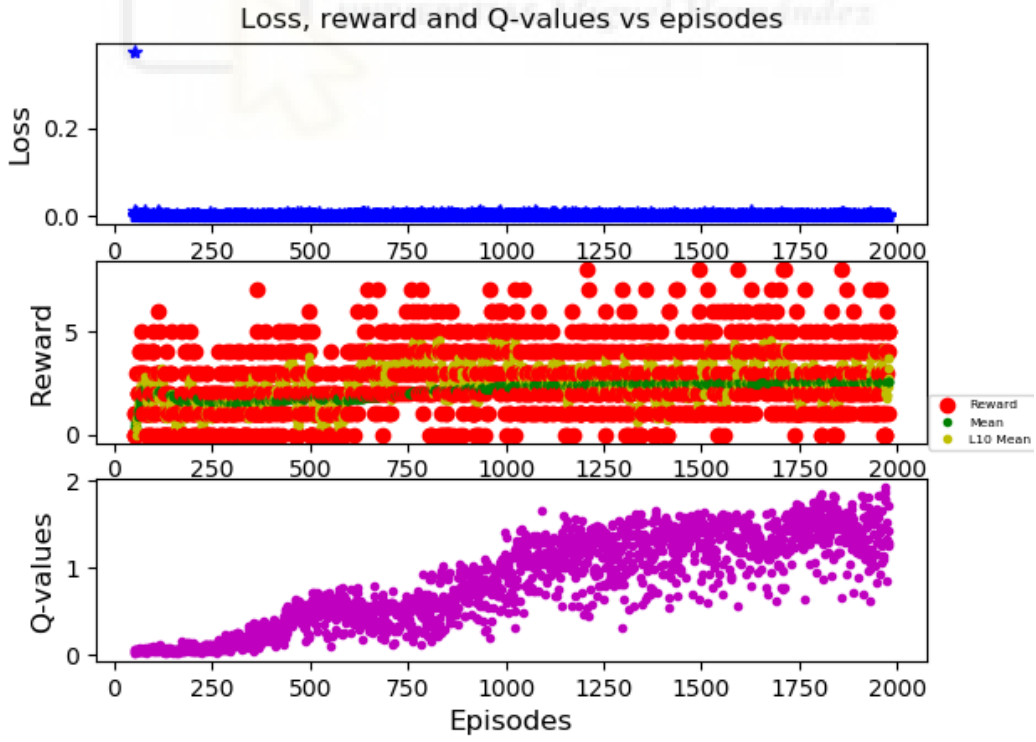

Figura 112: Grafica BreakoutDeterministic-v4, experimento 1, sesión 14

Análisis de las gráficas de las sesiones de entrenamiento: lo primero que se aprecia es que **la gráfica del *loss* forma una línea prácticamente recta, lo cual indica que las redes neuronales funcionan correctamente.** Las oscilaciones que se producen en los valores del *loss* son mínimas.

Si analizamos las **puntuaciones**, encontramos que **la recta de la media** (verde oscuro), en general, **tiende a subir pero muy lentamente**; lo cual indicaría que hay una mejoría (y por tanto un aprendizaje), aunque muy pequeño. **La media L10 presenta**, al igual que en el caso de *Pong*, **altibajos**, lo que nuevamente indicaría que las puntuaciones no aumentan o disminuyen linealmente; esto nos indica que los resultados no son constantes, y, por tanto, el aprendizaje tampoco.

En cuanto a la gráfica de *Q-values*, vemos que a pesar de tener cierta dispersión (sobretodo con el avance del tiempo), lo cual indica una oscilación en los valores, **tiende a aumentar**; además, no se aprecian grandes "cortes" en la gráfica; esto significa que la tasa de actualización de la red objetivo es buena. Si existieran estos cortes o grandes cambios, significaría que la tasa de actualización es inadecuada.

Los resultados del experimento final son los siguientes:

Fecha	*Experimento*	*Puntuación máx*	*Media max*	*Media L10*	*Media final*	*N.º Episodios*
27/05/19	Aleatorio	5	1,28	1,4	1,17	200
27/05/19	Experimento 1	10	2,67	2,5	2,25	200

Tabla 5: Tabla resultados finales BreakoutDeterministic-v4, experimento 1

Análisis resultados experimento final: la puntuación máxima de nuestro agente es justamente el doble del test aleatorio (10 la obtenida por el agente, 5 en aleatorio). **La media máxima es más del doble** (2,67 frente a 1,28). **La media L10 es más de un punto mayor con respecto a la prueba aleatoria.**

Finalmente, la media final obtenida es de 2,25 puntos por nuestro agente frente a 1,17, para el mismo número de episodios. Como se puede apreciar en este caso, **sí presenta una mejoría en los resultados frente al aleatorio**; si bien no son comparables a los obtenidos por *Google DeepMind* en sus papers [3] y [4], se aprecia que existe un aprendizaje.

Las gráficas obtenidas durante las pruebas son las siguientes:

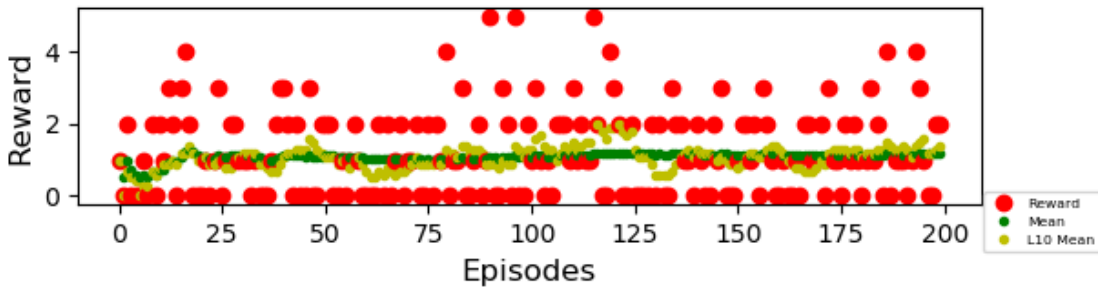

Figura 113: Gráfica BreakoutDeterministic-v4, test aleatorio

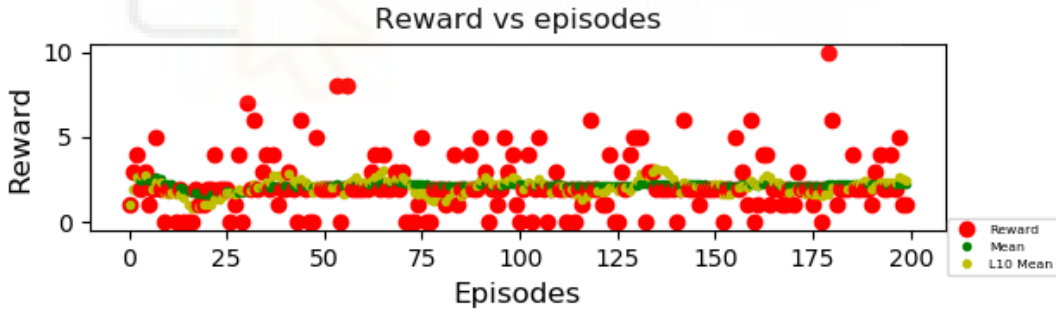

Figura 114: Gráfica BreakoutDeterministic-v4, test final experimento 1

Análisis gráficas test final experimento 1: si bien las gráficas a primera vista son bastante similares, cabe destacar que **en el caso aleatorio, encontramos mayor número de partidas en las que la puntuación es 0 que durante el test realizado con el agente.** Asimismo, la línea de la media, a pesar de en los dos casos mantenerse constante prácticamente, es un punto más alta en el caso de nuestro agente.

7.2.2.2 Experimento 2 (Google Colab)

Los hiperparámetros utilizados son los siguientes:

- Episodios máximos de entrenamiento: 100010
- Episodios de observación: 50
- Pasos para actualizar la red objetivo: 10000
- Tamaño de memoria: 100000
- Número de experiencias para *experience replay*: 32
- Ratio de aprendizaje: 0.00025
- Cuadros hasta epsilon mínimo: 300000
- Epsilon final: 0.1
- Gamma: 0,95

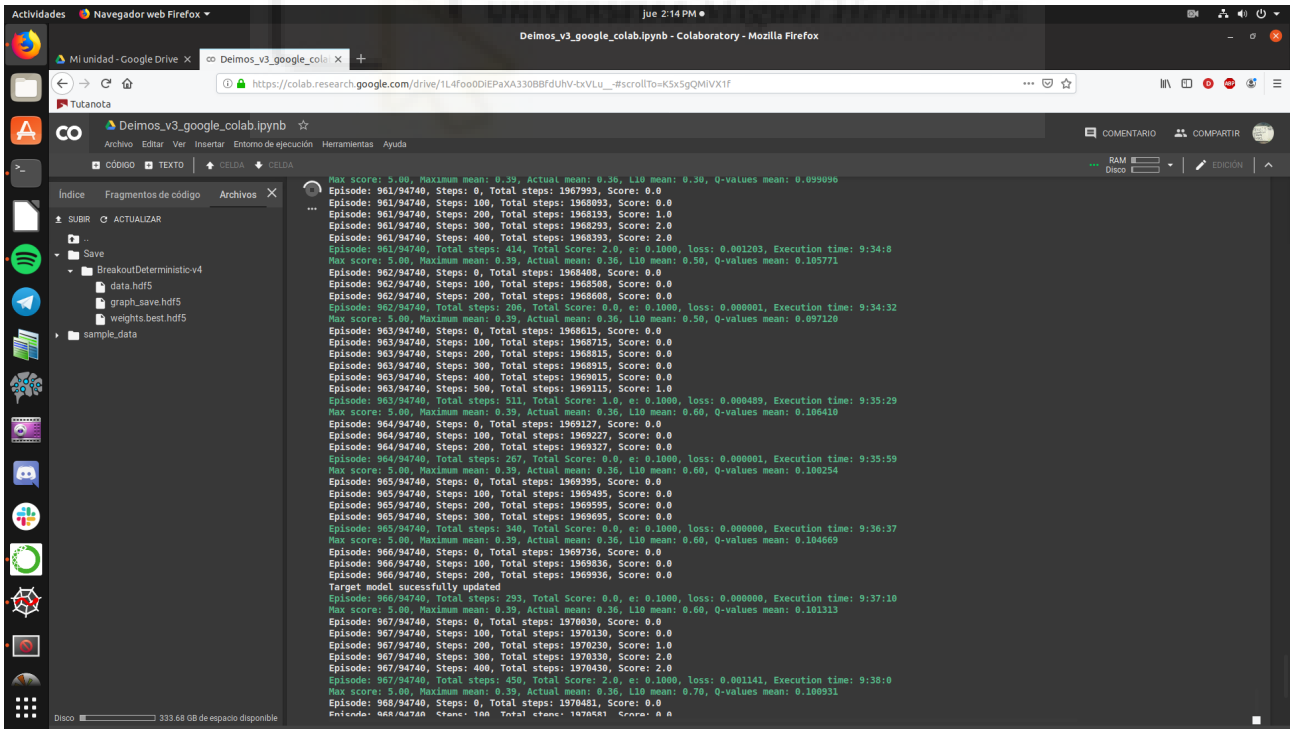

Figura 115: Captura de pantalla de Google Colab durante una sesión de entrenamiento

A continuación se presenta la gráfica donde se recogen los datos obtenidos durante las sesiones de entrenamiento:

Fecha	Puntuación máx	Media máx	Media L10	Media final	Loss	Media Q-Values	Tiempo sesión	Tiempo total	N.º episodios sesión	Episodios totales	Experiencias sesión	Experiencias totales
07/05/19	7	1,32	0,6	No recogido	0,008085	1,28	No recogido	No recogido	640	640	140453	140453
08/05/19	8	1,57	0,9	1,17	0,00383	0,077579	08:38:47	8h 38min 47s	580	1220	120171	260624
09/05/19	4	2	0,2	0,46	0,000008	0,044498	11:11:18	19h 50min 05s	590	1810	159777	420401
14/05/19	6	1	0,2	0,36	0	0,025723	11:07:40	30h 57min 45s	520	2330	171792	592193
16/05/19	4	1,5	0,2	0,4	0,0000015	0,069888	11:32:26	42h 29min 12s	530	2860	176071	768264
17/05/19	5	0,6	0,6	0,37	0	0,041033	10:51:07	53h 20min 19s	430	3290	139114	907378
20/05/19	6	No recogido	No recogido	No recogido	No recogido	No recogido	No recogido	No recogido	540	3830	176873	1084251
21/05/19	4	0,61	0,4	0,43	0	0,024299	11:20:14	64h 40min 33s	500	4330	161688	1245939
22/05/19	4	1	0,3	0,41	0	0,021477	11:44:48	76h 25min 21s	450	4780	148405	1394344

Tabla 6: Resultados entrenamiento Breakout, experimento 2 (Google Colab)

Análisis resultados obtenidos durante las sesiones de entrenamiento: el experimento se paró antes de tiempo debido a los malos resultados obtenidos, ya que no merecía la pena continuarlo a tenor de los resultados que se iban obteniendo; las puntuaciones, sorprendentemente, son peores conforme avanza el entrenamiento. La puntuación más baja es de 4 puntos, obtenida en varias sesiones, entre ellas las dos últimas.
La puntuación media máxima obtenida se observa que también tiende a disminuir conforme avanza el entrenamiento; el máximo se sitúa en 2, y el mínimo registrado en 0,6.
En cuanto a **la puntuación media final**, se observa que también es bastante baja; el máximo registrado es de 1,17 y el mínimo de 0,36, lo que indica que el algoritmo no está aprendiendo correctamente.
Analizando el *loss*, se obtienen puntuaciones de 0 o cercanas a 0, de lo que se desprende que la red neuronal en principio no muestra ningún problema de funcionamiento.
Los *Q-values* vemos que tienden a disminuir; otro indicador de que el aprendizaje no es correcto.
Los tiempos de las sesiones de entrenamiento son menores que en otros experimentos debido a que *Google Colab* nos limita el uso por sesión a 12 horas, además de haberse producido varios cortes, haciéndonos perder información o la sesión entera en muchos casos.

Las gráficas obtenidas son las siguientes:

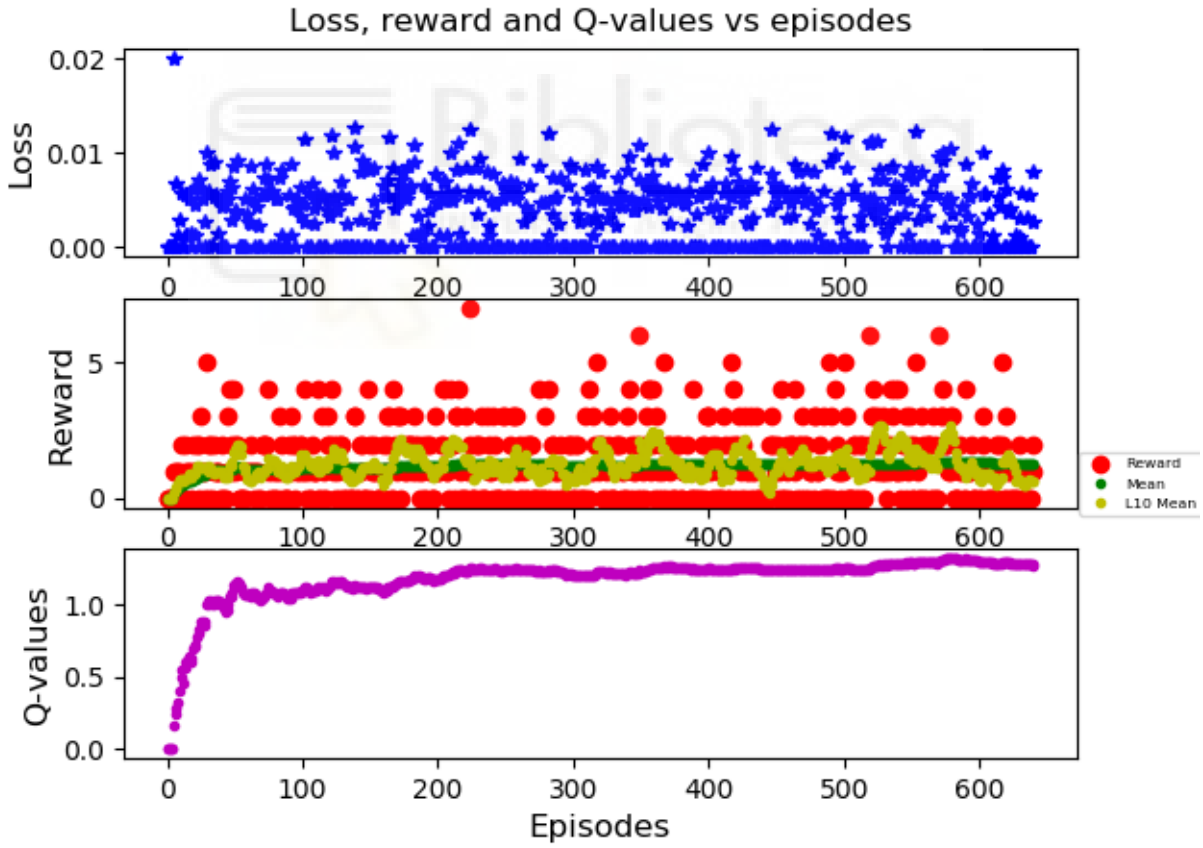

Figura 116: Gráfica BreakoutDeterministic-v4, sesión 1, experimento 2 (Google Colab)

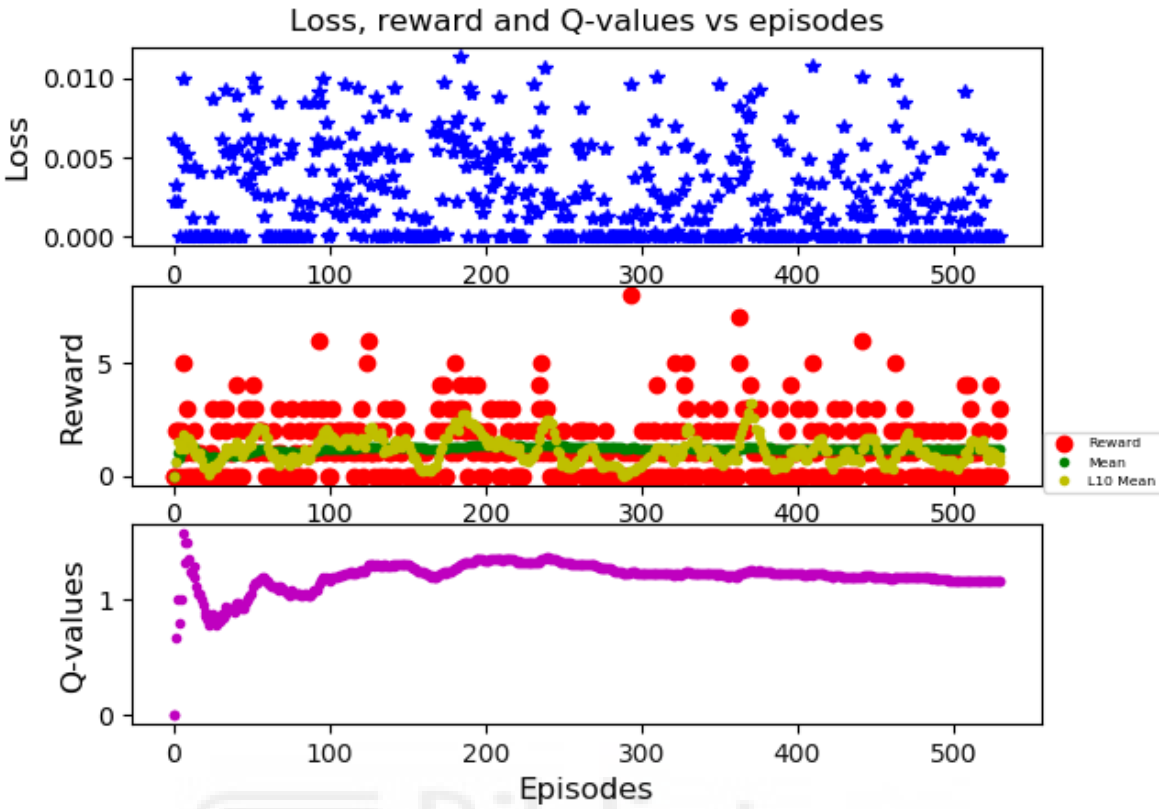

Figura 117: Gráfica BreakoutDeterministic-v4, sesión 2, experimento 2 (Google Colab)

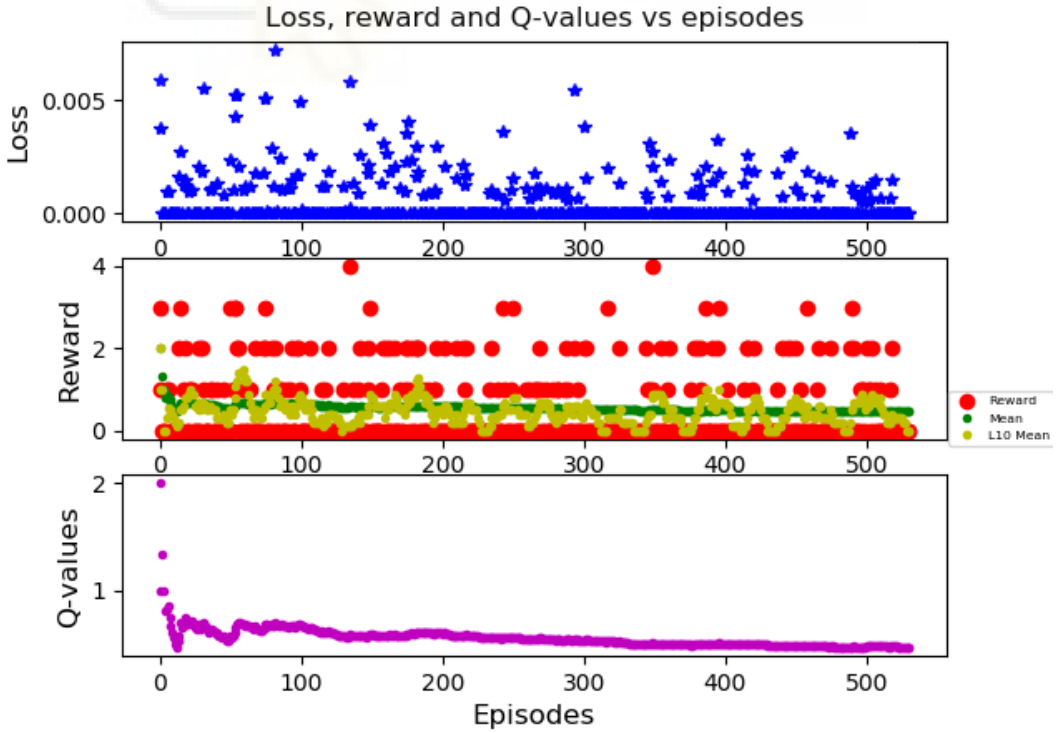

Figura 118: Gráfica BreakoutDeterministic-v4, sesión 3, experimento 2 (Google Colab)

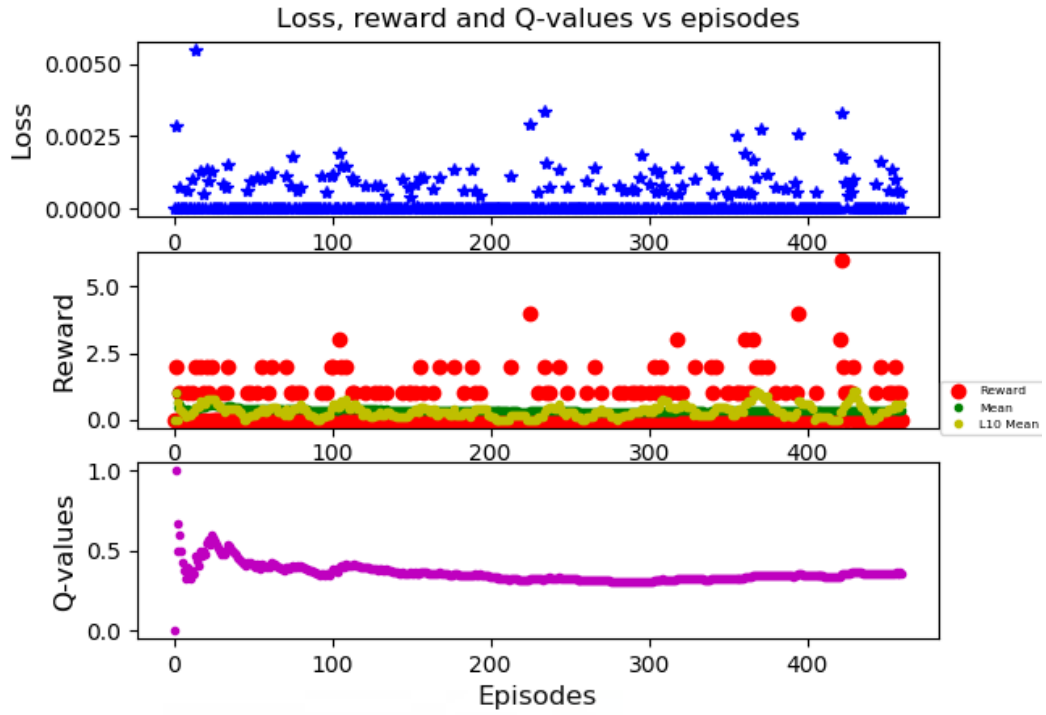

Figura 119: Gráfica BreakoutDeterministic-v4, sesión 4, experimento 2 (Google Colab)

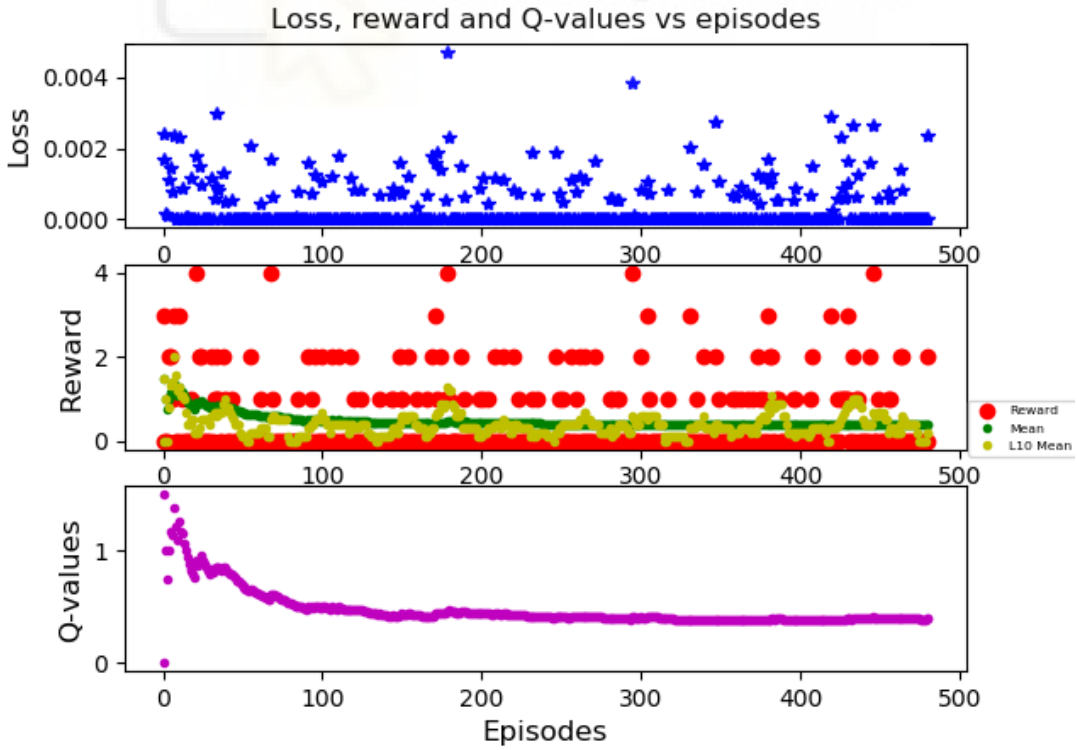

Figura 120: Gráfica BreakoutDeterministic-v4, sesión 5, experimento 2 (Google Colab)

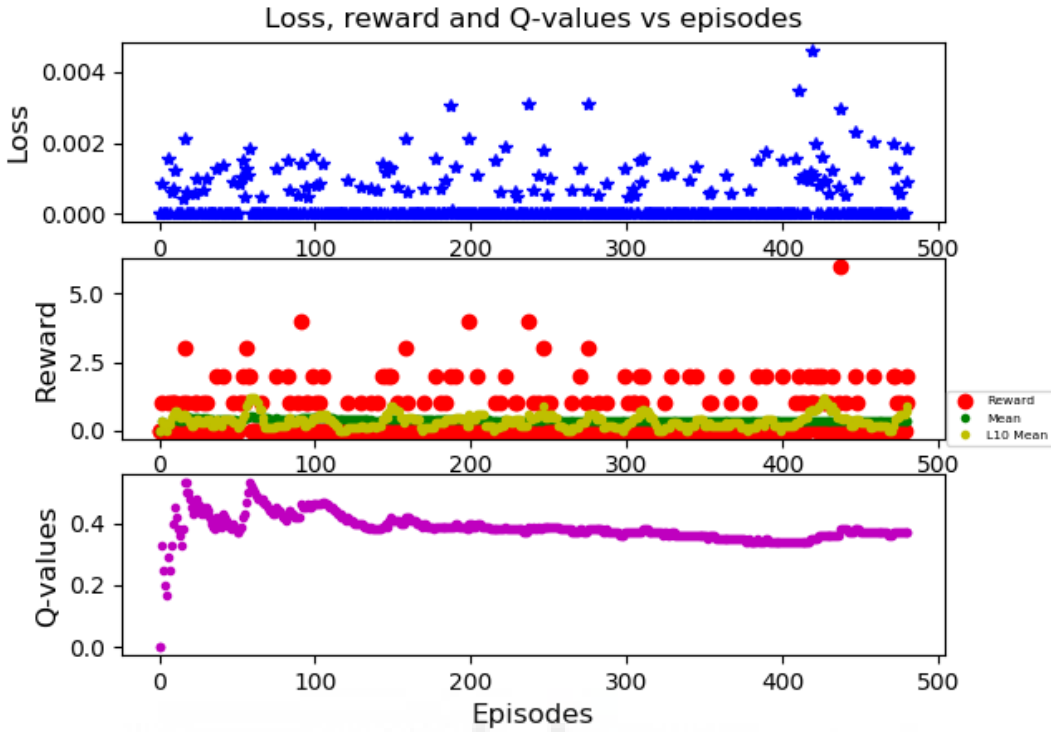

Figura 121: Gráfica BreakoutDeterministic-v4, sesión 6, experimento 2 (Google Colab)

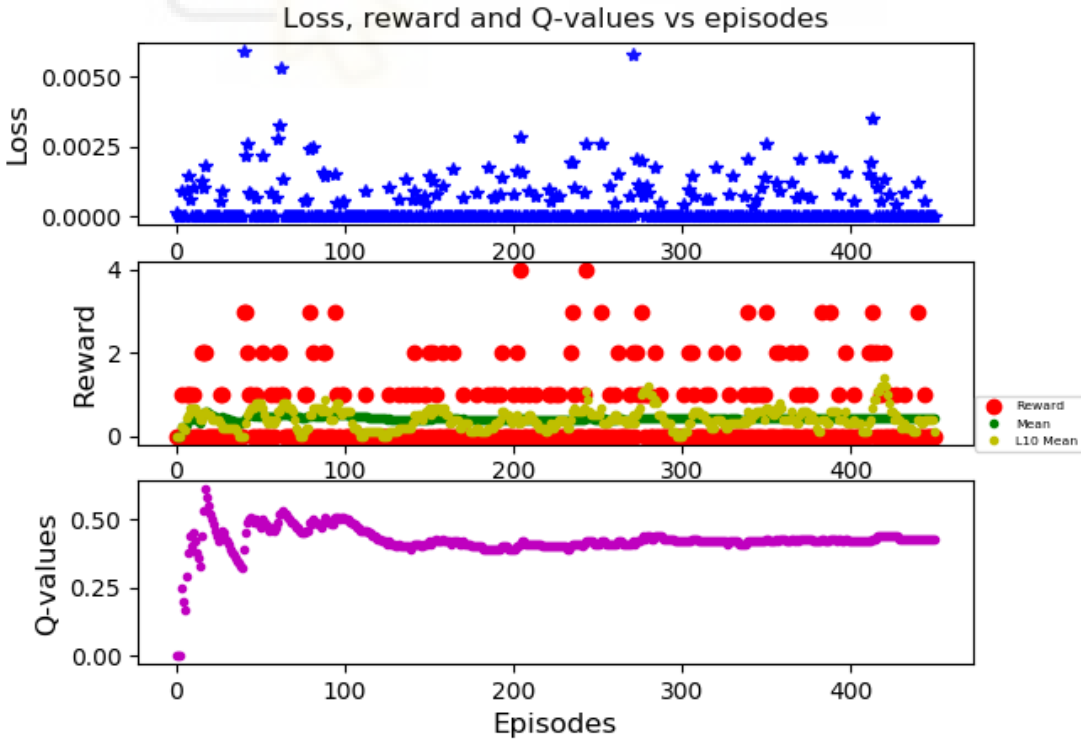

Figura 122: Gráfica BreakoutDeterministic-v4, sesión 7, experimento 2 (Google Colab)

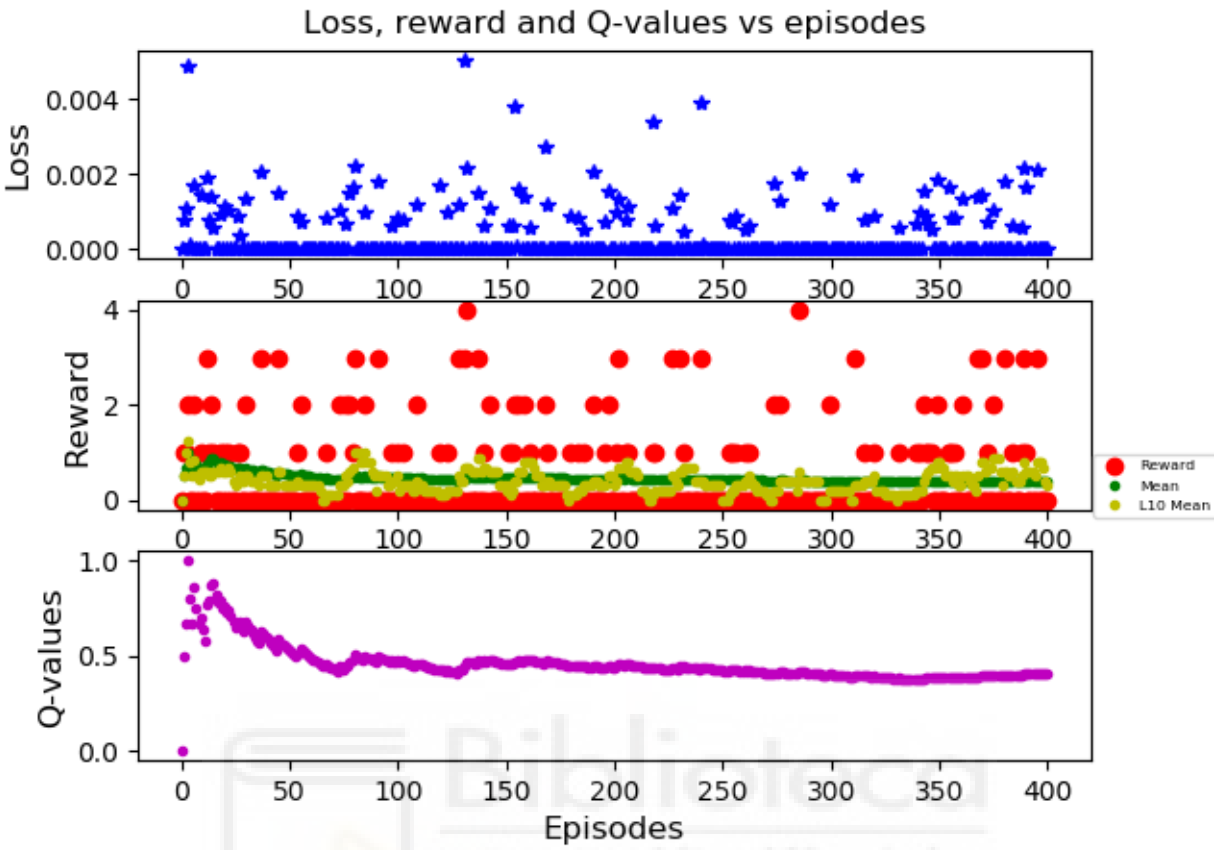

Figura 123: Gráfica BreakoutDeterministic-v4, sesión 8, experimento 2 (Google Colab)

Análisis de las gráficas obtenidas durante las sesiones de entrenamiento: como se observa del análisis de la tabla de resultados de las sesiones de entrenamiento, vemos que **el *loss* es bastante bajo, con pequeñas oscilaciones** (cosa habitual). **Las puntuaciones muestran poca dispersión, con gran cantidad de puntuaciones cero o muy bajas**; en consecuencia, **la línea que dibuja la media es bastante baja y plana**, es decir, no se presenta un aprendizaje; **la media L10**, si bien oscila en algunos casos, **sigue siendo bastante baja**.

En cuanto a los *Q-values*, en vez de aumentar, **oscilan** en un principio **hasta estabilizarse** prácticamente **en una línea recta**, lo que indica que **no hay una mejoría en la calidad de la selección de acciones** y por tanto no existe un aprendizaje; el agente con estos parámetros no es capaz de extraer información que le haga encontrar una estrategia para aumentar su puntuación.

Los **resultados** obtenidos en las **pruebas finales** son los siguientes:

Fecha	*Experimento*	*Puntuación máx*	*Media max*	*Media L10*	*Media final*	*N.º Episodios*
27/05/19	Aleatorio	5	1,28	1,4	1,17	200
29/05/19	Experimento 2	7	2	0	0,4	200

Tabla 7: Resultados finales BreakoutDeterministic-v4, experimento 2

Análisis resultados obtenidos experimento final: como era de esperar a tenor de los resultados obtenidos en el entrenamiento, excepto en la puntuación máxima y en la media máxima, **el experimento aleatorio consigue mejores resultados que los resultados obtenidos por el agente.**

Estos malos resultados tienen una explicación: hemos modificado el hiperparámetro gamma de 0.99 a 0.95. Este hiperparámetro **es el encargado de darle mayor o menor importancia a la posible recompensa futura.** Cuanto más bajo sea, menos se tendrá en cuenta una recompensa a largo plazo y más una recompensa más inmediata. **El sentido de este experimento es precisamente demostrar, con datos, como una pequeña variación de solo 0,04 en un hiperparámetro puede conseguir unos resultados muy diferentes en comparación con los de otros experimentos,** lo cual a su vez demuestra la dificultad de encontrar un ajuste correcto de los distintos hiperparámetros que presenta el agente, de ahí la baja reproducibilidad de buenos resultados.

Las gráficas obtenidas son las siguientes:

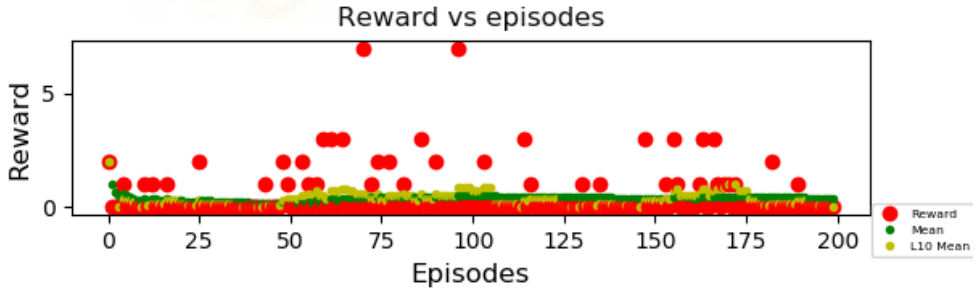

Figura 124: Gráfica experimento 2, BreakoutDeterministic-v4, sesión final

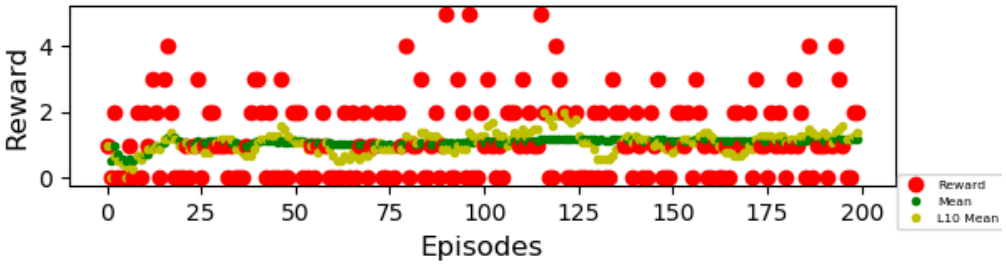

Figura 125: Resultados obtenidos experimento 2 BreakoutDeterministic-v4, sesión aleatoria

7.2.2.3 Experimento 3

Los hiperparámetros utilizados son los siguientes:

- Episodios máximos de entrenamiento: 100010
- Episodios de observación: 50
- Pasos para actualizar la red objetivo: 10000
- Tamaño de memoria: 100000
- Número de experiencias para *experience replay*: 32
- Ratio de aprendizaje: 0.00001
- Cuadros hasta epsilon mínimo: 2000000
- Epsilon final: 0.1
- Gamma: 0.99

La tabla obtenida tras la sesiones de entrenamiento es la siguiente:

Fecha	Puntuación máx	Media máx	Media L10	Media final	Loss	Media Q-Values	Tiempo sesión	Tiempo total	N.º episodios sesión	Episodios totales	Experiencias sesión	Experiencias totales
23/05/19	8	1,18	1,1	1,18	0,000196	0,588179	16:13:18	16h 13min 18s	3070	3070	607765	607765
24/05/19	7	2,4	1,2	1,4	0,000216	0,799078	15:58:59	32h 12min 17s	2830	5900	592976	1200741
25/05/19	8	2	2,4	1,53	0,001425	1,087698	15:50:31	48h 02min 48s	2620	8520	590747	1791488
26/05/19	7	1,41	2,7	1,41	0,0027	0,908643	16:24:12	64h 27min 00s	2310	10830	624451	2415939
27/05/19	8	1,9	2,3	1,77	0,000266	1,295865	16:59:26	81h 26min 26s	2040	12870	628625	3044564
28/05/19	8	2,27	0,9	2,21	0,002039	1,198143	17:07:59	98h 34min 25s	2000	14870	636800	3681364
29/05/19	7	2,38	2,3	2,11	0,002087	1,533321	16:46:31	115h 20min 56s	2060	16930	630668	4312032
30/05/19	7	2,42	1,8	2,22	0,000488	1,560521	16:33:26	131h 54min 12s	1970	18900	617111	4929143
31/05/19	10	2,42	2,5	2,31	0,001228	1,847051	40:59:07	148h 53min 19s	4790	23690	1510210	6439353
03/06/19	7	3	3,3	2,36	0,0003	1,044547	16:27:12	165h 20min 31s	1960	25690	621315	7060668
04/06/19	9	2,55	2,2	2,39	0,002107	0,761836	16:24:19	181h 44min 50s	1970	27620	617533	7678201
05/06/19	8	3	1,7	2,36	0,000649	1,021045	16:47:00	198h	2020	29640	635396	8313597

								31min 18s				
06/06/19	8	2,4	3,2	2,38	0,001457	1,092677	15:59:28	214h 31min 18s	1970	31610	601052	8914649
07/06/19	10	2,74	3	2,66	0,004047	1,725233	39:04:21	253h 35min 39s	4630	36240	1473544	10388193

Tabla 8: Resultados obtenidos durante las sesiones de entrenamiento del experimento 3

Análisis resultados sesiones de entrenamiento: la puntuación máxima alcanzada es de 10 puntos, siendo la menor de 7 puntos; en ese aspecto, no se obtienen mejoras con respecto a otros experimentos realizados. **La media máxima es de 2,74 puntos,** alcanzados en la última sesión; l**a media L10 máxima alcanzada es de 3,20 puntos**, siendo la mínima de 0,9; indicativo de que **existe una fluctuación de las puntuaciones durante las sesiones de entrenamiento.**
En cuanto a la media final, encontramos que la mínima es de 1,18 puntos, alcanzada en la primera sesión, **y la máxima es de 2,66**, alcanzada en la sesión final. **Esto podría indicar una progresión en las puntuaciones**, es decir, que el agente está aprendiendo.
El loss se mantiene en valores cercanos a cero, lo que indica un buen funcionamiento del agente.
Los Q-values se mantienen en valores aceptables; el mínimo se presenta en la primera sesión de entrenamiento, mientras que el Q-value final máximo registrado se presenta en la novena sesión.
La sesión más corta de entrenamiento tuvo una duración de 15 horas, 50 minutos y 31 segundos, mientras que la más larga fue de 40 horas, 59 minutos y 7 segundos. **El tiempo total de entrenamiento fue de 253 horas, 35 minutos y 39 segundos.**
Los episodios máximos completados durante una sesión son de 4790, y en la sesión en la que menos episodios se completaron se hicieron un total de 1960, siendo esa sesión de una duración de 16 horas, 27 minutos y 12 segundos.
El menor número de experiencias vistas corresponde a la sesión de menor duración, habiendo visto 590747 experiencias. En cambio, la sesión de más duración, registró el mayor número de experiencias, con 1510210.

A continuación **se presentan las gráficas de entrenamiento de las sesiones registradas**; las gráficas de las sesiones 1 y 3 no fueron registradas correctamente, por lo que no quedaron guardadas:

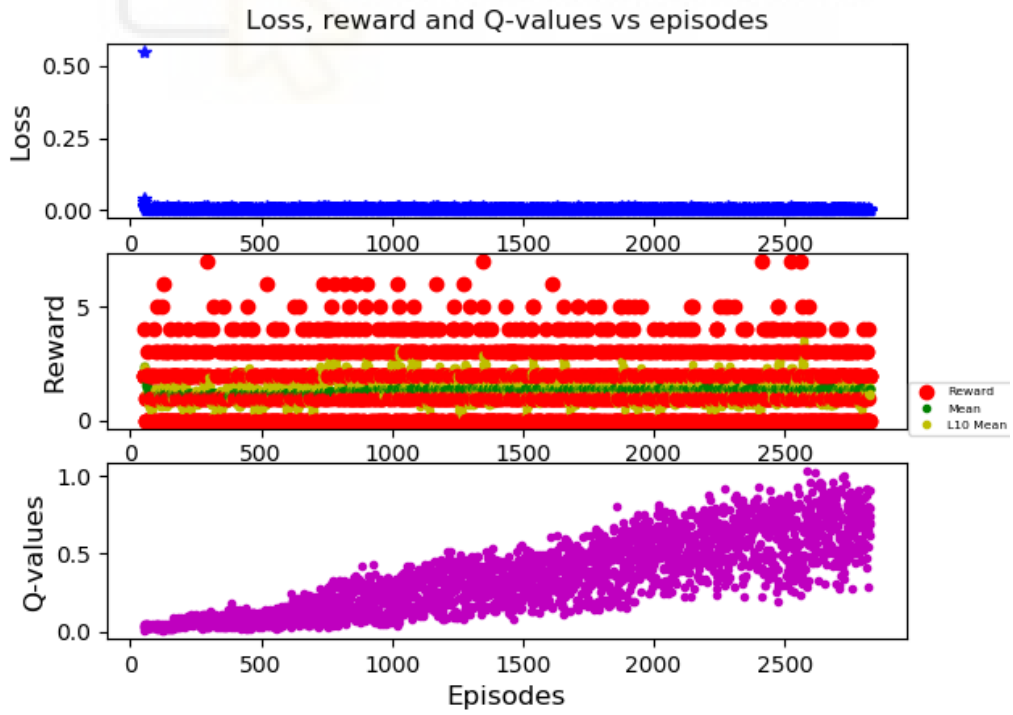

Figura 126: Grafica experimento 3 BreakoutDeterministic-v4, sesión 2

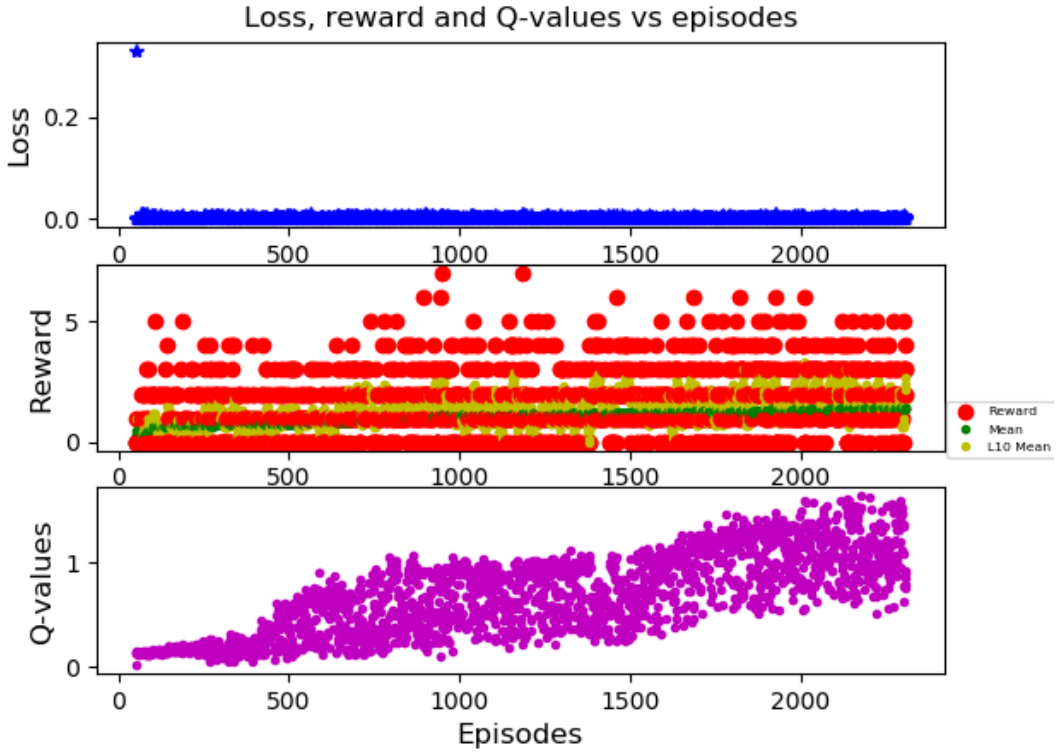

Figura 127: Gráfica experimento 3, BreakoutDeterministic-v4, sesión 4

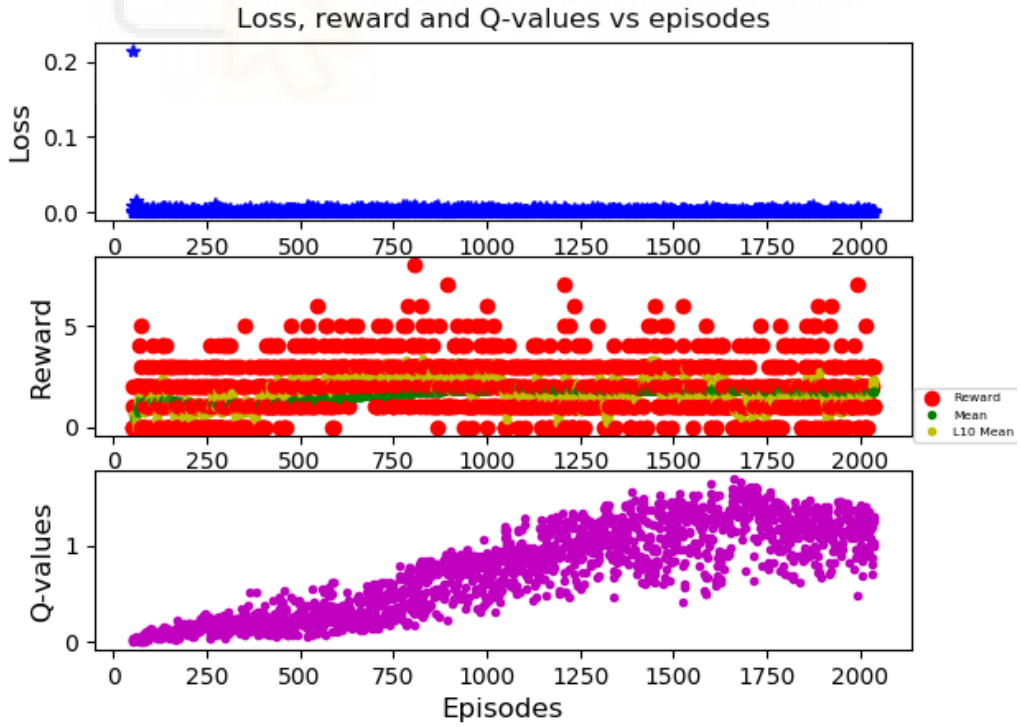

Figura 128: Gráfica experimento 3, BreakoutDeterministic-v4, sesión 5

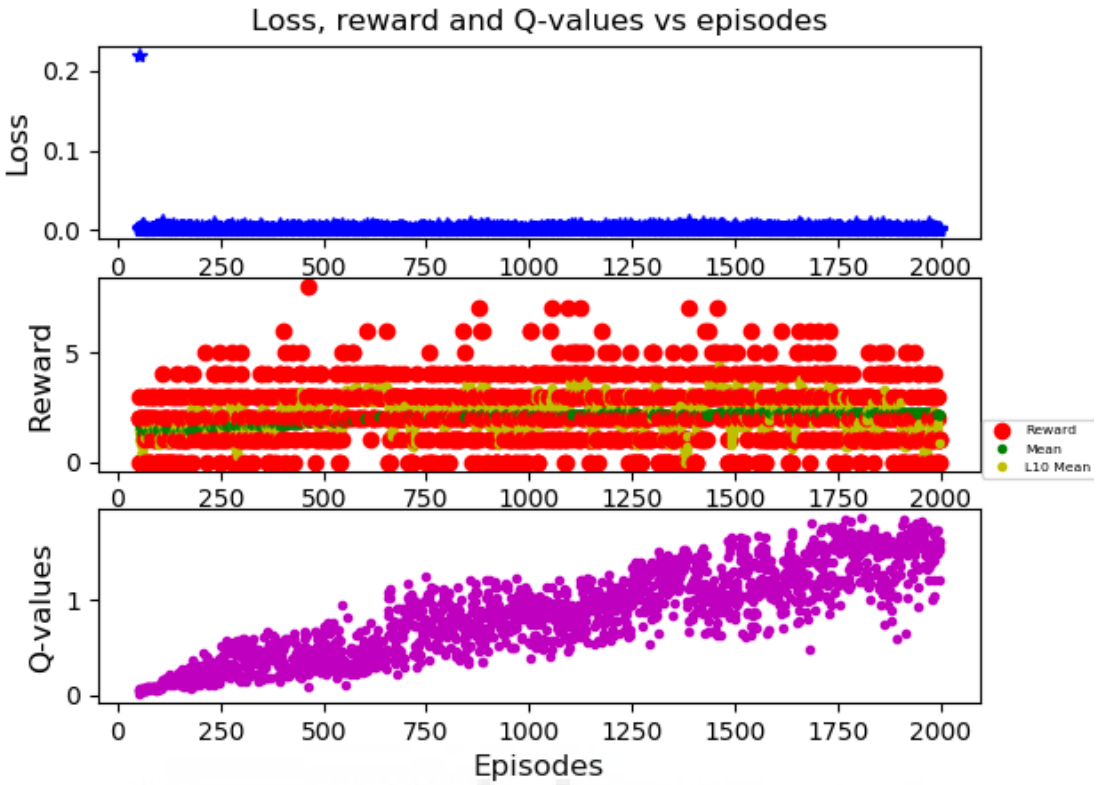

Figura 129: Gráfica experimento 3, BreakoutDeterministic-v4, sesión 6

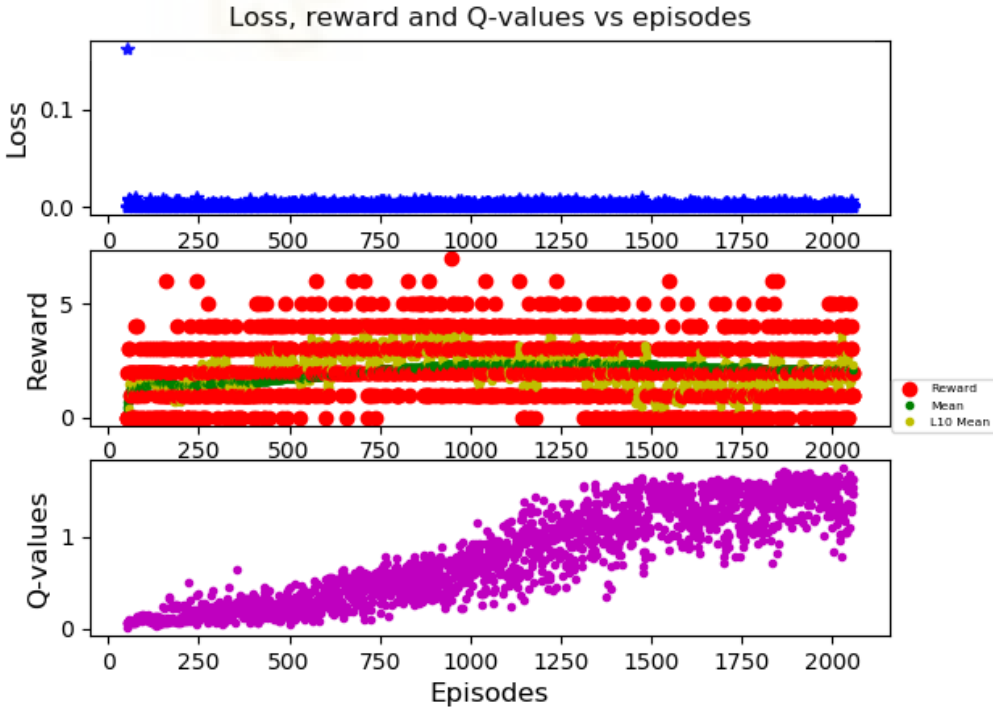

Figura 130: Gráfica experimento 3, BreakoutDeterministic-v4, sesión 7

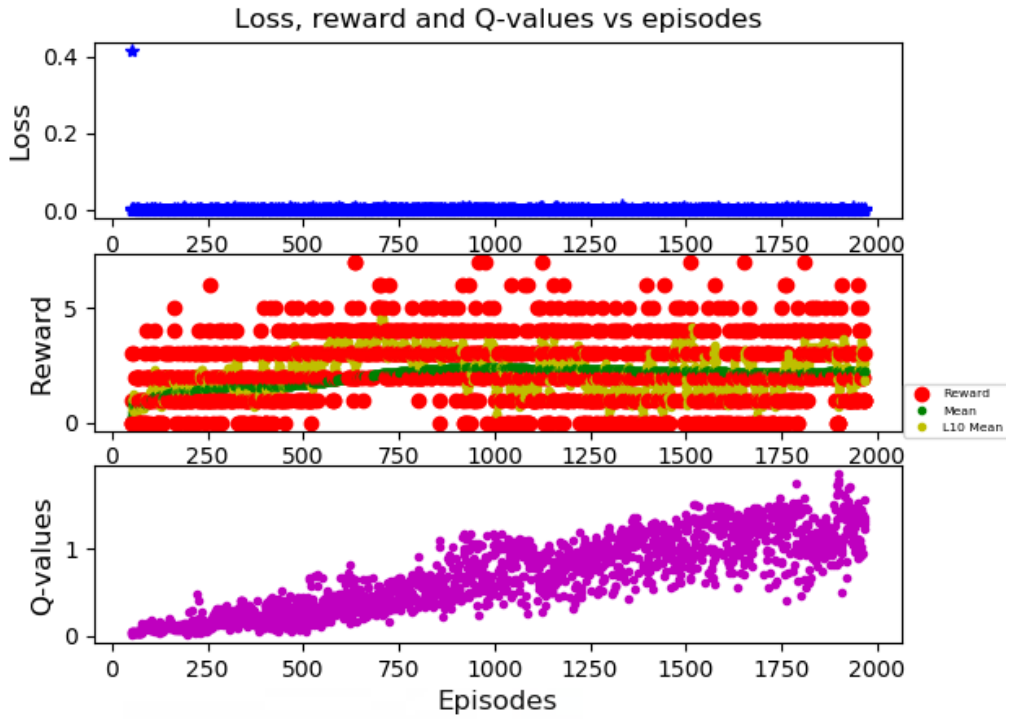

Figura 131: Gráfica experimento 3, BreakoutDeterministic-v4, sesión 8

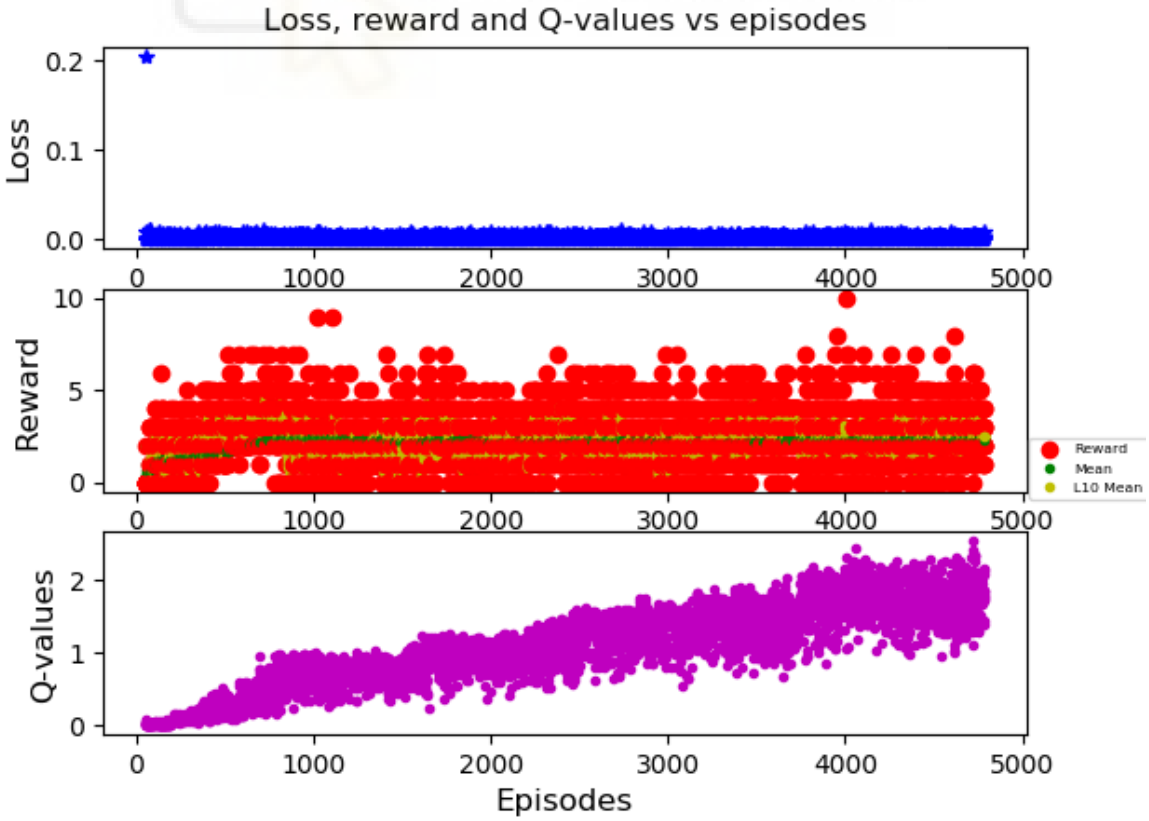

Figura 132: Gráfica experimento 3, BreakoutDeterministic-v4, sesión 9

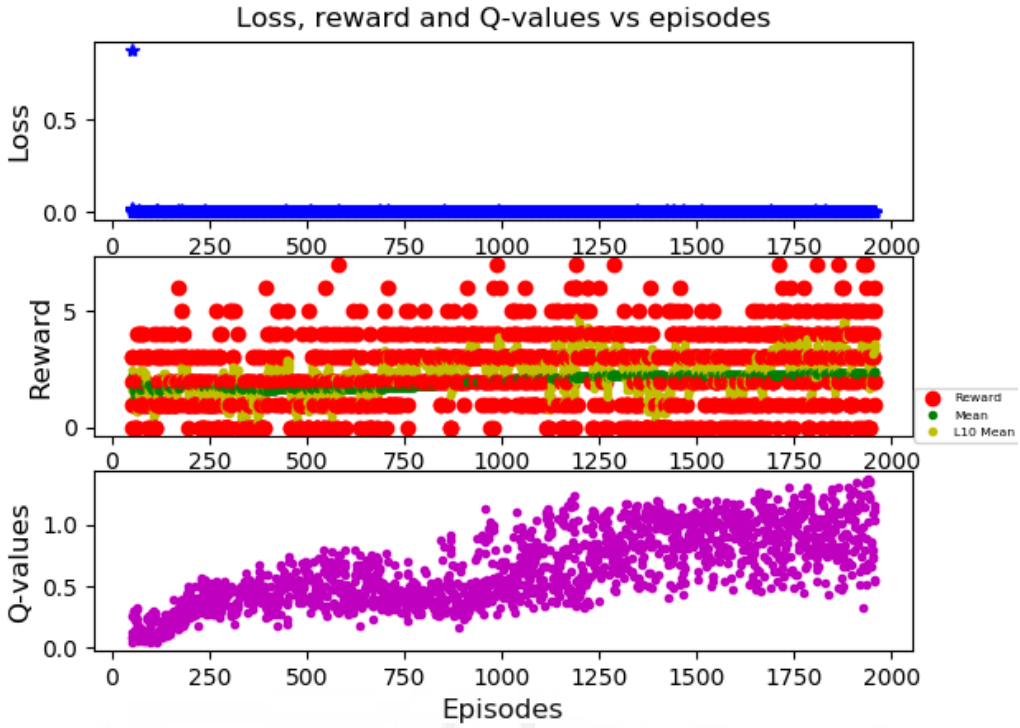

Figura 133: Gráfica experimento 3, BreakoutDeterministic-v4, sesión 10

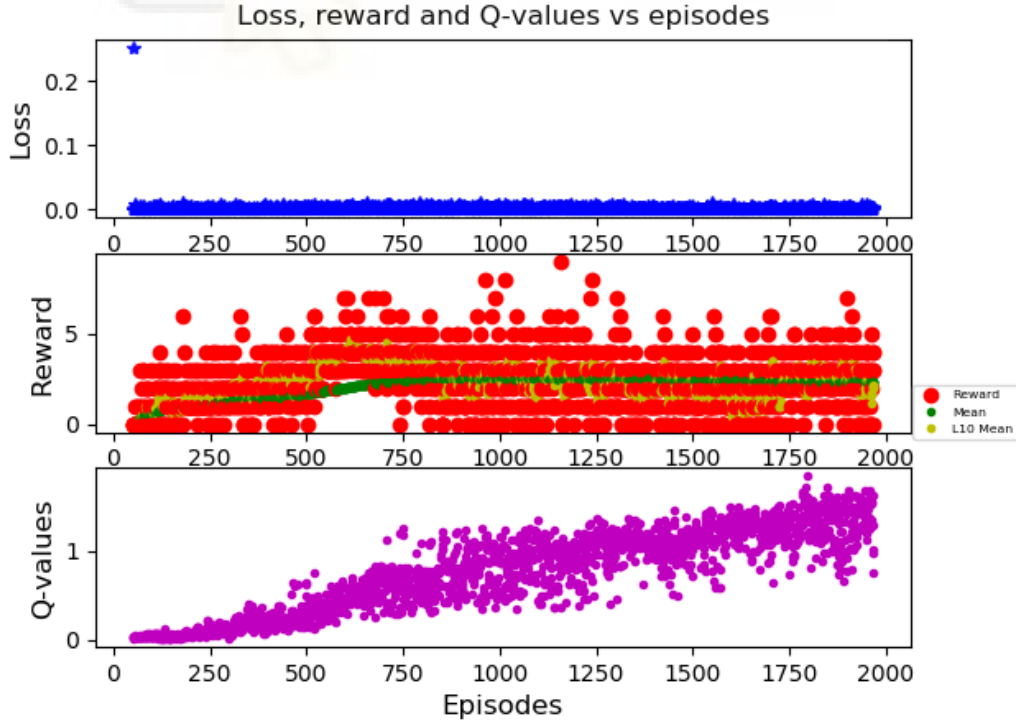

Figura 134: Gráfica experimento 3, BreakoutDeterministic-v4, sesión 11

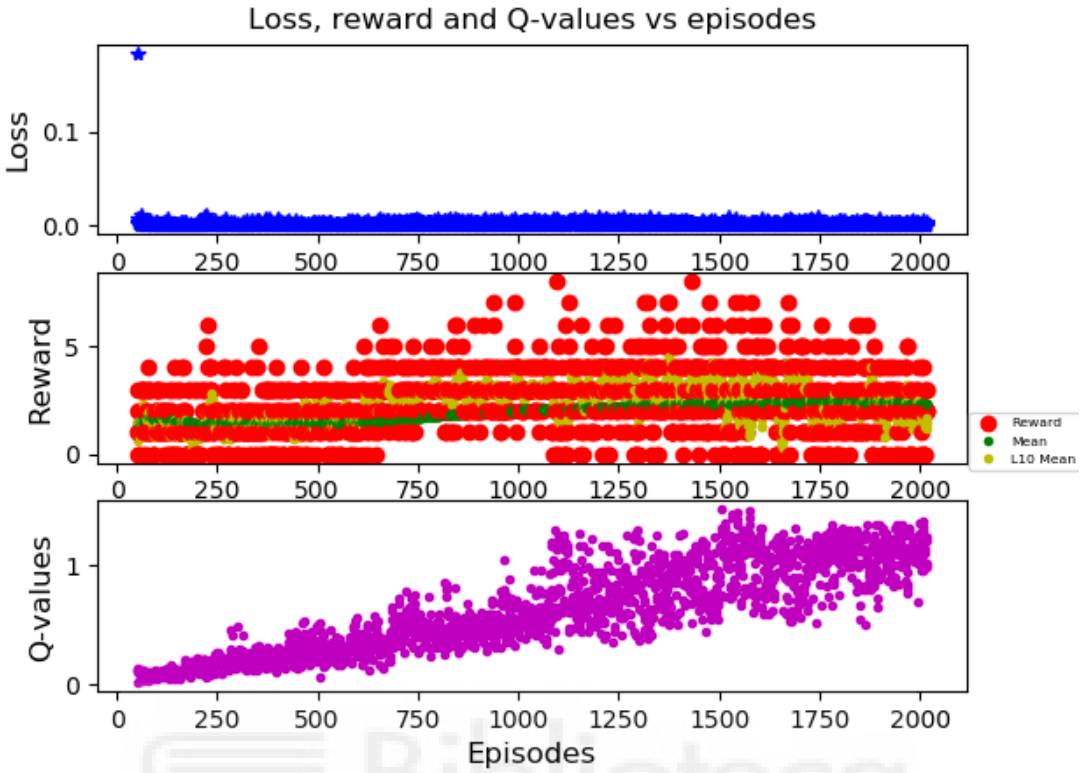

Figura 135: Gráfica experimento 3, BreakoutDeterministic-v4, sesión 12

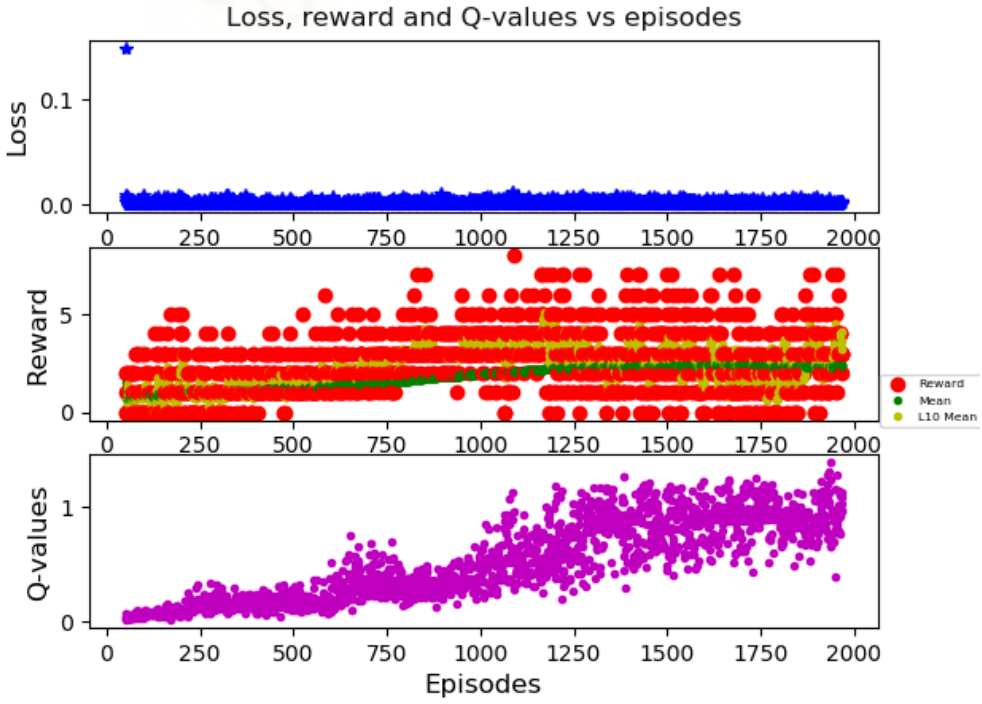

Figura 136: Gráfica experimento 3, BreakoutDeterministic-v4, sesión 13

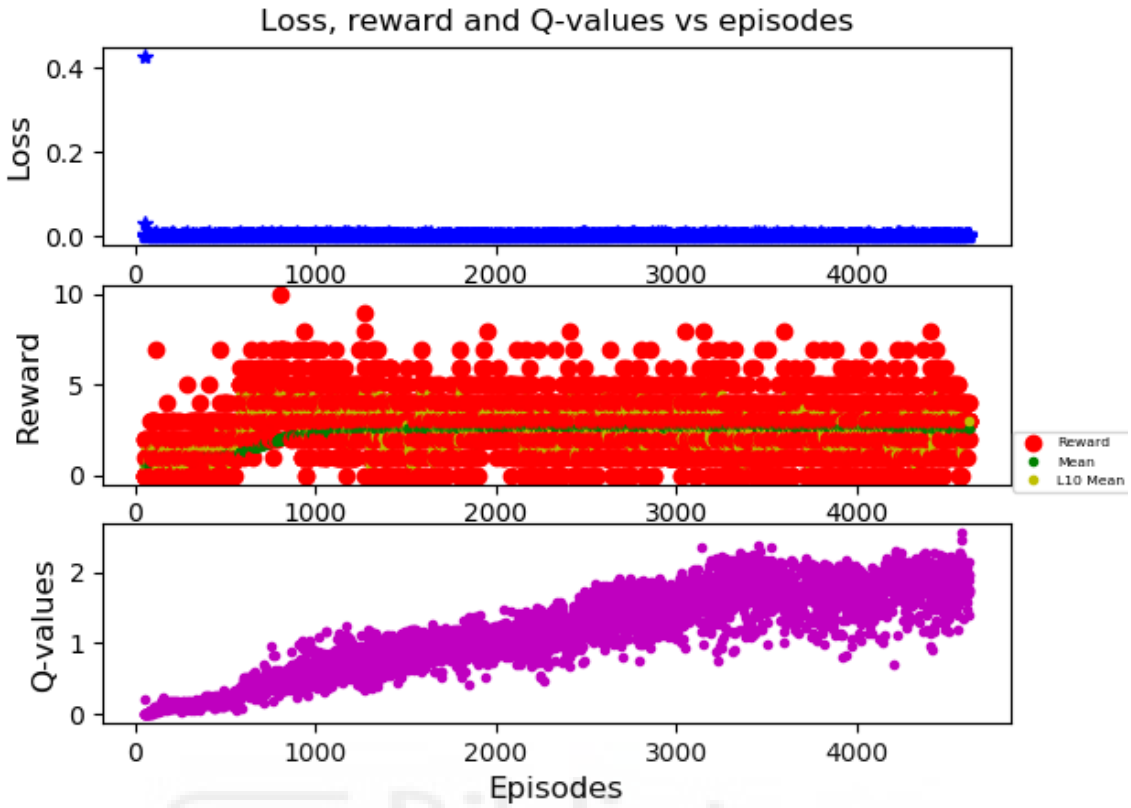

Figura 137: Gráfica experimento 3, BreakoutDeterministic-v4, sesión 14

Análisis gráficas obtenidas durante las sesiones de entrenamiento: en todas las gráficas superiores, correspondientes al ***loss***, vemos que se mantiene en **valores cercanos a cero**.

Estudiando las **puntuaciones**, **vemos como a partir de la sesión 5 encontramos grandes tramos donde no hay puntuaciones de 0**; asimismo, si analizamos **la media L10 se observa como oscilan las puntuaciones por encima y debajo de la media.** Esto indica que **las puntuaciones entre partidas no tienen continuidad**, es decir, que si en una partida obtenemos 5 puntos, en la siguiente la puntuación alcanzada no tiene por qué ser similar. La media encontramos que tiende a subir durante todas las sesiones de entrenamiento, hasta estabilizarse.

Los *Q-values* tienden a hacerse más grandes conforme entrenamos, pero también a ofrecer unas oscilaciones mayores, ya que se observa como en la gráfica conforme avanzan, la línea tiende a hacerse más gruesa.

En cuanto a los resultados obtenidos en la prueba final, se presenta la siguiente tabla:

Fecha	*Experimento*	*Puntuación máx*	*Media max*	*Media L10*	*Media final*	*N.º Episodios*
27/05/19	Aleatorio	5	1,28	1,4	1,17	200
13/06/19	Experimento 3	4	1,14	1	1,09	200

Tabla 9: Resultados obtenidos experimento 3 BreakoutDeterministic-v4, sesión final

Análisis resultados obtenidos prueba final: sorprendentemente, **a pesar de que la media final durante las sesiones de entrenamiento es mejor en este experimento que en el experimento 1, los resultados finales obtenidos son, en todos los casos, peores a la referencia** (aleatorio). Esto demuestra que los ajustes elegidos en este caso, no mejoran el resultado obtenido en el experimento 1, sino que además los empeoran.

Las gráficas obtenidas son las siguientes:

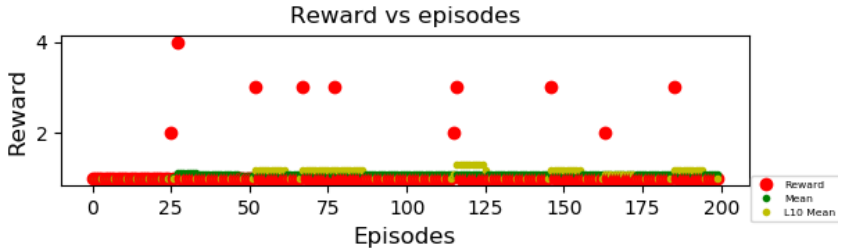

Figura 138: Resultados obtenidos experimento 3 BreakoutDeterministic-v4, sesión final

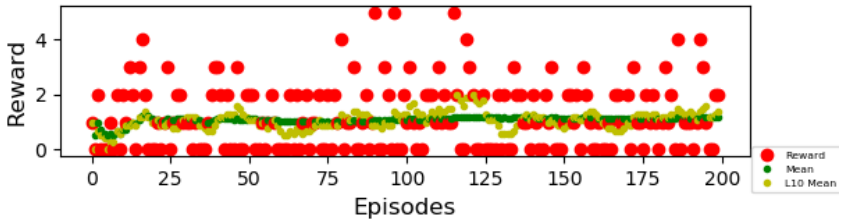

Figura 139: Resultados obtenidos experimento 3 BreakoutDeterministic-v4, sesión aleatoria

7.2.2.4 Experimento 4 (Google Colab)

Los hiperparámetros utilizados son los siguientes:

- Episodios máximos de entrenamiento: 100010
- Episodios de observación: 50
- Pasos para actualizar la red objetivo: 10000
- Tamaño de memoria: 100000
- Número de experiencias para *experience replay*: 32
- Ratio de aprendizaje: 0.00025
- Cuadros hasta epsilon mínimo: 300000
- Epsilon final: 0.1
- Gamma: 0.99

Los resultados obtenidos durante las sesiones de entrenamiento son los siguientes:

Fecha	Puntuación máx	Media máx	Media L10	Media final	Loss	Media Q-Values	Tiempo sesión	Tiempo total	N.º episodios sesión	Episodios totales	Experiencias sesión	Experiencias totales
23/05/19	7	1,33	1,1	1,32	0,000382	0,06586	11:39:09	11h 39min 09s	780	780	158554	158554
24/05/19	6	2	0,1	0,8	0	0,154145	11:40:50	23h 19min 59s	800	1580	172648	331202
25/05/19	6	1,5	0,1	0,6	0	0,019465	10:54:01	34h 14min 00s	530	2110	166634	497836
27/05/19	5	1,75	0,3	0,42	0	0,07381	11:33:33	45h 47min 33s	1130	3240	378176	876012
28/05/19	4	1	0,3	0,38	0	0,040314	11:29:45	57h 17min 18s	1160	4400	384833	1260845
29/05/19	6	3	0,44	0,4	0	0,032301	11:46:11	69h 03min 29s	1170	5570	386520	1647365
30/05/19	6	0,39	0,4	0,36	0	0,102003	11:42:53	80h 46min 12s	1160	6730	389882	2037247
31/05/19	8	1,2	0,1	0,43	0	0,041058	11:34:25	92h 20min 37s	1180	7910	390976	2428223
03/06/19	5	2,33	0,3	0,43	0	0,045946	11:20:32	103h 41min 09s	1150	9060	385711	2813934
04/06/19	6	0,67	0,9	0,37	0,000001	0,071057	11:13:41	114h 54min 50s	1160	10220	389914	3203848
05/06/19	5	1	0,5	0,39	0,000002	0,04333	11:29:44	126h 23min 34s	1220	11440	406317	3610165
06/06/19	5	1	0,3	0,39	0	0,045795	11:17:07	137h 40min	1110	12550	369792	3979957

								41s				

Tabla 10: Resultados obtenidos durante las sesiones de entrenamiento del experimento 4 (Google Colab)

Análisis resultados sesiones de entrenamiento: las puntuaciones máximas oscilan entre 7 y 4; puntuaciones inferiores a las obtenidas en otros experimentos. **La media máxima** obtenida está entre 2 y 0,39. Cabe destacar el valor mínimo (0,39) siendo muy bajo.

La media L10 (la media de las 10 últimas partidas) se encuentra entre 1,1 y 0,1, siendo también valores muy bajos.

Para los valores de las **medias finales**, encontramos que el máximo ha sido de 1,32 y el mínimo de 0,36. Cabe destacar que en la sesión final, el resultado obtenido es de 0,39 en la media final, y en la primera sesión es cuando se consigue el máximo.

Con todos estos datos, podemos asegurar que **el agente, con estos ajustes, no está funcionando correctamente, ni consigue extraer información del entorno.** Asimismo, se demuestra como unos pequeños cambios, pueden dar lugar al mal funcionamiento del agente, demostrando la dificultad de conseguir resultados satisfactorios así como la baja reproducibilidad de los mismos.

En cuanto al *loss*, los valores se mantienen muy cercanos a cero. Cabe destacar que en muchos casos el valor obtenido es 0; puesto que el programa solo recoge los seis últimos decimales (se han considerado más que suficientes), los valores 0 probablemente tendrán un valor muy pequeño.

Estudiando **los *Q-values*** medios finales, se observa que **son valores muy pequeños**; tanto el máximo como el mínimo se encuentran por debajo del 1 (0,154145 y 0,032301 respectivamente).

Los tiempos de sesión, son inferiores en todos los casos a 12 horas debido a que es el límite que se puede ejecutar un programa en *google colab*; por ello, la sesión máxima tiene una duración de 11 horas, 46 minutos y 11 segundos, siendo el menor tiempo registrado de 10 horas, 54 minutos y 1 segundo. El tiempo total acumulado entre todas las sesiones es de 137 horas, 40 minutos y 41 segundos.

Los episodios máximos vistos oscilan entre 1220 y 530; la sesión con menos episodios (530) coincide con la de menor duración.

Para las experiencias vistas en cada sesión, los valores obtenidos están entre 158554 para la sesión en la que se vio un menor número de experiencias (correspondiente a la primera sesión), siendo el máximo de 406317, sesión en la que se obtuvieron un mayor número de episodios.

Las gráficas obtenidas durante las sesiones de entrenamiento son las siguientes:

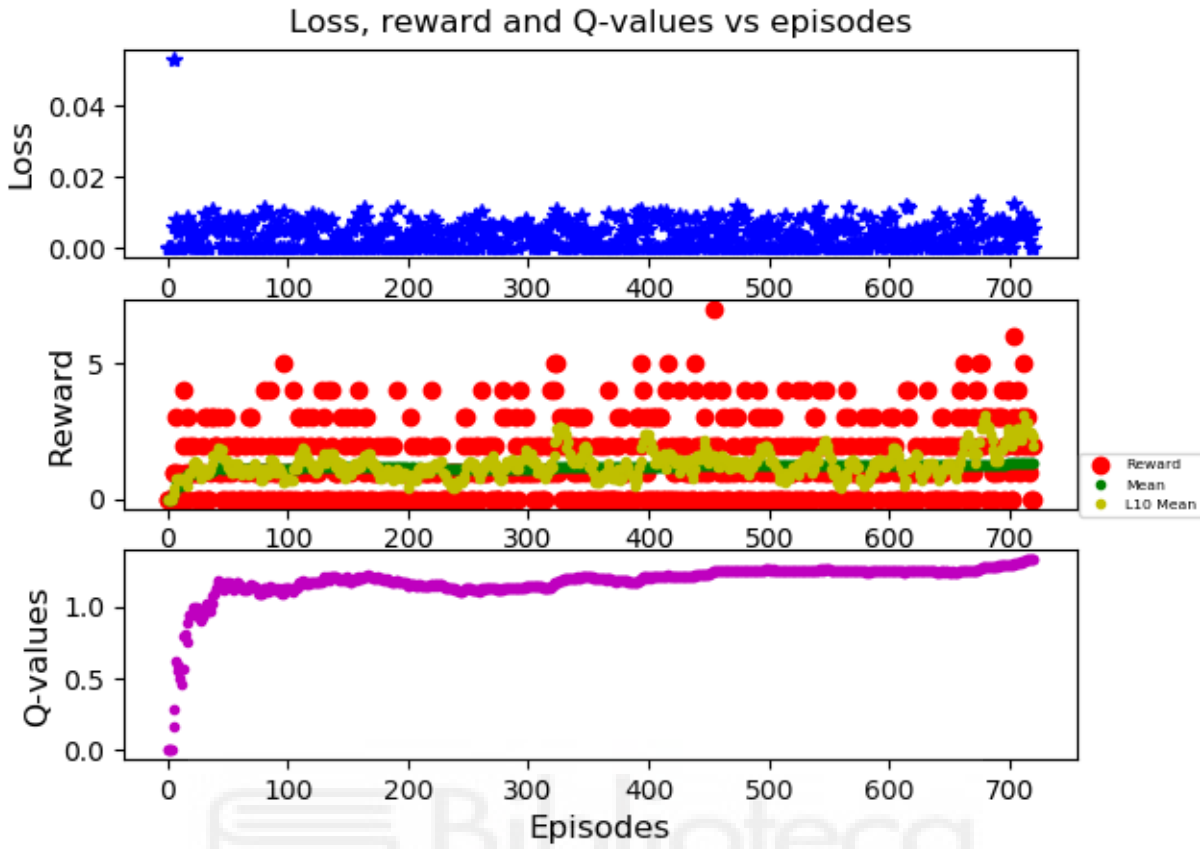

Figura 140: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 1

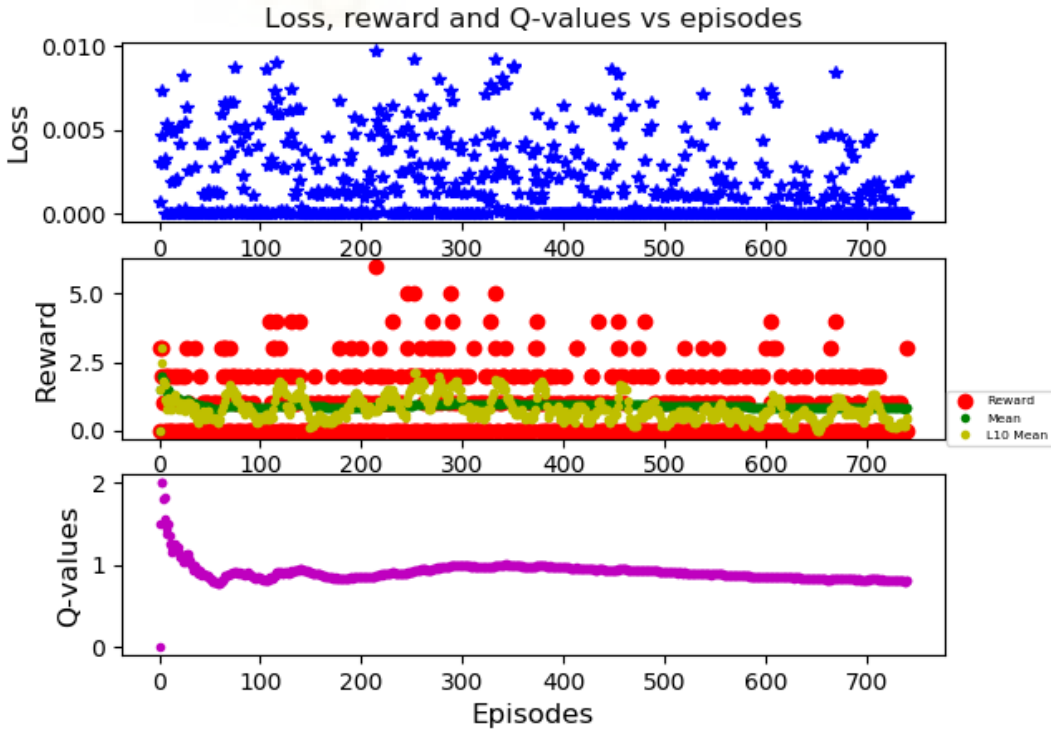

Figura 141: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 2

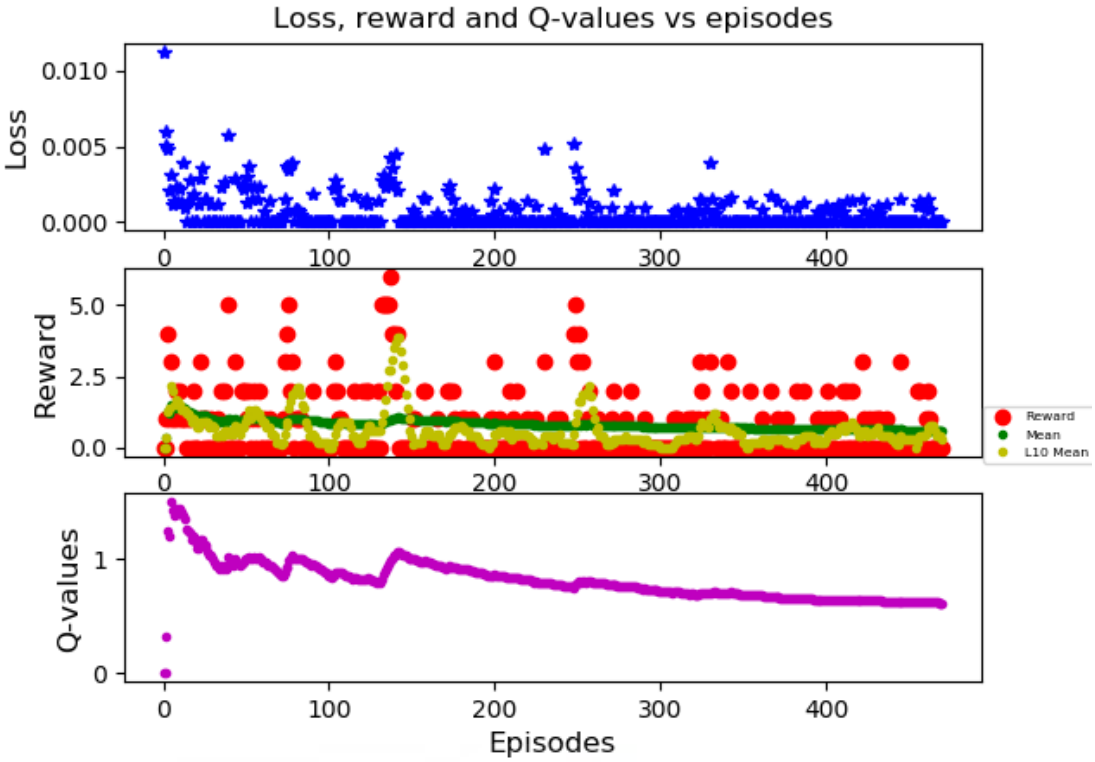

Figura 142: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 3

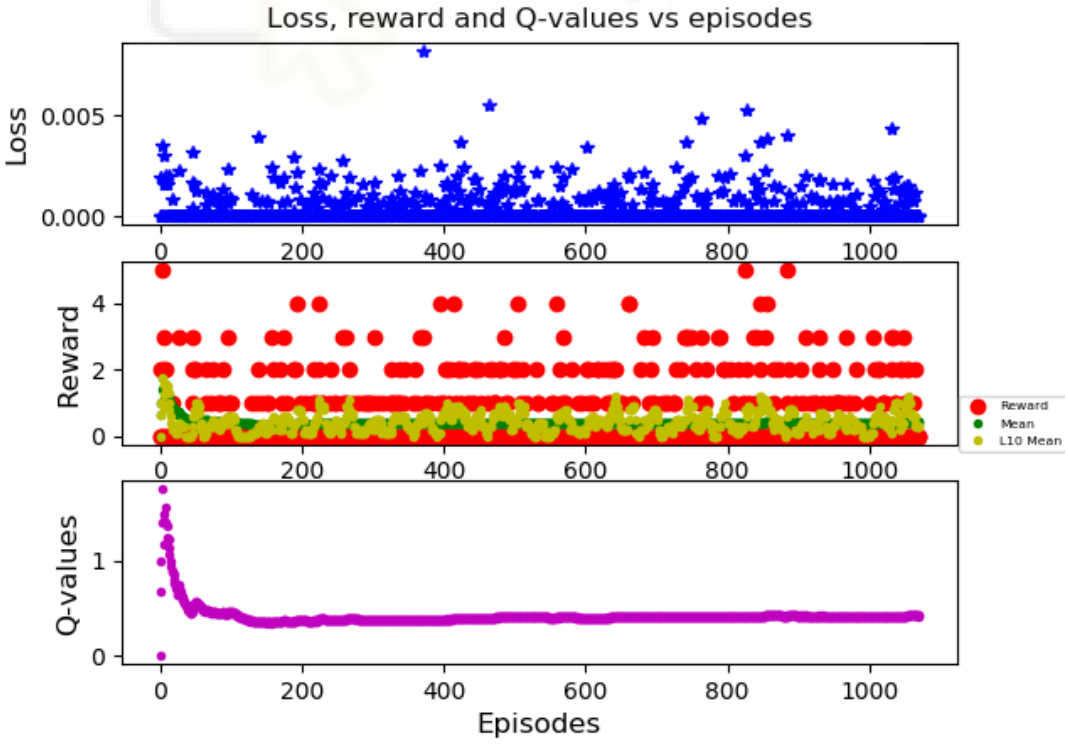

Figura 143: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 4

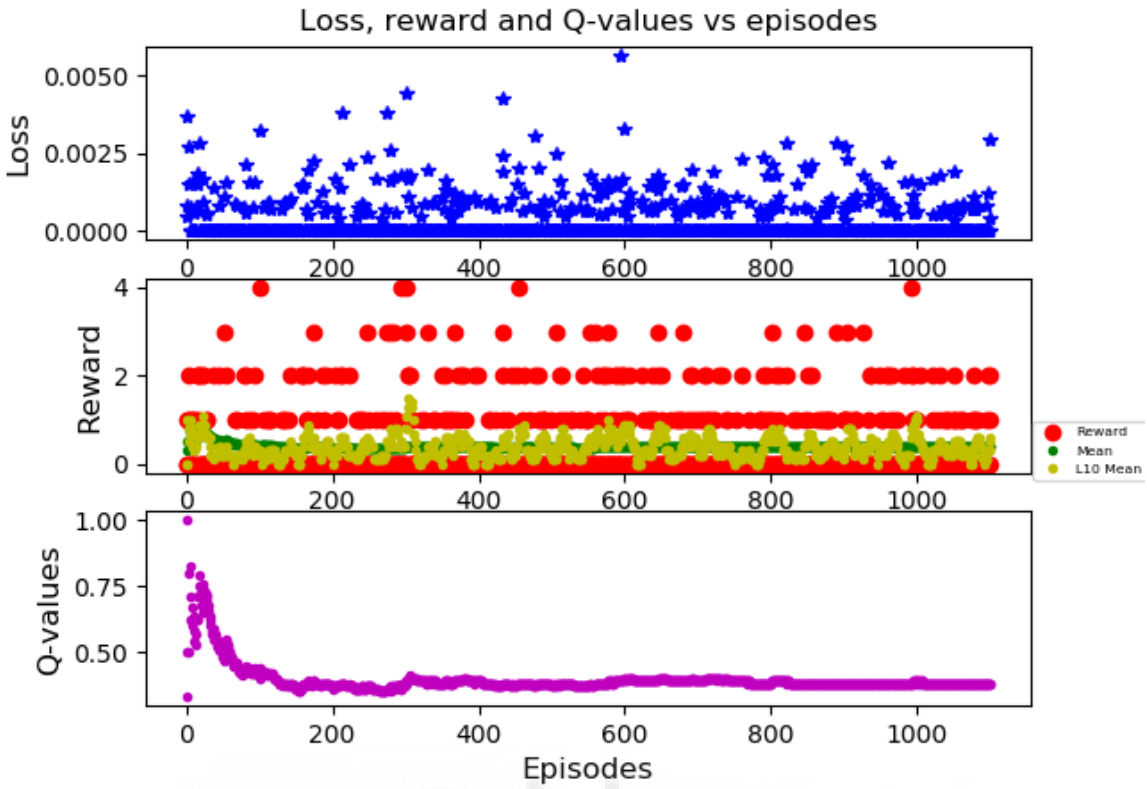

Figura 144: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 5

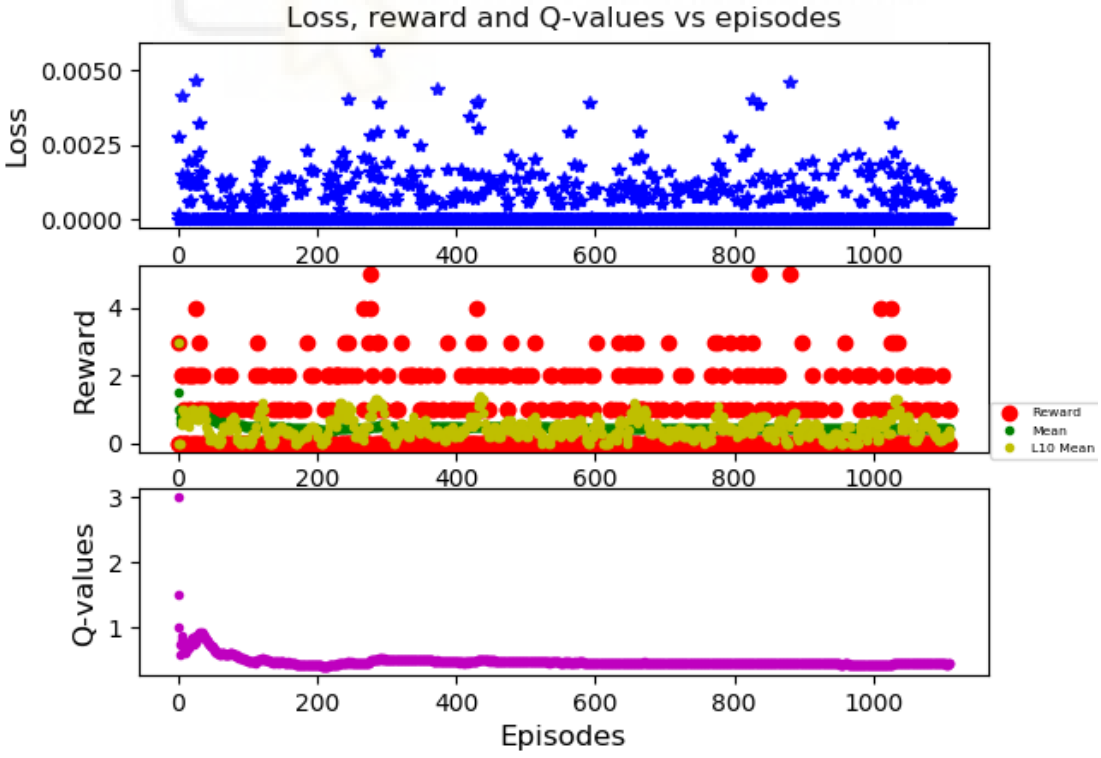

Figura 145: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 6

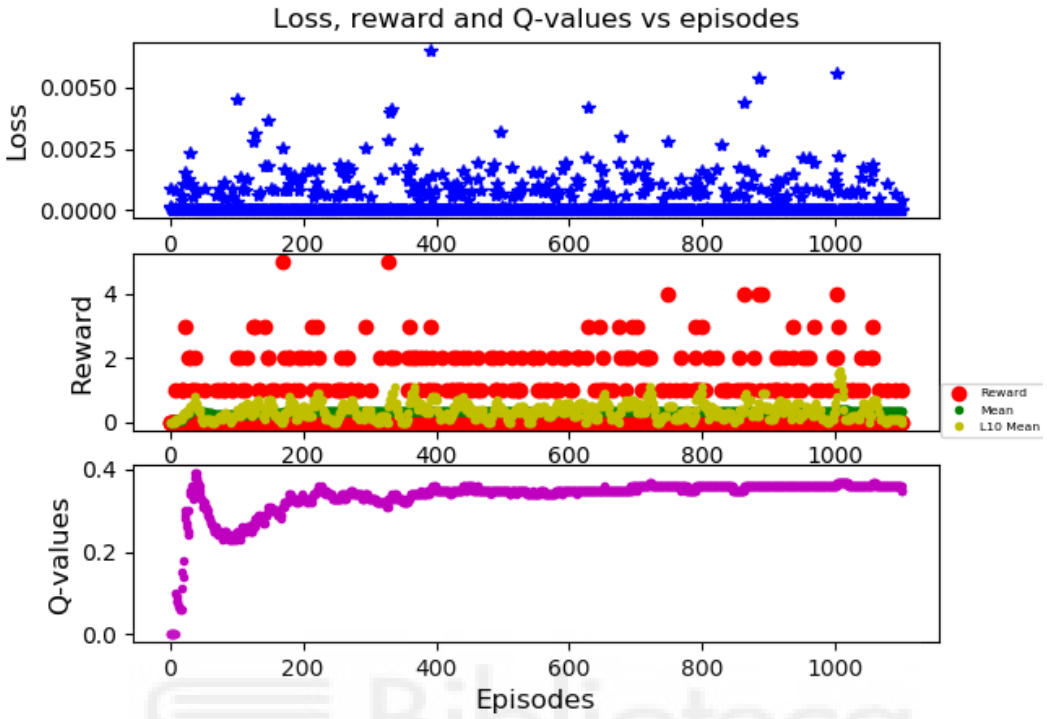

Figura 146: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 7

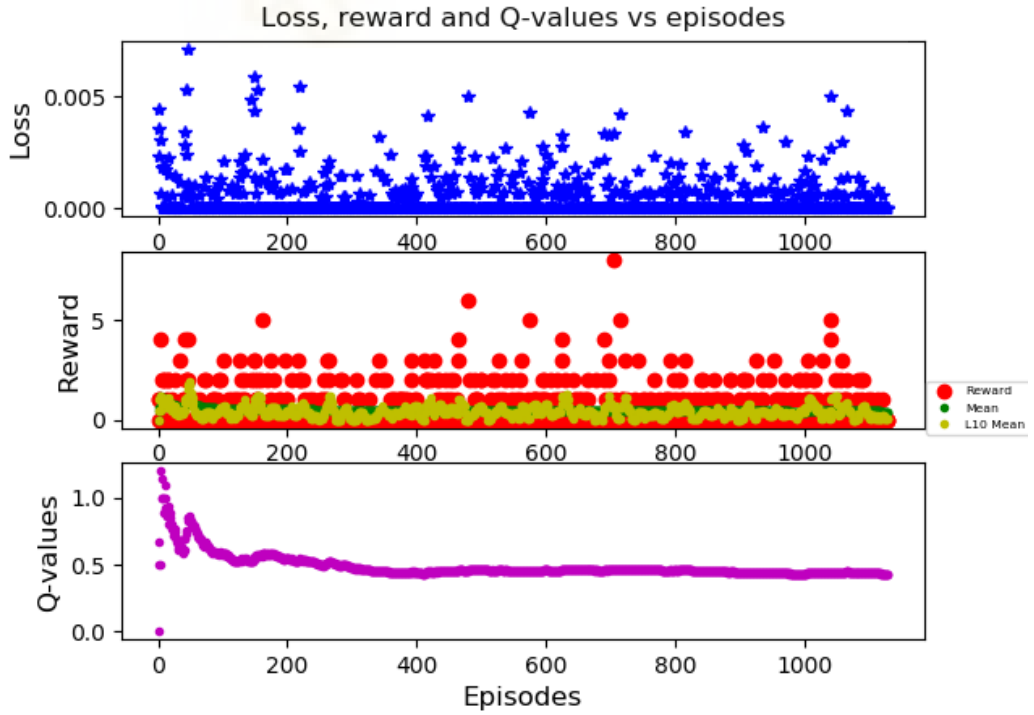

Figura 147: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 8

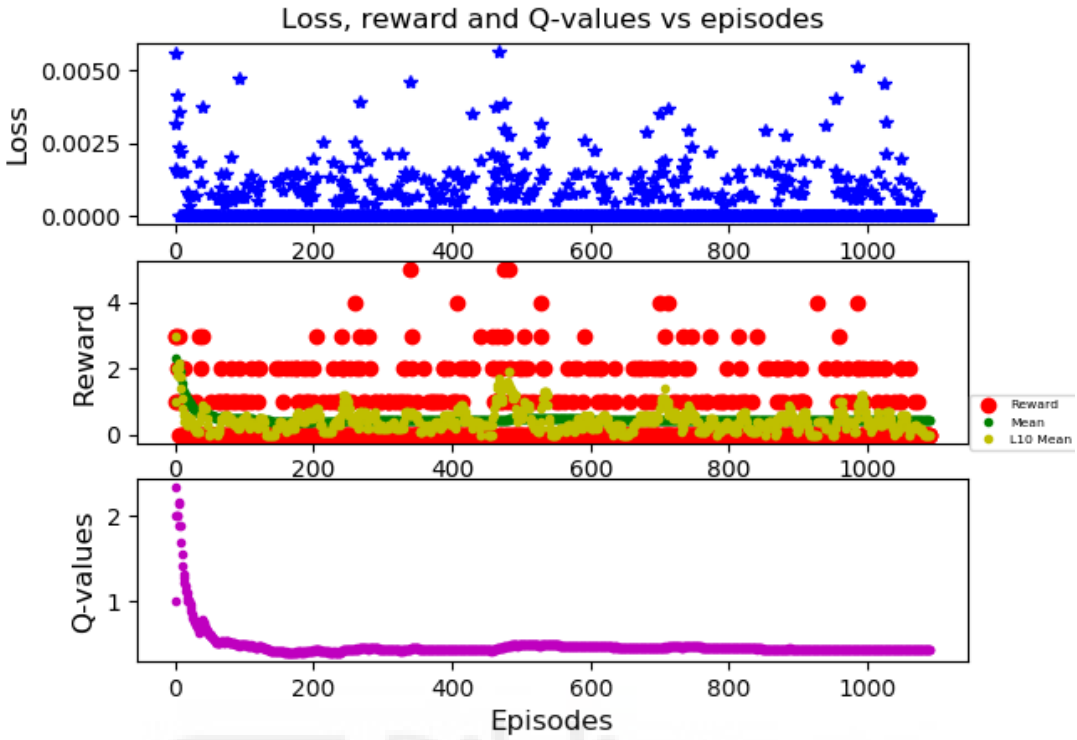

Figura 148: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 9

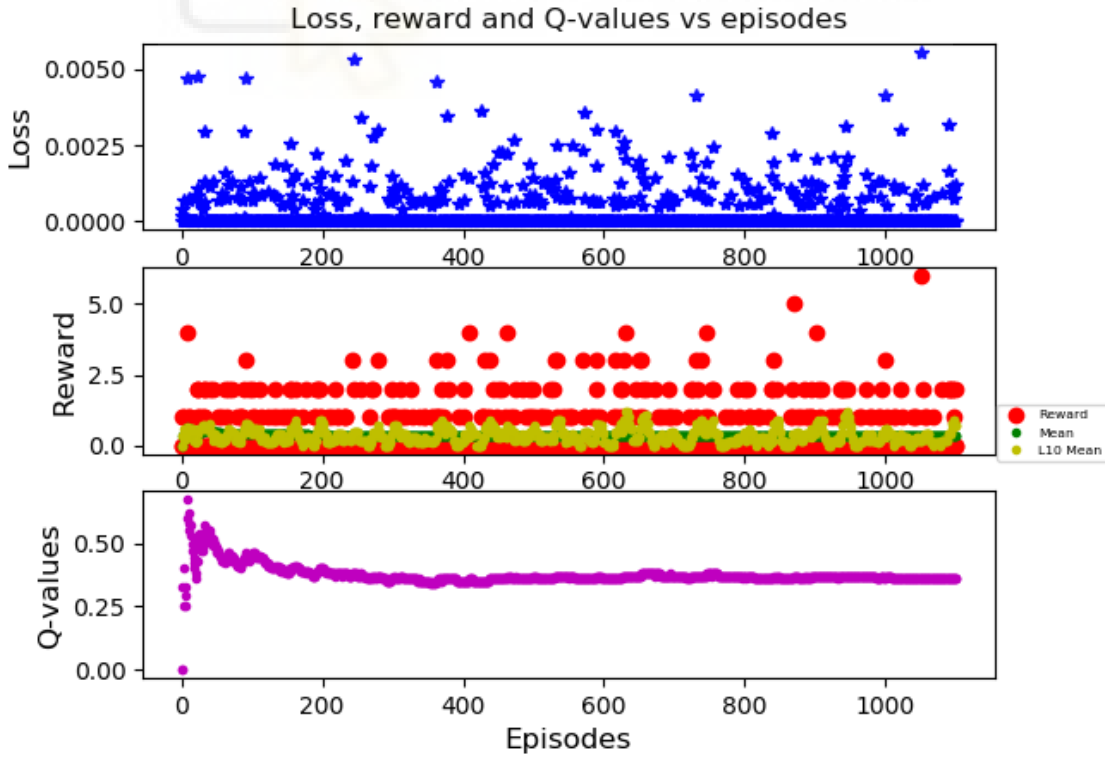

Figura 149: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 10

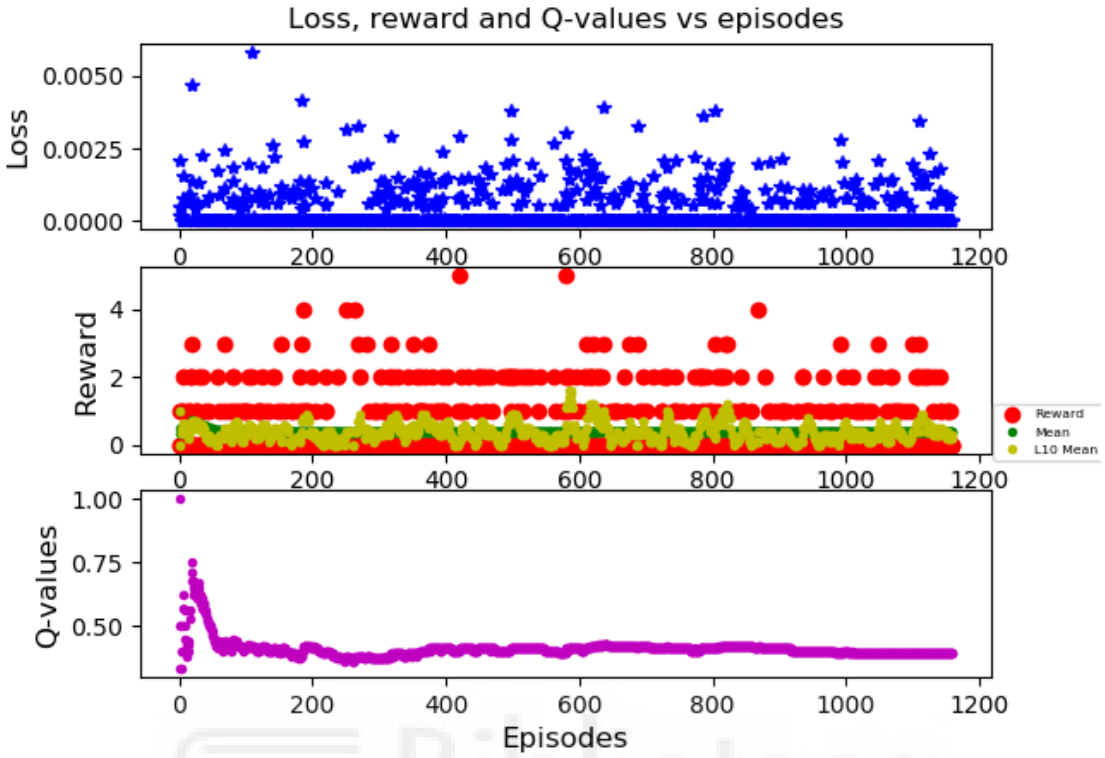

Figura 150: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 11

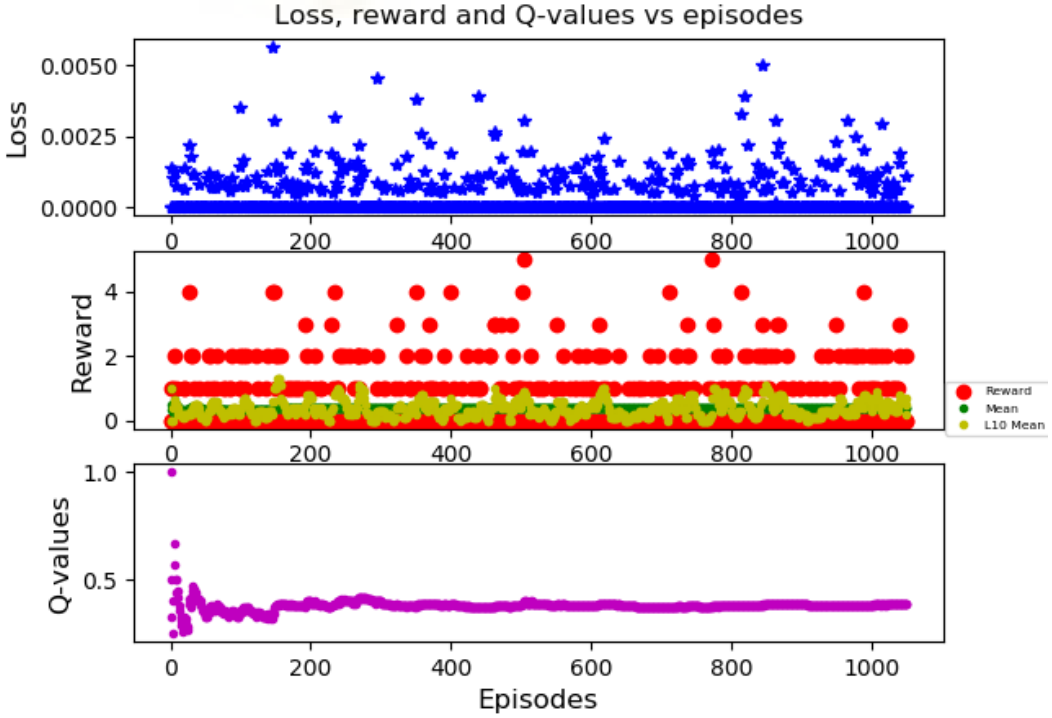

Figura 151: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 12

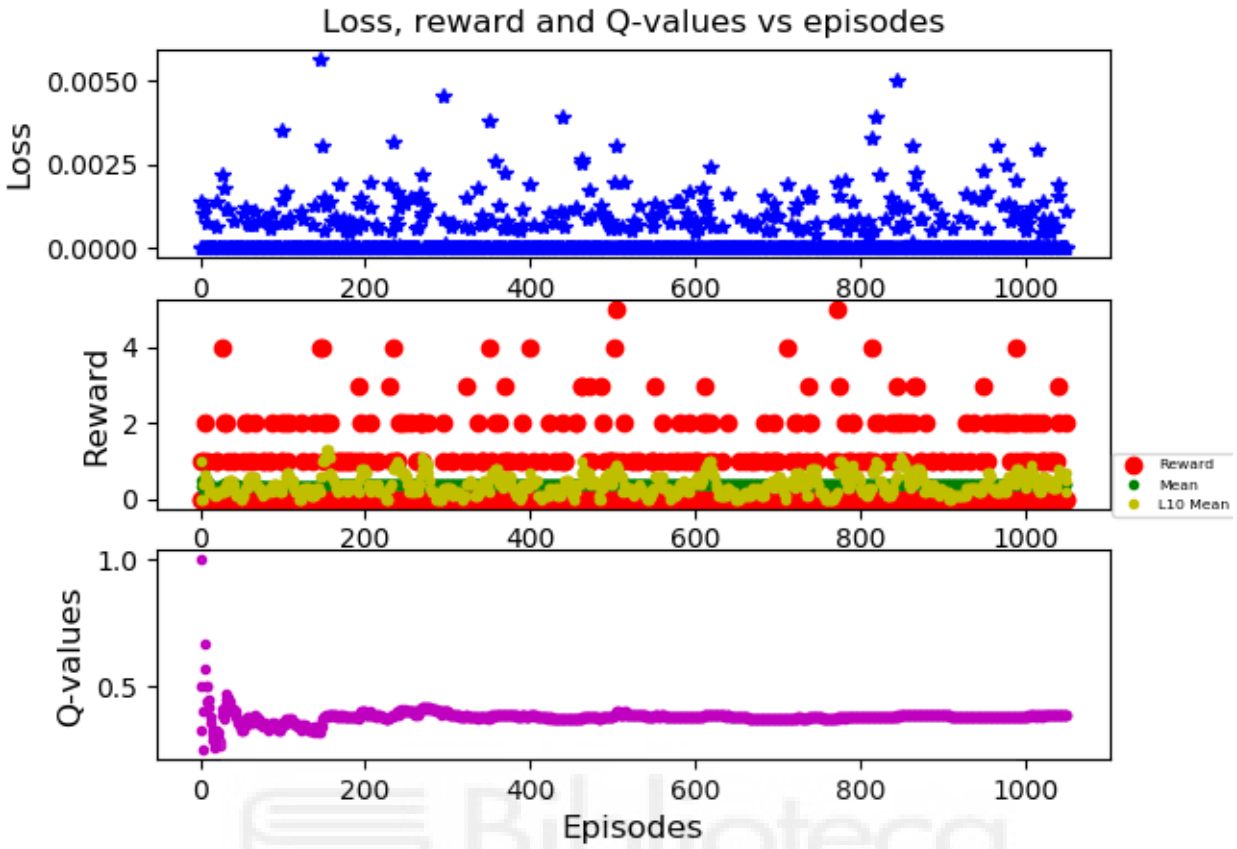

Figura 152: Gráfica experimento 4, BreakoutDeterministic-v4, sesión 13

Análisis gráficas obtenidas durante las sesiones de entrenamiento: los valores del *loss* se mantienen en valores correctos, cercanos a cero en todas las sesiones, sin grandes variaciones.

Las puntuaciones, como se desprende da la información recogida en la tabla, **son bastante bajas**, en las que abunda partidas con puntuaciones de 0; por ello, **tanto la media como la media L10 son bastante bajas.** La media es una línea prácticamente recta en la mayoría de sesiones, lo que implica que no existe un aprendizaje. **La media L10 presenta oscilaciones**, característica presente en otros experimentos.

En cuanto a los *Q-values*, se observa que en la mayoría de sesiones presentan unas pequeñas oscilaciones al principio de la sesión, para luego estabilizarse en una línea prácticamente recta.

Los **resultados** obtenidos en la **sesión final** son los siguientes:

Fecha	*Experimento*	*Puntuación máx*	*Media max*	*Media L10*	*Media final*	*N.º Episodios*
27/05/19	Aleatorio	5	1,28	1,4	1,17	200
13/06/19	Experimento 4	7	1,25	0,3	0,38	200

Tabla 11: Resultados obtenidos experimento 4 BreakoutDeterministic-v4, sesión final

Análisis resultados obtenidos sesión final: si bien la puntuación máxima es mayor por dos puntos a la del experimento aleatorio, y la media máxima es similar (1,28 en el caso aleatorio frente a 1,25 en el caso de nuestro agente con estos ajustes), **los valores de la media L10 y la media final se desploman a valores por debajo de uno** (0,3 para la media L10 y 0,38 para la media final) frente a los obtenidos aleatoriamente (1,4 y 1,17 respectivamente).

Por tanto, con estos ajustes, se observa que **el agente no funciona correctamente.**

Las gráficas obtenidas durante la sesión final **son las siguientes:**

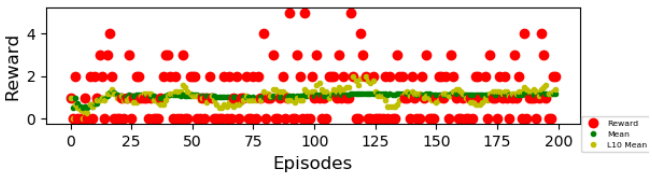

Figura 153: Resultados obtenidos experimento 4 BreakoutDeterministic-v4, sesión aleatoria

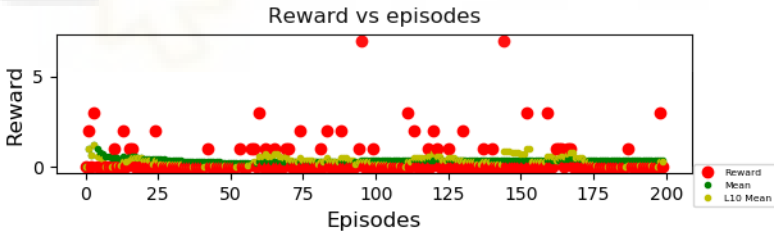

Figura 154: Resultados obtenidos experimento 4 BreakoutDeterministic-v4, sesión final

Análisis gráficas obtenidas sesión final: vemos como, si bien l**a línea de la media es plana en ambos casos** (no existe un aprendizaje), en el caso del agente aleatorio, su valor medio es más alto que en el caso de nuestro agente.

Encontramos también que **el agente aleatorio presenta menos puntuaciones cero que el agente entrenado.** Asimismo, **la media L10, fluctúa en el agente aleatorio,** mientras que en el agente entrenado prácticamente se mantiene con la media; esto es indicativo de que no existe ni empeoramiento ni mejoría.

7.2.2.5 Experimento 5

Los hiperparámetros utilizados son los siguientes:

- Episodios máximos de entrenamiento: 100010
- Episodios de observación: 50
- Pasos para actualizar la red objetivo: 10000
- Tamaño de memoria: 200000
- Número de experiencias para *experience replay*: 32
- Ratio de aprendizaje: 0.00025
- Cuadros hasta epsilon mínimo: 1000000
- Epsilon final: 0.1
- Gamma: 0.99

Los resultados obtenidos durante las sesiones de entrenamiento se presentan en la siguiente tabla:

Fecha	Puntuación máx	Media máx	Media L10	Media final	Loss	Media Q-Values	Tiempo sesión	Tiempo total	N.º episodios sesión	Episodios totales	Experiencias sesión	Experiencias totales
10/06/19	9	1,35	1,7	1,34	0,000018	0,289223	15:43:12	15h 43min 12s	2870	2870	586841	586841
11/06/19	8	2	2,1	1,37	0,002217	0,413951	16:19:00	32h 02min 12s	2580	5450	611487	1198328
12/06/19	9	1,92	1,1	1,52	0,000702	0,321952	16:20:39	48h 22min 51s	2100	7550	612355	1810683
13/06/19	7	2,36	1,2	1,54	0	0,279588	16:17:30	64h 40min 21s	2100	9650	603904	2414587
14/06/19	10	2,27	0,4	1,75	0	0,33533	17:20:26	82h 00min 47s	2200	11850	656446	3071033
17/06/19	8	2,92	1,9	1,75	0,002554	0,313402	17:39:03	99h 39min 50s	2230	14080	664601	3735634
18/06/19	8	3	1,3	1,72	0,000016	0,224045	17:28:46	117h 08min 36s	2150	16230	652725	4388359
19/06/19	8	2,25	2,7	1,65	0,001164	0,26095	17:10:51	134h 19min 27s	2160	18390	646904	5035263
20/06/19	8	2,5	0,6	1,34	0,004905	0,255127	16:52:04	151h 11min 31s	2150	20540	625912	5661175
21/06/19	8	3,59	0,1	1,62	0,000977	0,334712	21:38:58	172h 50min 29s	2590	23130	818921	6480096
25/06/19	8	3,62	4,5	1,82	0,498905	0,002579	24:41:03	197h 31min 32s	2960	26090	924653	7404749
25/06/19	8	3	0	1,71	0	0,266732	15:11:52	212h 43min 24s	1920	28010	572598	7977347

26/06/19	8	2,1	3,1	1,66	0,0262299	0,002426	18:43:57	231h 27min 21s	2240	30250	706658	8684005
27/06/19	6	2,65	2,1	1,43	0,264239	0,00378	15:20:18	246h 47min 39s	1960	32210	561313	9245318
28/06/19	8	3,36	3,6	1,59	0,272463	0,004885	19:12:09	265h 59min 48s	2410	34620	732437	9977755

Tabla 12: Resultados obtenidos sesiones de entrenamiento experimento 5

Análisis resultados obtenidos durante las sesiones de entrenamiento: el entrenamiento ha ocupado **15 sesiones de entrenamiento,** en las cuales **la puntuación más alta corresponde a 10 puntos, y la más baja a 6. La puntuación media máxima más alta fue de 3,62 puntos, mientras que la más baja fue de 1,35;** como se aprecia, existe una diferencia de más de 2 puntos entre ellas.

Para los valores de **la media L10** nos encontramos con el mismo caso: **existe una gran diferencia entre las puntuaciones máximas y mínimas** (4,5 y 0 puntos respectivamente), lo cual nos lleva a pensar que **existe una gran oscilación de los valores, probablemente debido al ratio de aprendizaje de la red neuronal.**

En cuanto a **la media final,** los valores son bastante parecidos, aún así, bastante bajos: 1,82 máximo y 1,34 de valor mínimo.

El loss se encuentra en valores aceptables, cercanos a cero, lo cual indica que no hay ningún tipo de problema en la red neuronal.

Al analizar **la media de los *Q-values*,** nos encontramos con el mismo caso que en las puntuaciones: existe una gran variación entre el valor máximo obtenido (0,413951) y el mínimo (0,002426).

Los tiempos por sesión se encuentran en valores similares a otros experimentos, siendo el tiempo por sesión más alto superior a 24 horas, y el más corto de poco más de 15 horas. El tiempo total acumulado entre sesiones es de casi 266 horas.

Los episodios máximos y mínimos vistos durante todas las sesiones de entrenamiento coinciden con los tiempos máximos y mínimos por sesión, siendo el máximo de 2960 episodios y el mínimo de 1920.

Por último, para el número de experiencias vistas, encontramos que el máximo (924653 experiencias) corresponde a la sesión más larga, mientras que el mínimo (561313 experiencias) corresponde a la sesión en la que se alcanzó la puntuación mínima, no a la de menor tiempo.

Se presentan a continuación las **gráficas registradas durante las sesiones de entrenamiento:**

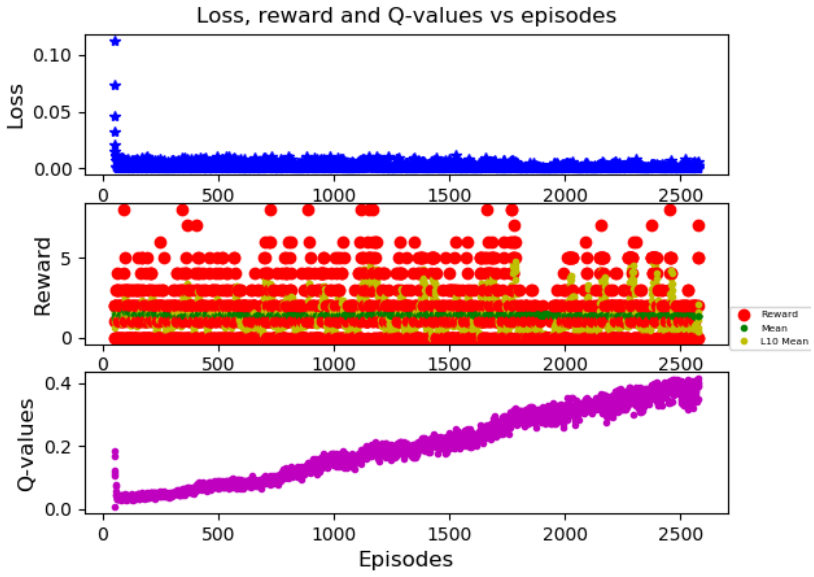

Figura 155: Gráfica sesión 2 experimento 5 BreakoutDeterministic-v4

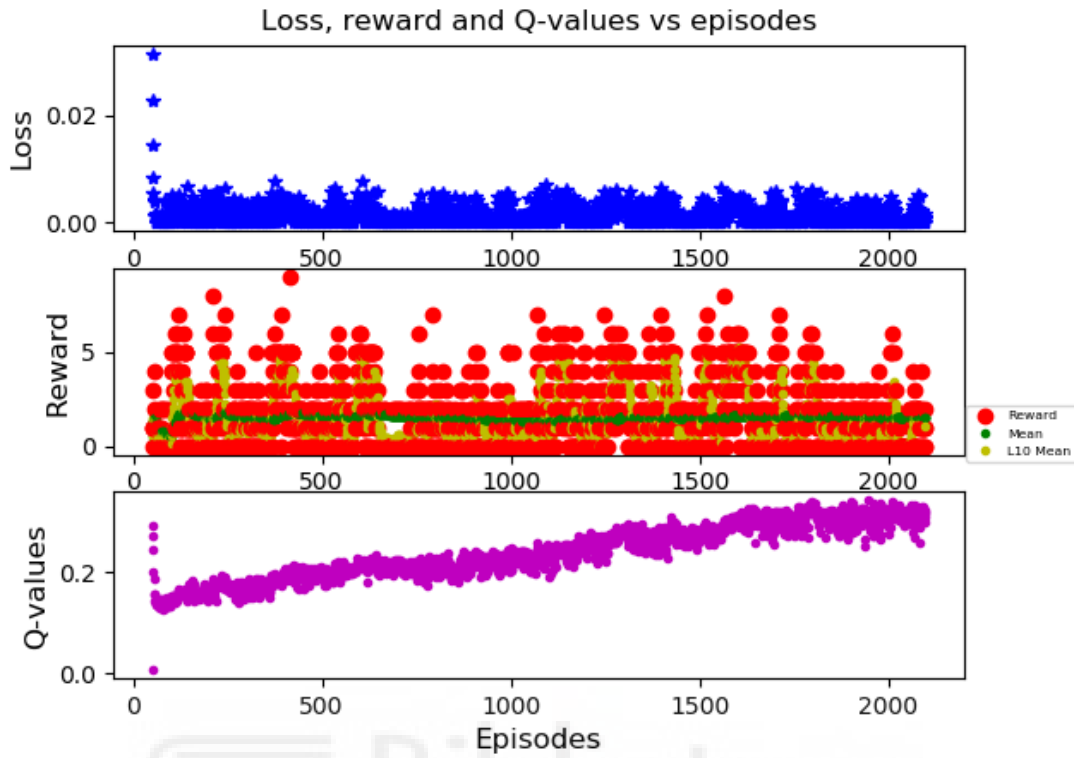

Figura 156: Gráfica sesión 3 experimento 5 BreakoutDeterministic-v4

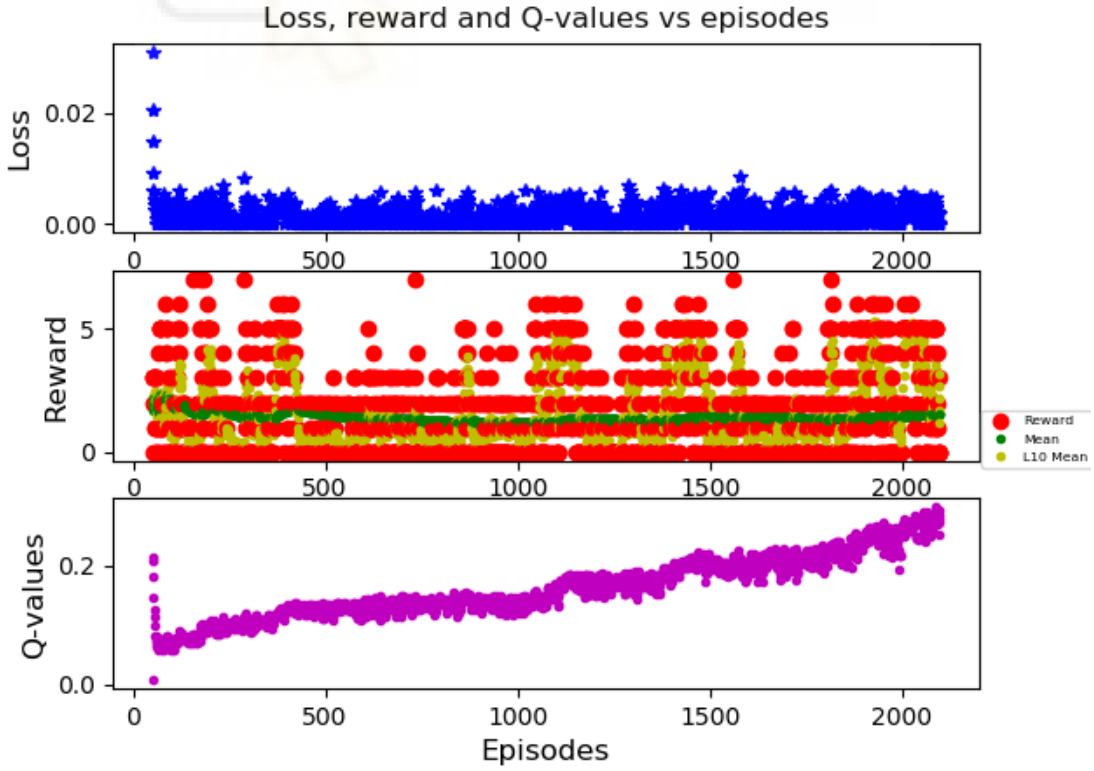

Figura 157: Gráfica sesión 4 experimento 5 BreakoutDeterministic-v4

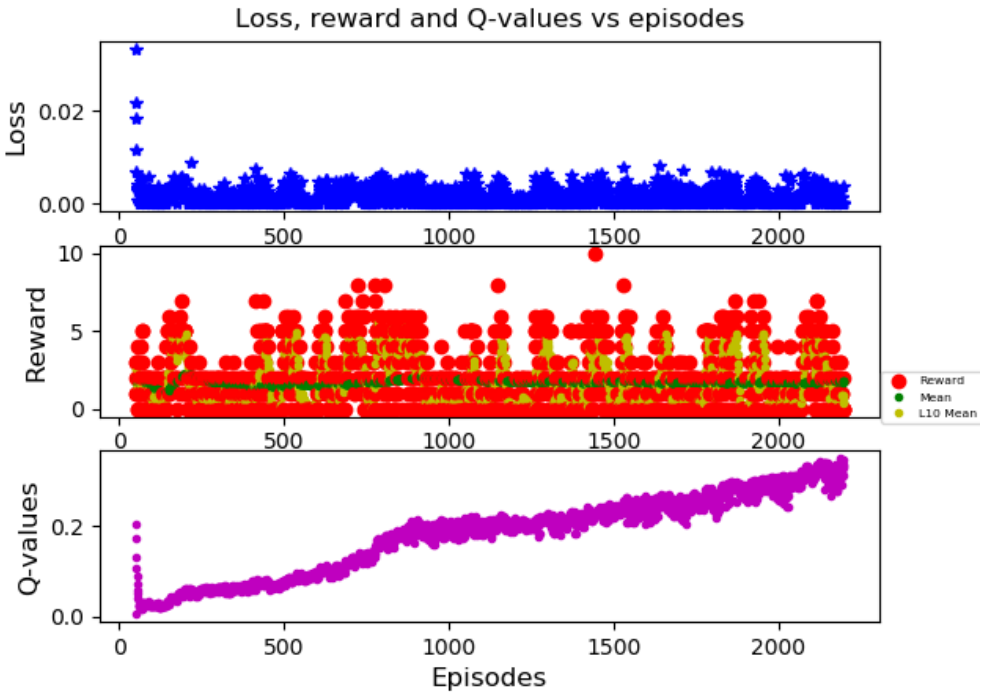

Figura 158: Gráfica sesión 5 experimento 5 BreakoutDeterministic-v4

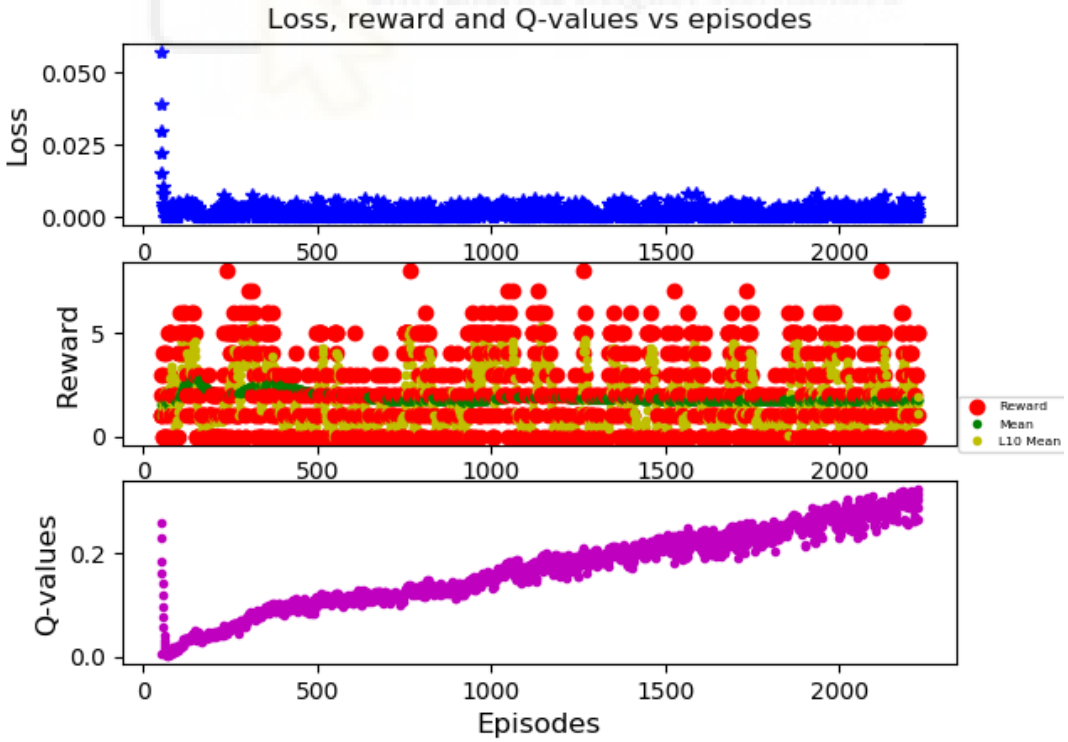

Figura 159: Gráfica sesión 6 experimento 5 BreakoutDeterministic-v4

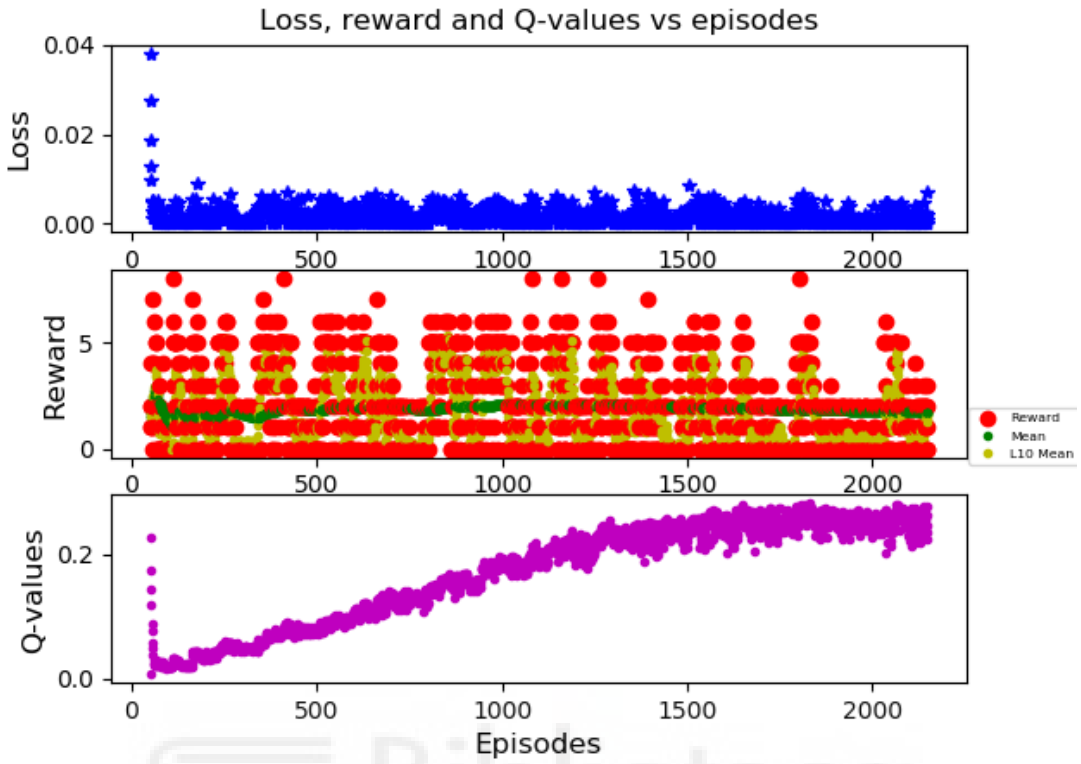

Figura 160: Gráfica sesión 7 experimento 5 BreakoutDeterministic-v4

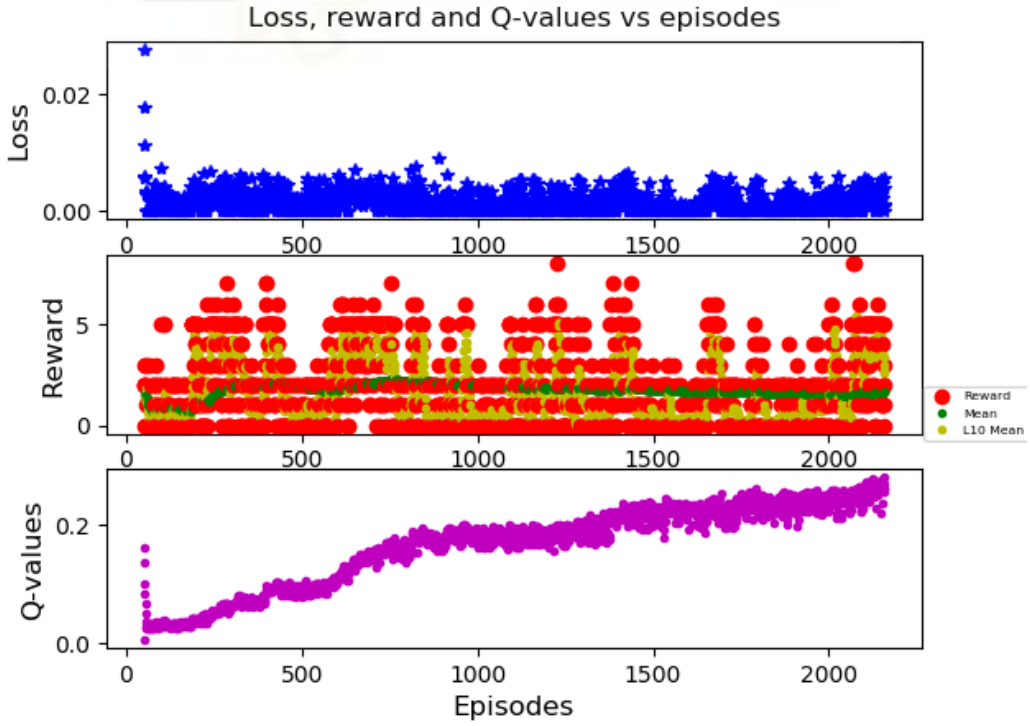

Figura 161: Gráfica sesión 8 experimento 5 BreakoutDeterministic-v4

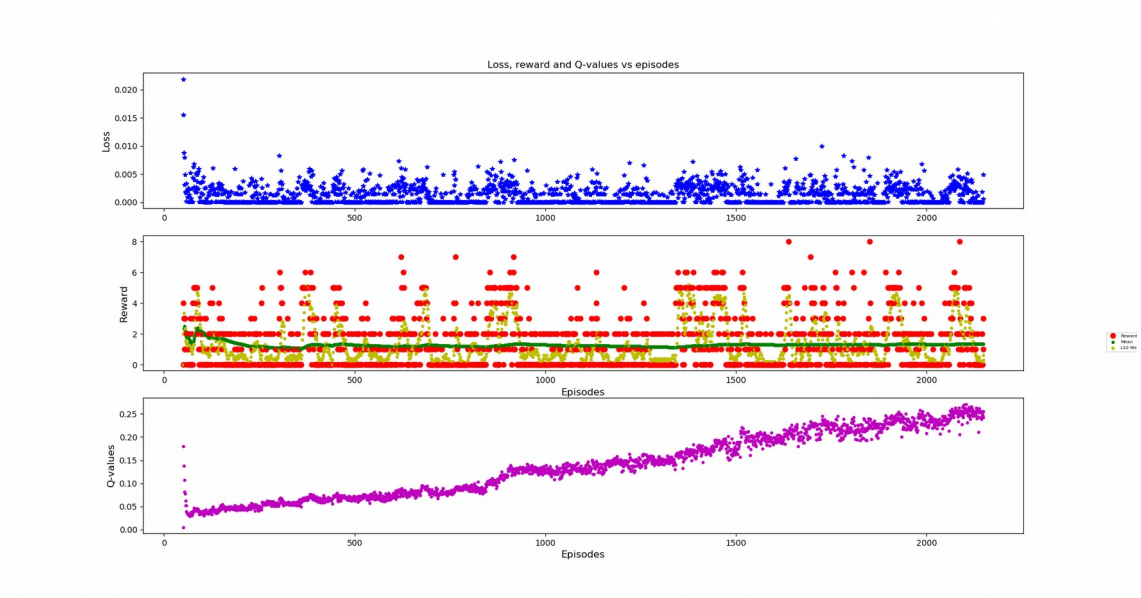

Figura 162: Gráfica sesión 9 experimento 5 BreakoutDeterministic-v4

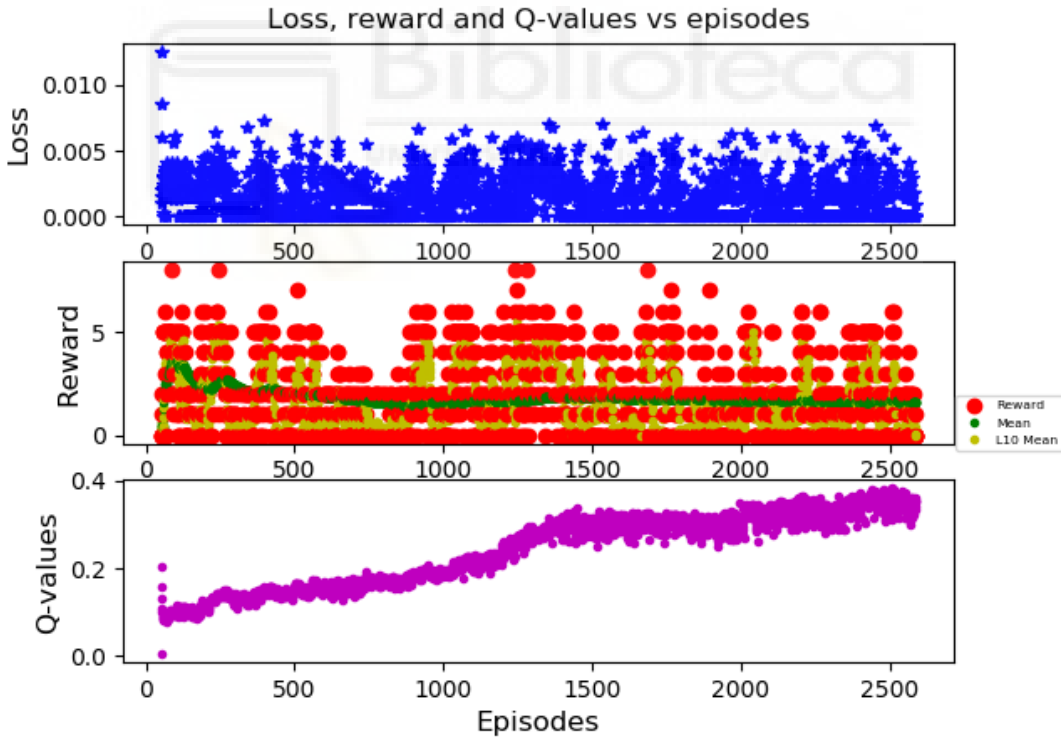

Figura 163: Gráfica sesión 10 experimento 5 BreakoutDeterministic-v4

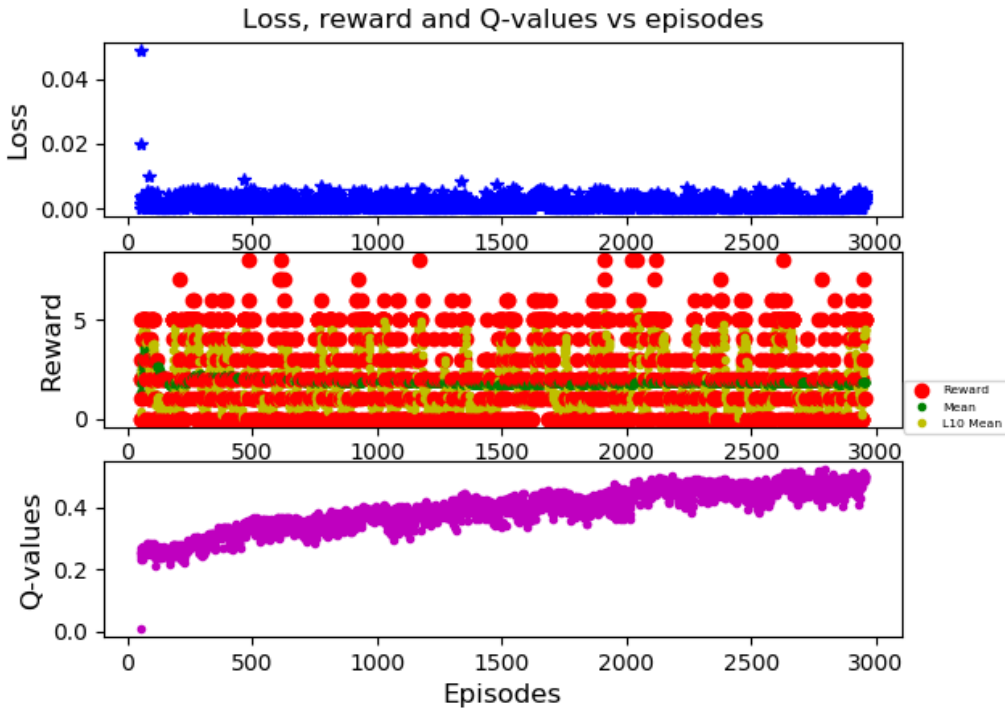

Figura 164: Gráfica sesión 11 experimento 5 BreakoutDeterministic-v4

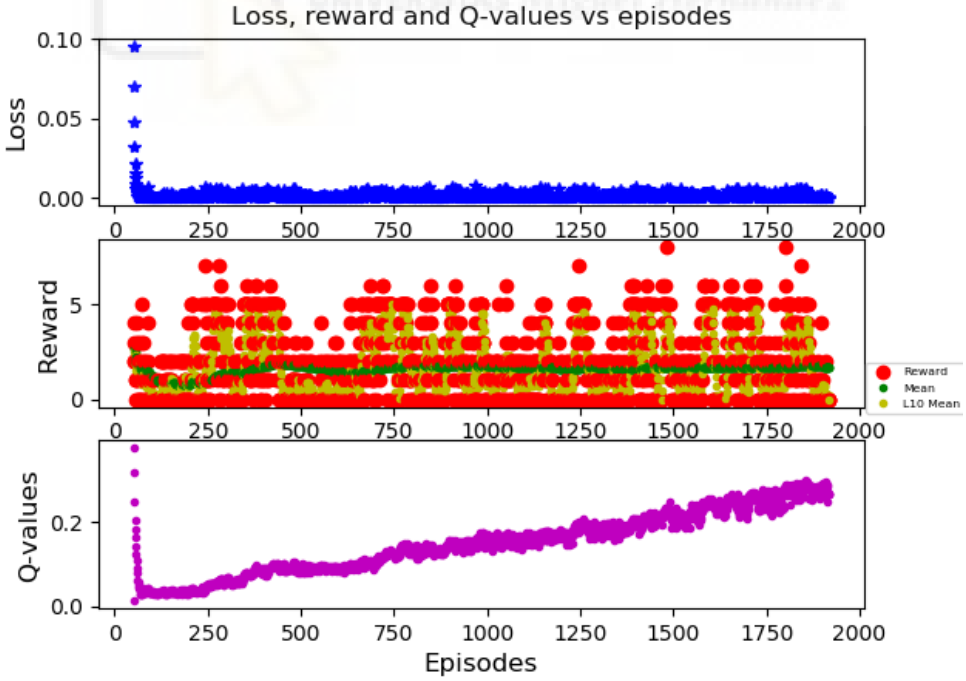

Figura 165: Gráfica sesión 12 experimento 5 BreakoutDeterministic-v4

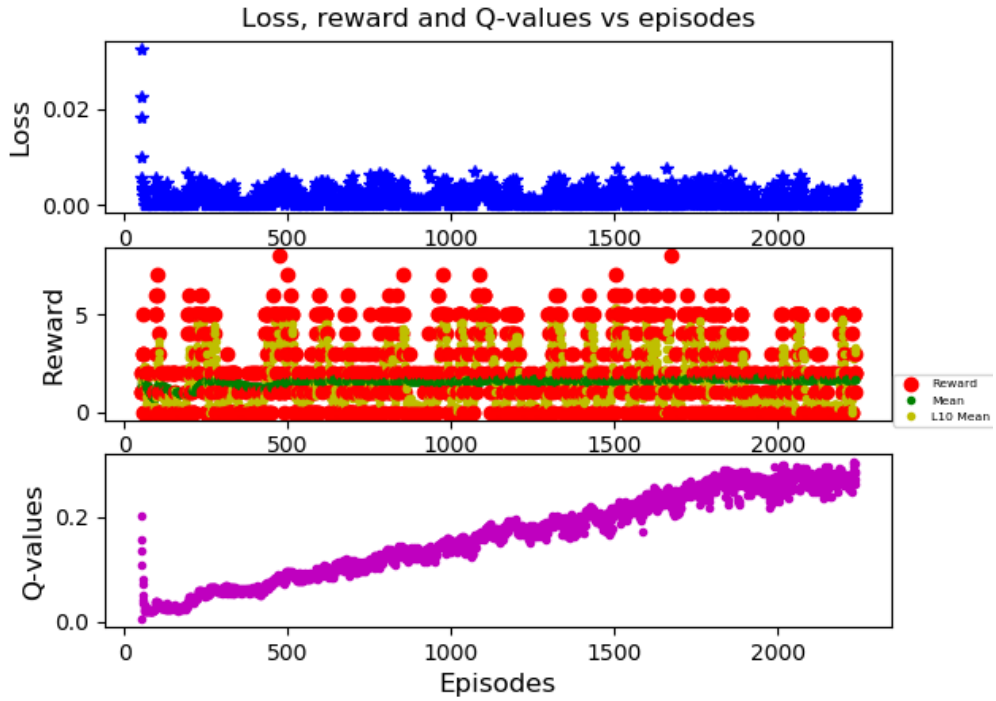

Figura 166: Gráfica sesión 13 experimento 5 BreakoutDeterministic-v4

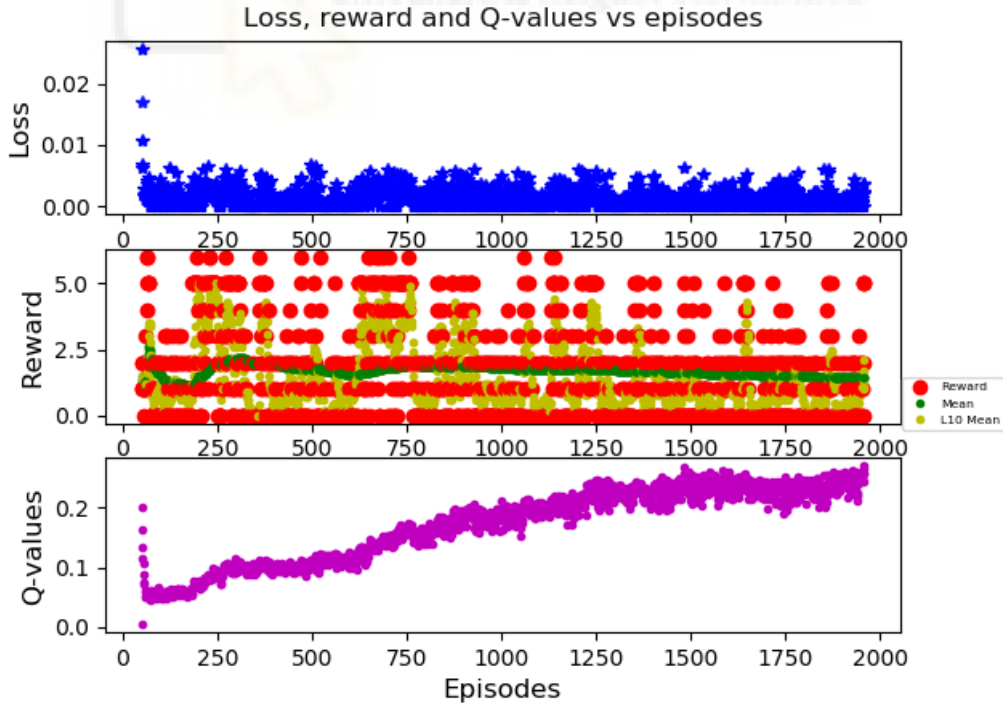

Figura 167: Gráfica sesión 14 experimento 5 BreakoutDeterministic-v4

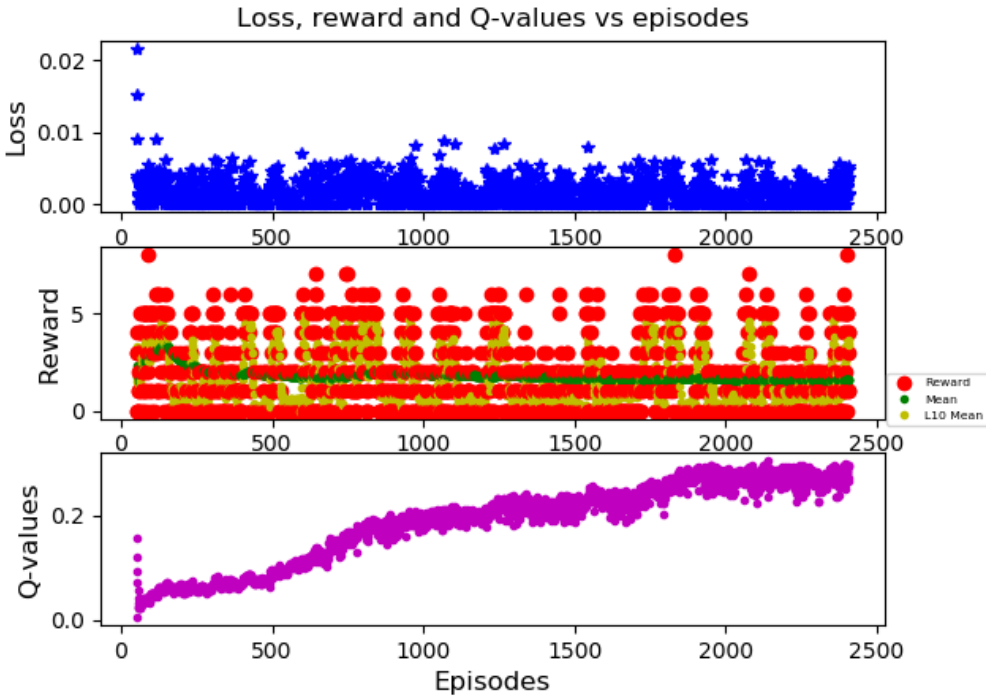

Figura 168: Gráfica sesión 15 experimento 5 BreakoutDeterministic-v4

Análisis de las gráficas obtenidas durante las sesiones de entrenamiento: el *loss* se encuentra, como se desprende de las gráficas, **en valores aceptables**, cercanos a cero.

Para las **puntuaciones**, se observa que efectivamente, **existen fluctuaciones muy grandes de la media L10**. Esto es indicativo de que **tiene rachas de puntuaciones en las que consigue muy buenas puntuaciones, para luego tener rachas muy malas;** si nos fijamos, en casi todas las gráficas registradas encontramos con grandes grupos de puntuaciones cero durante toda la partida.

Este tipo de oscilaciones son contraproducentes para el aprendizaje; nos interesa un aprendizaje más equilibrado, y como se ha comentado en el análisis de los resultados de las sesiones de entrenamiento, **probablemente esté debido al ratio de aprendizaje.** Un ratio de aprendizaje más bajo es probable que elimine este tipo de oscilaciones.

Si pasamos al análisis de **los *Q-values* medios**, en todos los casos, **tienden a ascender**, lo cual indica que la confianza de nuestro agente va en aumento conforme avanza la sesión de entrenamiento.

Los resultados finales son los siguientes:

Fecha	*Experimento*	*Puntuación máx*	*Media max*	*Media L10*	*Media final*	*N.º Episodios*
27/05/19	Aleatorio	5	1,28	1,4	1,17	200
01/07/19	Experimento 5	7	1,14	0,7	0,64	200

Tabla 13: Resultados finales experimento 5

Análisis resultados obtenidos prueba final: se observa que, en general, **los resultados obtenidos son bastante pobres en comparación a los del agente totalmente aleatorio.** En el único valor en el que **el agente destaca** es **en la puntuación máxima** (7 frente a 5). **En el resto de valores, nuestro agente se encuentra por debajo, obteniendo valores en la media L10 y en la media final de aproximadamente la mitad de los obtenidos por el agente completamente aleatorio.** La media máxima sí que se acerca al valor obtenido por el agente aleatorio, pero nuevamente se encuentra por debajo del valor aleatorio (1,28 para el agente aleatorio y 1,14 para el agente de inteligencia artificial).

Por tanto, **no podemos afirmar que con esta configuración de hiperparámetros, se produzca aprendizaje.**

Se presentan a continuación las **gráficas obtenidas de las sesiones de prueba finales aleatorias y de nuestro agente:**

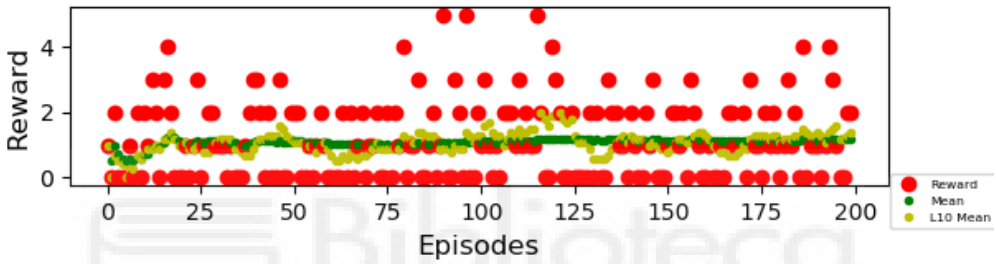

Figura 169: Gráfica prueba final BreakoutDeterministic-v4, agente aleatorio

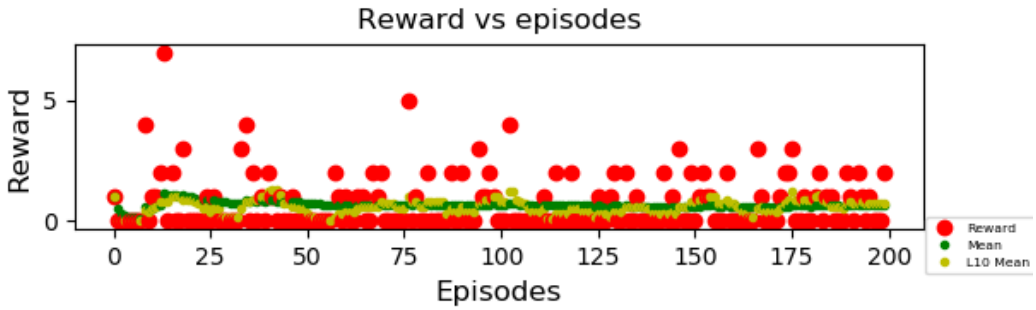

Figura 170: Gráfica prueba final BreakoutDeterministic-v4, experimento 5

Análisis gráficas prueba final: como se esperaba, a tenor de los resultados obtenidos, l**a media obtenida por el agente aleatorio es mayor que la obtenida por el agente de inteligencia artificial; asimismo, encontramos menos valores cero en el caso aleatorio.**

7.2.2.5 Experimento 6 (Google Colab)

Los hiperparámetros utilizados son los siguientes:

- Episodios máximos de entrenamiento: 100010
- Episodios de observación: 50
- Pasos para actualizar la red objetivo: 10000
- Tamaño de memoria: 200000
- Número de experiencias para *experience replay*: 48
- Ratio de aprendizaje: 0.00025
- Cuadros hasta epsilon mínimo: 1000000
- Epsilon final: 0.1
- Gamma: 0.99

Los datos obtenidos durante las distintas sesiones de entrenamiento quedan recogidos en la siguiente tabla:

Fecha	Puntuación máx	Media máx	Media L10	Media final	Loss	Media Q-Values	Tiempo sesión	Tiempo total	N.º episodios sesión	Episodios totales	Experiencias sesión	Experiencias totales
11/06/19	7	1,24	1,8	1,24	0,007131	0,097385	11:35:11	11h 35min 11s	1410	1410	280939	280939
12/06/19	7	1,42	0,7	1,23	0,007113	0,192101	11:13:38	22h 48min 49s	1250	2660	252327	533266
13/06/19	9	1,69	1,2	1,25	0,0039999	0,119973	11:20:27	34h 09min 16s	1180	3840	249585	782851
14/06/19	6	1,44	0,6	0,74	0	0,081756	11:42:03	45h 51min 19s	1120	4960	257365	1040216
17/06/19	4	1,5	0,7	0,43	0	0,060592	11:22:55	57h 14min 14s	800	5760	254464	1294680
18/06/19	6	4,24	0,4	0,61	0,001897	0,037967	11:21:43	68h 35min 57s	710	6470	237093	1531773
20/06/19	4	1,13	0,4	0,38	0,000001	0,044313	11:42:34	80h 18min 31s	800	7270	No registrado	No registrado
21/06/19	4	0,45	0,4	0,42	0,000952	0,059151	11:33:52	91h 52min 23s	750	8020	No registrado	No regisrado
24/06/19	4	1,5	0,3	0,39	0	0,042288	11:10:57	103h 03min 20s	770	8790	No registrado	No registrado
25/06/19	4	No recogido	No recogido	No recogido	No recogido	No recogido	No recogido	No recogido	510	9300	No registrado	No registrado

Tabla 14: Resultados obtenidos experimento 6 (Google Colab) durante las sesiones de entrenamiento

Análisis resultados obtenidos durante las sesiones de entrenamiento: encontramos que **la diferencia entre la puntuación máxima alcanzada** (9 puntos) **y la mínima** (4) **es bastante grande.** Además, **la puntuación mínima se repite varias veces,** mientras que la puntuación máxima solo se alcanza en la sesión 3. Se observa que **las puntuaciones, en vez de mejorar, han ido empeorando.**
La media máxima más alta registrada **es de 4,24 puntos, siendo la mínima de 0,45**; se aprecia una gran diferencia entre las mismas.
En cuanto a la media L10, encontramos un valor máximo de 1,8 puntos en la primera sesión, y el mínimo, de 0,3 puntos, en la última sesión de entrenamiento. **Los valores de la media final** presentan un patrón similar: se consigue el valor más alto en la tercera sesión de entrenamiento (1,25 puntos), mientras que el mínimo se alcanza en la sesión séptima (0,38). Las sesiones posteriores presentan valores muy bajos. **Todos estos datos hacen pensar que la red neuronal, a pesar de que los valores del *loss* se encuentran cercanos al cero, está en un mínimo local y es incapaz de salir del mismo.**
Encontramos que **la media de los *Q-values* es baja**, pero además **existe una gran diferencia entre la más grande registrada** (0,192101) **y la mínima** (0,037967); también la puntuación mínima fue registrada en una sesión posterior a la máxima, encontrándonos además que en las sesiones posteriores a esta última los valores obtenidos son muy bajos también.
Los tiempos de sesiones, puesto que se ha entrenado bajo *Google Colab*, son cercanos pero inferiores a las 12 horas; el máximo número de episodios vistos en una sesión es de 1410, alcanzado en la primera sesión; el mínimo es de 510, alcanzado en la última sesión.
El máximo número de experiencias vistas lo encontramos en la primera sesión (280939), y el mínimo (237093) en la sexta sesión. No existe gran diferencia entre la cantidad de experiencias vistas entre sesiones.

Las gráficas recogidas durante las sesiones de entrenamiento son las siguientes:

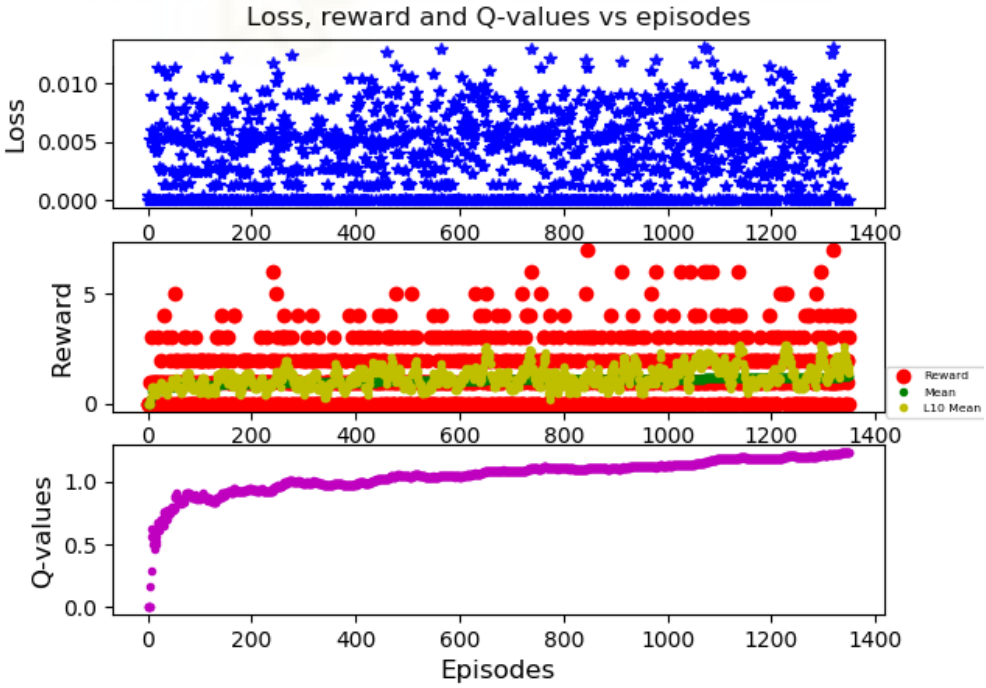

Figura 171: Gráfica experimento 6, sesión 2, BreakoutDeterministic-v4

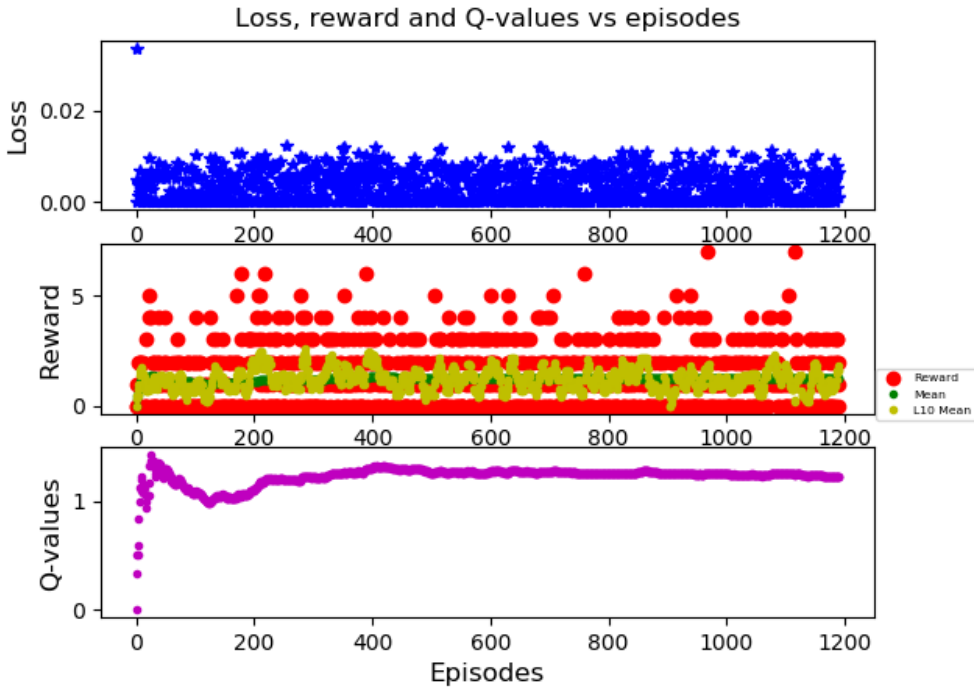

Figura 172: Gráfica experimento 6, sesión 3, BreakoutDeterministic-v4

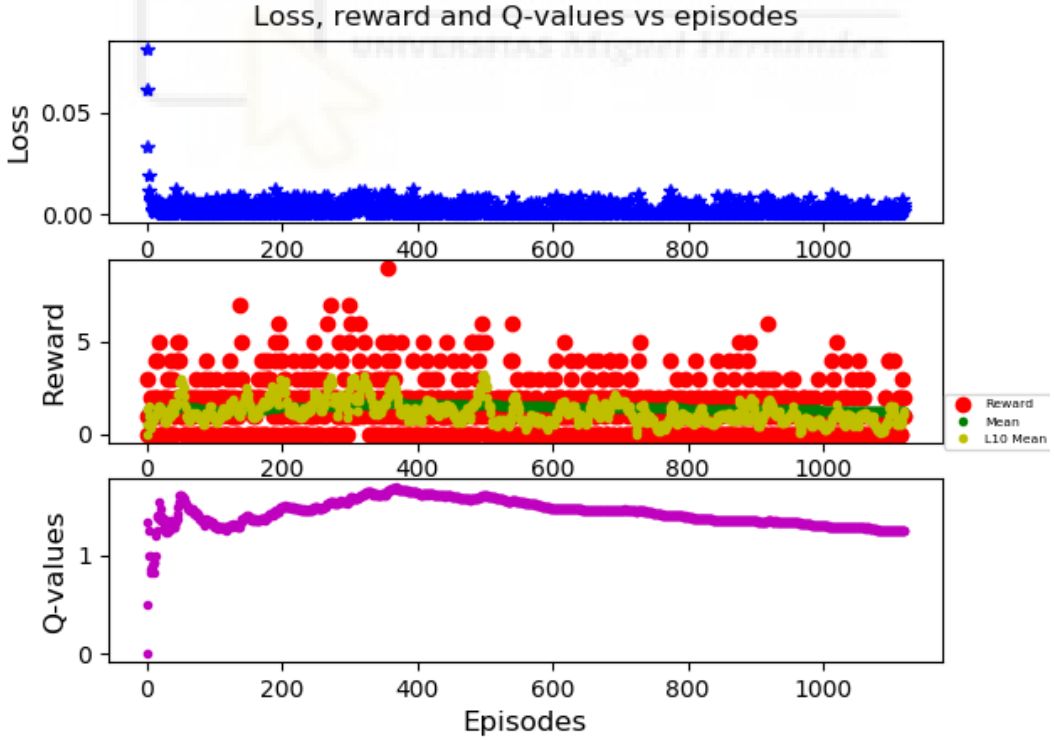

Figura 173: Gráfica experimento 6, sesión 4, BreakoutDeterministic-v4

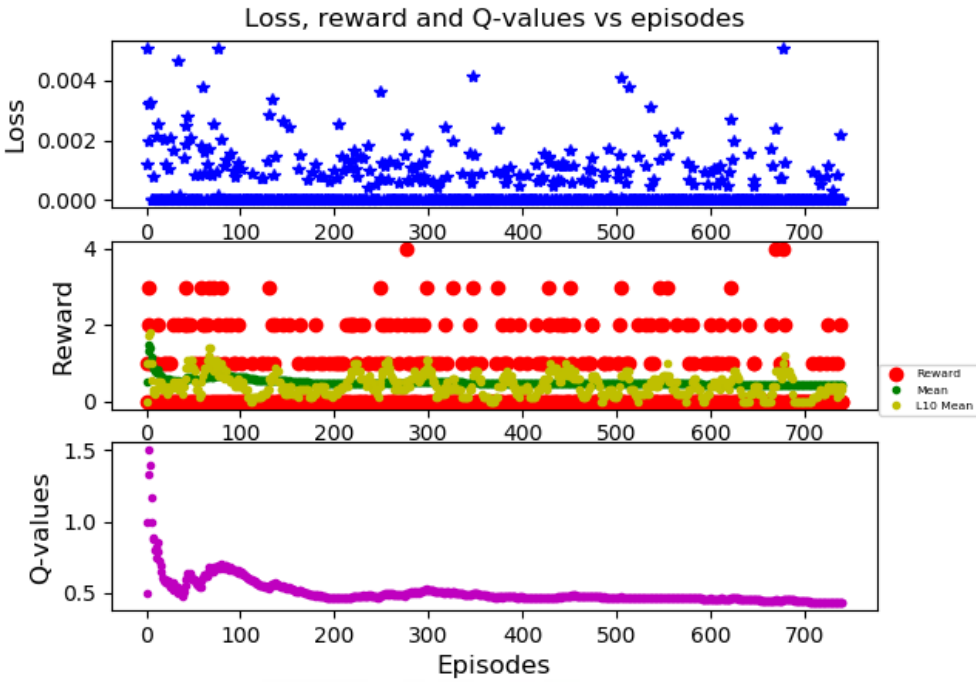

Figura 174: Gráfica experimento 6, sesión 6, BreakoutDeterministic-v4

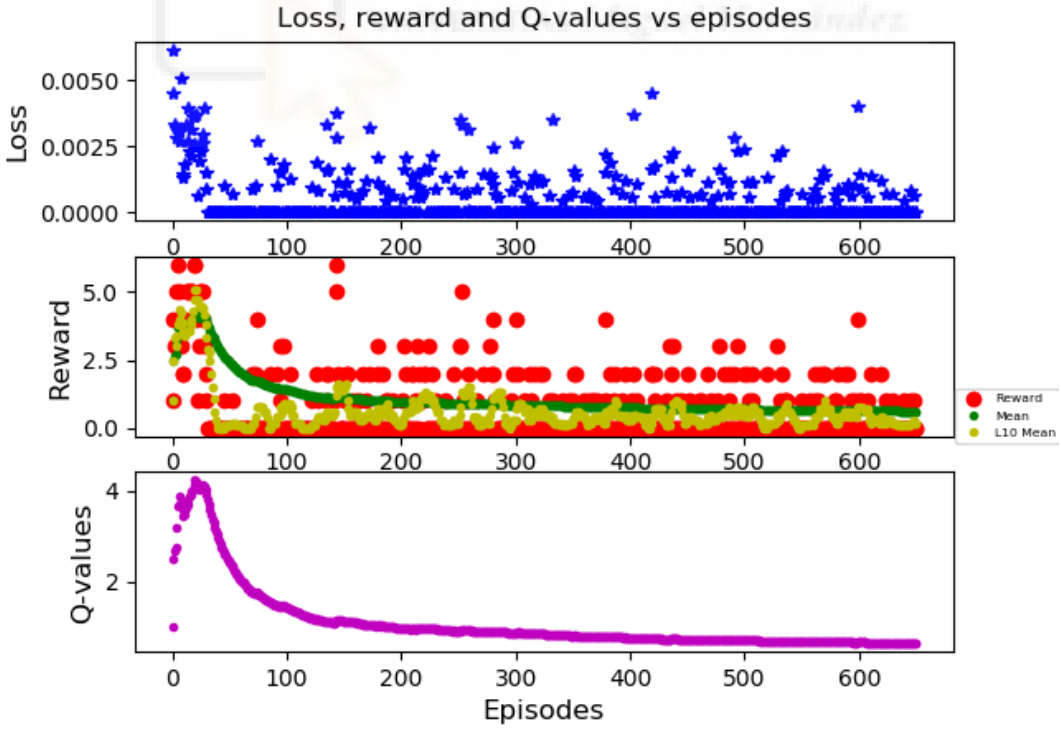

Figura 175: Gráfica experimento 6, sesión 7, BreakoutDeterministic-v4

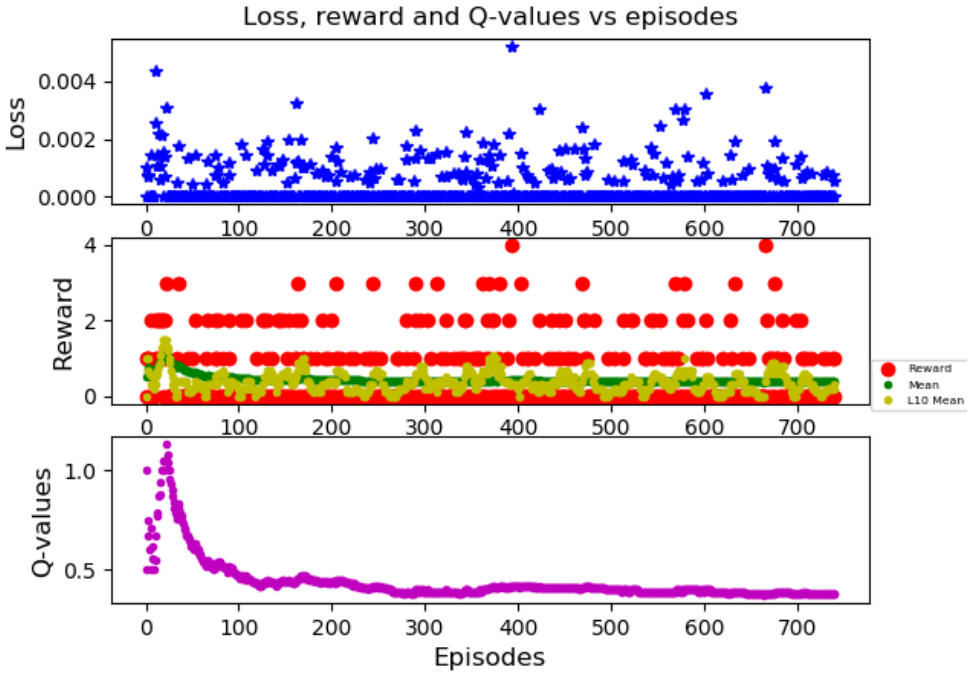

Figura 176: Gráfica experimento 6, sesión 8, BreakoutDeterministic-v4

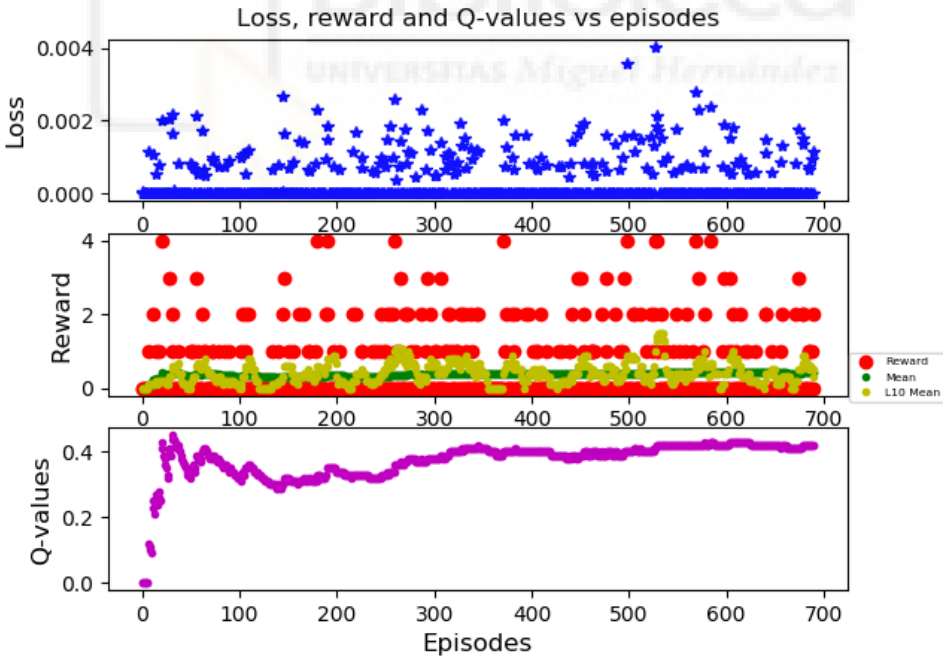

Figura 177: Gráfica experimento 6, sesión 9, BreakoutDeterministic-v4

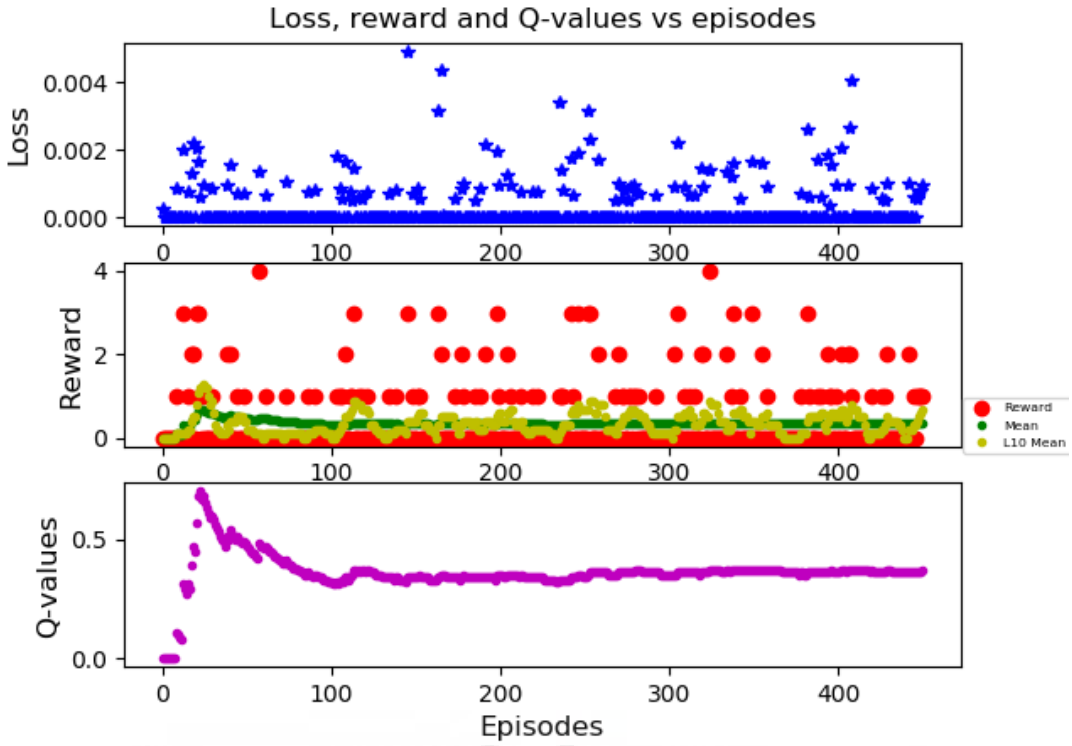

Figura 178: Gráfica experimento 6, sesión 10, BreakoutDeterministic-v4

Análisis gráficas obtenidas durante las sesiones de entrenamiento: el *loss* se mantiene en valores cercanos a cero, sin grandes oscilaciones. **Para las puntuaciones**, encontramos que **muchas de las mismas son cero o uno**, lo que lógicamente provoca que la media sea bastante baja. **Se aprecian oscilaciones de la media L10,** si bien no tan pronunciadas como en el experimento 5, sí que nos **indican que probablemente la tasa de aprendizaje no sea correcta.**
La gráfica de los *Q-values* medios en algunos casos, en vez de subir, tiende a bajar o se estabiliza. En estos casos, lo que nos indica es que **no está aprendiendo**, es decir, el agente no es capaz de encontrar una estrategia mejor.
Por tanto, no podemos afirmar que, con estos ajustes, se produzca un aprendizaje.

Los resultados finales obtenidos son los siguientes:

Fecha	*Experimento*	*Puntuación máx*	*Media max*	*Media L10*	*Media final*	*N.º Episodios*
27/05/19	Aleatorio	5	1,28	1,4	1,17	200
01/07/19	Experimento 6	7	1,18	0,6	0,38	200

Tabla 15: Resultados finales experimento 6, BreakoutDeterministic-v4

Análisis resultados obtenidos prueba final: la puntuación máxima que obtiene nuestro agente, **es superior a la obtenida por el agente aleatorio** (7 frente a 5). Aparte de ese parámetro, **el resto son inferiores**; la media máxima, si bien se acerca a la conseguida por el aleatorio (1,18 frente a 1,28) no llega a superarla.

Las mayores diferencias se aprecian en la media L10 (1,4 el agente aleatorio frente a 0,6 nuestro agente) **y la media final** (0,38 frente a 1,17). En ambos casos, la diferencia es más del doble, siendo mucho mayor en el caso de la media final, lo que nos indica que el aprendizaje no se produce.

Las gráficas obtenidas de las pruebas finales son las siguientes:

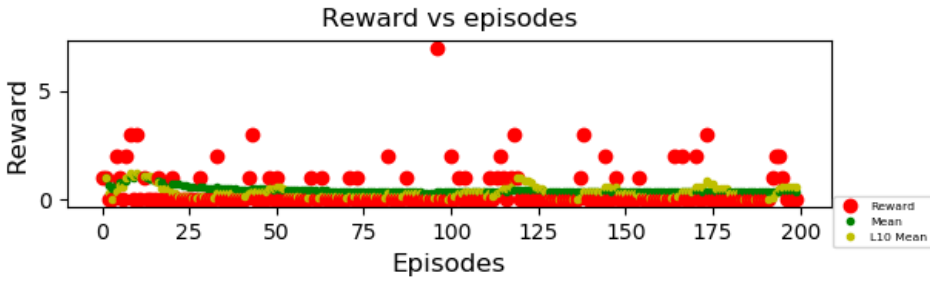

Figura 179: Gráfica experimento 6, sesión final agente, BreakoutDeterministic-v4

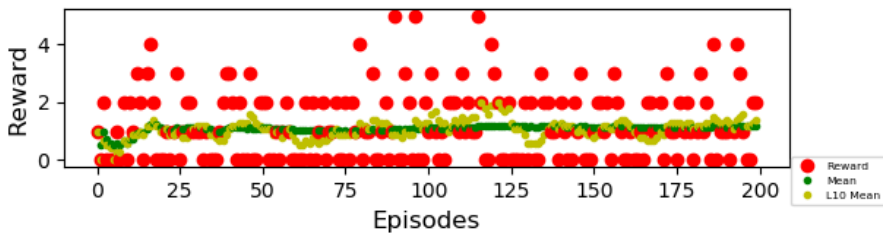

Figura 180: Gráfica experimento 6, sesión aleatoria, BreakoutDeterministic-v4

Análisis gráficas experimento final: tal y como se viene desprendiendo del resto de datos y experimentos, **el agente de inteligencia artificial no logra superar al agente en modo puramente aleatorio.** Se aprecia que nuestro agente marca muchas más puntuaciones cero que el agente aleatorio; si bien la media L10 oscila menos que en el caso aleatorio, tanto esta como la media son más bajas que las obtenidas por el agente aleatorio.

8. Conclusiones y futuros proyectos

El aprendizaje por refuerzo mediante redes neuronales convolucionales, como queda demostrado en este trabajo, es en la actualidad un método ineficiente y de difícil ajuste, además de que los problemas que resuelve pueden ser resueltos por otros métodos, los cuales obtienen mejores resultados en muchos casos y con menor coste en tiempo. **Sin embargo, su principal virtud es su adaptabilidad: es capaz de adaptarse a nuevas circunstancias, lo cual le da una versatilidad que no es posible lograr mediante ningún otro método de aprendizaje existente en la actualidad**; un claro ejemplo de esto lo tenemos en el paper [4], en el que con el mismo algoritmo y prácticamente con los mismos ajustes, es capaz de adaptarse y aprender a jugar a 49 juegos distintos de *Atari 2600*. Este tipo de logros no es posible conseguirlos con aprendizaje supervisado ni no supervisado, los cuales están limitados a realizar una tarea eficientemente.

Si bien este método es aún experimental y con poco recorrido, **debido a su potencial merece un estudio e investigación en profundidad, para perfeccionar sus capacidades y rendimiento.**

En cuanto a los resultados obtenidos, **no se ha logrado unos resultados similares a los obtenidos en [4], esto es debido a la cantidad de hiperparámetros que existen y que deben ajustarse, lo cual causa una baja reproducibilidad del experimento**; además de que el algoritmo es muy sensible a los pequeños cambios como queda patente en algunos experimentos. Aún así, queda demostrado que, si bien los resultados son mejorables, **con un ajuste adecuado sí existe un aprendizaje con respecto a un entorno completamente aleatorio.** Dedicando más tiempo y recursos se podrían ajustar mejor los ajustes a estos hiperparámetros, consiguiendo probablemente mejores resultados.

Se presentan a continuación algunos **posibles usos y mejoras que se proponen como posibles proyectos futuros:**

- **Inicializar el programa con experiencias, previamente guardadas, de partidas realizadas por un humano:** Uno de los mayores problemas a los que nos enfrentamos en cualquier tipo de tarea de inteligencia artificial es la calidad y cantidad de datos. Los datos deben ser de buena calidad si queremos obtener buenos resultados, además de tener los suficientes para que nuestro agente sea capaz de extraer información de los mismos. En el caso de aprendizaje por refuerzo, la cantidad no es un problema, ya que generamos nuevas experiencias (datos) conforme progresa el entrenamiento (en teoría de mejor calidad cada vez, ya que se supone que el agente va aprendiendo y mejorando).

  La mejora que se propone es usar como inicio datos de partidas jugadas por un jugador humano: hacer que el agente extraiga información de las mismas durante un tiempo, para posteriormente, cuando consiga unas puntuaciones aceptables, comience a aprender de sus propias experiencias generadas.

- **Aplicar *prioritized experience replay* al agente para ver si el entrenamiento mejora sus resultados:** el sistema de *experience replay* implementado actualmente es el mismo que se describe en [4]; es decir, para aprender escogemos un número determinado de experiencias de manera aleatoria; el problema que presenta este método es que no tenemos conocimiento

ninguno de si las experiencias que el agente utiliza son de buena calidad o no; para solucionar este problema, en [202] se propone el método conocido como *PER (siglas de Prioritized Experience Replay)*.

El *PER* se basa en reforzar aquellos comportamientos menos comunes, es decir, implementa un sistema de búsqueda e indexación de las experiencias almacenadas, para ver el número de veces que se han utilizado, si tienen cierta similitud entre ellas, etc. **El *PER* da prioridad a las experiencias menos comunes**, repitiéndolas más veces que aquellas que son más comunes. De esta manera busca aprender aquello que se sale de lo normal. Si pensamos en una situación en la vida real, en la que ocurre algo que nos sorprende, pensaremos más tiempo en ello que en si no sucede [203].

Por ejemplo, si cruzamos la calle presenciamos un atropello, probablemente pasemos días pensando en cómo se podía haber evitado, si podríamos haber hecho algo, etc. En cambio si al cruzar la calle no pasa absolutamente nada, es probable que no dediquemos tiempo a pensar en ese hecho.

Este tipo de mejora **no se ha llevado a cabo para el agente presentado para este trabajo fin de máster debido a la complejidad de su implementación.**

- **Creación de una *Rainbow DQN*:** En el paper *"Rainbow: Combining improvements in Deep Reinforcement Learning"* [204] se mencionan una serie de mejoras a la DQN básica, tales como:
  - Double Q-Learning
  - Prioritized experience replay (PER)
  - Dueling Networks
  - Multi-Step Learning
  - Distributional RL
  - Noisy Nets

  Con todas estas mejoras implementadas al mismo tiempo, se consiguen unos resultados muy superiores a la DQN tradicional, así como a otros algoritmos.

- **Usar aprendizaje por refuerzo con SLAM:** en el paper *"Playing Doom with SLAM-Augmented Deep Reinforcement Learning"* [205] **se plantea un método en el que se usa SLAM para aumentar la información recibida del entorno por una DQN mientras juega al videojuego *Doom*.** De esta forma, nuestro agente tiene un mayor conocimiento del entorno, por lo que puede tomar mejores decisiones.

Figura 181: Captura videojuego Doom

[206]

Sería interesante el estudio de este método en robots en la vida real, para desarrollar tareas tales como moverse entre edificios o esquivando obstáculos, ya que la capacidad de adaptación a nuevas circunstancias del entorno con una DQN es uno de sus puntos fuertes.

- **Investigar otros algoritmos de aprendizaje por refuerzo:** existen otros algoritmos de aprendizaje por refuerzo como A2C, A3C, o aprendizaje por refuerzo jerárquico.

  Las siglas de A2C corresponden a *Advantage actor-critic* y A3C a A*synchronous advantage actor-critic*. A3C obtiene mejores resultados que la DQN (excepto en el caso de *Rainbow DQN)* [204]. Se puede encontrar una explicación simple de como funciona A3C en [207].

  En cuanto al **aprendizaje por refuerzo jerárquico**, se trata de descomponer el problema en pequeños pasos. Poniendo un ejemplo, si estamos en la primera planta de un edificio, en una habitación y queremos ir a la segunda planta del edificio de enfrente, con los algoritmos actuales abordamos el problema globalmente; con aprendizaje jerárquico, abordaríamos primero como salir de la habitación, después como bajar a la planta baja, como salir del edificio, etc, hasta llegar a la segunda planta del otro edificio.

  Otro ejemplo: si estamos ante un examen, el algoritmo actual abordaría el examen en su totalidad como un problema; con un algoritmo de aprendizaje por refuerzo jerárquico, lo descompondría en las distintas preguntas del examen, apartados, etc.

Este tipo de planteamiento resuelve el problema de los algoritmos de aprendizaje por refuerzo de que no son capaces de escalar bien en entornos muy grandes; podría presentar una posible solución al mismo.

[208]

9. Anexos

Anexo I: Anaconda

En este anexo, se explicará **como instalar Anaconda. Tanto su descarga como uso son completamente gratuitos.**

I.Instalación de anaconda

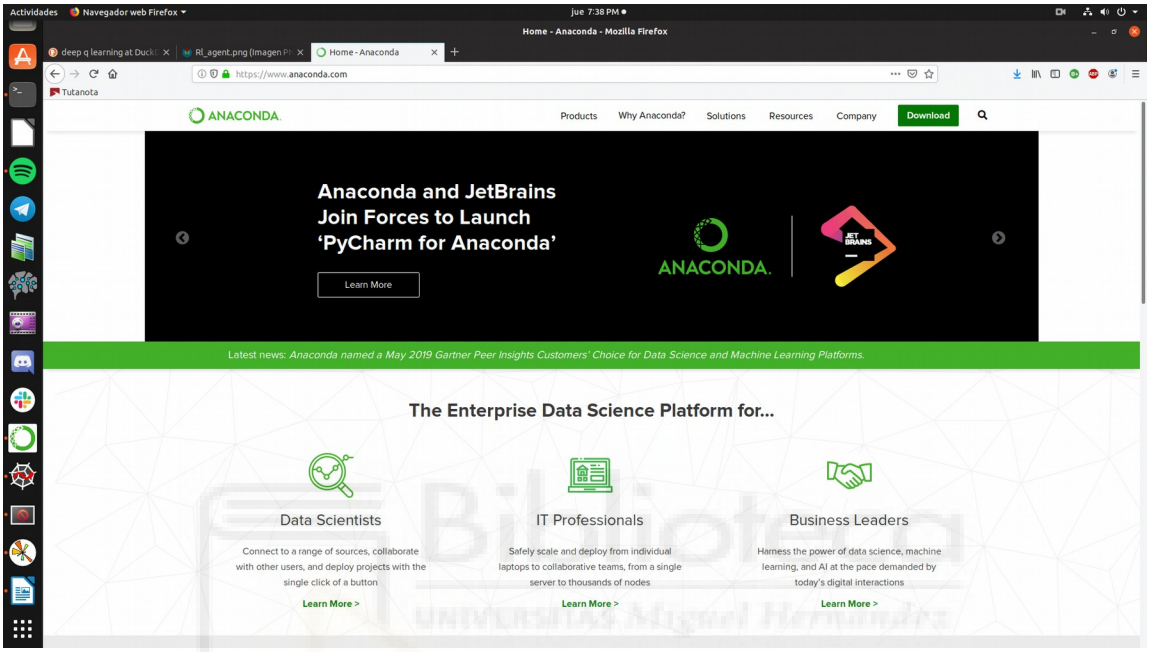

Figura 182: Página principal de Anaconda

1. Pulsamos sobre el **botón "Download" para ir a la sección de descargas.**

Figura 183: Botón de descarga de Anaconda

2. Puesto que en nuestro caso estamos bajo Linux en un sistema de 64 bits, **seleccionamos el icono de Linux** (normalmente la web detectará que estamos bajo Linux y no será necesario seleccionarlo).

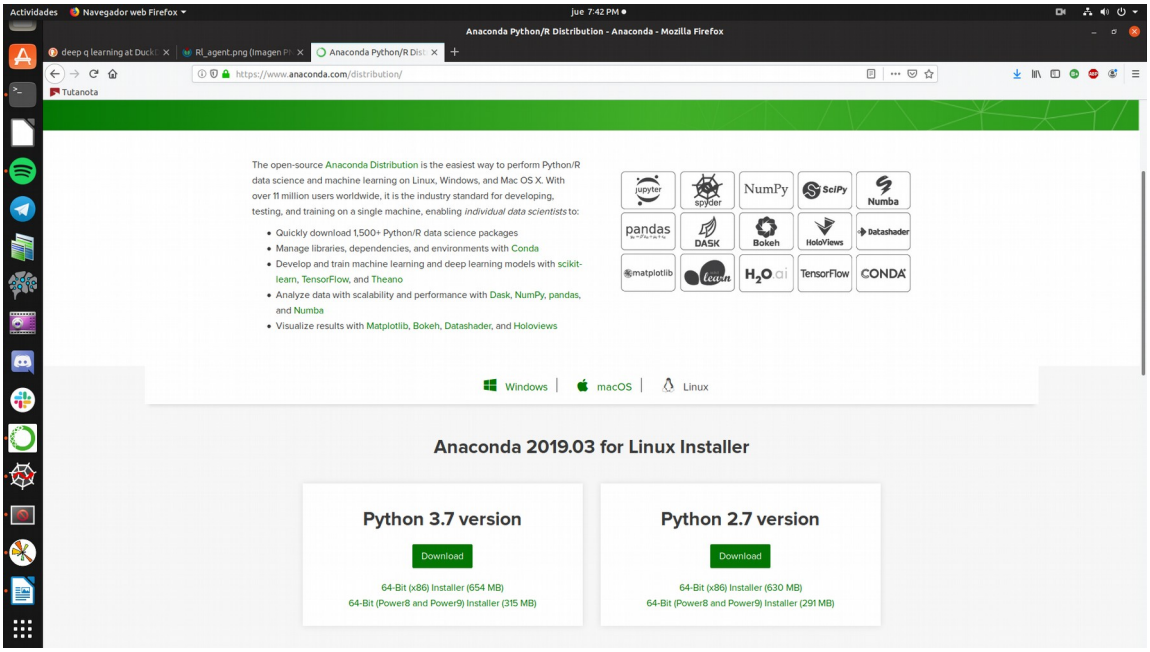

Figura 184: Captura de pantalla de la sección de descargas de Anaconda

3. Aparecerán dos opciones: la versión 3.7 de Python o la versión 2.7; **seleccionamos la 3.7 debido a que ofrece mejor compatibilidad con Tensorflow**; además estamos utilizando Python 3.6, por lo que ofrece compatibilidad.
4. Hacemos click en ***"64-bit(x86) installer"***
5. **Esperamos a que finalice la descarga**
6. **El archivo tiene la extensión *.sh*.** Existen **dos maneras de ejecutar el archivo** de instalación:
    1. Bien **haciendo click derecho sobre el archivo**, y en propiedades seleccionando la pestaña permisos, marcar *permitir ejecutar el archivo como un programa;* después debemos ejecutar el archivo en la terminal.

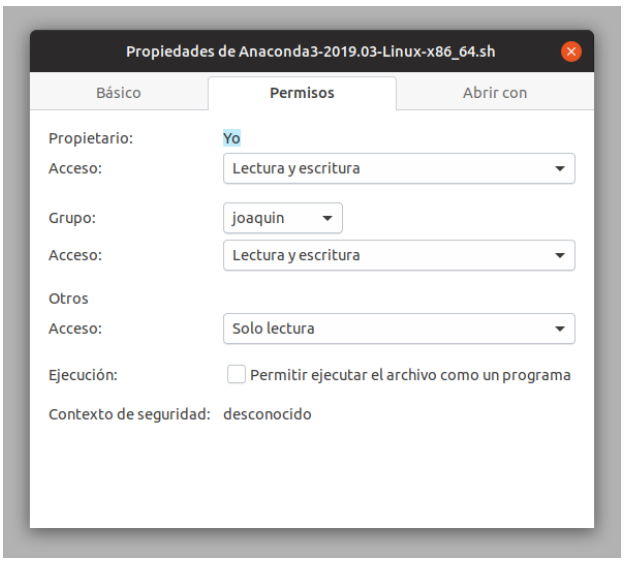

Figura 185: Ventana donde se muestran los permisos del archivo

2. **Desde la consola**, entramos en la carpeta donde hemos descargado el archivo, y escribimos `chmod +x file.sh`, donde sustituimos `file.sh` por el nombre del archivo correspondiente. Después, para ejecutarlo, debemos escribir en la consola `./file.sh`, sustituyendo *file.sh* por el nombre de nuestro archivo.

7. **Para ejecutar anaconda, escribiremos en consola *anaconda-navigator*** y se ejecutará automáticamente.

En [209] encontramos la **guía de instalación oficial** por si es necesario para clarificar algún paso.

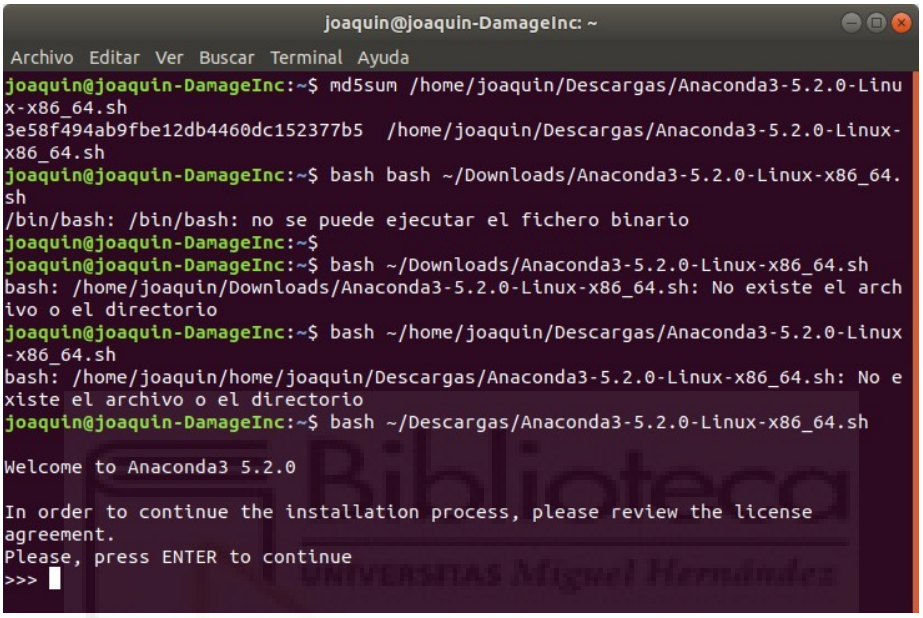

Figura 186: Captura pantalla instalación Anaconda, comprobación MD5 y comienzo de la instalación.

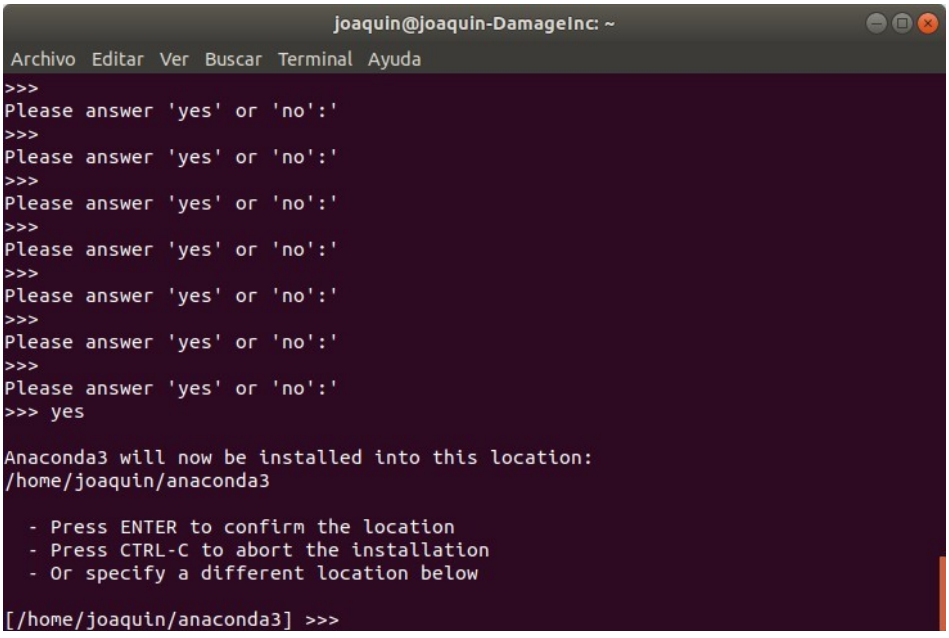

Figura 187: Captura de pantalla instalación Anaconda, directorio de instalación

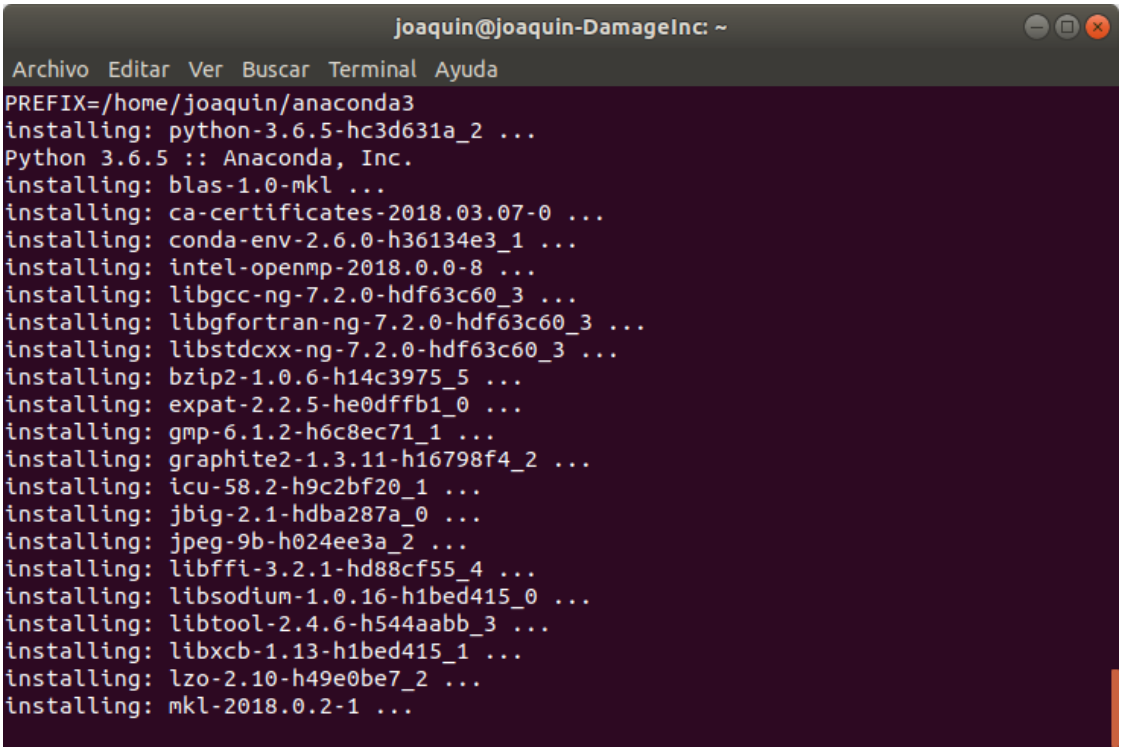

Figura 188: Captura de pantalla instalación Anaconda. Instalación de paquetes

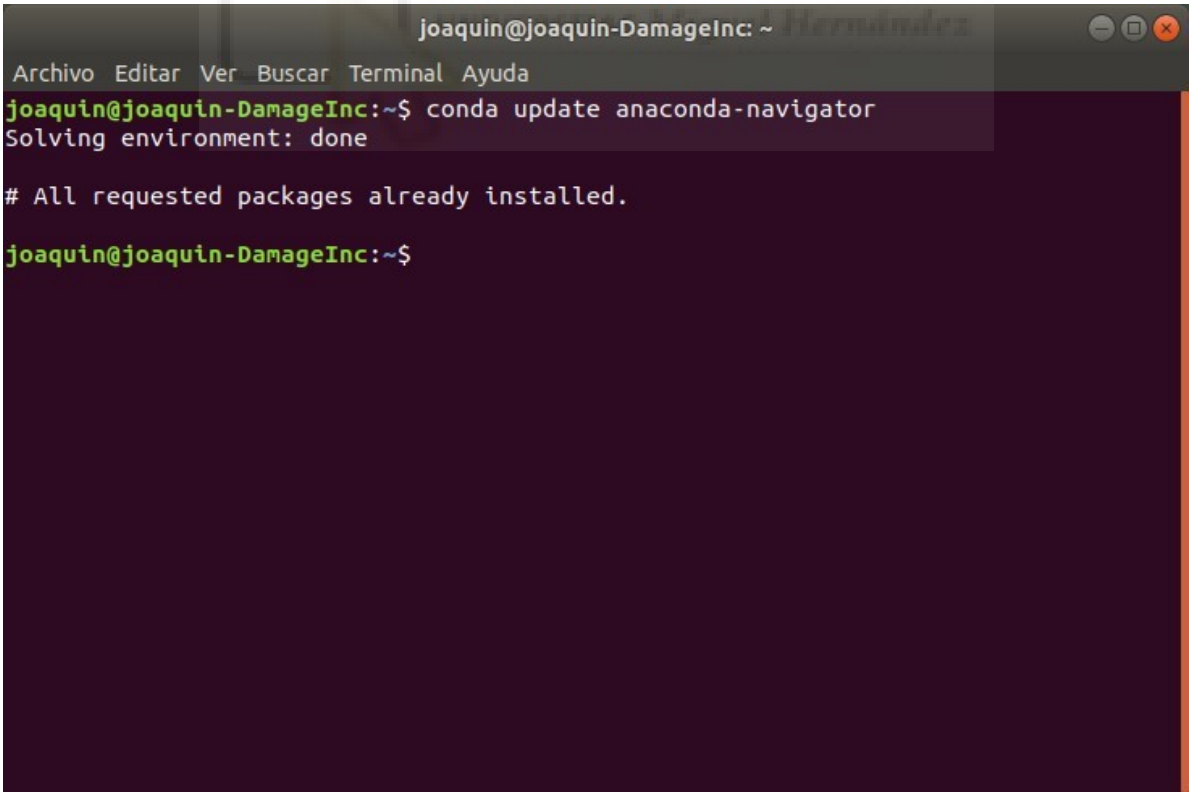

Figura 189: Captura de pantalla instalación Anaconda. Comprobación de que la versión es la más actualizada

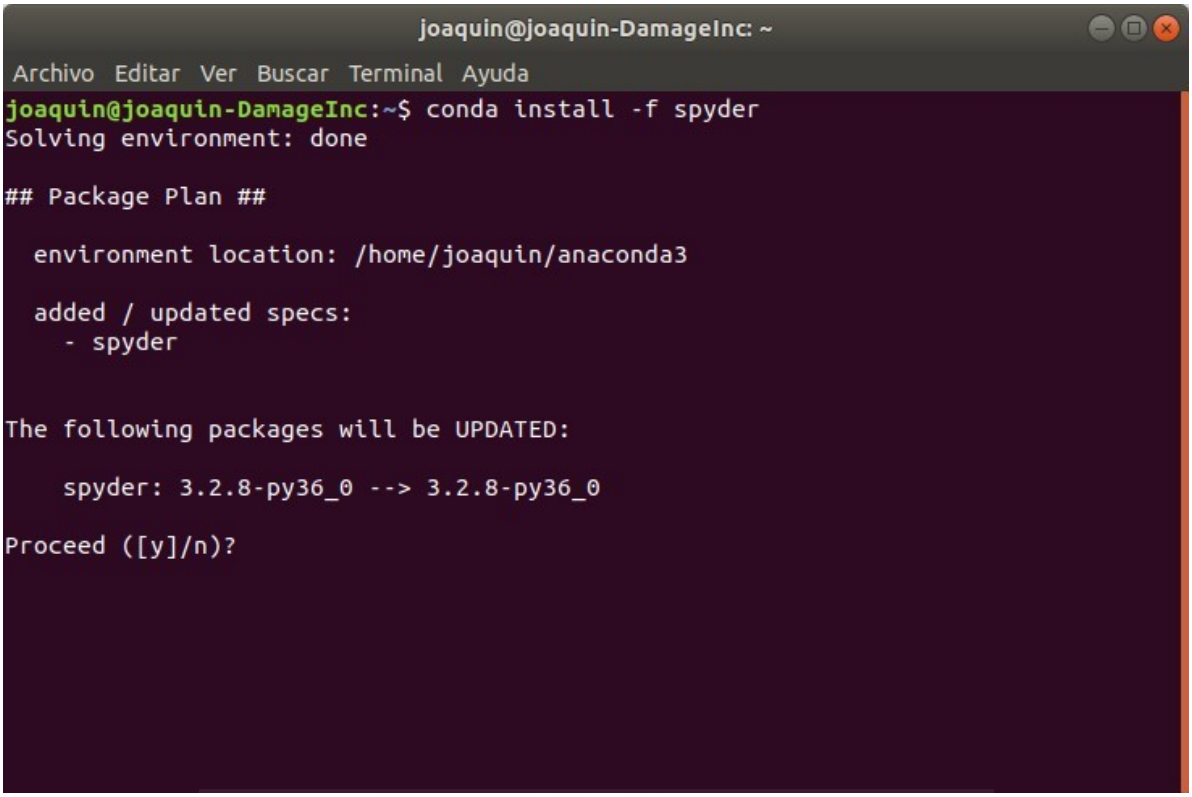

Figura 190: Instalación de Spyder desde Anaconda. Los paquetes para Anaconda se instalan con la instrucción conda install y el nombre del paquete o aplicación. También dispone de modo gráfico

Anexo II: Instalación de Tensorflow en anaconda (linux)

En este anexo se describirá el proceso de **instalación de tensorflow en anaconda**.

II.I. Requisitos previos

Antes de realizar la instalación de tensorflow, **necesitaremos haber instalado previamente**:

- **Ubuntu 16.04 o posterior**
- **Drivers de Nvidia instalados**
- **Anaconda correctamente instalado**

  [210]

II.II. Instalación de drivers Nvidia en linux

Lo primero que debemos hacer es instalar los drivers de Nvidia. Para ello, **seguiremos los siguientes pasos:**

1. Entramos a la **web de Nvidia,** en la **sección drivers:** https://www.nvidia.com/Download/index.aspx?lang=en-us
2. **Seleccionamos nuestra tarjeta**, sistema operativo, etc, y pulsamos el **botón *search***.

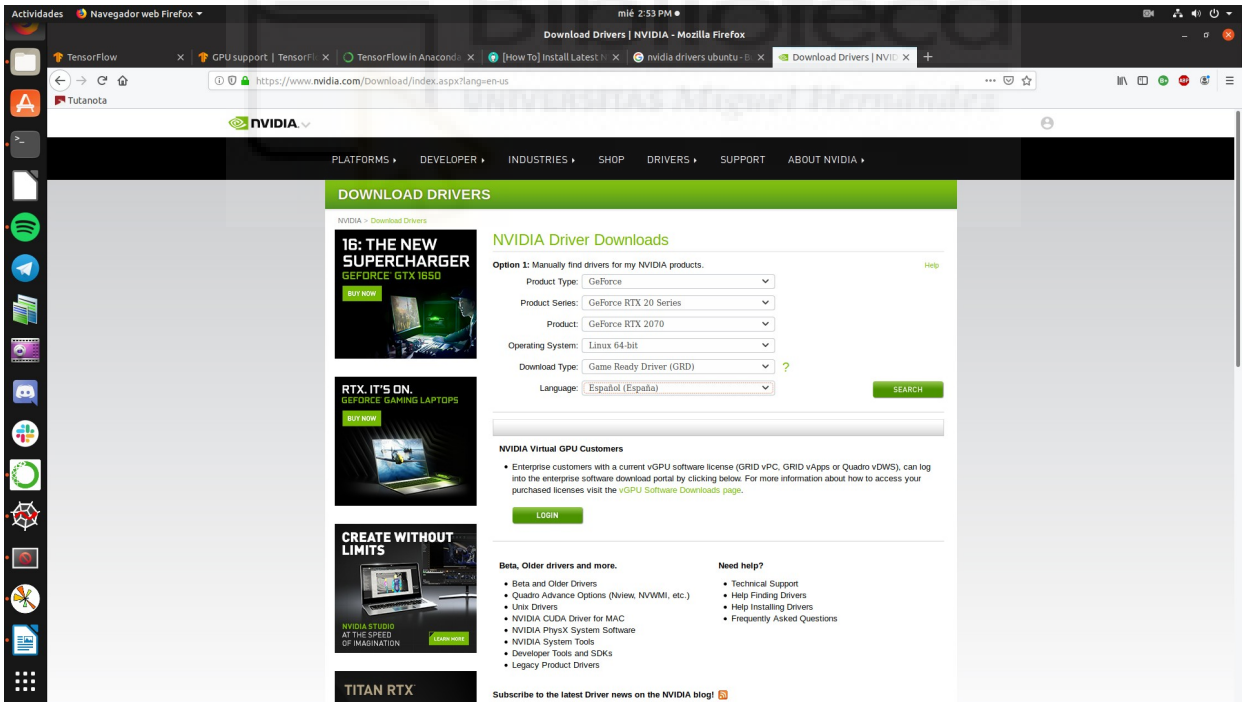

Figura 191: Captura de pantalla sección descargas de drivers de Nvidia, con la selección correctamente realizada para la descarga

3. Tras pulsar el botón *search*, nos aparecerá una **página con los últimos drivers**. Pulsamos sobre la pestaña ***productos soportados*** y **comprobamos que** efectivamente **sea compatible con la tarjeta gráfica que tenemos instalada en nuestro ordenador.**

Figura 192: Comprobación de que los drivers que vamos a descargar efectivamente son compatibles con nuestra tarjeta gráfica

4. Tras pulsar el **botón *descargar ahora*** y esperar a que se descargue el **archivo con la extensión *.run***, procederemos a su instalación.

5. Primero, deberemos **hacer el archivo *.run* ejecutable**; para ello **abriremos el terminal, y desde el directorio en el que esté el archivo descargado, escribiremos:**

   ```
   chmod +x Nombre_del_archivo_que_hemos_descargado.run
   ```

6. **Para proceder a la instalación del archivo, simplemente escribimos** `./Nombre_del_archivo_que_hemos_descargado.run` nuevamente **desde el directorio en el que está el archivo.**

7. **Seguimos las instrucciones que aparecerán en pantalla** y esperamos que finalice la instalación.

8. **Reiniciamos el PC** para que los cambios se hagan efectivos.

[211]

II.III. Instalación de Tensorflow bajo anaconda

Para realizar la **instalación de tensorflow bajo anaconda**, simplemente tendremos que **escribir en el terminal:**

```
conda create -n tensorflow_env tensorflow
```

si queremos instalar la versión sin soporte para GPU; **si en cambio, queremos utilizar la GPU** (el cual es nuestro caso), **escribiremos en el terminal:**

```
conda create -n tensorflow_gpuenv tensorflow-gpu
```

Tras finalizar la instalación, tendremos el entorno creado; **si arrancamos anaconda escribiendo en el terminal**

```
anaconda-navigator
```

para iniciar el entorno gráfico de anaconda; encontraremos una **pestaña donde pone *applications on*** con un desplegable. Si pulsamos ese desplegable, deberá estar disponible el **entorno de tensorflow.** Bastará con seleccionarlo y esperar a que cargue para poder utilizar las aplicaciones que tengamos instaladas en el entorno de tensorflow.

[212]

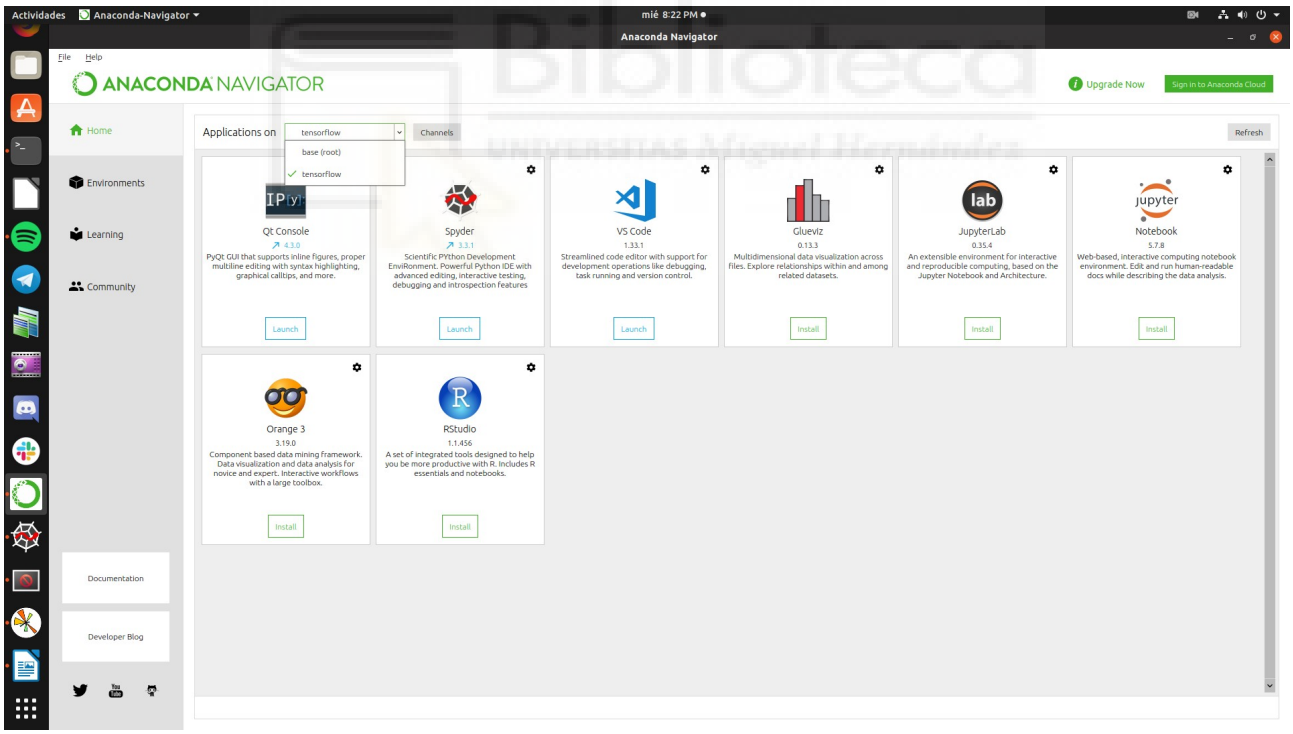

Figura 193: Captura de pantalla de anaconda. Se aprecia como en el desplegable aparece el entorno "base" y "tensorflow"

II.IV. Instalación de CUDA en anaconda

En el caso de que durante la instalación de tensorflow no se hayan instalado todos los paquetes necesarios para su uso con la GPU, **necesitaremos de los paquetes CUDA (cudatoolkit y cuDNN).** Para instalar el paquete de herramientas de cuda en anaconda, primero deberemos activar el entorno correspondiente (en este caso el de tensorflow). Para ello, escribiremos en el terminal:

```
conda activate tensorflow_gpuenv
```

Si hemos puesto otro nombre a nuestro entorno, introduciremos el nombre correspondiente. Tras esto, **escribiremos por consola:**

```
conda install -c anaconda cudatoolkit
```

[213]

II.V. Instalación de cuDNN SDK en linux

El otro paquete que debemos instalar bajo anaconda, en caso de que no se haya instalado durante la instalación de tensorflow, es cuDNN. **Este paquete el el encargado de permitirnos ejecutar redes neuronales utilizando la tarjeta gráfica, lo cual acelerará la ejecución de cualquier agente de inteligencia artificial.**

Para instalar este paquete, tras haber activado nuestro entorno con:

```
conda activate tensorflow_gpuenv
```

escribiremos en el terminal:

```
conda install -c anaconda cudnn
```
[214]

Anexo III. Problemas surgidos durante la instalación o relacionados con ella

En este anexo **se detallarán los problemas surgidos durante la instalación de los componentes y programas utilizados durante la realización de este trabajo** final de máster, o durante la utilización de los mismos.

III.I Pantalla en negro en Spyder tras la instalación

Tras la instalación de anaconda, se produjo un error en el cual **Spyder**, el entorno de desarrollo integrado utilizado durante la realización de todo este trabajo fin de máster, **al iniciarse presentaba la pantalla en negro, impidiendo su utilización.**

Este problema **surgió durante su uso en Ubuntu 16.04**; no se volvió a experimentar tras realizar la actualización del sistema operativo a la versión 18.10.

La solución al problema consistió en lo siguiente:

1. **Instalar el paquete pyopengl**: para ello, introduciremos la siguiente orden en el terminal:

   ```
   pip install PyOpenGL PyOpenGL_accelerate
   ```

   Esto debemos hacerlo tanto en el entorno normal como en tensorflow; para activar tensorflow desde la línea de comandos, debemos escribir:

   ```
   source activate tensorflow
   ```

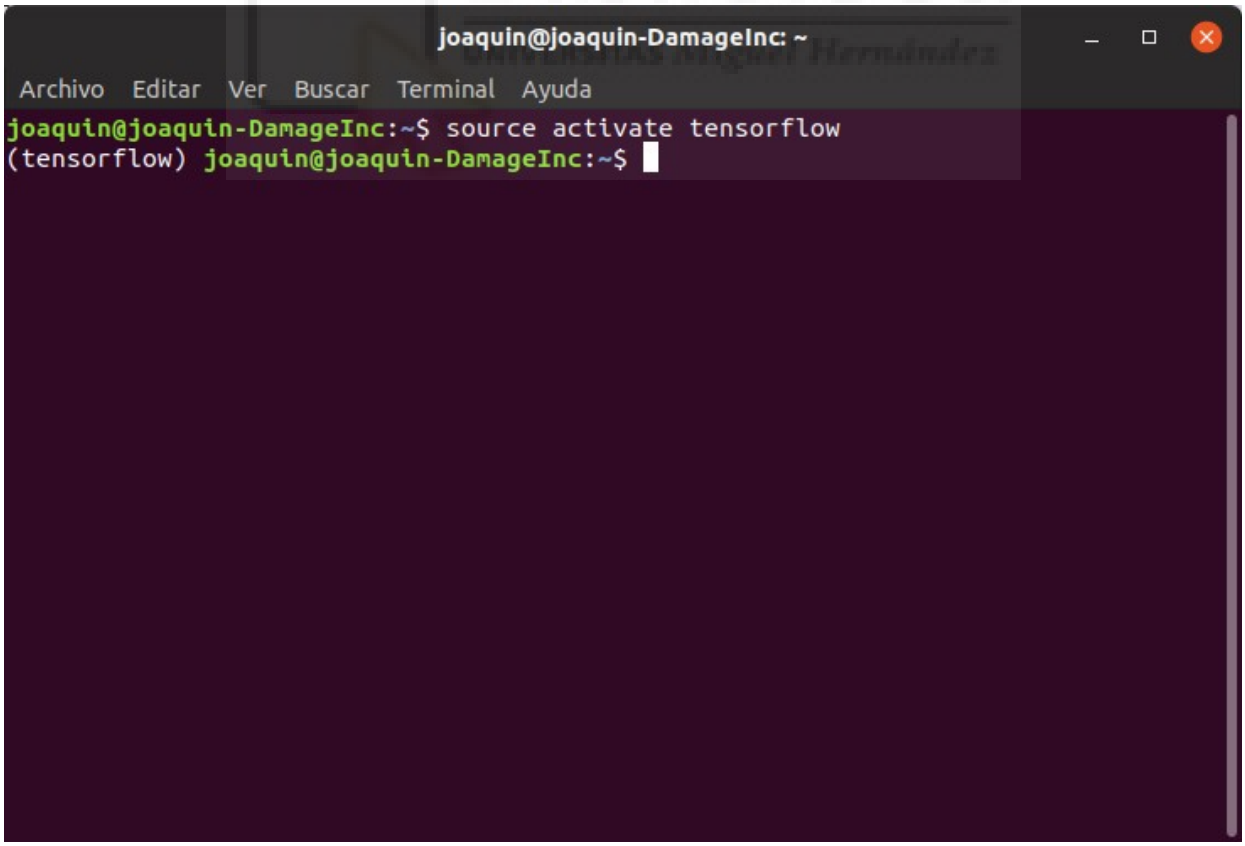

Figura 194: Captura de pantalla del terminal, en ella se muestra como activar tensorflow y el aspecto de la línea de comandos tras activarlo

2. **Añadimos**

   ```
   from OpenGL import GL
   ```

   en el archivo *start.py* de Spyder; en nuestro caso al estar instalado bajo anaconda, **la ruta para encontrar el archivo es *anaconda3/lib/python3.5/site-packages/spyder/app/start.py***

[215]

III.II Error al cambiar de gráfica

Durante la realización del proyecto, **se cambiaron las dos tarjetas gráficas Nvidia Geforce 1050 Ti OC por una Nvidia Geforce RTX 2070, la cual nos ofrece mayor rendimiento.**

Figura 195: Una de las Nvidia GeForce GTX1050Ti Oc sustituidas

Figura 196: Tarjeta gráfica Nvidia GeForce RTX2070, que sustituyó a la anterior 1050Ti

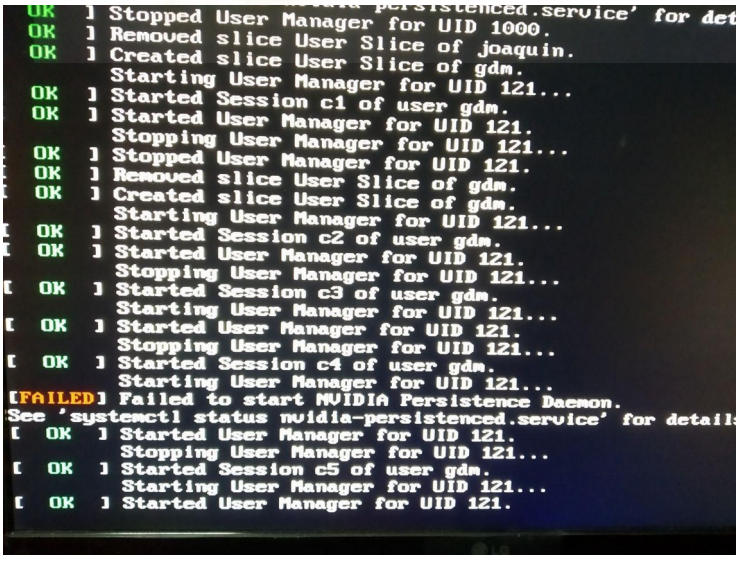

Figura 197: Foto de la pantalla durante el chequeo inicial. Se aprecia como falla al iniciar el Nvidia persistence daemon, lo que nos da una pista sobre cual es el fallo.

Al realizar el cambio de tarjeta, instalar los drivers, las librerías CUDA y cuDNN [216], s**e produjo un error que nos impidió entrar a Ubuntu** (el proyecto está realizado íntegramente bajo este sistema operativo). **La solución consistió en utilizar un disco externo, arrancar desde ahí Ubuntu como *Live CD* y desde ahí borrar tanto los drivers como las librerías CUDA y cuDNN. Tras esto, se realizó el inicio normalmente, procediendo a instalar de nuevo los drivers** (el problema sucedía tras instalar cuDNN y reiniciar).

Investigando a fondo la cuestión, se llega a la conclusión de que **el fallo había sido producido por una falta de compatibilidad con el sistema operativo** (Se estaba usando Ubuntu 17.10 LTS pero las librerías CUDA no existían para esa distribución, solo para 16.04, para 18.04 o 18.10). Así que el primer paso fue actualizar el sistema operativo a la versión 18.10 [217].

Una vez realizada la actualización, se procede a comprobar que todo funciona correctamente en anaconda; resultando en un **fallo de núcleo en tensorflow en el que no puede encontrar el algoritmo de la convolución.**

Investigando, **el problema residía en que anaconda no soporta la versión de cuDNN que necesita CUDA 10.1** (la más novedosa que soporta la nueva tarjeta gráfica instalada). Anaconda soporta actualmente la versión de cuDNN 7.3.1; la versión más reciente de cuDNN es la 7.5.1 y aún no es soportada por la misma.

Se actualiza tensorflow desde anaconda a la última versión estable (1.13; existe una versión alpha actualmente 2.0) **pero el problema persiste.**

Solución: instalar en anaconda tensorflow 1.8.0; ya que como se indica en [218] podría solucionar el problema.

Tras instalar la versión 1.8.0 de tensorflow, esta soporta cuDNN 7.3.1, así que nuestro problema quedó solucionado.

Anexo IV. Uso de Google Colaboratory

IV.I ¿Qué es Google Colaboratory?

Google Colaboratory (o Google Colab) es una **herramienta de investigación para la educación y exploración del aprendizaje automático**. El entorno en el que **se basa es un bloc de notas Jupyter**, el cual se puede usar en línea sin configuración prácticamente. **Es gratuito**, compatible con la mayoría de navegadores, y está testeado en Chrome, Firefox y Safari.

Lo interesante de este proyecto es que **pone a nuestra disposición tarjetas gráficas Nvidia Tesla K80 de gran potencia** para nuestro uso (durante 12 horas seguidas por sesión), así como TPUs de Google, totalmente gratis.

Los blocs creados se almacenan en Google Drive, y se pueden compartir de la misma forma que las hojas de cálculo o documentos de Google. También **se pueden importar archivos directamente a Colaboratory**. Los blocs de notas de Jupyter, se almacenan en su formato (.ipynb). El código se ejecuta en una máquina virtual exclusiva para nuestra cuenta; estas máquinas virtuales son recicladas cuando están inactivas durante un tiempo prolongado.

Actualmente, Colab es **compatible con Python 2.7 y Python 3.6**. En el futuro, Google se plantea expandir la compatibilidad del mismo a otros lenguajes como R o Scala [219].

IV.II Ejecución de código en Google Colaboratory

Lo primero que debemos hacer es **abrir** nuestro **Google Drive**; si hemos iniciado previamente ya nuestra libreta, haciendo doble click sobre ella; si no debemos ir a Mi unidad, y o bien definir una carpeta y después **seleccionar *Mi Unidad* > *Más* > *Colaboratory*** o directamente seleccionarlo si queremos que se cree directamente el archivo en nuestro drive.

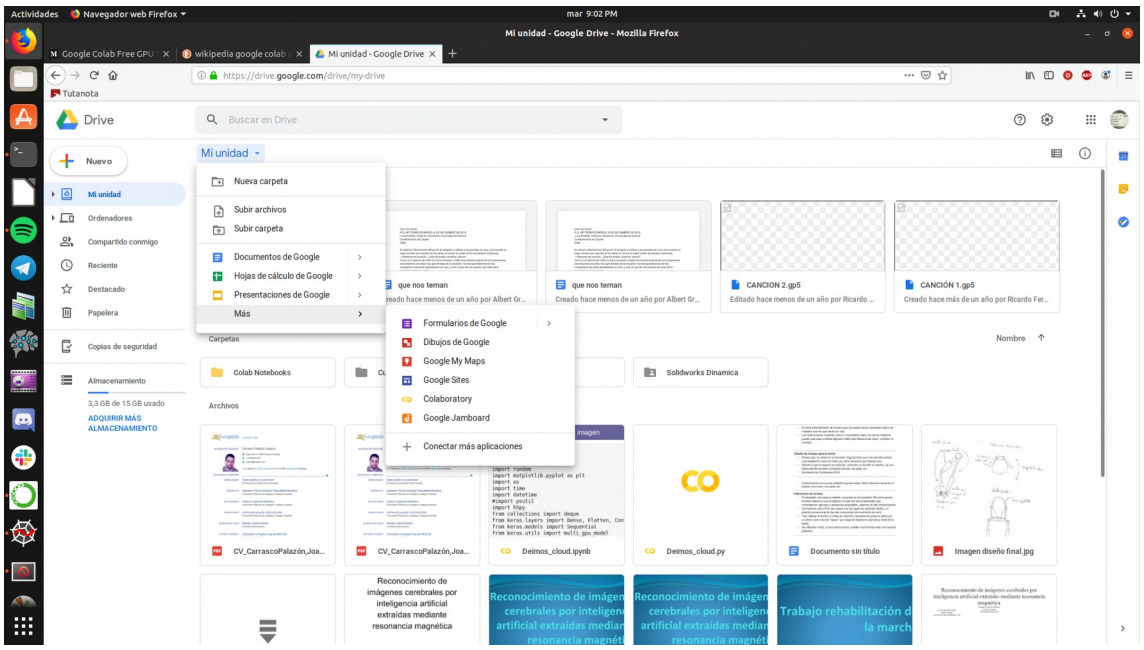

Figura 198: Captura de Pantalla Google Drive. En ella se muestra donde como acceder a Colaboratory.

Una vez hemos hecho click sobre Colaboratory, se nos abrirá una pestaña similar a esta:

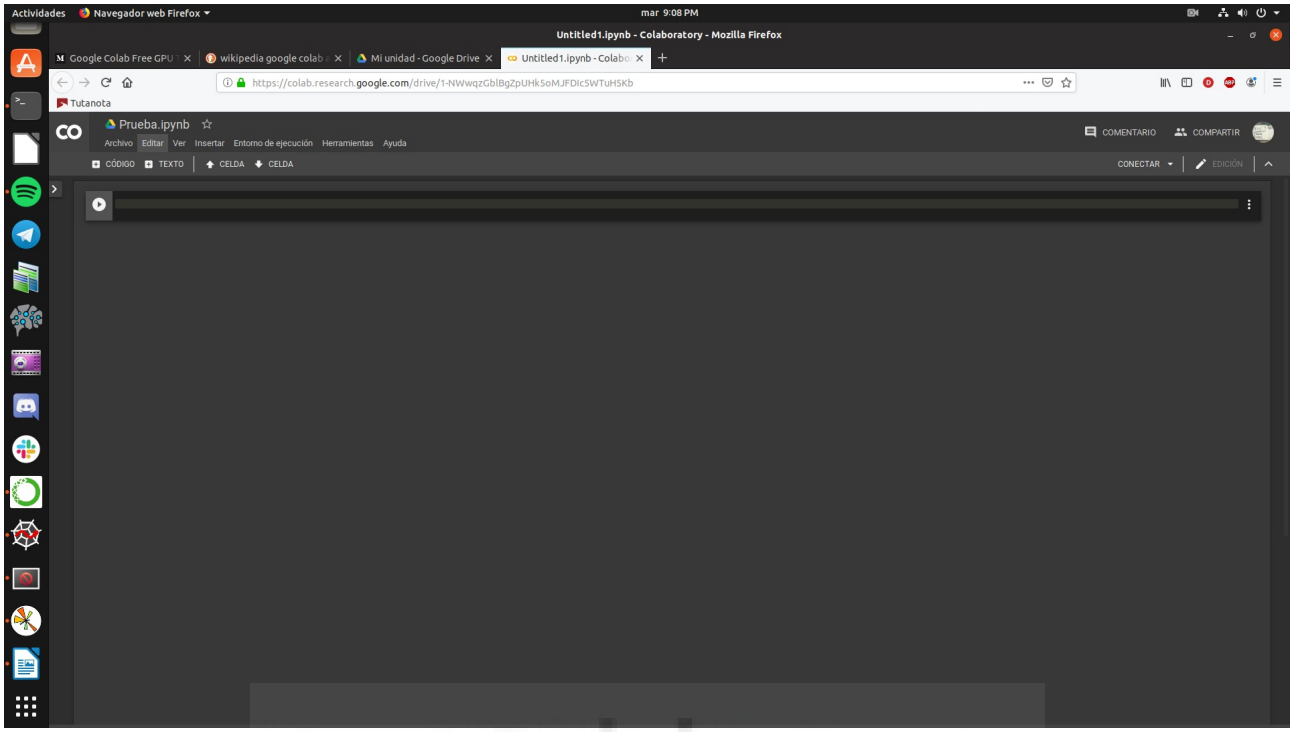

Figura 199: Captura de pantalla Google Colab, de una libreta vacía

En la misma, si hacemos click en el título con la extensión *.ipynb* podremos cambiar el nombre. Para este anexo se ha elegido *Prueba.ipynb*.

Nota: *el color del tema por defecto es blanco, para cambiar el tema a negro debemos hacer click en Herramientas>Preferencias… y ahí elegir en el tema entre las tres posibles (light, dark o adaptive) la que más nos guste.*

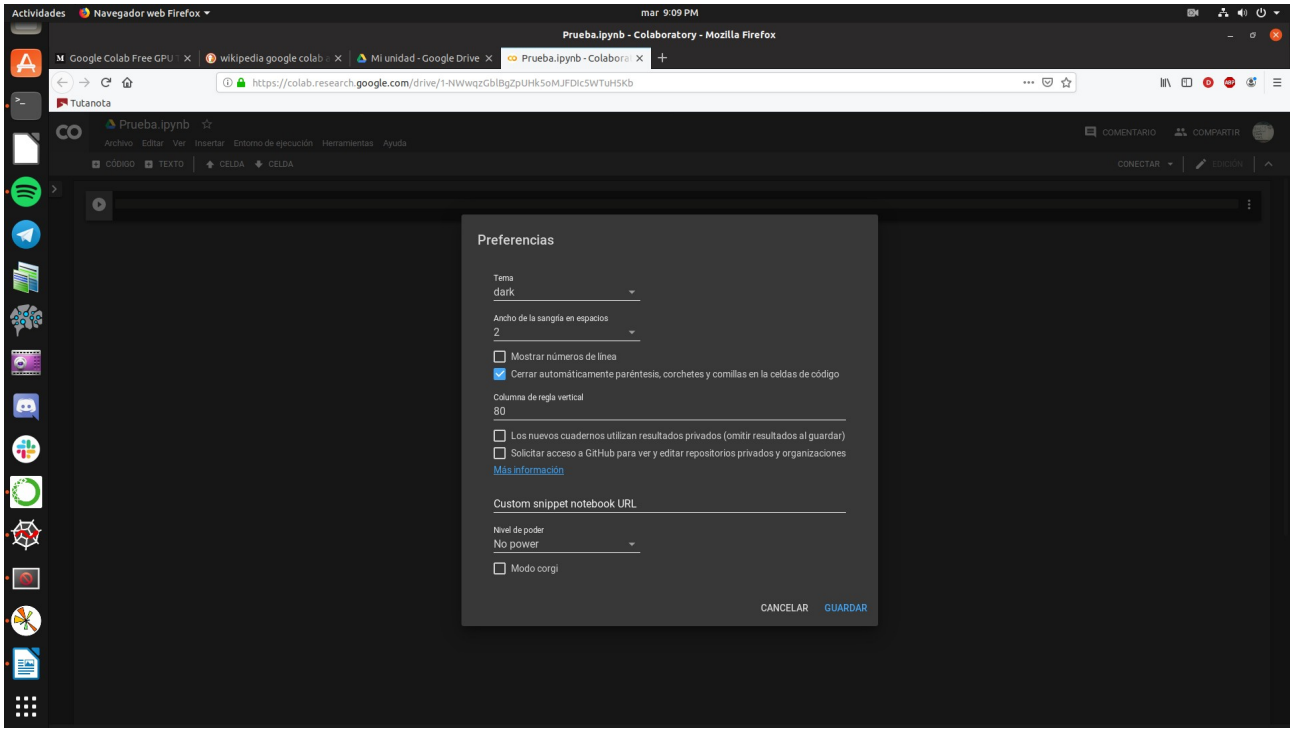

Figura 200: Captura de pantalla donde se muestra como cambiar el tema en Colab

Una vez realizado esto, puesto que **para este proyecto utilizaremos la GPU que pone a nuestra disposición Google**, deberemos seleccionarla. Para ello haremos click en ***Entorno de Ejecución>Cambiar tipo de entorno de ejecución*** y en las opciones disponibles, en ***Acelerador por Hardware* seleccionar *GPU*.**

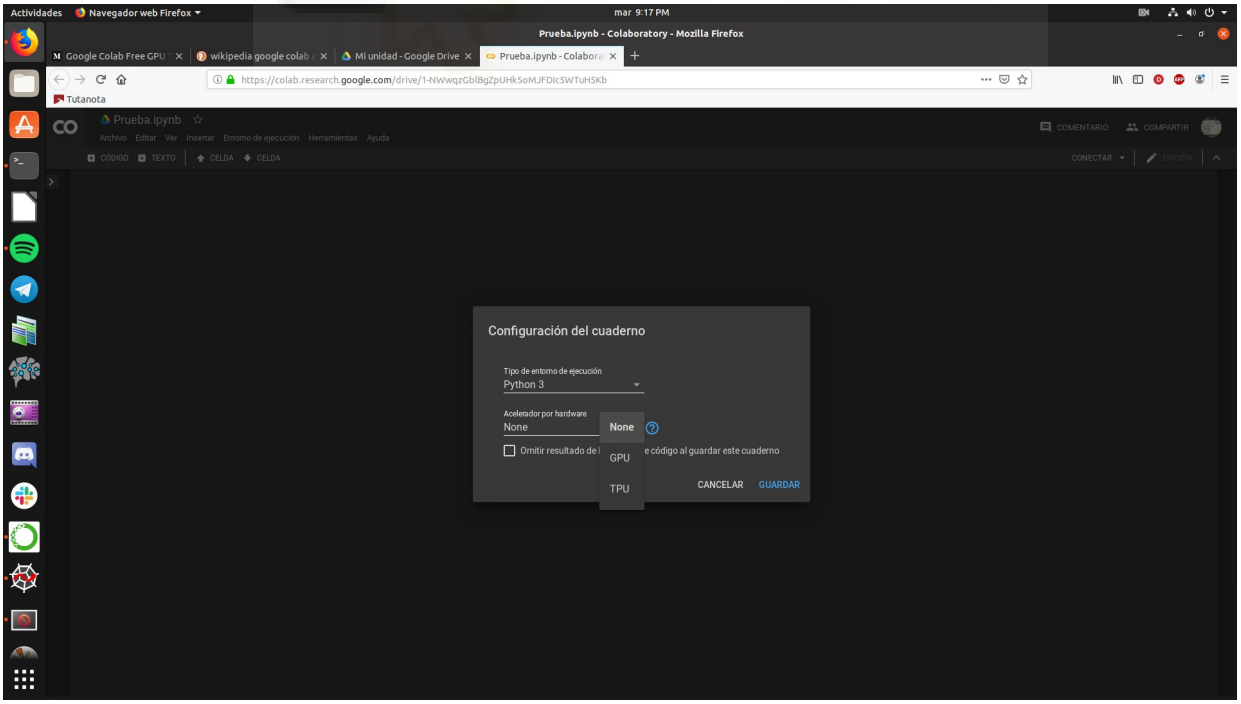

Figura 201: Captura de pantalla Google Colab, en ella se muestran los distintos entornos de ejecución disponibles

Comprobamos que todo funciona correctamente, ejecutando un sencillo código de prueba; para ejecutar el código debemos hacer click en el botón de play en la línea o líneas de código que queramos ejecutar; debajo nos aparecerá la salida que genera ese código:

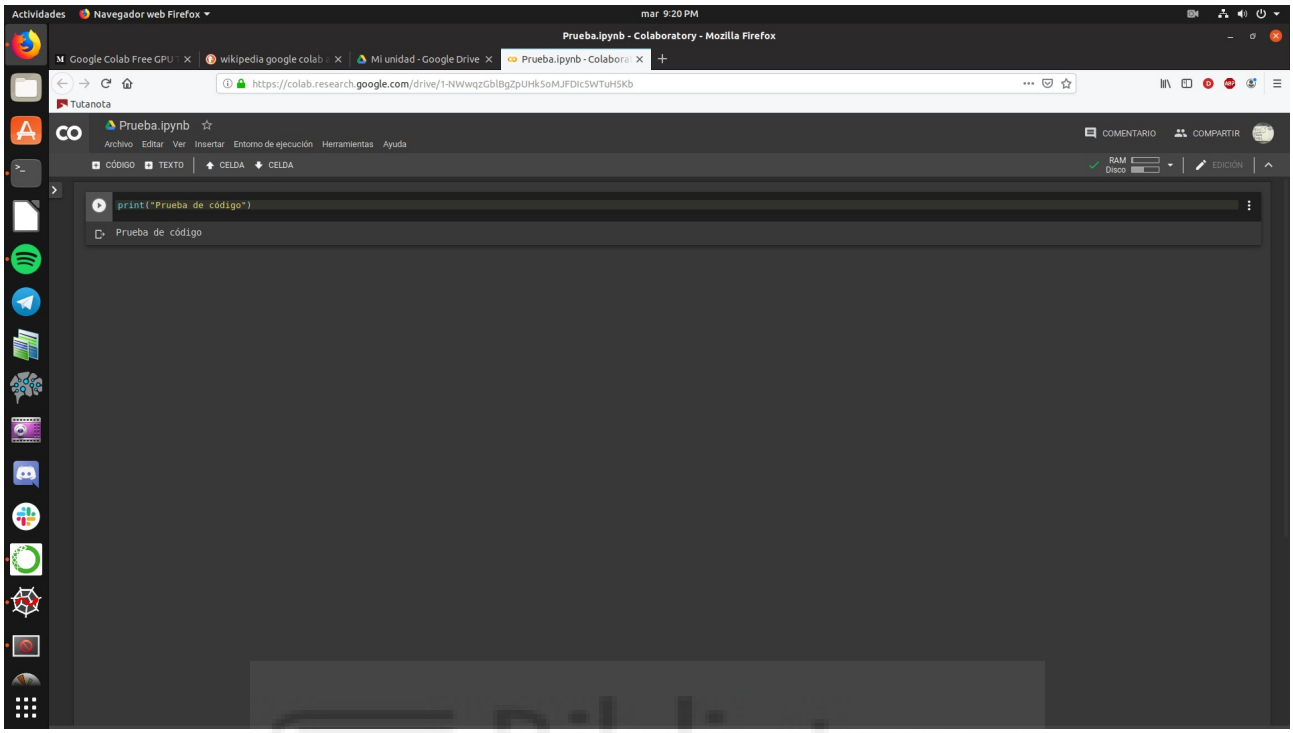

Figura 202: Captura de pantalla prueba código Google Colab. En ella se muestra el resultado de la ejecución de una simple línea de código

IV.III Instalación de librerías adicionales en Colab

Antes de proceder a la ejecución de nuestro código, **necesitaremos instalar ciertas librerías que no están instaladas por defecto. Las librerías que necesitamos instalar son:**

- **Keras:** para ello ejecutaremos la siguiente instrucción:

```
!pip install -q keras
     import keras
```

- **OpenCV:**

```
!apt-get -qq install -y libsm6 libxext6 && pip install -q -U opencv-
python
     import cv2
```

- **OpenAI-Gym:**

```
!pip install gym
```

[220]

Con estas tres librerías ya deberíamos ser capaces de ejecutar nuestro algoritmo. En caso de necesitar alguna librería más, ejecutamos `!pip install o !apt-get` **y el nombre de la librería.**

Hay que recordar que **debemos guardar los archivos antes de que termine la sesión; si la sesión se corta, los archivos generados se perderán. El programa genera tres archivos: uno con los pesos de las neuronas, otro con los datos para realizar las gráficas posteriormente, y otro con datos del programa** tales como el número de pasos, episodios, etc. Los archivos de datos y pesos de las neuronas son exactamente iguales que los generados cuando se ejecuta desde local.

Al guardar los archivos tras una sesión (recordemos que el programa realiza un guardado cada 10 episodios), nos permite continuar con el entrenamiento: solo debemos ejecutar de nuevo el código, esperar a que genere la estructura de archivos (ya que la misma se borra; por tanto es como empezar de nuevo), parar la ejecución, subir los archivos de datos y pesos de las neuronas, y ejecutar de nuevo. De esta forma, el programa leerá los datos almacenados previamente y continuará la sesión por donde se quedó tras el último punto de guardado.

Anexo V. Entornos disponibles en OpenAI-Gym

Se presenta la **lista completa de entornos disponibles en OpenAI-Gym**. La nomenclatura de los entornos sigue la siguiente norma, tal y como se explica en [221]:

- **v0 tiene una probabilidad de repetir la acción automáticamente del 0,25**, es decir, que el 25% de las veces la acción anterior se repetirá en vez de utilizar una acción nueva.
- **v4 tiene una probabilidad de repetición del 0%**, es decir, siempre realizará la acción que se decida.
- **Los entornos** `Deterministic` **tienen un *frameskip* fijo de 4**, mientras que en los demás se elimina un cuadro entre el segundo y el quinto aleatoriamente.
- **Existen los entornos** `NoFrameskip`, en los que **no se eliminan los cuadros**; en caso de que queramos realizar nosotros mismos el *frameskip*, deberemos utilizar este entorno.
- Los entornos `ram` extraen la información de la memoria; este tipo de entornos no nos interesan para este proyecto, ya que lo que se busca es extraer información a través de imágenes.

La instalación de los entornos se explica en [222] y [223]. Asimismo, **algunos entornos puede que no estén disponibles ya que se necesita la instalación de motores físicos adicionales** como en el caso de *MuJoCo* (siglas de *Multi-Joint dynamics with Contact*) [224].

La lista completa de entornos se puede obtener tras importar los entornos de OpenAI-gym en Python (`from gym import envs`) y escribir en consola `print(envs.registry.all())`,

tal y como se muestra en [225]:

- `Copy-v0`
- `RepeatCopy-v0`
- `ReversedAddition-v0`
- `ReversedAddition3-v0`
- `DuplicatedInput-v0`
- `Reverse-v0`
- `CartPole-v0`
- `CartPole-v1`
- `MountainCar-v0`
- `MountainCarContinuous-v0`
- `Pendulum-v0`
- `Acrobot-v1`
- `LunarLander-v2`
- `LunarLanderContinuous-v2`
- `BipedalWalker-v2`
- `BipedalWalkerHardcore-v2`
- `CarRacing-v0`
- `Blackjack-v0`
- `KellyCoinflip-v0`
- `KellyCoinflipGeneralized-v0`
- `FrozenLake-v0`
- `FrozenLake8x8-v0`
- `CliffWalking-v0`
- `Nchain-v0`
- `Roulette-v0`
- `Taxi-v2`
- `GuessingGame-v0`

- HotterColder-v0
- Reacher-v2
- Pusher-v2
- Thrower-v2
- Striker-v2
- InvertedPendulum-v2
- InvertedDoublePendulum-v2
- HalfCheetah-v2
- Hopper-v2
- Swimmer-v2
- Walker2d-v2
- Ant-v2
- Humanoid-v2
- HumanoidStandup-v2
- FetchSlide-v1
- FetchPickAndPlace-v1
- FetchReach-v1
- FetchPush-v1
- HandReach-v0
- HandManipulateBlockRotateZ-v0
- HandManipulateBlockRotateParallel-v0
- HandManipulateBlockRotateXYZ-v0
- HandManipulateBlockFull-v0
- HandManipulateBlock-v0
- HandManipulateEggRotate-v0
- HandManipulateEggFull-v0
- HandManipulateEgg-v0
- HandManipulatePenRotate-v0
- HandManipulatePenFull-v0
- HandManipulatePen-v0
- FetchSlideDense-v1
- FetchPickAndPlaceDense-v1
- FetchReachDense-v1
- FetchPushDense-v1
- HandReachDense-v0
- HandManipulateBlockRotateZDense-v0
- HandManipulateBlockRotateParallelDense-v0
- HandManipulateBlockRotateXYZDense-v0
- HandManipulateBlockFullDense-v0
- HandManipulateBlockDense-v0
- HandManipulateEggRotateDense-v0
- HandManipulateEggFullDense-v0
- HandManipulateEggDense-v0
- HandManipulatePenRotateDense-v0
- HandManipulatePenFullDense-v0
- HandManipulatePenDense-v0
- AirRaid-v0
- AirRaid-v4
- AirRaidDeterministic-v0
- AirRaidDeterministic-v4
- AirRaidNoFrameskip-v0
- AirRaidNoFrameskip-v4
- AirRaid-ram-v0
- AirRaid-ram-v4
- AirRaid-ramDeterministic-v0
- AirRaid-ramDeterministic-v4
- AirRaid-ramNoFrameskip-v0
- AirRaid-ramNoFrameskip-v4
- Alien-v0
- Alien-v4
- AlienDeterministic-v0

- AlienDeterministic-v4
- AlienNoFrameskip-v0
- AlienNoFrameskip-v4
- Alien-ram-v0
- Alien-ram-v4
- Alien-ramDeterministic-v0
- Alien-ramDeterministic-v4
- Alien-ramNoFrameskip-v0
- Alien-ramNoFrameskip-v4
- Amidar-v0
- Amidar-v4
- AmidarDeterministic-v0
- AmidarDeterministic-v4
- AmidarNoFrameskip-v0
- AmidarNoFrameskip-v4
- Amidar-ram-v0
- Amidar-ram-v4
- Amidar-ramDeterministic-v0
- Amidar-ramDeterministic-v4
- Amidar-ramNoFrameskip-v0
- Amidar-ramNoFrameskip-v4
- Assault-v0
- Assault-v4
- AssaultDeterministic-v0
- AssaultDeterministic-v4
- AssaultNoFrameskip-v0
- AssaultNoFrameskip-v4
- Assault-ram-v0
- Assault-ram-v4
- Assault-ramDeterministic-v0
- Assault-ramDeterministic-v4
- Assault-ramNoFrameskip-v0
- Assault-ramNoFrameskip-v4
- Asterix-v0
- Asterix-v4
- AsterixDeterministic-v0
- AsterixDeterministic-v4
- AsterixNoFrameskip-v0
- AsterixNoFrameskip-v4
- Asterix-ram-v0
- Asterix-ram-v4
- Asterix-ramDeterministic-v0
- Asterix-ramDeterministic-v4
- Asterix-ramNoFrameskip-v0
- Asterix-ramNoFrameskip-v4
- Asteroids-v0
- Asteroids-v4
- AsteroidsDeterministic-v0
- AsteroidsDeterministic-v4
- AsteroidsNoFrameskip-v0
- AsteroidsNoFrameskip-v4
- Asteroids-ram-v0
- Asteroids-ram-v4
- Asteroids-ramDeterministic-v0
- Asteroids-ramDeterministic-v4
- Asteroids-ramNoFrameskip-v0
- Asteroids-ramNoFrameskip-v4
- Atlantis-v0
- Atlantis-v4
- AtlantisDeterministic-v0
- AtlantisDeterministic-v4

- AtlantisNoFrameskip-v0
- AtlantisNoFrameskip-v4
- Atlantis-ram-v0
- Atlantis-ram-v4
- Atlantis-ramDeterministic-v0
- Atlantis-ramDeterministic-v4
- Atlantis-ramNoFrameskip-v0
- Atlantis-ramNoFrameskip-v4
- BankHeist-v0
- BankHeist-v4
- BankHeistDeterministic-v0
- BankHeistDeterministic-v4
- BankHeistNoFrameskip-v0
- BankHeistNoFrameskip-v4
- BankHeist-ram-v0
- BankHeist-ram-v4
- BankHeist-ramDeterministic-v0
- BankHeist-ramDeterministic-v4
- BankHeist-ramNoFrameskip-v0
- BankHeist-ramNoFrameskip-v4
- BattleZone-v0
- BattleZone-v4
- BattleZoneDeterministic-v0
- BattleZoneDeterministic-v4
- BattleZoneNoFrameskip-v0
- BattleZoneNoFrameskip-v4
- BattleZone-ram-v0
- BattleZone-ram-v4
- BattleZone-ramDeterministic-v0
- BattleZone-ramDeterministic-v4
- BattleZone-ramNoFrameskip-v0
- BattleZone-ramNoFrameskip-v4
- BeamRider-v0
- BeamRider-v4
- BeamRiderDeterministic-v0
- BeamRiderDeterministic-v4
- BeamRiderNoFrameskip-v0
- BeamRiderNoFrameskip-v4
- BeamRider-ram-v0
- BeamRider-ram-v4
- BeamRider-ramDeterministic-v0
- BeamRider-ramDeterministic-v4
- BeamRider-ramNoFrameskip-v0
- BeamRider-ramNoFrameskip-v4
- Berzerk-v0
- Berzerk-v4
- BerzerkDeterministic-v0
- BerzerkDeterministic-v4
- BerzerkNoFrameskip-v0
- BerzerkNoFrameskip-v4
- Berzerk-ram-v0
- Berzerk-ram-v4
- Berzerk-ramDeterministic-v0
- Berzerk-ramDeterministic-v4
- Berzerk-ramNoFrameskip-v0
- Berzerk-ramNoFrameskip-v4
- Bowling-v0
- Bowling-v4
- BowlingDeterministic-v0
- BowlingDeterministic-v4
- BowlingNoFrameskip-v0

- BowlingNoFrameskip-v4
- Bowling-ram-v0
- Bowling-ram-v4
- Bowling-ramDeterministic-v0
- Bowling-ramDeterministic-v4
- Bowling-ramNoFrameskip-v0
- Bowling-ramNoFrameskip-v4
- Boxing-v0
- Boxing-v4
- BoxingDeterministic-v0
- BoxingDeterministic-v4
- BoxingNoFrameskip-v0
- BoxingNoFrameskip-v4
- Boxing-ram-v0
- Boxing-ram-v4
- Boxing-ramDeterministic-v0
- Boxing-ramDeterministic-v4
- Boxing-ramNoFrameskip-v0
- Boxing-ramNoFrameskip-v4
- Breakout-v0
- Breakout-v4
- BreakoutDeterministic-v0
- BreakoutDeterministic-v4
- BreakoutNoFrameskip-v0
- BreakoutNoFrameskip-v4
- Breakout-ram-v0
- Breakout-ram-v4
- Breakout-ramDeterministic-v0
- Breakout-ramDeterministic-v4
- Breakout-ramNoFrameskip-v0
- Breakout-ramNoFrameskip-v4
- Carnival-v0
- Carnival-v4
- CarnivalDeterministic-v0
- CarnivalDeterministic-v4
- CarnivalNoFrameskip-v0
- CarnivalNoFrameskip-v4
- Carnival-ram-v0
- Carnival-ram-v4
- Carnival-ramDeterministic-v0
- Carnival-ramDeterministic-v4
- Carnival-ramNoFrameskip-v0
- Carnival-ramNoFrameskip-v4
- Centipede-v0
- Centipede-v4
- CentipedeDeterministic-v0
- CentipedeDeterministic-v4
- CentipedeNoFrameskip-v0
- CentipedeNoFrameskip-v4
- Centipede-ram-v0
- Centipede-ram-v4
- Centipede-ramDeterministic-v0
- Centipede-ramDeterministic-v4
- Centipede-ramNoFrameskip-v0
- Centipede-ramNoFrameskip-v4
- ChopperCommand-v0
- ChopperCommand-v4
- ChopperCommandDeterministic-v0
- ChopperCommandDeterministic-v4
- ChopperCommandNoFrameskip-v0
- ChopperCommandNoFrameskip-v4

- ChopperCommand-ram-v0
- ChopperCommand-ram-v4
- ChopperCommand-ramDeterministic-v0
- ChopperCommand-ramDeterministic-v4
- ChopperCommand-ramNoFrameskip-v0
- ChopperCommand-ramNoFrameskip-v4
- CrazyClimber-v0
- CrazyClimber-v4
- CrazyClimberDeterministic-v0
- CrazyClimberDeterministic-v4
- CrazyClimberNoFrameskip-v0
- CrazyClimberNoFrameskip-v4
- CrazyClimber-ram-v0
- CrazyClimber-ram-v4
- CrazyClimber-ramDeterministic-v0
- CrazyClimber-ramDeterministic-v4
- CrazyClimber-ramNoFrameskip-v0
- CrazyClimber-ramNoFrameskip-v4
- DemonAttack-v0
- DemonAttack-v4
- DemonAttackDeterministic-v0
- DemonAttackDeterministic-v4
- DemonAttackNoFrameskip-v0
- DemonAttackNoFrameskip-v4
- DemonAttack-ram-v0
- DemonAttack-ram-v4
- DemonAttack-ramDeterministic-v0
- DemonAttack-ramDeterministic-v4
- DemonAttack-ramNoFrameskip-v0
- DemonAttack-ramNoFrameskip-v4
- DoubleDunk-v0
- DoubleDunk-v4
- DoubleDunkDeterministic-v0
- DoubleDunkDeterministic-v4
- DoubleDunkNoFrameskip-v0
- DoubleDunkNoFrameskip-v4
- DoubleDunk-ram-v0
- DoubleDunk-ram-v4
- DoubleDunk-ramDeterministic-v0
- DoubleDunk-ramDeterministic-v4
- DoubleDunk-ramNoFrameskip-v0
- DoubleDunk-ramNoFrameskip-v4
- ElevatorAction-v0
- ElevatorAction-v4
- ElevatorActionDeterministic-v0
- ElevatorActionDeterministic-v4
- ElevatorActionNoFrameskip-v0
- ElevatorActionNoFrameskip-v4
- ElevatorAction-ram-v0
- ElevatorAction-ram-v4
- ElevatorAction-ramDeterministic-v0
- ElevatorAction-ramDeterministic-v4
- ElevatorAction-ramNoFrameskip-v0
- ElevatorAction-ramNoFrameskip-v4
- Enduro-v0
- Enduro-v4
- EnduroDeterministic-v0
- EnduroDeterministic-v4
- EnduroNoFrameskip-v0
- EnduroNoFrameskip-v4
- Enduro-ram-v0

- Enduro-ram-v4
- Enduro-ramDeterministic-v0
- Enduro-ramDeterministic-v4
- Enduro-ramNoFrameskip-v0
- Enduro-ramNoFrameskip-v4
- FishingDerby-v0
- FishingDerby-v4
- FishingDerbyDeterministic-v0
- FishingDerbyDeterministic-v4
- FishingDerbyNoFrameskip-v0
- FishingDerbyNoFrameskip-v4
- FishingDerby-ram-v0
- FishingDerby-ram-v4
- FishingDerby-ramDeterministic-v0
- FishingDerby-ramDeterministic-v4
- FishingDerby-ramNoFrameskip-v0
- FishingDerby-ramNoFrameskip-v4
- Freeway-v0
- Freeway-v4
- FreewayDeterministic-v0
- FreewayDeterministic-v4
- FreewayNoFrameskip-v0
- FreewayNoFrameskip-v4
- Freeway-ram-v0
- Freeway-ram-v4
- Freeway-ramDeterministic-v0
- Freeway-ramDeterministic-v4
- Freeway-ramNoFrameskip-v0
- Freeway-ramNoFrameskip-v4
- Frostbite-v0
- Frostbite-v4
- FrostbiteDeterministic-v0
- FrostbiteDeterministic-v4
- FrostbiteNoFrameskip-v0
- FrostbiteNoFrameskip-v4
- Frostbite-ram-v0
- Frostbite-ram-v4
- Frostbite-ramDeterministic-v0
- Frostbite-ramDeterministic-v4
- Frostbite-ramNoFrameskip-v0
- Frostbite-ramNoFrameskip-v4
- Gopher-v0
- Gopher-v4
- GopherDeterministic-v0
- GopherDeterministic-v4
- GopherNoFrameskip-v0
- GopherNoFrameskip-v4
- Gopher-ram-v0
- Gopher-ram-v4
- Gopher-ramDeterministic-v0
- Gopher-ramDeterministic-v4)
- Gopher-ramNoFrameskip-v0
- Gopher-ramNoFrameskip-v4
- Gravitar-v0
- Gravitar-v4
- GravitarDeterministic-v0
- GravitarDeterministic-v4
- GravitarNoFrameskip-v0
- GravitarNoFrameskip-v4
- Gravitar-ram-v0
- Gravitar-ram-v4

- Gravitar-ramDeterministic-v0
- Gravitar-ramDeterministic-v4
- Gravitar-ramNoFrameskip-v0
- Gravitar-ramNoFrameskip-v4
- Hero-v0
- Hero-v4
- HeroDeterministic-v0
- HeroDeterministic-v4
- HeroNoFrameskip-v0
- HeroNoFrameskip-v4
- Hero-ram-v0
- Hero-ram-v4
- Hero-ramDeterministic-v0
- Hero-ramDeterministic-v4
- Hero-ramNoFrameskip-v0
- Hero-ramNoFrameskip-v4
- IceHockey-v0
- IceHockey-v4
- IceHockeyDeterministic-v0
- IceHockeyDeterministic-v4
- IceHockeyNoFrameskip-v0
- IceHockeyNoFrameskip-v4
- IceHockey-ram-v0
- IceHockey-ram-v4
- IceHockey-ramDeterministic-v0
- IceHockey-ramDeterministic-v4
- IceHockey-ramNoFrameskip-v0
- IceHockey-ramNoFrameskip-v4
- Jamesbond-v0
- Jamesbond-v4
- JamesbondDeterministic-v0
- JamesbondDeterministic-v4
- JamesbondNoFrameskip-v0
- JamesbondNoFrameskip-v4
- Jamesbond-ram-v0
- Jamesbond-ram-v4
- Jamesbond-ramDeterministic-v0
- Jamesbond-ramDeterministic-v4
- Jamesbond-ramNoFrameskip-v0
- Jamesbond-ramNoFrameskip-v4
- JourneyEscape-v0
- JourneyEscape-v4
- JourneyEscapeDeterministic-v0
- JourneyEscapeDeterministic-v4
- JourneyEscapeNoFrameskip-v0
- JourneyEscapeNoFrameskip-v4
- JourneyEscape-ram-v0
- JourneyEscape-ram-v4
- JourneyEscape-ramDeterministic-v0
- JourneyEscape-ramDeterministic-v4
- JourneyEscape-ramNoFrameskip-v0
- JourneyEscape-ramNoFrameskip-v4
- Kangaroo-v0
- Kangaroo-v4
- KangarooDeterministic-v0
- KangarooDeterministic-v4
- KangarooNoFrameskip-v0
- KangarooNoFrameskip-v4
- Kangaroo-ram-v0
- Kangaroo-ram-v4
- Kangaroo-ramDeterministic-v0

- Kangaroo-ramDeterministic-v4
- Kangaroo-ramNoFrameskip-v0
- Kangaroo-ramNoFrameskip-v4
- Krull-v0
- Krull-v4
- KrullDeterministic-v0
- KrullDeterministic-v4
- KrullNoFrameskip-v0
- KrullNoFrameskip-v4
- Krull-ram-v0
- Krull-ram-v4
- Krull-ramDeterministic-v0
- Krull-ramDeterministic-v4
- Krull-ramNoFrameskip-v0
- Krull-ramNoFrameskip-v4
- KungFuMaster-v0
- KungFuMaster-v4
- KungFuMasterDeterministic-v0
- KungFuMasterDeterministic-v4
- KungFuMasterNoFrameskip-v0
- KungFuMasterNoFrameskip-v4
- KungFuMaster-ram-v0
- KungFuMaster-ram-v4
- KungFuMaster-ramDeterministic-v0
- KungFuMaster-ramDeterministic-v4
- KungFuMaster-ramNoFrameskip-v0
- KungFuMaster-ramNoFrameskip-v4
- MontezumaRevenge-v0
- MontezumaRevenge-v4
- MontezumaRevengeDeterministic-v0
- MontezumaRevengeDeterministic-v4
- MontezumaRevengeNoFrameskip-v0
- MontezumaRevengeNoFrameskip-v4
- MontezumaRevenge-ram-v0
- MontezumaRevenge-ram-v4
- MontezumaRevenge-ramDeterministic-v0
- MontezumaRevenge-ramDeterministic-v4
- MontezumaRevenge-ramNoFrameskip-v0
- MontezumaRevenge-ramNoFrameskip-v4
- MsPacman-v0
- MsPacman-v4
- MsPacmanDeterministic-v0
- MsPacmanDeterministic-v4
- MsPacmanNoFrameskip-v0
- MsPacmanNoFrameskip-v4
- MsPacman-ram-v0
- MsPacman-ram-v4
- MsPacman-ramDeterministic-v0
- MsPacman-ramDeterministic-v4
- MsPacman-ramNoFrameskip-v0
- MsPacman-ramNoFrameskip-v4
- NameThisGame-v0
- NameThisGame-v4
- NameThisGameDeterministic-v0
- NameThisGameDeterministic-v4
- NameThisGameNoFrameskip-v0
- NameThisGameNoFrameskip-v4
- NameThisGame-ram-v0
- NameThisGame-ram-v4
- NameThisGame-ramDeterministic-v0
- NameThisGame-ramDeterministic-v4

- NameThisGame-ramNoFrameskip-v0
- NameThisGame-ramNoFrameskip-v4
- Phoenix-v0
- Phoenix-v4
- PhoenixDeterministic-v0
- PhoenixDeterministic-v4
- PhoenixNoFrameskip-v0
- PhoenixNoFrameskip-v4
- Phoenix-ram-v0
- Phoenix-ram-v4
- Phoenix-ramDeterministic-v0
- Phoenix-ramDeterministic-v4
- Phoenix-ramNoFrameskip-v0
- Phoenix-ramNoFrameskip-v4
- Pitfall-v0
- Pitfall-v4
- PitfallDeterministic-v0
- PitfallDeterministic-v4
- PitfallNoFrameskip-v0
- PitfallNoFrameskip-v4
- Pitfall-ram-v0
- Pitfall-ram-v4
- Pitfall-ramDeterministic-v0
- Pitfall-ramDeterministic-v4
- Pitfall-ramNoFrameskip-v0
- Pitfall-ramNoFrameskip-v4
- Pong-v0
- Pong-v4
- PongDeterministic-v0
- PongDeterministic-v4
- PongNoFrameskip-v0
- PongNoFrameskip-v4
- Pong-ram-v0
- Pong-ram-v4
- Pong-ramDeterministic-v0
- Pong-ramDeterministic-v4
- Pong-ramNoFrameskip-v0
- Pong-ramNoFrameskip-v4
- Pooyan-v0
- Pooyan-v4
- PooyanDeterministic-v0
- PooyanDeterministic-v4
- PooyanNoFrameskip-v0
- PooyanNoFrameskip-v4
- Pooyan-ram-v0
- Pooyan-ram-v4
- Pooyan-ramDeterministic-v0
- Pooyan-ramDeterministic-v4
- Pooyan-ramNoFrameskip-v0
- Pooyan-ramNoFrameskip-v4
- PrivateEye-v0
- PrivateEye-v4
- PrivateEyeDeterministic-v0
- PrivateEyeDeterministic-v4
- PrivateEyeNoFrameskip-v0
- PrivateEyeNoFrameskip-v4
- PrivateEye-ram-v0
- PrivateEye-ram-v4
- PrivateEye-ramDeterministic-v0
- PrivateEye-ramDeterministic-v4
- PrivateEye-ramNoFrameskip-v0

- PrivateEye-ramNoFrameskip-v4
- Qbert-v0
- Qbert-v4
- QbertDeterministic-v0
- QbertDeterministic-v4
- QbertNoFrameskip-v0
- QbertNoFrameskip-v4
- Qbert-ram-v0
- Qbert-ram-v4
- Qbert-ramDeterministic-v0
- Qbert-ramDeterministic-v4
- Qbert-ramNoFrameskip-v0
- Qbert-ramNoFrameskip-v4
- Riverraid-v0
- Riverraid-v4
- RiverraidDeterministic-v0
- RiverraidDeterministic-v4
- RiverraidNoFrameskip-v0
- RiverraidNoFrameskip-v4
- Riverraid-ram-v0
- Riverraid-ram-v4
- Riverraid-ramDeterministic-v0
- Riverraid-ramDeterministic-v4
- Riverraid-ramNoFrameskip-v0
- Riverraid-ramNoFrameskip-v4
- RoadRunner-v0
- RoadRunner-v4
- RoadRunnerDeterministic-v0
- RoadRunnerDeterministic-v4
- RoadRunnerNoFrameskip-v0
- RoadRunnerNoFrameskip-v4
- RoadRunner-ram-v0
- RoadRunner-ram-v4
- RoadRunner-ramDeterministic-v0
- RoadRunner-ramDeterministic-v4
- RoadRunner-ramNoFrameskip-v0
- RoadRunner-ramNoFrameskip-v4
- Robotank-v0
- Robotank-v4
- RobotankDeterministic-v0
- RobotankDeterministic-v4
- RobotankNoFrameskip-v0
- RobotankNoFrameskip-v4
- Robotank-ram-v0
- Robotank-ram-v4
- Robotank-ramDeterministic-v0
- Robotank-ramDeterministic-v4
- Robotank-ramNoFrameskip-v0
- Robotank-ramNoFrameskip-v4
- Seaquest-v0
- Seaquest-v4
- SeaquestDeterministic-v0
- SeaquestDeterministic-v4
- SeaquestNoFrameskip-v0
- SeaquestNoFrameskip-v4
- Seaquest-ram-v0
- Seaquest-ram-v4
- Seaquest-ramDeterministic-v0
- Seaquest-ramDeterministic-v4
- Seaquest-ramNoFrameskip-v0
- Seaquest-ramNoFrameskip-v4

- Skiing-v0
- Skiing-v4
- SkiingDeterministic-v0
- SkiingDeterministic-v4
- SkiingNoFrameskip-v0
- SkiingNoFrameskip-v4
- Skiing-ram-v0
- Skiing-ram-v4
- Skiing-ramDeterministic-v0
- Skiing-ramDeterministic-v4
- Skiing-ramNoFrameskip-v0
- Skiing-ramNoFrameskip-v4
- Solaris-v0
- Solaris-v4
- SolarisDeterministic-v0
- SolarisDeterministic-v4
- SolarisNoFrameskip-v0
- SolarisNoFrameskip-v4
- Solaris-ram-v0
- Solaris-ram-v4
- Solaris-ramDeterministic-v0
- Solaris-ramDeterministic-v4
- Solaris-ramNoFrameskip-v0
- Solaris-ramNoFrameskip-v4
- SpaceInvaders-v0
- SpaceInvaders-v4
- SpaceInvadersDeterministic-v0
- SpaceInvadersDeterministic-v4
- SpaceInvadersNoFrameskip-v0
- SpaceInvadersNoFrameskip-v4
- SpaceInvaders-ram-v0
- SpaceInvaders-ram-v4
- SpaceInvaders-ramDeterministic-v0
- SpaceInvaders-ramDeterministic-v4
- SpaceInvaders-ramNoFrameskip-v0
- SpaceInvaders-ramNoFrameskip-v4
- StarGunner-v0
- StarGunner-v4
- StarGunnerDeterministic-v0
- StarGunnerDeterministic-v4
- StarGunnerNoFrameskip-v0
- StarGunnerNoFrameskip-v4
- StarGunner-ram-v0
- StarGunner-ram-v4
- StarGunner-ramDeterministic-v0
- StarGunner-ramDeterministic-v4
- StarGunner-ramNoFrameskip-v0
- StarGunner-ramNoFrameskip-v4
- Tennis-v0
- Tennis-v4
- TennisDeterministic-v0
- TennisDeterministic-v4
- TennisNoFrameskip-v0
- TennisNoFrameskip-v4
- Tennis-ram-v0
- Tennis-ram-v4
- Tennis-ramDeterministic-v0
- Tennis-ramDeterministic-v4
- Tennis-ramNoFrameskip-v0
- Tennis-ramNoFrameskip-v4
- TimePilot-v0

- TimePilot-v4
- TimePilotDeterministic-v0
- TimePilotDeterministic-v4
- TimePilotNoFrameskip-v0
- TimePilotNoFrameskip-v4
- TimePilot-ram-v0
- TimePilot-ram-v4
- TimePilot-ramDeterministic-v0
- TimePilot-ramDeterministic-v4
- TimePilot-ramNoFrameskip-v0
- TimePilot-ramNoFrameskip-v4
- Tutankham-v0
- Tutankham-v4
- TutankhamDeterministic-v0
- TutankhamDeterministic-v4
- TutankhamNoFrameskip-v0
- TutankhamNoFrameskip-v4
- Tutankham-ram-v0
- Tutankham-ram-v4
- Tutankham-ramDeterministic-v0
- Tutankham-ramDeterministic-v4
- Tutankham-ramNoFrameskip-v0
- Tutankham-ramNoFrameskip-v4
- UpNDown-v0
- UpNDown-v4
- UpNDownDeterministic-v0
- UpNDownDeterministic-v4
- UpNDownNoFrameskip-v0
- UpNDownNoFrameskip-v4
- UpNDown-ram-v0
- UpNDown-ram-v4
- UpNDown-ramDeterministic-v0
- UpNDown-ramDeterministic-v4
- UpNDown-ramNoFrameskip-v0
- UpNDown-ramNoFrameskip-v4
- Venture-v0
- Venture-v4
- VentureDeterministic-v0
- VentureDeterministic-v4
- VentureNoFrameskip-v0
- VentureNoFrameskip-v4
- Venture-ram-v0
- Venture-ram-v4
- Venture-ramDeterministic-v0
- Venture-ramDeterministic-v4
- Venture-ramNoFrameskip-v0
- Venture-ramNoFrameskip-v4
- VideoPinball-v0
- VideoPinball-v4
- VideoPinballDeterministic-v0
- VideoPinballDeterministic-v4
- VideoPinballNoFrameskip-v0
- VideoPinballNoFrameskip-v4
- VideoPinball-ram-v0
- VideoPinball-ram-v4
- VideoPinball-ramDeterministic-v0
- VideoPinball-ramDeterministic-v4
- VideoPinball-ramNoFrameskip-v0
- VideoPinball-ramNoFrameskip-v4
- WizardOfWor-v0
- WizardOfWor-v4

- WizardOfWorDeterministic-v0
- WizardOfWorDeterministic-v4
- WizardOfWorNoFrameskip-v0
- WizardOfWorNoFrameskip-v4
- WizardOfWor-ram-v0
- WizardOfWor-ram-v4
- WizardOfWor-ramDeterministic-v0
- WizardOfWor-ramDeterministic-v4
- WizardOfWor-ramNoFrameskip-v0
- WizardOfWor-ramNoFrameskip-v4
- YarsRevenge-v0
- YarsRevenge-v4
- YarsRevengeDeterministic-v0
- YarsRevengeDeterministic-v4
- YarsRevengeNoFrameskip-v0
- YarsRevengeNoFrameskip-v4
- YarsRevenge-ram-v0
- YarsRevenge-ram-v4
- YarsRevenge-ramDeterministic-v0
- YarsRevenge-ramDeterministic-v4
- YarsRevenge-ramNoFrameskip-v0
- YarsRevenge-ramNoFrameskip-v4
- Zaxxon-v0
- Zaxxon-v4
- ZaxxonDeterministic-v0
- ZaxxonDeterministic-v4
- ZaxxonNoFrameskip-v0
- ZaxxonNoFrameskip-v4
- Zaxxon-ram-v0
- Zaxxon-ram-v4
- Zaxxon-ramDeterministic-v0
- Zaxxon-ramDeterministic-v4
- Zaxxon-ramNoFrameskip-v0
- Zaxxon-ramNoFrameskip-v4
- CubeCrash-v0
- CubeCrashSparse-v0
- CubeCrashScreenBecomesBlack-v0
- MemorizeDigits-v0

Anexo VI. Códigos de los agentes de inteligencia artificial y programas accesorios creados para la realización de este trabajo fin de máster.

En este anexo se presentará en su totalidad el **código utilizado para todos los programas y agentes en su versión final**. Los códigos en formato *.py* estarán disponibles en el repositorio https://bitbucket.org/jocapal/tfm_repo/src/master/ para su utilización. El repositorio será de libre acceso.

VI.I Deimos_v3_agent.py

Se presenta aquí el código del agente principal, con los comentarios correspondientes para ayudar a su correcta comprensión.

```
#                                DEIMOS-V3

#                            Version Single-GPU

#                            V3.0 (23/04/2019)

# Reestructurado el programa con respecto a la v2.15: Ahora se compone de 3 archivos, para hacer el programa mas limpio.

# Eliminado todo lo referente a TensorBoard ya que no se va a utilizar.

# Añadida info de memoria cada 10 episodios para tener un control mas preciso sobre que esta pasando.

# Arreglado el hecho de que la puntuacion media y maxima en Pong sea siempre 0.

#                              24/04/2019

# Movidas funciones a archivo de funciones auxiliares por limpieza.

# Ahora el agente solo guardara los ficheros de la CNN si ha terminado el periodo de observacion.

#                              27/04/2019

# Solucionado problema con el contador de los episodios, que no los contaba correctamente.


# 1. IMPORTS Y FROMS:

import gym

import numpy as np

import time

import h5py

import random

from Deimos_v3_aux import image_processing, elapsed_time, memory_usage, directory_creation, path_check, load_episodes, load_steps
```

```
from Deimos_v3_metrics import print_progress, print_episode_final, graph_plot
from collections import deque
from keras.layers import Dense, Flatten, Conv2D
from keras.models import Sequential
from keras.optimizers import Adam

# 2. VARIABLES INICIALES:
game = "BreakoutDeterministic-v4"
start_time = time.time()
env = gym.make(game)
env.reset()
steps =  0
total_reward = 0
i = 0
i_update = i
episodes_sum = 0
exp_batch = []
weights = "/weights.best.hdf5"
mem_h5 = "/data.hdf5"
save_path = "Save/"
filepath = save_path+game+weights
save_filepath = save_path+game+mem_h5
if 'Pong' in game:
    med_sup = -21
    max_reward = -21
else:
    med_sup = 0
    max_reward = 0
l10 = deque((), maxlen = 10)
Q_values = deque()
episodes = 100010
observation_episodes = 50
update_network = 10000
directory_creation(save_path, game)
path_check(save_filepath)
```

```
# 3. AI: Aqui definiremos el agente como tal; especificaremos cada seccion del
agente, explicando que realiza cada parte del mismo.
class AI:
# 3.1 Variables iniciales; estas variables estan disponibles para su acceso
# desde cualquier punto del programa escribiendo AI.nombre_de_la_variable.
# Ejemplo: AI.four_batch_size
    four_batch_size = 4
    four_memory_state = deque((), maxlen = four_batch_size)
    four_memory_action = deque((), maxlen = four_batch_size)
    four_memory_next_state = deque((), maxlen = four_batch_size)
    four_memory_reward = deque((), maxlen = four_batch_size)
    four_memory_done = deque((), maxlen = four_batch_size)
    four_memory_lives = deque((), maxlen = four_batch_size)
    four_memory_batch = deque((), maxlen = 1)
    four_batch_state = deque((), maxlen = four_batch_size)
    mem_size = 200000
    mem_batch = deque((), maxlen = mem_size)
    replay_sample_size = 32
    sample_batch = []
    s_batch = []
    cnn_shape = (105,80,4)
    batch_shape = (1,105,80,4)
    learning_rate = 0.00025
    action_size = env.action_space.n
    initial_epsilon = 1
    final_epsilon = 0.1
    epsilon_frames = 1000000
    epsilon_decay = (initial_epsilon-final_epsilon)/epsilon_frames
    epsilon = initial_epsilon
    loss_list = []
    gamma = 0.99
    loss_plt = 0


# 3.2 Variables de inicio: estas variables son comunes dentro de la clase AI, es
decir, del agente.
```

```
    def __init__(self):
        self.model = self.CNN()
        self.target_model = self.target_CNN()


# Aqui intentamos cargar el valor de epsilon en caso de que exista; si no es
asi, utilizaremos el valor inicial de epsilon.
        try:
            savedata = h5py.File(save_filepath, 'r')
            saved_epsilon = savedata['epsilon']
            AI.epsilon = saved_epsilon[()]
            savedata.close()
            print("\033[1;32;38mExploration rate sucessfully loaded")
        except:
            AI.epsilon = AI.epsilon
            savedata.close()
            print("\033[1;31;38mError: exploration rate values cant be loaded")


# 3.3 Definimos la red neuronal convolucional:
    def CNN(self):
        model = Sequential()
        model.add(Conv2D(32, (8, 8), strides=(4,4), input_shape = AI.cnn_shape,
activation='relu'))
        model.add(Conv2D(64, (4, 4), strides=(2,2), activation='relu'))
        model.add(Conv2D(64, (3, 3), strides=(1,1), activation='relu'))
        model.add(Flatten())
        model.add(Dense(512, activation='relu'))
        model.add(Dense(AI.action_size, activation='linear'))
        model.compile(loss="mse", optimizer=Adam(lr=AI.learning_rate))
        model.summary()
        print("Model CNN construction complete")
        return model


# 3.4 Definimos la red neuronal objetivo; debe ser identica a la CNN.
    def target_CNN(self):
        target_model = Sequential()
```

```
        target_model.add(Conv2D(32, (8, 8), strides=(4,4), input_shape =
AI.cnn_shape, activation='relu'))
        target_model.add(Conv2D(64, (4, 4), strides=(2,2), activation='relu'))
        target_model.add(Conv2D(64, (3, 3), strides=(1,1), activation='relu'))
        target_model.add(Flatten())
        target_model.add(Dense(512, activation='relu'))
        target_model.add(Dense(AI.action_size, activation='linear'))
        target_model.compile(loss="mse", optimizer=Adam(lr=AI.learning_rate))
        target_model.summary()
        print("Target CNN construction complete")
        print("\033[1;32;38mCNN construction completed sucessfully")
        return target_model


# 3.5 Con esta funcion, cargamos los valores de los pesos en la red neuronal; de
esta forma, podremos retomar el entrenamiento en caso de que paremos el
programa:
    def load(self):
        model = self.model
        try:
            model.load_weights(filepath)
            print("\033[1;32;38mModel weights sucessfully loaded")
            print("\033[1;31;38mALL SYSTEMS NOMINAL")
        except:
            print("\033[1;31;38mError: Model values cant be loaded\
033[1;37;38m")


# 3.6 En esta funcion, guardamos los valores de epsilon, los pesos de las
neuronas y el numero de episodios para posteriormente cargarlos en caso de que
paremos el programa. Guardamos cada 10 episodios.
    def save(self, i, steps):
        model = self.model
        exploration_rate = AI.epsilon
        if episodes % 10 == 0:
            model.save(filepath)
            print("\033[1;37;38mModel saved")
            savedata = h5py.File(save_filepath, 'w')
```

```
            saved_epsilon = savedata.create_dataset('epsilon', data =
AI.epsilon)
            saved_i = savedata.create_dataset('i', data = i_update)
            saved_steps = savedata.create_dataset('steps', data = step_counter)
            savedata.close()
        return exploration_rate, saved_epsilon, saved_i, saved_steps


# 3.7 Con esta funcion copiamos los pesos de la red neuronal principal a la red
neuronal objetivo; este proceso se realiza cada determinado numero de pasos.
    def update_net(self):
        model = self.model
        model_weights = model.get_weights()
        target_model = self.target_model
        target_model.set_weights(model_weights)
        print("Target model sucessfully updated")


# 3.8 en la funcion batcher, creamos un paquete especifico con los estados para
la seleccion de la accion.
    def batcher(four_batch_state):
        AI.s_batch = []
        AI.s_batch = np.asarray(AI.four_batch_state)
        AI.s_batch.shape = AI.batch_shape


# 3.9 En esta funcion creamos los paquetes de cuatro experiencias; cuando estan
hechos los paquetes de 4, los enviamos a las diferentes funciones del programa
para su posterior procesamiento.
    def four_batcher(self, exp_batch):
        AI.four_batch_state.append(exp_batch[2])
        if len(AI.four_batch_state) == AI.four_batch_size:
            AI.batcher(AI.four_batch_state)


# 3.10 Esta funcion prepara los distintos paquetes extraidos de las experiencias
y los almacena en la memoria para su posterior utilizacion.
    def four_memory(self, exp_batch):
       AI.four_memory_state.append(exp_batch[0])
       AI.four_memory_action.append(exp_batch[1])
       AI.four_memory_next_state.append(exp_batch[2])
```

```
        AI.four_memory_reward.append(exp_batch[3])
        AI.four_memory_done.append(exp_batch[4])
        AI.four_memory_lives.append(exp_batch[5])


        if len(AI.four_memory_state) == AI.four_batch_size:
            AI.four_memory_batch.append((AI.four_memory_state,
AI.four_memory_action,
AI.four_memory_next_state, AI.four_memory_reward, AI.four_memory_done,
AI.four_memory_lives))
            AI.memory(AI.four_memory_batch)
```

```
# 3.11 En la funcion memory almacenamos todas las experiencias en paquetes de 4
listas para su posterior procesamiento. El limite de memoria esta definido por
la variable AI.mem_size; esto es importante ya que dependiendo de la ram de la
que dispongamos, el limite se debera ajustar. Cuando se llena, la funcion
automaticamente elimina el elemento mas a la izquierda, es decir, el elemento
mas antiguo (AI.mem_batch.popleft())
```

```
    def memory(self):
        if len(AI.mem_batch) < AI.mem_size:
            AI.mem_batch.append(AI.four_memory_batch)
        else:
            AI.mem_batch.popleft()
            AI.mem_batch.append(AI.four_memory_batch)
```

```
# 3.12 En la funcion replay_sampler, cogemos las experiencias acumuladas desde
la memoria, y las procesamos para prepararlas para introducirlas en el
entrenamiento; hacemos una seleccion aleatoria de entre los elementos de la
memoria, dependiendo del tamaño de la misma; si es menor que lo especificado en
AI.replay_sample_size, tomamos una muestra del tamaño de la propia memoria (es
decir, todas las muestras); si el tamaño de memoria es mayor, tomamos un numero
de muestras del tamaño establecido en AI.replay_sample_size.
```

```
    def replay_sampler(self, mem_batch):
        AI.sample_batch = []


        if len(AI.mem_batch) < AI.replay_sample_size:
            AI.sample_batch.append(random.sample(AI.mem_batch,
len(AI.mem_batch)))
        else:
            AI.sample_batch.append(random.sample(AI.mem_batch,
AI.replay_sample_size))
```

```
        AI.sample_batch = np.asarray(AI.sample_batch)
        AI.sample_batch.shape = (AI.replay_sample_size, 6, 4)


# 3.13 En action_selection, como su nombre indica, seleccionamos una accion;
para ello, usamos un paquete de 4 experiencias procesado anteriormente en la
funcion batcher; este paquete lo enviamos a la red neuronal principal. Despues
comparamos el valor de epsilon (nuestro ratio de exploracion) con un numero
aleatorio. Si el valor aleatorio es menor que el valor de epsilon, tomaremos una
accion aleatoria en vez de la calculada con la CNN.
    def action_selection(self, s_batch):
        random_value = np.random.rand()
        if random_value < AI.epsilon or i < observation_episodes:
            best_action = env.action_space.sample()
            best_action_value = 0
        else:
            action_values = self.model.predict(s_batch)
            best_action = np.argmax(action_values[0])
            best_action_value = action_values[0, best_action]
# Decrementamos epsilon hasta su valor minimo:
        if i > observation_episodes:
            if AI.epsilon > AI.final_epsilon:
                AI.epsilon -= AI.epsilon_decay
        return best_action, best_action_value


# 3.14 En la funcion training, usamos las experiencias acumuladas anteriormente
para entrenar la red; aqui se implementa el algoritmo Q-Learning propiamente
dicho. Tambien se calcula el loss, que es la diferencia entre el estado actual y
el estado predicho en un futuro; esta medida nos da una idea de cuan preciso
esta siendo nuestro proceso.
    def training(self, sample_batch):
        AI.training_s_batch = []
        AI.training_a_batch = []
        AI.training_ns_batch = []
        AI.training_r_batch = []
        AI.training_d_batch = []
        Q_values = []
        loss_batch = []
        batch_size = len(AI.sample_batch)
```

```
        Q_values = np.zeros((batch_size, AI.action_size))
        new_Q = np.zeros((batch_size, AI.action_size))

        for m in range(batch_size):
            AI.training_s_batch =
np.asarray(list(AI.sample_batch[m,0])).reshape(AI.batch_shape)
            AI.training_a_batch = np.asarray(list(AI.sample_batch[m,1]))
            AI.training_a_batch = AI.training_a_batch[:,0].astype(int)
            AI.training_ns_batch =
np.asarray(list(AI.sample_batch[m,2])).reshape(AI.batch_shape)
            AI.training_r_batch = np.asarray(list(AI.sample_batch[m,3]))
            AI.training_d_batch = np.asarray(list(AI.sample_batch[m,4]))

            loss_batch.append(AI.training_s_batch)

            Q_values[m] =
self.model.predict(AI.training_s_batch.reshape(AI.batch_shape))
            Q_values[m, AI.training_a_batch] = AI.training_r_batch
# Si ha perdido una vida, se le penaliza; de esta forma se "fuerza" al agente a
que aprenda a no morir. Para penalizarlo no se le dan mas recompensas si ha
perdido una vida. Para Pong, penalizamos si su puntuacion es negativa:
            if 'Pong' in game:
                    new_Q[m] =
self.target_model.predict(AI.training_ns_batch.reshape(AI.batch_shape))
                    Q_values[m, AI.training_a_batch] += (AI.gamma *
np.max(new_Q))
            else:
                if np.all(AI.training_d_batch) == False:
                    new_Q[m] =
self.target_model.predict(AI.training_ns_batch.reshape(AI.batch_shape))
                    Q_values[m, AI.training_a_batch] += (AI.gamma *
np.max(new_Q))

        loss =
self.model.fit(np.asarray(loss_batch).reshape(batch_size,105,80,4), Q_values,
batch_size=batch_size, verbose=0, shuffle=False)
        loss_history = loss.history['loss'][0]
        AI.loss_list.append(loss_history)
```

```
        if 'Pong' in game:
            if done == True:
                AI.loss_plt = np.mean(AI.loss_list)
                AI.loss_list = []
        else:
            if lives == 0:
                AI.loss_plt = np.mean(AI.loss_list)
                AI.loss_list = []


# Antes de entrar en el bucle, cargamos el numero de episodios, epsilon, los pasos, asi como los pesos de las neuronas:
agent = AI()
episodes = load_episodes(save_filepath, episodes, i_update)[0]
i_update = load_episodes(save_filepath, episodes, i_update)[1]
step_counter = load_steps(save_filepath)
agent.load()
# 4. Bucle de entrenamiento: aqui entrenamos el programa por un numero de episodios determinados; dentro de cada episodio, ponemos un limite de pasos (acciones) que puede realizar antes de acabar el episodio. El episodio tambien puede acabar porque el agente fracasa y no consigue el objetivo.
Puesto que para que el resto del proceso funcione, si acabamos de arrancar el programa, bien sea por primera vez o tras apagarlo, todos los buffers estaran vacios, lo cual provocaria un error (no hay ninguna imagen que procesar, ni ningun batch que introducir a las CNN). Para solucionarlo, tomamos el primer estado (el estado 0) y lo introducimos un numero de veces suficiente como para que el programa funcione adecuadamente. Desde el bucle tambien se monitoriza el avance del mismo, tanto viendo como progresa visualmente env.render(), como creando una grafica (reward_plot); tambien mostramos el valor de epsilon para darnos una idea de cuanto ha avanzado el programa.
Otras metricas que se toman, a nivel de debug, es el tamaño de AI.mem_batch,ya que si se nos desborda puede bloquear el ordenador, por eso, al menos en las primeras sesiones es recomendable controlar el tamaño de memoria.
Para ver la ram libre, asi como el uso de la CPU (aunque al usar procesamiento en paralelo con la GPU es secundario), usamos las utilidades proporcionadas por la libreria psutil (psutil.cpu_percent y psutil.virtual_memory). Se muestra por pantalla el episodio en el que esta y el numero de pasos que lleva del mismo cada 100 pasos, como control.
for i in range (episodes):
    state = env.reset()
    total_reward = 0
    if i > observation_episodes:
```

```
        if i % 10 == 0:
            agent.save(i_update, AI.epsilon)
            memory_usage()
            print("\033[1;37;38mMemory length: ", len(agent.mem_batch))
    if i == 0 and steps == 0:
        random_action = env.action_space.sample()
        state1, reward1, done1, lives1 = env.step(random_action)
        done1 = False
        action1 = [random_action, 0.0]
        lives1 = lives1['ale.lives']
        next_state1 = state1
        exp_batch = (image_processing(state1), action1,
image_processing(next_state1), reward1, done1, lives1)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        reward1 = np.sign(reward1)
        total_reward += reward1
        del state1, action1, next_state1, reward1, done1, exp_batch,
random_action
    no_op_counter = 0
    no_op_max = np.random.randint(4,30, dtype=int)


    for steps in range (10000):
# NO-OP al inicio, como en el paper de DeepMind:
        if no_op_counter < no_op_max:
            action = [0, 0.0]
            no_op_counter += 1
        else:
            action = agent.action_selection(AI.s_batch)
        Q_values.append(action[1])
        next_state, reward, done, lives = env.step(action[0])
        lives = lives['ale.lives']
        reward = np.sign(reward)
```

```
# Si pierde una vida, penalizamos evitando que consiga recompensa maxima; asi lo
"enseñamos" a no perder vidas.
        if (lives < lives1):
            done == True
        total_reward += reward
        exp_batch = (image_processing(state), action,
image_processing(next_state), reward, done, lives)
        agent.four_batcher(exp_batch)
        agent.four_memory(exp_batch)
        state = next_state
        exp_batch = []
        step_counter += 1
        env.render()
        if i >= observation_episodes:
            agent.replay_sampler(AI.mem_batch)
            agent.training(AI.sample_batch)
            if step_counter % update_network == 0:
                agent.update_net()
            if 'Pong' in game:
                if done == True:
                  i_update += 1
                  episodes_sum += total_reward
                  episode_time = time.time()
                  if len(l10) < 10:
                      l10.append(total_reward)
                  if total_reward > max_reward:
                    max_reward = total_reward
                  else:
                    l10.popleft()
                    l10.append(total_reward)
                  l10_mean = np.mean(l10)
                  Q_values_mean = np.mean(Q_values)
                  Q_values = []
                  med_episodes = round((episodes_sum/((i+1)-
observation_episodes)),2)
                  h,m,s = elapsed_time(start_time, episode_time)
```

```
                    if med_episodes > med_sup:
                        med_sup = med_episodes
                    epsilon = AI.epsilon
                    loss_plt = AI.loss_plt
                    print_episode_final(i, episodes, steps, total_reward,
AI.epsilon, AI.loss_plt, h,m,s, max_reward, med_sup, med_episodes, l10_mean,
Q_values_mean)
                    graph_plot(i, AI.loss_plt, total_reward, med_episodes,
l10_mean, Q_values_mean, observation_episodes)
                    break
# Si es cualquier otro juego:
            else:
                if lives == 0:
                    i_update += 1
                    episodes_sum += total_reward
                    episode_time = time.time()
                    if len(l10) < 10:
                        l10.append(total_reward)
                    if total_reward > max_reward:
                        max_reward = total_reward
                    else:
                        l10.popleft()
                        l10.append(total_reward)
                    l10_mean = np.mean(l10)
                    Q_values_mean = np.mean(Q_values)
                    Q_values = []
                    med_episodes = round((episodes_sum/((i+1)-
observation_episodes)),2)
                    h,m,s = elapsed_time(start_time, episode_time)
                    if med_episodes > med_sup:
                        med_sup = med_episodes
                    print_episode_final(i, episodes, steps, total_reward,
AI.epsilon, AI.loss_plt, h,m,s, max_reward, med_sup, med_episodes, l10_mean,
Q_values_mean)
                    graph_plot(i, AI.loss_plt, total_reward, med_episodes,
l10_mean, Q_values_mean, observation_episodes)
        if steps % 100 == 0:
```

```
            print_progress(i, episodes, steps, step_counter, total_reward)
# Adecuamos aqui de nuevo para Pong:
        if 'Pong' in game:
            if done == True:
                break
        else:
            if lives == 0:
                break
```

VI.II Deimos_v3_agent_multi.py

Se presenta a continuación la versión del programa principal, en su versión con soporte Multi-GPU.

```
#                                DEIMOS-V3

#                            Version Multi-GPU

#                            V3.0 (23/04/2019)

# Reestructurado el programa con respecto a la v2.15: Ahora se compone de 3 archivos, para hacer el programa mas limpio.

# Eliminado todo lo referente a TensorBoard ya que no se va a utilizar.

# Añadida info de memoria cada 10 episodios para tener un control mas preciso sobre que esta pasando.

# Arreglado el hecho de que la puntuacion media y maxima en Pong sea siempre 0.


# 1. IMPORTS Y FROMS:

import gym

import numpy as np

import time

import h5py

import random

from Deimos_v3_aux import image_processing, elapsed_time, memory_usage, directory_creation, path_check, load_episodes, load_steps

from Deimos_v3_metrics import print_progress, print_episode_final, graph_plot

from collections import deque

from keras.layers import Dense, Flatten, Conv2D

from keras.models import Sequential

from keras.utils import multi_gpu_model

from keras.optimizers import Adam

from tensorflow.python.client import device_lib


# 2. VARIABLES INICIALES:

game = "PongDeterministic-v4"

start_time = time.time()

env = gym.make(game)

env.reset()

steps =  0

total_reward = 0

i = 0
```

```
i_update = i
episodes_sum = 0
exp_batch = []
weights = "/weights.best.hdf5"
mem_h5 = "/data.hdf5"
save_path = "Save/"
filepath = save_path+game+weights
save_filepath = save_path+game+mem_h5
if 'Pong' in game:
    med_sup = -21
    max_reward = -21
else:
    med_sup = 0
    max_reward = 0
l10 = deque((), maxlen = 10)
Q_values = deque()
episodes_start = 100010
episodes = episodes_start
observation_episodes = 50
update_network = 10000
directory_creation(save_path, game)
path_check(save_filepath)
gpu_number = len(device_lib.list_local_devices())-1
print("Found {} GPUs in the system".format(gpu_number))


# 3. AI: Aqui definiremos el agente como tal; especificaremos cada seccion del
agente, explicando que realiza cada parte del mismo.
class AI:
# 3.1 Variables iniciales; estas variables estan disponibles para accederse
desde cualquier punto del programa escribiendo AI.nombre_de_la_variable.
# Ejemplo: AI.four_counter
    four_batch_size = 4
    four_memory_state = deque((), maxlen = four_batch_size)
    four_memory_action = deque((), maxlen = four_batch_size)
    four_memory_next_state = deque((), maxlen = four_batch_size)
    four_memory_reward = deque((), maxlen = four_batch_size)
```

```
    four_memory_done = deque((), maxlen = four_batch_size)
    four_memory_lives = deque((), maxlen = four_batch_size)
    four_memory_batch = deque((), maxlen = 1)
    four_batch_state = deque((), maxlen = four_batch_size)
    mem_size = 100000
    mem_batch = deque((), maxlen = mem_size)
    replay_sample_size = 32
    sample_batch = []
    s_batch = []
    cnn_shape = (105,80,4)
    batch_shape = (1,105,80,4)
    learning_rate = 0.00001
    action_size = env.action_space.n
    initial_epsilon = 1
    final_epsilon = 0.1
    epsilon_frames = 300000
    epsilon_decay = (initial_epsilon-final_epsilon)/epsilon_frames
    epsilon = initial_epsilon
    loss_list = []
    gamma = 0.99
    loss_plt = 0


# 3.2 Variables de inicio: estas variables son comunes dentro de la clase AI, es
decir, del agente.
    def __init__(self):
        self.parallel_model = self.CNN()
        self.parallel_target_model = self.target_CNN()


# Aqui intentamos cargar el valor de epsilon en caso de que exista; si no es
asi, cargamos el valor inicial de epsilon.
        try:
            savedata = h5py.File(save_filepath, 'r')
            saved_epsilon = savedata['epsilon']
            AI.epsilon = saved_epsilon[()]
            savedata.close()
            print("\033[1;32;38mExploration rate sucessfully loaded")
```

```
        except:
            AI.epsilon = AI.epsilon
            savedata.close()
            print("\033[1;31;38mError: exploration rate values cant be loaded")


# 3.3 Definimos la red neuronal convolucional; se ha añadido el procesamiento
multi-GPU:
    def CNN(self):
        model = Sequential()
        model.add(Conv2D(32, (8, 8), strides=(4,4), input_shape = AI.cnn_shape,
activation='relu'))
        model.add(Conv2D(64, (4, 4), strides=(2,2), activation='relu'))
        model.add(Conv2D(64, (3, 3), strides=(1,1), activation='relu'))
        model.add(Flatten())
        model.add(Dense(512, activation='relu'))
        model.add(Dense(AI.action_size, activation='linear'))
        parallel_model = multi_gpu_model(model, gpus=gpu_number)
        parallel_model.compile(loss="mse", optimizer=Adam(lr=AI.learning_rate))
        parallel_model.summary()
        print("Model CNN construction complete")
        return parallel_model


# 3.4 Definimos la red neuronal objetivo; debe ser identica a la normal.
    def target_CNN(self):
        target_model = Sequential()
        target_model.add(Conv2D(32, (8, 8), strides=(4,4), input_shape =
AI.cnn_shape, activation='relu'))
        target_model.add(Conv2D(64, (4, 4), strides=(2,2), activation='relu'))
        target_model.add(Conv2D(64, (3, 3), strides=(1,1), activation='relu'))
        target_model.add(Flatten())
        target_model.add(Dense(512, activation='relu'))
        target_model.add(Dense(AI.action_size, activation='linear'))
        parallel_target_model = multi_gpu_model(target_model, gpus=gpu_number)
        parallel_target_model.compile(loss="mse",
optimizer=Adam(lr=AI.learning_rate))
        parallel_target_model.summary()
```

```
        print("Target CNN construction complete")
        print("\033[1;32;38mCNN construction completed sucessfully")
        return parallel_target_model


# 3.5 Con esta funcion, cargamos los valores de los pesos en la red neuronal; de esta forma, podremos retomar el entrenamiento en caso de que paremos el programa:
    def load(self):
        model = self.parallel_model
        try:
            model.load_weights(filepath)
            print("\033[1;32;38mModel weights sucessfully loaded")
            print("\033[1;31;38mALL SYSTEMS NOMINAL")
        except:
            print("\033[1;31;38mError: Model values cant be loaded\
033[1;37;38m")


# 3.6 En esta funcion, guardamos los valores de epsilon, los pesos de las neuronas y el numero de episodios para posteriormente cargarlos en caso de que paremos el programa. Guardamos cada 10 episodios.
    def save(self, i, steps):
        model = self.parallel_model
        exploration_rate = AI.epsilon
        if episodes % 10 == 0:
            model.save(filepath)
            print("\033[1;37;38mModel saved")
            savedata = h5py.File(save_filepath, 'w')
            saved_epsilon = savedata.create_dataset('epsilon', data =
AI.epsilon)
            saved_i = savedata.create_dataset('i', data = i_update)
            saved_steps = savedata.create_dataset('steps', data = step_counter)
            savedata.close()
        return exploration_rate, saved_epsilon, saved_i, saved_steps


# 3.7 Con esta funcion copiamos los pesos de la red neuronal principal a la red neuronal objetivo; este proceso se realiza cada determinado numero de pasos.
    def update_net(self):
```

```
        model = self.parallel_model
        model_weights = model.get_weights()
        target_model = self.parallel_target_model
        target_model.set_weights(model_weights)
        print("Target model sucessfully updated")


# 3.8 en la funcion batcher, creamos un paquete especifico con los estados para
la seleccion de la accion.
    def batcher(four_batch_state):
        AI.s_batch = []
        AI.s_batch = np.asarray(AI.four_batch_state)
        AI.s_batch.shape = AI.batch_shape


# 3.9 En esta funcion creamos los paquetes de cuatro experiencias; cuando estan
hechos los paquetes de 4, los enviamos a las diferentes funciones del programa
para su posterior procesamiento.
    def four_batcher(self, exp_batch):
        AI.four_batch_state.append(exp_batch[2])
        if len(AI.four_batch_state) == AI.four_batch_size:
            AI.batcher(AI.four_batch_state)


# 3.10 Esta funcion prepara los distintos paquetes extraidos de las experiencias
y los almacena en la memoria para su posterior utilizacion.
    def four_memory(self, exp_batch):
       AI.four_memory_state.append(exp_batch[0])
       AI.four_memory_action.append(exp_batch[1])
       AI.four_memory_next_state.append(exp_batch[2])
       AI.four_memory_reward.append(exp_batch[3])
       AI.four_memory_done.append(exp_batch[4])
       AI.four_memory_lives.append(exp_batch[5])


       if len(AI.four_memory_state) == AI.four_batch_size:
            AI.four_memory_batch.append((AI.four_memory_state,
AI.four_memory_action,
AI.four_memory_next_state, AI.four_memory_reward,
AI.four_memory_done, AI.four_memory_lives))
            AI.memory(AI.four_memory_batch)
```

```
# 3.11 En la funcion memory almacenamos todas las experiencias en paquetes de 4 listas para su posterior procesamiento. El limite de memoria esta definido por la variable AI.mem_size; esto es importante ya que dependiendo de la ram de la que dispongamos, el limite se debera ajustar. Cuando se llena, la funcion automaticamente elimina el elemento mas a la izquierda, es decir, el elemento mas antiguo (AI.mem_batch.popleft())

    def memory(self):

        if len(AI.mem_batch) < AI.mem_size:

            AI.mem_batch.append(AI.four_memory_batch)

        else:

            AI.mem_batch.popleft()

            AI.mem_batch.append(AI.four_memory_batch)


# 3.12 En la funcion replay_sampler, cogemos las experiencias acumuladas desde la memoria, y las procesamos para prepararlas para introducirlas en el entrenamiento; hacemos una seleccion aleatoria de entre los elementos de la memoria, dependiendo del tamaño de la misma; si es menor que lo especificado en AI.replay_sample_size, tomamos una muestra del tamaño de la propia memoria (es decir, todas las muestras); si el tamaño de memoria es mayor, tomamos un numero de muestras del tamaño establecido en AI.replay_sample_size.

    def replay_sampler(self, mem_batch):

        AI.sample_batch = []


        if len(AI.mem_batch) < AI.replay_sample_size:

            AI.sample_batch.append(random.sample(AI.mem_batch,
len(AI.mem_batch)))

        else:

            AI.sample_batch.append(random.sample(AI.mem_batch,
AI.replay_sample_size))


        AI.sample_batch = np.asarray(AI.sample_batch)

        AI.sample_batch.shape = (AI.replay_sample_size, 6, 4)


# 3.13 En action_selection, como su nombre indica, seleccionamos una accion; para ello, usamos un paquete de 4 experiencias procesado anteriormente en la funcion batcher; este paquete lo enviamos a la red neuronal principal.

Despues comparamos el valor de epsilon (nuestro ratio de exploracion) con un numero aleatorio. Si el valor aleatorio es menor que el valor de epsilon, tomaremos una accion aleatoria en vez de la calculada con la CNN.

    def action_selection(self, s_batch):

        random_value = np.random.rand()
```

```
        if random_value < AI.epsilon or i < observation_episodes:
            best_action = env.action_space.sample()
            best_action_value = 0
        else:
            action_values = self.parallel_model.predict(s_batch)
            best_action = np.argmax(action_values[0])
            best_action_value = action_values[0, best_action]
# Decrementamos epsilon hasta su valor minimo:
        if i > observation_episodes:
            if AI.epsilon > AI.final_epsilon:
                AI.epsilon -= AI.epsilon_decay
        return best_action, best_action_value


# 3.14 En la funcion training, usamos las experiencias acumuladas anteriormente
para entrenar la red; aqui se implementa el algoritmo Q-Learning propiamente
dicho. Tambien se calcula el loss, que es la diferencia entre el estado actual y
el estado predicho en un futuro; esta medida nos da una idea de cuan preciso
esta siendo nuestro proceso.
    def training(self, sample_batch):
        AI.training_s_batch = []
        AI.training_a_batch = []
        AI.training_ns_batch = []
        AI.training_r_batch = []
        AI.training_d_batch = []
        Q_values = []
        loss_batch = []
        batch_size = len(AI.sample_batch)
        Q_values = np.zeros((batch_size, AI.action_size))
        new_Q = np.zeros((batch_size, AI.action_size))


        for m in range(batch_size):
            AI.training_s_batch =
np.asarray(list(AI.sample_batch[m,0])).reshape(AI.batch_shape)
            AI.training_a_batch = np.asarray(list(AI.sample_batch[m,1]))
            AI.training_a_batch = AI.training_a_batch[:,0].astype(int)
            AI.training_ns_batch =
np.asarray(list(AI.sample_batch[m,2])).reshape(AI.batch_shape)
```

```
            AI.training_r_batch = np.asarray(list(AI.sample_batch[m,3]))
            AI.training_d_batch = np.asarray(list(AI.sample_batch[m,1]))


            loss_batch.append(AI.training_s_batch)


            Q_values[m] =
self.parallel_model.predict(AI.training_s_batch.reshape(AI.batch_shape))
            Q_values[m, AI.training_a_batch] = AI.training_r_batch
# Si ha perdido una vida, se le penaliza; de esta forma se "fuerza" al agente a
que aprenda a no morir. Para penalizarlo no se le dan mas recompensas si ha
perdido una vida.
            if 'Pong' in game:
                    new_Q[m] =
self.parallel_target_model.predict(AI.training_ns_batch.reshape(AI.batch_shape))
                    Q_values[m, AI.training_a_batch] += (AI.gamma *
np.max(new_Q))
            else:
                if np.all(AI.training_d_batch) == False:
                    new_Q[m] =
self.parallel_target_model.predict(AI.training_ns_batch.reshape(AI.batch_shape))
                    Q_values[m, AI.training_a_batch] += (AI.gamma *
np.max(new_Q))


        loss =
self.parallel_model.fit(np.asarray(loss_batch).reshape(batch_size,105,80,4),
Q_values, batch_size=batch_size, verbose=0, shuffle=False)
        loss_history = loss.history['loss'][0]
        AI.loss_list.append(loss_history)


        if 'Pong' in game:
            if done == True:
                AI.loss_plt = np.mean(AI.loss_list)
                AI.loss_list = []
        else:
            if lives == 0:
                AI.loss_plt = np.mean(AI.loss_list)
                AI.loss_list = []
agent = AI()
```

```
episodes = load_episodes(save_filepath, episodes, i_update, episodes_start)

step_counter = load_steps(save_filepath)

agent.load()


# 4. Bucle de entrenamiento: aqui entrenamos el programa por un numero de episodios determinados; dentro de cada episodio, ponemos un limite de pasos (acciones) que puede realizar antes de acabar el episodio. El episodio tambien puede acabar porque el agente fracasa y no consigue el objetivo.

Puesto que para que el resto del proceso funcione, si acabamos de arrancar el programa, bien sea por primera vez o tras apagarlo, todos los buffers estaran vacios, lo cual provocaria un error (no hay ninguna imagen que procesar, ni ningun batch que introducir a las CNN). Para solucinarlo, tomamos el primer estado (el estado 0) y lo introducimos un numero de veces suficiente como para que el programa funcione adecuadamente. Desde el bucle tambien se monitoriza el avance del mismo, tanto viendo como progresa visualmente env.render(), como creando una grafica (reward_plot); tambien mostramos el valor de epsilon para darnos una idea de cuanto ha avanzado el programa.

Otras metricas que se toman, a nivel de debug, es el tamaño de AI.mem_batch, ya que si se nos desborda puede bloquear el ordenador, por eso, al menos en las primeras sesiones es recomendable controlar el tamaño de memoria.

Para ver la ram libre, asi como el uso de la CPU (aunque al usar procesamiento en paralelo con la GPU es secundario), usamos las utilidades proporcionadas por la libreria psutil (psutil.cpu_percent y psutil.virtual_memory). Se muestra por pantalla el episodio en el que esta y el numero de pasos que lleva del mismo cada 100 pasos, como control.

for i in range (episodes):
    state = env.reset()
    total_reward = 0
    if i % 10 == 0:
        agent.save(i_update, AI.epsilon)
        memory_usage()
        print("\033[1;37;38mMemory length: ", len(agent.mem_batch))
    if i == 0 and steps == 0:
        random_action = env.action_space.sample()
        state1, reward1, done1, lives1 = env.step(random_action)
        done1 = False
        action1 = [random_action, 0.0]
        lives1 = lives1['ale.lives']
        next_state1 = state1
        exp_batch = (image_processing(state1), action1,
image_processing(next_state1), reward1, done1, lives1)
```

```
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        reward1 = np.sign(reward1)
        total_reward += reward1
        del state1, action1, next_state1, reward1, done1, exp_batch,
random_action
    no_op_counter = 0
    no_op_max = np.random.randint(4,30, dtype=int)


    for steps in range (10000):
# NO-OP al inicio, como en el paper de DeepMind:
        if no_op_counter < no_op_max:
            action = [0, 0.0]
            no_op_counter += 1
        else:
            action = agent.action_selection(AI.s_batch)
        Q_values.append(action[1])
        next_state, reward, done, lives = env.step(action[0])
        lives = lives['ale.lives']
        reward = np.sign(reward)
# Si pierde una vida, penalizamos evitando que consiga recompensa maxima; asi lo
"enseñamos" a no perder vidas.
        if (lives < lives1):
            done == True
        total_reward += reward
        exp_batch = (image_processing(state), action,
image_processing(next_state), reward, done, lives)
        agent.four_batcher(exp_batch)
        agent.four_memory(exp_batch)
        state = next_state
        exp_batch = []
        step_counter += 1
        env.render()
        if i >= observation_episodes:
```

```
            agent.replay_sampler(AI.mem_batch)
            agent.training(AI.sample_batch)
            if step_counter % update_network == 0:
                agent.update_net()
            if 'Pong' in game:
                if done == True:
                  i_update += 1
                  episodes_sum += total_reward
                  episode_time = time.time()
                  if len(l10) < 10:
                      l10.append(total_reward)
                  if total_reward > max_reward:
                    max_reward = total_reward
                  else:
                    l10.popleft()
                    l10.append(total_reward)
                  l10_mean = np.mean(l10)
                  Q_values_mean = np.mean(Q_values)
                  Q_values = []
                  med_episodes = round((episodes_sum/((i+1)-
observation_episodes)),2)
                  h,m,s = elapsed_time(start_time, episode_time)
                  if med_episodes > med_sup:
                      med_sup = med_episodes
                  epsilon = AI.epsilon
                  loss_plt = AI.loss_plt
                  print_episode_final(i, episodes, steps, total_reward, epsilon,
loss_plt, h,m,s, max_reward, med_sup, med_episodes, l10_mean, Q_values_mean)
                  graph_plot(i, loss_plt, total_reward, med_episodes, l10_mean,
Q_values_mean, observation_episodes)
                  break
# Si es cualquier otro juego:
            else:
                if lives == 0:
                    i_update += 1
                    episodes_sum += total_reward
```

```
                    episode_time = time.time()
                    if len(l10) < 10:
                        l10.append(total_reward)
                    if total_reward > max_reward:
                        max_reward = total_reward
                    else:
                        l10.popleft()
                        l10.append(total_reward)
                    l10_mean = np.mean(l10)
                    Q_values_mean = np.mean(Q_values)
                    Q_values = []
                    med_episodes = round((episodes_sum/((i+1)-
observation_episodes)),2)
                    h,m,s = elapsed_time(start_time, episode_time)
                    if med_episodes > med_sup:
                        med_sup = med_episodes
                    print_episode_final(i, episodes, steps, total_reward,
epsilon, loss_plt, h,m,s, max_reward, med_sup, med_episodes, l10_mean,
Q_values_mean)
                    graph_plot(i, loss_plt, total_reward, med_episodes,
l10_mean, Q_values_mean, observation_episodes)
        if steps % 100 == 0:
            print_progress(i, episodes, steps, step_counter, total_reward)
# Adecuamos aqui de nuevo para Pong:
        if 'Pong' in game:
            if done == True:
                break
        else:
            if lives == 0:
                break
```

VI.III Deimos_v3_agent_test.py

En este apartado se presenta el programa encargado de realizar los tests una vez entrenado el agente.

```
import gym
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
import h5py
from collections import deque
from keras.layers import Dense, Flatten, Conv2D
from keras.models import Sequential
from keras.optimizers import Adam

game = "BreakoutDeterministic-v4"
env = gym.make(game)
env.reset()
steps =  0
total_reward = 0
i = 0
i_update = i
episodes_sum = 0
exp_batch = []
weights = "/weights.best.hdf5"
mem_h5 = "/data.hdf5"
save_path = "Save/"
experiment = "/Experimento_2/"
filepath = save_path+game+experiment+weights
save_filepath = save_path+game+experiment+mem_h5
med_sup = 0
max_reward = 0
l10 = deque((), maxlen = 10)
Q_values = deque()

if 'Pong' in game:
    med_sup = -21
```

```
    max_reward = -21
else:
    med_sup = 0
    max_reward = 0

if os.path.isfile(save_filepath) is False:
    savedata = h5py.File(save_filepath, 'w')
    savedata.close()
    print("Restoration point created.")
else:
    print("Restoration point found.")

class AI:

    four_batch_size = 4
    four_batch_state = deque((), maxlen = four_batch_size)
    s_batch = []
    cnn_shape = (105,80,4)
    batch_shape = (1,105,80,4)
    learning_rate = 0.00025
    action_size = env.action_space.n
    epsilon = 0.05
    gamma = 0.95

    def __init__(self):
        self.parallel_model = self.CNN()

    def CNN(self):
        model = Sequential()
        model.add(Conv2D(32, (8, 8), strides=(4,4), input_shape = AI.cnn_shape,
activation='relu'))
        model.add(Conv2D(64, (4, 4), strides=(2,2), activation='relu'))
        model.add(Conv2D(64, (3, 3), strides=(1,1), activation='relu'))
        model.add(Flatten())
        model.add(Dense(512, activation='relu'))
```

```
        model.add(Dense(AI.action_size, activation='linear'))
        parallel_model =  model
        parallel_model.compile(loss="mse", optimizer=Adam(lr=AI.learning_rate))
        parallel_model.summary()
        print("Model CNN construction complete")
        return parallel_model

    def load(self):
        model = self.parallel_model
        try:
            model.load_weights(filepath)
            print("\033[1;32;38mModel weights sucessfully loaded")
            print("\033[1;31;38mALL SYSTEMS NOMINAL")
        except:
            print("\033[1;31;38mError: Model values cant be loaded\
033[1;37;38m")

    def batcher(four_batch_state):
        AI.s_batch = []
        AI.s_batch = np.asarray(AI.four_batch_state)
        AI.s_batch.shape = AI.batch_shape

    def four_batcher(self, exp_batch):
        AI.four_batch_state.append(exp_batch[2])
        if len(AI.four_batch_state) == AI.four_batch_size:
            AI.batcher(AI.four_batch_state)

    def action_selection(self, s_batch):
        random_value = np.random.rand()
        if random_value < AI.epsilon:
            best_action = env.action_space.sample()
            best_action_value = 0
        else:
            action_values = self.parallel_model.predict(s_batch)
            best_action = np.argmax(action_values[0])
```

```
            best_action_value = action_values[0, best_action]
        return best_action, best_action_value

def image_processing(image):
    processed_image = cv2.resize(image, (105,80))
    processed_image = cv2.cvtColor(processed_image, cv2.COLOR_RGB2GRAY)
    processed_image = processed_image/255.0
    return processed_image

agent = AI()
episodes = 200
agent.load()

for i in range (episodes):
    state = env.reset()
    total_reward = 0
    step_counter = 0
    if i == 0 and steps == 0:
        random_action = env.action_space.sample()
        state1, reward1, done1, lives1 = env.step(random_action)
        done1 = False
        action1 = [random_action, 0.0]
        lives1 = lives1['ale.lives']
        next_state1 = state1
        exp_batch = (image_processing(state1), action1,
image_processing(next_state1), reward1, done1, lives1)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        reward1 = np.sign(reward1)
        total_reward += reward1
        del state1, action1, next_state1, reward1, done1, exp_batch,
random_action
    for steps in range (10000):
        action = agent.action_selection(AI.s_batch)
```

```
        Q_values.append(action[1])
        next_state, reward, done, lives = env.step(action[0])
        lives = lives['ale.lives']
        total_reward += reward
        exp_batch = (image_processing(state), action,
image_processing(next_state), reward, done, lives)
        agent.four_batcher(exp_batch)
        state = next_state
        exp_batch = []
        step_counter += 1
        env.render()
        if done == True:
            i_update += 1
            episodes_sum += total_reward
            if len(l10) < 10:
                l10.append(total_reward)
            if total_reward > max_reward:
                max_reward = total_reward
            else:
                l10.popleft()
                l10.append(total_reward)
            l10_mean = np.mean(l10)
            Q_values_mean = np.mean(Q_values)
            Q_values = []
            med_episodes = round(episodes_sum/((i+1)),2)
            if med_episodes > med_sup:
                med_sup = med_episodes
            print("\033[1;32;38mEpisode: {}/{}, Total steps: {}, Total Score:
{}"
                      .format(i, episodes, steps, total_reward))
            print("\033[1;32;38mMax score: {:.2f}, Maximum mean: {:.2f}, Actual
mean: {:.2f}, L10 mean: {:.2f}" .format(max_reward, med_sup, med_episodes,
l10_mean))


            plt.subplot(3,1,1)
            plt.title('Reward vs episodes')
```

```
            plt.plot(i, total_reward, 'ro', label = 'Reward')
            plt.plot(i, med_episodes, 'g.', label = 'Mean')
            plt.plot(i, l10_mean, 'y.', label = 'L10 Mean')
            plt.xlabel('Episodes', fontsize=12)
            plt.ylabel('Reward', fontsize=12)
            if i == 0:
                plt.legend(loc='upper right', bbox_to_anchor=(1.14,0.10),
prop={'size':5.5})
            plt.pause(0.05)
            plt.show()
            break


        if steps % 100 == 0:
                    print("\033[1;37;38mEpisode: {}/{}, Steps: {}, Total steps:
{}, Score: {}"
                        .format(i, episodes, steps, step_counter, total_reward))

env.close()
```

VI.IV Deimos_v3_aux.py

En este apartado se presenta el archivo que contiene las funciones auxiliares, es decir aquellas que directamente no pertenecen al agente pero son necesarias para su buen funcionamiento.

```
#                                DEIMOS-V3

#                            Funciones auxiliares
#

# Este archivo contiene todas las funciones auxiliares que se han podido mover
al mismo sin alterar el correcto funcionamiento del agente.

#                                23/04/2019

# Añadida funcion para obtener tamaño de memoria y uso de la CPU cada 10
episodios.

#                                24/04/2019

# Añadidas funciones directory_creation, path_check, load_episodes, load_steps

#                                27/04/2019

# Ahora la funcion load_episodes devuelve dos valores, el de los episodios y el
de los episodios completados.


import cv2

import psutil

import os

import h5py


# PROCESAMIENTO DE LA IMAGEN

# En image_processing, cogemos cada una de las imagenes que genera el programa y
las procesamos para dejarlas listas para el resto de tratamiento de datos.

# El procesamiento consiste en lo siguiente: un cambio de tamaño a 105x80 (la
imagen original es de tamaño 210x160), pasar la imagen a blanco y negro y a
escalar los valores de los pixeles a 1 al dividir entre 255.

def image_processing(image):

    processed_image = cv2.resize(image, (105,80))

    processed_image = cv2.cvtColor(processed_image, cv2.COLOR_RGB2GRAY)

    processed_image = processed_image/255.0

    return processed_image


# Esta pequeña funcion nos dara el tiempo de ejecucion conforme vaya avanzando:

def elapsed_time(start_time, episode_time):
```

```
    execution_time = episode_time-start_time
    m, s = divmod(execution_time, 60)
    h, m = divmod(m, 60)
    h = int(h)
    m = int(m)
    s = int(s)
    return h, m, s


# Esta funcion nos dira el uso de la CPU, uso de memoria y tamaño del batch de
memoria:
def memory_usage():
    print("CPU usage: ", psutil.cpu_percent())
    print(psutil.virtual_memory())


# Esta funcion comprobará si existe la estructura de archivos para el guardado;
si no existe, la creara.
def directory_creation(save_path, game):
    try:
        os.makedirs(save_path+game)
        print("Directory", save_path+game, "created.")
    except FileExistsError:
        print("Directory ", save_path+game, "already exists. Variables would be
loaded from there.")


# Esta funcion comprueba que existe un archivo de guardado. En caso contrario,
lo crea:
def path_check(save_filepath):
    if os.path.isfile(save_filepath) is False:
        savedata = h5py.File(save_filepath, 'w')
        savedata.close()
        print("Restoration point created.")
    else:
        print("Restoration point found.")


# Esta funcion carga los episodios guardados antes de empezar el bucle de
programa:
def load_episodes(save_filepath, epidoses, i_update):
```

```
    episodes = 100010
    try:
        savedata = h5py.File(save_filepath, 'r')
        saved_i = savedata['i']
        i_update = saved_i[()]
        savedata.close()
        episodes = episodes-i_update
        print("\033[1;32;38mEpisodes sucessfully loaded")
    except:
        episodes = episodes
        savedata.close()
        print("\033[1;31;38mEpisodes could not be loaded. Starting at episode
0")
    return episodes, i_update


# Esta funcion carga los pasos que hemos dado:
def load_steps(save_filepath):
    try:
        savedata = h5py.File(save_filepath, 'r')
        saved_steps = savedata['steps']
        step_counter = saved_steps[()]
        savedata.close()
        print("\033[1;32;38mSteps sucessfully loaded")
    except:
        step_counter = 0
        savedata.close()
        print("\033[1;31;38mSteps could not be loaded. Starting at step 0")
    return step_counter
```

VI.V Deimos_v3_google_colab_graph.py

Este archivo lo ejecutaremos después de cada sesión en *Google Colab*; tras haber obtenido los datos, creará la gráfica correspondiente a esa sesión de entrenamiento.

```
import matplotlib.pyplot as plt
import h5py

game = "BreakoutDeterministic-v4"
save_path = "Save/"
graph_savefile = "/Backup_google_colab/graph_save.hdf5"
graph_filepath = save_path+game+graph_savefile
saved_l10 = []
saved_max_score = 0.0
saved_max_mean = 0
saved_actual_mean = 0
saved_q_mean = 0
saved_reward = 0
saved_episodes = 0
saved_loss = []
i = []
try:
    savefile = h5py.File(graph_filepath, 'r')
    saved_l10 = savefile['l10']
    saved_max_score = savefile['max_score']
    saved_max_mean = savefile['max_mean']
    saved_actual_mean = savefile['med_episodes']
    saved_q_mean = savefile['q_mean']
    saved_reward = savefile['reward']
    saved_episodes = savefile['i']
    saved_loss = savefile['loss_list']
except:
    print("Data cannot be loaded")

max_score = saved_max_score[()]
max_mean = saved_max_mean[()]
```

```
Q_values_mean = list(saved_actual_mean[()])
total_reward = list(saved_reward[()])
episodes = saved_episodes[()]-49
loss_plt = list(saved_loss[()])
med_episodes = list(saved_actual_mean[()])
l10_mean = list(saved_l10[()])
savefile.close()
for x in range(episodes):
    i.append(x)

plt.ion()
plt.subplot(3,1,1)
plt.plot(i, loss_plt, 'b*')
plt.title('Loss, reward and Q-values vs episodes')
plt.ylabel('Loss', fontsize=12)
plt.subplot(3,1,2)
plt.plot(i, total_reward, 'ro', label = 'Reward')
plt.plot(i, med_episodes, 'g.', label = 'Mean')
plt.plot(i, l10_mean, 'y.', label = 'L10 Mean')
plt.legend(loc='upper right', bbox_to_anchor=(1.14,0.30), prop={'size':5.5})
plt.xlabel('Episodes', fontsize=12)
plt.ylabel('Reward', fontsize=12)
plt.subplot(3,1,3)
plt.plot(i, Q_values_mean, '.m')
plt.xlabel('Episodes', fontsize=12)
plt.ylabel('Q-values', fontsize=12)
plt.show()
```

VI.VI Deimos_v3_google_colab.py

Este archivo contiene la versión preparada para ejecutar en *Google Colab*; funciona exactamente igual que *Deimos_v3_agent.py*. Para ejecutarla en *Google Colab*, deberemos o bien cambiar la extensión a *.ipynb* o copiar todo el código a una libreta con esa extensión.

```
import gym
import cv2
import numpy as np
import random
import os
import time
import datetime
import h5py
from collections import deque
from keras.layers import Dense, Flatten, Conv2D
from keras.models import Sequential
from keras.optimizers import Adam


game = "BreakoutDeterministic-v4"
date = str(datetime.datetime.now().strftime('%d-%m-%Y-%H:%M:%S'))
start_time = time.time()
env = gym.make(game)
env.reset()
steps =  0
total_reward = 0
i = 0
i_update = i
episodes_sum = 0
exp_batch = []
weights = "/weights.best.hdf5"
mem_h5 = "/data.hdf5"
save_path = "Save/"
graph_savefile = "/graph_save.hdf5"
filepath = save_path+game+weights
save_filepath = save_path+game+mem_h5
graph_filepath = save_path+game+graph_savefile
```

```
reward_list = []
q_mean_list = []
loss_list = []
l10_list = []
mean_list = []

if 'Pong' in game:
    med_sup = -21
    max_reward = -21
else:
    med_sup = 0
    max_reward = 0
l10 = deque((), maxlen = 10)
Q_values = deque()
observation_episodes = 50
update_network = 10000

try:
    os.makedirs(save_path+game)
    print("Directory", save_path+game, "created.")
except FileExistsError:
    print("Directory ", save_path+game, "already exists. Variables would be
loaded from there.")

if os.path.isfile(save_filepath) is False:
    savedata = h5py.File(save_filepath, 'w')
    savedata.close()
    print("Restoration point created.")
else:
    print("Restoration point found.")

class AI:

    four_batch_size = 4
    four_memory_state = deque((), maxlen = four_batch_size)
```

```
four_memory_action = deque((), maxlen = four_batch_size)
four_memory_next_state = deque((), maxlen = four_batch_size)
four_memory_reward = deque((), maxlen = four_batch_size)
four_memory_done = deque((), maxlen = four_batch_size)
four_memory_lives = deque((), maxlen = four_batch_size)
four_memory_batch = deque((), maxlen = 1)
four_batch_state = deque((), maxlen = four_batch_size)
mem_size = 100000
mem_batch = deque((), maxlen = mem_size)
replay_sample_size = 32
sample_batch = []
s_batch = []
cnn_shape = (105,80,4)
batch_shape = (1,105,80,4)
learning_rate = 0.00025
action_size = env.action_space.n
initial_epsilon = 1
final_epsilon = 0.1
epsilon_frames = 300000
epsilon_decay = (initial_epsilon-final_epsilon)/epsilon_frames
epsilon = initial_epsilon
loss_list = []
gamma = 0.99
loss_plt = 0

def __init__(self):
    self.model = self.CNN()
    self.target_model = self.target_CNN()

    try:
        savedata = h5py.File(save_filepath, 'r')
        saved_epsilon = savedata['epsilon']
        AI.epsilon = saved_epsilon[()]
        savedata.close()
        print("\033[1;32;38mExploration rate sucessfully loaded")
```

```
        except:
            AI.epsilon = AI.epsilon
            savedata.close()
            print("\033[1;31;38mError: exploration rate values cant be loaded")


    def CNN(self):
        model = Sequential()
        model.add(Conv2D(32, (8, 8), strides=(4,4), input_shape = AI.cnn_shape,
activation='relu'))
        model.add(Conv2D(64, (4, 4), strides=(2,2), activation='relu'))
        model.add(Conv2D(64, (3, 3), strides=(1,1), activation='relu'))
        model.add(Flatten())
        model.add(Dense(512, activation='relu'))
        model.add(Dense(AI.action_size, activation='linear'))
        model.compile(loss="mse", optimizer=Adam(lr=AI.learning_rate))
        model.summary()
        print("Model CNN construction complete")
        return model


    def target_CNN(self):
        target_model = Sequential()
        target_model.add(Conv2D(32, (8, 8), strides=(4,4), input_shape =
AI.cnn_shape, activation='relu'))
        target_model.add(Conv2D(64, (4, 4), strides=(2,2), activation='relu'))
        target_model.add(Conv2D(64, (3, 3), strides=(1,1), activation='relu'))
        target_model.add(Flatten())
        target_model.add(Dense(512, activation='relu'))
        target_model.add(Dense(AI.action_size, activation='linear'))
        target_model.compile(loss="mse", optimizer=Adam(lr=AI.learning_rate))
        target_model.summary()
        print("Target CNN construction complete")
        print("\033[1;32;38mCNN construction completed sucessfully")
        return target_model


    def load(self):
        model = self.model
```

```
        try:
            model.load_weights(filepath)
            print("\033[1;32;38mModel weights sucessfully loaded")
            print("\033[1;31;38mALL SYSTEMS NOMINAL")
        except:
            print("\033[1;31;38mError: Model values cant be loaded\
033[1;37;38m")


    def save(self, i, steps):
        model = self.model
        exploration_rate = AI.epsilon
        if episodes % 10 == 0:
            model.save(filepath)
            print("\033[1;37;38mModel saved")
            savedata = h5py.File(save_filepath, 'w')
            saved_epsilon = savedata.create_dataset('epsilon', data =
AI.epsilon)
            saved_i = savedata.create_dataset('i', data = i_update)
            saved_steps = savedata.create_dataset('steps', data = step_counter)
            savedata.close()
        return exploration_rate, saved_epsilon, saved_i, saved_steps


    def update_net(self):
        model = self.model
        model_weights = model.get_weights()
        target_model = self.target_model
        target_model.set_weights(model_weights)
        print("Target model sucessfully updated")


    def batcher(four_batch_state):
        AI.s_batch = []
        AI.s_batch = np.asarray(AI.four_batch_state)
        AI.s_batch.shape = AI.batch_shape


    def four_batcher(self, exp_batch):
        AI.four_batch_state.append(exp_batch[2])
```

```
        if len(AI.four_batch_state) == AI.four_batch_size:
            AI.batcher(AI.four_batch_state)

    def four_memory(self, exp_batch):
       AI.four_memory_state.append(exp_batch[0])
       AI.four_memory_action.append(exp_batch[1])
       AI.four_memory_next_state.append(exp_batch[2])
       AI.four_memory_reward.append(exp_batch[3])
       AI.four_memory_done.append(exp_batch[4])
       AI.four_memory_lives.append(exp_batch[5])

       if len(AI.four_memory_state) == AI.four_batch_size:
            AI.four_memory_batch.append((AI.four_memory_state,
AI.four_memory_action, AI.four_memory_next_state, AI.four_memory_reward,
AI.four_memory_done, AI.four_memory_lives))
            AI.memory(AI.four_memory_batch)

    def memory(self):
        if len(AI.mem_batch) < AI.mem_size:
            AI.mem_batch.append(AI.four_memory_batch)
        else:
            AI.mem_batch.popleft()
            AI.mem_batch.append(AI.four_memory_batch)

    def replay_sampler(self, mem_batch):
        AI.sample_batch = []

        if len(AI.mem_batch) < AI.replay_sample_size:
            AI.sample_batch.append(random.sample(AI.mem_batch,
len(AI.mem_batch)))
        else:
            AI.sample_batch.append(random.sample(AI.mem_batch,
AI.replay_sample_size))

        AI.sample_batch = np.asarray(AI.sample_batch)
        AI.sample_batch.shape = (AI.replay_sample_size, 6, 4)
```

```
    def action_selection(self, s_batch):
        random_value = np.random.rand()
        if random_value < AI.epsilon or i < observation_episodes:
            best_action = env.action_space.sample()
            best_action_value = 0
        else:
            action_values = self.model.predict(s_batch)
            best_action = np.argmax(action_values[0])
            best_action_value = action_values[0, best_action]
        if i > observation_episodes:
            if AI.epsilon > AI.final_epsilon:
                AI.epsilon -= AI.epsilon_decay
        return best_action, best_action_value

    def training(self, sample_batch):
        AI.training_s_batch = []
        AI.training_a_batch = []
        AI.training_ns_batch = []
        AI.training_r_batch = []
        AI.training_d_batch = []
        Q_values = []
        loss_batch = []
        batch_size = len(AI.sample_batch)
        Q_values = np.zeros((batch_size, AI.action_size))
        new_Q = np.zeros((batch_size, AI.action_size))

        for m in range(batch_size):
            AI.training_s_batch =
np.asarray(list(AI.sample_batch[m,0])).reshape(AI.batch_shape)
            AI.training_a_batch = np.asarray(list(AI.sample_batch[m,1]))
            AI.training_a_batch = AI.training_a_batch[:,0].astype(int)
            AI.training_ns_batch =
np.asarray(list(AI.sample_batch[m,2])).reshape(AI.batch_shape)
            AI.training_r_batch = np.asarray(list(AI.sample_batch[m,3]))
            AI.training_d_batch = np.asarray(list(AI.sample_batch[m,1]))
```

```
            loss_batch.append(AI.training_s_batch)


            Q_values[m] =
self.model.predict(AI.training_s_batch.reshape(AI.batch_shape))
            Q_values[m, AI.training_a_batch] = AI.training_r_batch


            if 'Pong' in game:
                    new_Q[m] =
self.target_model.predict(AI.training_ns_batch.reshape(AI.batch_shape))
                    Q_values[m, AI.training_a_batch] += (AI.gamma *
np.max(new_Q))
            else:
                if np.all(AI.training_d_batch) == False:
                    new_Q[m] =
self.target_model.predict(AI.training_ns_batch.reshape(AI.batch_shape))
                    Q_values[m, AI.training_a_batch] += (AI.gamma *
np.max(new_Q))


        loss =
self.model.fit(np.asarray(loss_batch).reshape(batch_size,105,80,4), Q_values,
batch_size=batch_size, verbose=0, shuffle=False)
        loss_history = loss.history['loss'][0]
        AI.loss_list.append(loss_history)


        if 'Pong' in game:
            if done == True:
                AI.loss_plt = np.mean(AI.loss_list)
                AI.loss_list = []
        else:
            if lives == 0:
                AI.loss_plt = np.mean(AI.loss_list)
                AI.loss_list = []


def image_processing(image):
    processed_image = cv2.resize(image, (105,80))
    processed_image = cv2.cvtColor(processed_image, cv2.COLOR_RGB2GRAY)
```

```
    processed_image = processed_image/255.0
    return processed_image


def elapsed_time(start_time, episode_time):
    execution_time = episode_time-start_time
    m, s = divmod(execution_time, 60)
    h, m = divmod(m, 60)
    h = int(h)
    m = int(m)
    s = int(s)
    return h, m, s


def save_graph(graph_filepath, l10_mean, max_reward, med_sup, med_episodes,
q_mean_list, reward_list, i, loss_list):
    savefile = h5py.File(graph_filepath, 'w')
    saved_l10 = savefile.create_dataset('l10', data = l10_list)
    saved_max_score = savefile.create_dataset('max_score', data = max_reward)
    saved_max_mean = savefile.create_dataset('max_mean', data = med_sup)
    saved_actual_mean = savefile.create_dataset('med_episodes', data =
mean_list)
    saved_q_mean = savefile.create_dataset('q_mean', data = q_mean_list)
    saved_reward = savefile.create_dataset('reward', data = reward_list)
    saved_episodes = savefile.create_dataset('i', data = i)
    saved_loss = savefile.create_dataset('loss_list', data = loss_list)
    savefile.close()
    return saved_l10, saved_max_score, saved_max_mean, saved_actual_mean,
saved_q_mean, saved_reward, saved_episodes, saved_loss


agent = AI()
episodes = 100010
try:
        savedata = h5py.File(save_filepath, 'r')
        saved_i = savedata['i']
        i_update = saved_i[()]
        savedata.close()
        episodes = episodes-i_update
```

```
        print("\033[1;32;38mEpisodes sucessfully loaded")
except:
        episodes = 100010
        savedata.close()
        print("\033[1;31;38mEpisodes could not be loaded. Starting at episode
0")
try:
    savedata = h5py.File(save_filepath, 'r')
    saved_steps = savedata['steps']
    step_counter = saved_steps[()]
    savedata.close()
    print("\033[1;32;38mSteps sucessfully loaded")
except:
    step_counter = 0
    savedata.close()
    print("\033[1;31;38mSteps could not be loaded. Starting at step 0")

agent.load()

for i in range (episodes):
    state = env.reset()
    total_reward = 0
    if i > observation_episodes:
        if i % 10 == 0:
            agent.save(i_update, AI.epsilon)
    if i == 0 and steps == 0:
        random_action = env.action_space.sample()
        state1, reward1, done1, lives1 = env.step(random_action)
        done1 = False
        action1 = [random_action, 0.0]
        lives1 = lives1['ale.lives']
        next_state1 = state1
        exp_batch = (image_processing(state1), action1,
image_processing(next_state1), reward1, done1, lives1)
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
```

```
        agent.four_batcher(exp_batch)
        agent.four_batcher(exp_batch)
        reward1 = np.sign(reward1)
        total_reward += reward1
        del state1, action1, next_state1, reward1, done1, exp_batch,
random_action
    no_op_counter = 0
    no_op_max = np.random.randint(4,30, dtype=int)


    for steps in range (10000):
        if no_op_counter < no_op_max:
            action = [0, 0.0]
            no_op_counter += 1
        else:
            action = agent.action_selection(AI.s_batch)
        Q_values.append(action[1])
        next_state, reward, done, lives = env.step(action[0])
        lives = lives['ale.lives']
        reward = np.sign(reward)
        if (lives < lives1):
            done == True
        total_reward += reward
        exp_batch = (image_processing(state), action,
image_processing(next_state), reward, done, lives)
        agent.four_batcher(exp_batch)
        agent.four_memory(exp_batch)
        state = next_state
        exp_batch = []
        step_counter += 1
        if i >= observation_episodes:
            agent.replay_sampler(AI.mem_batch)
            agent.training(AI.sample_batch)
            if step_counter % update_network == 0:
                agent.update_net()
            if 'Pong' in game:
                if done == True:
```

```
                    i_update += 1
                    episodes_sum += total_reward
                    episode_time = time.time()
                    if len(l10) < 10:
                        l10.append(total_reward)
                    if total_reward > max_reward:
                      max_reward = total_reward
                    else:
                      l10.popleft()
                      l10.append(total_reward)
                    l10_mean = np.mean(l10)
                    Q_values_mean = np.mean(Q_values)
                    q_mean_list.append(Q_values_mean)
                    Q_values = []
                    med_episodes = round((episodes_sum/((i+1)-
observation_episodes)),2)
                    h,m,s = elapsed_time(start_time, episode_time)
                    if med_episodes > med_sup:
                        med_sup = med_episodes
                    reward_list.append(total_reward)
                    loss_list.append(AI.loss_plt)
                    l10_list.append(l10_mean)
                    mean_list.append(med_episodes)
                    if i > observation_episodes:
                        if i % 10 == 0:
                            save_graph(graph_filepath, l10_mean, max_reward,
med_sup, med_episodes, q_mean_list, reward_list, i, loss_list)
                    print("\033[1;32;38mEpisode: {}/{}, Total steps: {}, Total
Score: {}, e: {:.4f}, loss: {:.6f}, Execution time: {}:{}:{}"
                        .format(i, episodes, steps, total_reward, agent.epsilon,
AI.loss_plt, h,m,s))
                    print("\033[1;32;38mMax score: {:.2f}, Maximum mean: {:.2f},
Actual mean: {:.2f}, L10 mean: {:.2f}, Q-values mean:
{:.6f}" .format(max_reward, med_sup, med_episodes, l10_mean, Q_values_mean))
            else:
                if lives == 0:
                    i_update += 1
```

```
                    episodes_sum += total_reward
                    episode_time = time.time()
                    if len(l10) < 10:
                        l10.append(total_reward)
                    if total_reward > max_reward:
                        max_reward = total_reward
                    else:
                        l10.popleft()
                        l10.append(total_reward)
                    l10_mean = np.mean(l10)
                    Q_values_mean = np.mean(Q_values)
                    q_mean_list.append(Q_values_mean)
                    Q_values = []
                    med_episodes = round((episodes_sum/((i+1)-
observation_episodes)),2)
                    h,m,s = elapsed_time(start_time, episode_time)
                    if med_episodes > med_sup:
                        med_sup = med_episodes
                    reward_list.append(total_reward)
                    loss_list.append(AI.loss_plt)
                    l10_list.append(l10_mean)
                    mean_list.append(med_episodes)
                    if i > observation_episodes:
                        if i % 10 == 0:
                            save_graph(graph_filepath, l10_mean, max_reward,
med_sup, med_episodes, q_mean_list, reward_list, i, loss_list)
                    print("\033[1;32;38mEpisode: {}/{}, Total steps: {}, Total
Score: {}, e: {:.4f}, loss: {:.6f}, Execution time: {}:{}:{}"
                      .format(i, episodes, steps, total_reward, agent.epsilon,
AI.loss_plt, h,m,s))
                    print("\033[1;32;38mMax score: {:.2f}, Maximum mean: {:.2f},
Actual mean: {:.2f}, L10 mean: {:.2f}, Q-values mean:
{:.6f}" .format(max_reward, med_sup, med_episodes, l10_mean, Q_values_mean))
        if steps % 100 == 0:
                    print("\033[1;37;38mEpisode: {}/{}, Steps: {}, Total steps:
{}, Score: {}"
                        .format(i, episodes, steps, step_counter, total_reward))
```

```
if 'Pong' in game:
    if done == True:
        break
else:
    if lives == 0:
        break
```

VI.VII Deimos_v3_metrics.py

En este archivo se recoge el código referente a las funciones encargadas de realizar distintas mediciones durante el funcionamiento del agente.

```
#                               DEIMOS-V3
#                               Metricas
# En este archivo guardaremos las funciones referentes a la obtencion de datos, graficas, etc.
#                               24/04/2019
# Arreglada leyenda de la grafica.


import matplotlib.pyplot as plt


def print_episode_final(i, episodes, steps, total_reward, epsilon, loss_plt, h,m,s, max_reward, med_sup, med_episodes, l10_mean, Q_values_mean):
    print("\033[1;32;38mEpisode: {}/{}, Total steps: {}, Total Score: {}, e: {:.4f}, loss: {:.6f}, Execution time: {}:{}:{}"
                    .format(i, episodes, steps, total_reward, epsilon, loss_plt, h,m,s))
    print("\033[1;32;38mMax score: {:.2f}, Maximum mean: {:.2f}, Actual mean: {:.2f}, L10 mean: {:.2f}, Q-values mean: {:.6f}" .format(max_reward, med_sup, med_episodes, l10_mean, Q_values_mean))



def print_progress(i, episodes, steps, step_counter, total_reward):
    print("\033[1;37;38mEpisode: {}/{}, Steps: {}, Total steps: {}, Score: {}"
                        .format(i, episodes, steps, step_counter, total_reward))


# Ploteamos las recompensas y el loss para comprobar como funciona todo:
def graph_plot(i, loss_plt, total_reward, med_episodes, l10_mean, Q_values_mean, observation_episodes):
    plt.ion()
    plt.subplot(3,1,1)
    plt.plot(i, loss_plt, 'b*')
    plt.title('Loss, reward and Q-values vs episodes')
    plt.ylabel('Loss', fontsize=12)
    plt.subplot(3,1,2)
```

```
    plt.plot(i, total_reward, 'ro', label = 'Reward')
    plt.plot(i, med_episodes, 'g.', label = 'Mean')
    plt.plot(i, l10_mean, 'y.', label = 'L10 Mean')
    if i == observation_episodes:
        plt.legend(loc='upper right', bbox_to_anchor=(1.14,0.30),
prop={'size':5.5})
    plt.xlabel('Episodes', fontsize=12)
    plt.ylabel('Reward', fontsize=12)
    plt.subplot(3,1,3)
    plt.plot(i, Q_values_mean, '.m')
    plt.xlabel('Episodes', fontsize=12)
    plt.ylabel('Q-values', fontsize=12)
    plt.pause(0.05)
    plt.show()
```

VI.VIII Deimos_v3_random_test.py

En este archivo ejecutaremos el test final aleatorio, obteniendo tras el mismo una gráfica con resultados, así como puntuaciones máximas, medias, etc, que luego se utilizarán para comparar con el test final realizado tras entrenar al agente.

```
import gym
import numpy as np
from collections import deque
import matplotlib.pyplot as plt

game = "PongDeterministic-v4"
env = gym.make(game)
env.reset()
episodes = 200
episodes_sum = 0
l10 = deque((), maxlen = 10)
if 'Pong' in game:
    med_sup = -21
    max_reward = -21
else:
    med_sup = 0
    max_reward = 0

for i in range(episodes):
    total_reward = 0
    env.reset()
    for steps in range(10000):
        action = env.action_space.sample()
        _, reward, done, _ = env.step(action)
        env.render()
        steps += 1
        total_reward += reward
        if done == True:
            episodes_sum += total_reward
            if len(l10) < 10:
```

```
                l10.append(total_reward)
            if total_reward > max_reward:
                max_reward = total_reward
            else:
                l10.popleft()
                l10.append(total_reward)
            l10_mean = np.mean(l10)
            med_episodes = round((episodes_sum/(i+1)),2)
            if med_episodes > med_sup:
                med_sup = med_episodes
            print("\033[1;32;38mMax score: {:.2f}, Maximum mean: {:.2f}, Actual
mean: {:.2f}, L10 mean: {:.2f}" .format(max_reward, med_sup, med_episodes,
l10_mean))
        if steps % 100 == 0:
            print("\033[1;37;38mEpisode: {}/{}, Total steps: {}, Total Score:
{}"
                      .format(i, episodes, steps, total_reward))

        if done == True:
            plt.subplot(3,1,1)
            plt.plot(i, total_reward, 'ro', label = 'Reward')
            plt.plot(i, med_episodes, 'g.', label = 'Mean')
            plt.plot(i, l10_mean, 'y.', label = 'L10 Mean')
            plt.xlabel('Episodes', fontsize=12)
            plt.ylabel('Reward', fontsize=12)
            if i == 0:
                plt.legend(loc='upper right', bbox_to_anchor=(1.14,0.10),
prop={'size':5.5})
            plt.pause(0.05)
            plt.show()
            break
env.close()
```

10. Bibliografía

Bibliografía

1: Future of life artificial intelligence benefits risk,
https://futureoflife.org/wp-content/uploads/2015/11/artificial_intelligence_benefits_risk.jpg
2: Imagen Google DeepMind DQN,
https://image.itmedia.co.jp/pcuser/articles/1607/24/yu_dqn2.jpg
3: Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, Playing Atari with Deep Reinforcement Learning,
2013,https://arxiv.org/pdf/1312.5602v1.pdf
4: Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis1, Human-level control through deep reinforcement learning,
2015,https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
5: Patente DQN de Google, https://patents.google.com/patent/US20150100530A1/en
6: Leonardoaraujosantos gitbook conv agent image,
https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/image_folder_5/conv_agent.png
7: Imagen SuperMario,
http://whatleydude.com/wp-content/uploads/2015/09/special___wallpaper___super_mario_bros___1_1_by_thelimomon-d6npbma.jpg
8: Forbes imagen AI, https://specials-images.forbesimg.com/imageserve/684285598/960x0.jpg?fit=scale
9: Computer Hoy, 10 usos de la inteligencia artificial que nadie se hubiera imaginado,
https://computerhoy.com/noticias/life/10-usos-inteligencia-artificial-que-nadie-hubiera-imaginado-79811
10: Imagen coche autónomo, https://upload.wikimedia.org/wikipedia/commons/thumb/6/65/Hands-free_Driving.jpg/1920px-Hands-free_Driving.jpg
11: Search enterprise AI, self-driving car,
https://searchenterpriseai.techtarget.com/definition/driverless-car
12: Imagen inteligencia artificial en videojuegos, http://slideplayer.com/slide/4380437/14/images/2/What+is+AI+in+Games+Techniques+used+in+computer+and+video+games+to+produce+the+illusion+of+intelligence+in+the+behavior+of+non-player+characters..jpg
13: Noticias universia, 5 usos cotidianos de la inteligencia artificial que quizás no conozcas,
http://noticias.universia.com.ar/cultura/noticia/2018/02/22/1158097/5-usos-cotidianos-inteligencia-artificial-quiza-conozcas.html
14: Imagen inteligencia artificial en fábricas,
https://www.latticesemi.com/-/media/LatticeSemi/Images/Blogs/AI-in-smart-factory.ashx?h=260&la=en&mh=432&mw=768&w=370
15: Giztab, Los 5 usos más comunes de la inteligencia artificial, https://www.giztab.com/que-usos-tiene-la-inteligencia-artificial/
16: Imagen reconocimiento facial,
https://mondrian.mashable.com/2014%252F03%252F19%252Fc7%252FFace_Recogn.9c9af.jpg%252F1200x627.jpg?signature=y7Y8OBgb3MFANYDSwBB_61Ruors=
17: Wikipedia Aplicaciones de la inteligencia artificial,
https://es.wikipedia.org/wiki/Aplicaciones_de_la_inteligencia_artificial
18: , https://image.slidesharecdn.com/1655tilly-170405112708/95/my-robot-can-learn-using-reinforcement-learning-to-teach-my-robot-5-638.jpg?cb=1491391721

19: Hongzi Mao, Mohammad Alizadeh, Ishai Menache, Srikanth Kandula, Massachusetts Institute of Technology, Microsoft Research, Resource Management with Deep Reinforcement Learning, ,https://people.csail.mit.edu/alizadeh/papers/deeprm-hotnets16.pdf
20: Towardsdatascience applications of reinforcement learning in real world, https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12
21: I. Arel, C. Liu, T. Urbanik, A.G. Kohls, Reinforcement learning-based multi-agentsystem for network traffic signal control, ,http://web.eecs.utk.edu/~itamar/Papers/IET_ITS_2010.pdf
22: Imagen control tráfico IA, https://3c1703fe8d.site.internapcdn.net/newman/csz/news/800/2017/3-2-artificialin.jpg
23: Jens Kober, J. Andrew Bagnell, Jan Peters, Reinforcement Learning in Robotics:A Survey, ,https://www.ias.informatik.tu-darmstadt.de/uploads/Publications/Kober_IJRR_2013.pdf
24: Sergey Levine, Chelsea Finn, Trevor Darrell, Pieter Abbeel, End-to-End Training of Deep Visuomotor Policies, ,https://arxiv.org/pdf/1504.00702.pdf
25: Zhenpeng Zhou, Xiaocheng Li, and Richard N. Zare, Optimizing Chemical Reactions with Deep Reinforcement Learning, ,https://pubs.acs.org/doi/full/10.1021/acscentsci.7b00492
26: Imagen lenguajes de programación, https://pcquest.com/wp-content/uploads/2016/12/programming-languages.jpg
27: Icono C, https://png.icons8.com/color/1600/c-programming
28: Wikipedia C, https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)
29: Toward Data science what is the best programming language for machine learning, https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7
30: Quora Is C language a good choice for machine learning, https://www.quora.com/Is-C-language-a-good-choice-for-machine-learning
31: Tensorflow installation C language, https://www.tensorflow.org/install/lang_c
32: Wikipedia caffe, https://en.wikipedia.org/wiki/Caffe_(software)
33: Web de theano, http://deeplearning.net/software/theano/
34: , http://www.itavalon.com/wp-content/uploads/2016/05/c-plus-logo-300x169.png
35: Wikipedia C++, https://es.wikipedia.org/wiki/C%2B%2B
36: Uptowork C vs C++, https://www.upwork.com/hiring/development/c-vs-c-plus-plus/
37: , https://upload.wikimedia.org/wikipedia/en/thumb/3/30/Java_programming_language_logo.svg/1200px-Java_programming_language_logo.svg.png
38: Wikipedia Java, https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)
39: Machine Learning mastery Java machine learning, https://machinelearningmastery.com/java-machine-learning/
40: Tensorflow Java, https://www.tensorflow.org/install/lang_java
41: Deeplearning4j, https://deeplearning4j.org/
42: IBM encuesta mejor lenguaje para machine learning y data science, https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Language_Is_Best_For_Machine_Learning_And_Data_Science?lang=en
43: , https://www.solvetic.com/uploads/monthly_01_2016/tutorials-1415-0-60642300-1452279191.jpg
44: Wikipedia Python, https://es.wikipedia.org/wiki/Python
45: Data Flair Features of Python, https://data-flair.training/blogs/features-of-python/
46: , https://cdn.shopify.com/s/files/1/1668/0637/products/BSYjGT6BceyC9KPXjMy853j4Sp68Dt-left_large.png?v=1496897723
47: The Zen of Python, http://www.thezenofpython.com/
48: Python whats new on 3.7, https://docs.python.org/3/whatsnew/3.7.html

49: Web de NumPy, http://www.numpy.org/
50: Instalacion de tensorflow en Python, https://www.tensorflow.org/install
51: Web de Theano, http://deeplearning.net/software/theano/
52: Requisitos instalacion caffe, http://caffe.berkeleyvision.org/installation.html#prerequisites
53: Instalacion keras Python, https://keras.io/#installation
54: , https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/R_logo.svg/1200px-R_logo.svg.png
55: Wikipedia R lenguaje de programacion, https://es.wikipedia.org/wiki/R_(lenguaje_de_programaci%C3%B3n
56: Pagina web de R, seccion about, https://www.r-project.org/about.html
57: Datacamp tutorial neural network models R, https://www.datacamp.com/community/tutorials/neural-network-models-r
58: R Bloggers CNN in R, https://www.r-bloggers.com/convolutional-neural-networks-in-r/
59: R Keras, https://keras.rstudio.com/
60: Datascience+ neuralment train and test neural networks using R, https://datascienceplus.com/neuralnet-train-and-test-neural-networks-using-r/
61: , https://15e6l2wqu7j3wc4xz1ngopsw-wpengine.netdna-ssl.com/wp-content/uploads/2017/11/MATLAB.jpg
62: Wikipedia MATLAB, https://es.wikipedia.org/wiki/MATLAB
63: Página web de mathworks, https://es.mathworks.com/
64: , https://upload.wikimedia.org/wikipedia/commons/thumb/6/6a/Gnu-octave-logo.svg/1200px-Gnu-octave-logo.svg.png
65: Wikipedia Octave, https://es.wikipedia.org/wiki/GNU_Octave
66: DeepLearning Matlab Toolbox, https://es.mathworks.com/products/deep-learning.html
67: Toolbox para DeepLearning para Matlab , https://github.com/rasmusbergpalm/DeepLearnToolbox
68: Función nnet de Octave, https://octave.sourceforge.io/nnet/overview.html
69: , https://www.techdotmatrix.com/wp-content/uploads/2016/10/Programming-languages.jpg
70: Wikipedia lista lenguajes para inteligencia artificial, https://en.wikipedia.org/wiki/List_of_programming_languages_for_artificial_intelligence
71: , https://www.assignmenthelp.net/images/programming/machine-programming.png
72: Imagen robot trabajando en PC, http://www.megatronicslab.com/wp-content/uploads/2019/05/ct-bsi-sap-machine-learning-hiring-20170621-1024x576.jpg
73: Imagen código binario, https://www.zsl.org/sites/default/files/image/2014-02/software_108744983.jpg
74: Logo Anaconda, https://www.anaconda.com/wp-content/uploads/2018/06/cropped-Anaconda_horizontal_RGB-1-600x102.png
75: Wikipedia Anaconda (Python distribution), https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)
76: Web oficial de Anaconda, Distribucion, https://www.anaconda.com/distribution/
77: Documentacion Anaconda, https://docs.anaconda.com/anaconda/
78: Logotipo Spyder, https://upload.wikimedia.org/wikipedia/commons/thumb/7/7e/Spyder_logo.svg/1200px-Spyder_logo.svg.png
79: Wikipedia Spyder software, https://en.wikipedia.org/wiki/Spyder_(software)
80: Web de Spyder, https://www.spyder-ide.org/
81: Tensorflow logo, https://upload.wikimedia.org/wikipedia/commons/thumb/1/11/TensorFlowLogo.svg/1200px-TensorFlowLogo.svg.png

82: Web de tensorflow, estudio de casos, https://www.tensorflow.org/about/case-studies/
83: Wikipedia Tensorflow, https://en.wikipedia.org/wiki/TensorFlow
84: Logotipo Keras, http://adventuresinmachinelearning.com/wp-content/uploads/2017/05/keras-logo-small-wb-1.png
85: Web oficial de keras, https://keras.io/
86: Wikipedia Keras, https://en.wikipedia.org/wiki/Keras
87: Logotipo OpenCV, https://upload.wikimedia.org/wikipedia/commons/thumb/3/32/OpenCV_Logo_with_text_svg_version.svg/1200px-OpenCV_Logo_with_text_svg_version.svg.png
88: Wikipedia OpenCV, https://en.wikipedia.org/wiki/OpenCV
89: Web oficial de OpenCV, sección about, https://opencv.org/about.html
90: Web de NumPy, http://www.numpy.org/
91: Wikipedia NumPy, https://en.wikipedia.org/wiki/NumPy
92: Web de OpenAI, seccion about, https://openai.com/about/
93: Wikipedia OpenAI, https://en.wikipedia.org/wiki/OpenAI
94: Logotipo OpenAI-Gym, https://cdn-images-1.medium.com/max/1600/1*ZHISh_zLYIlJPTq_6lX5LQ.png
95: Imagen cylon OpenAI, http://canadajournal.net/technology/openai-elon-musk-opens-free-training-gym-artificial-intelligence-46883-2016/
96: Web de Matplotlib, https://matplotlib.org/
97: Wikipedia Matplotlib, https://en.wikipedia.org/wiki/Matplotlib
98: Logotipo matplotlib, https://upload.wikimedia.org/wikipedia/commons/thumb/8/84/Matplotlib_icon.svg/1024px-Matplotlib_icon.svg.png
99: Logo Google Colaboratory, https://cdn-images-1.medium.com/max/1600/1*PW6-aildLxUPlZcvLtJtEg.jpeg
100: Google Colaboratory FAQ, https://research.google.com/colaboratory/faq.html
101: Towards data science Google Colaboratory, https://towardsdatascience.com/google-colaboratory-simplifying-data-science-workflow-c70059386323
102: Imagen hardware ordenador, https://wallpapertag.com/wallpaper/full/2/d/7/944930-download-computer-hardware-wallpaper-1920x1200.jpg
103: Imagen placa base, https://cdn.wccftech.com/wp-content/uploads/2017/10/Intel-Z370-Motherboard.jpg
104: Imagen PCIe, https://linuxtidbits.files.wordpress.com/2009/01/pci-express-slots.gif?w=560
105: Tim Dettmers, A full hardware guide to Deep Learning, http://timdettmers.com/2018/12/16/deep-learning-hardware-guide/
106: Cotscomputers, PCIe lanes explained, https://cotscomputers.com/blog/pcie-lanes/
107: Articulosgamer que es una CPU, https://articulosgamer.com/que-es-una-cpu/
108: Blog algorithmia, hardware for machine learning, https://blog.algorithmia.com/hardware-for-machine-learning/
109: Imagen AMD Ryzen 5, https://images.techhive.com/images/article/2017/03/amd-ryzen-5-primary-100713506-large.jpg
110: , einfochips, everything you need to know about hardware requirements for machine learning,
111: Imagen Memoria RAM, https://www.fixmypcfree.com/wp-content/uploads/2013/09/ram.jpg
112: Imagen msi Nvidia RTX2070, https://www.geeks3d.com/public/jegx/2018q4/msi/geforce-rtx-2070-ventus-8gb-img04.jpg
113: Imagen configuración múltiples GPUs, https://www.extremetech.com/wp-content/uploads/2016/06/multi-GPU-SLI.jpg
114: Tim Dettmers, which gpu for deep learning, http://timdettmers.com/2019/04/03/which-gpu-for-deep-learning/

115: Imagen AMD vs Nvidia, https://i.ytimg.com/vi/c7vAN3IVBdY/maxresdefault.jpg
116: Infoworld, what is cuda parallel programming for gpus,
https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html
117: Logotipo Nvidia CUDA,
http://jtes.net/wp-content/uploads/2015/05/NVIDIA_CUDA_V_2C_r.jpg
118: Imagen Inteligencia Artificial, https://wallpapertag.com/wallpaper/full/a/8/c/751417-artificial-intelligence-wallpapers-1920x1080-pictures.jpg
119: Wikipedia Inteligencia Artificial, https://es.wikipedia.org/wiki/Inteligencia_artificial
120: Wikipedia Alan Turing, https://es.wikipedia.org/wiki/Alan_Turing
121: Wikipedia Historia de la Inteligencia Artificial,
https://es.wikipedia.org/wiki/Historia_de_la_inteligencia_artificial
122: Xataka Que es la inteligencia artificial, https://www.xataka.com/robotica-e-ia/que-inteligencia-artificial
123: Omicrono Que es la Inteligencia Artificial, https://omicrono.elespanol.com/2018/10/que-es-la-inteligencia-artificial/
124: Real Academia de la lengua Española, definición de Inteligencia Artificial, https://dle.rae.es/?id=LqtyoaQ
125: Imagen cerebro-ordenador, https://www-tc.pbs.org/wgbh/nova/assets/img/full-size/artificial-intelligence-merl.jpg
126: Zeolearn understanding different types of artificial intelligence,
https://www.zeolearn.com/magazine/understanding-different-types-of-artificial-intelligence-technology
127: Imagen clasificacion IA,
https://simplecore.intel.com/newsroom/wp-content/uploads/sites/11/2018/05/artificial-intelligence-terms.jpg
128: Imagen Machine Learning,
https://cdn-images-1.medium.com/max/1200/1*M9le42saJxWlOYyYvhKtPA.jpeg
129: Wikipedia Aprendizaje automático, https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico
130: Imagen clasificación unsupervised learning, supervised learning y reinforcement learning,
https://wordstream-files-prod.s3.amazonaws.com/s3fs-public/machine-learning.png
131: Imagen esquema aprendizaje supervisado,
https://i0.wp.com/vinodsblog.com/wp-content/uploads/2018/04/SMLP1.png?resize=1300%2C721&ssl=1
132: Wikipedia Aprendizaje supervisado, https://es.wikipedia.org/wiki/Aprendizaje_supervisado
133: Medium Tipos de aprendizaje automático, https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2
134: Esquema explicativo aprendizaje no supervisado, https://community.singularitynet.io/uploads/db8677/original/1X/3f814ab62d70a1013e1e23bef356f941bdeff44e.jpg
135: Wikipedia Aprendizaje no supervisado,
https://es.wikipedia.org/wiki/Aprendizaje_no_supervisado
136: Imagen algunos algoritmos de aprendizaje no supervisado, https://i1.wp.com/vinodsblog.com/wp-content/uploads/2018/11/Common-Algorithms-in-Unsupervised-Learning.png?fit=958%2C541&ssl=1
137: Wikipedia Aprendizaje por refuerzo, https://es.wikipedia.org/wiki/Aprendizaje_por_refuerzo
138: Wikipedia conductismo, https://es.wikipedia.org/wiki/Conductismo
139: Topics in Reinforcement Learning, Reinforcement Learning Repository at UMass, Amherst,
http://www-anw.cs.umass.edu/rlr/
140: Esquema simplificado RL,
https://bigdata-madesimple.com/wp-content/uploads/2018/02/Machine-Learning-Explained3.png

141: Imagen ejemplo simplificado aprendizaje por refuerzo, https://cdn.marutitech.com/wp-content/uploads/2017/04/RL1.jpg
142: Deep Reinforcement Learning doesn't work yet, https://www.alexirpan.com/2018/02/14/rl-hard.html
143: Reinforcement learning never worked and 'deep' only helped a bit, https://himanshusahni.github.io/2018/02/23/reinforcement-learning-never-worked.html
144: OpenAI Faulty reward functions, https://himanshusahni.github.io/2018/02/23/reinforcement-learning-never-worked.html
145: Imagen funcionamiento simplificado aprendizaje por refuerzo, https://keon.io/images/deep-q-learning/rl.png
146: Imagen Deep Learning, https://i0.wp.com/slator.com/assets/2016/03/deep-learning-pic.png?fit=1299%2C861&ssl=1
147: Machine Learning Mastery what is deep learning, https://machinelearningmastery.com/what-is-deep-learning/
148: Wikipedia aprendizaje profundo, https://es.wikipedia.org/wiki/Aprendizaje_profundo
149: Imagen extracción características convnet, https://cdn-images-1.medium.com/max/2600/1*8C49dLMgINot63DvafDD3g.jpeg
150: Imagen descomposición funcionamiento interno convnet, https://ujwlkarn.files.wordpress.com/2016/08/conv_all.png
151: Wikipedia Redes neuronales convolucionales, https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales
152: Wikipedia convolución, https://es.wikipedia.org/wiki/Convoluci%C3%B3n
153: Numerentur CNN, http://numerentur.org/convolucionales/
154: Stanford Lecture 6: CNNs and Deep Q Learning, https://web.stanford.edu/class/cs234/CS234Win2018/slides/cs234_2018_l6.pdf
155: Wikipedia Rectifier (neural networks), https://en.wikipedia.org/wiki/Rectifier_(neural_networks)
156: Image Classification using Convolutional Neural Networks in Keras, https://www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/
157: Imagen extracción características mediante CNN, https://cdn-images-1.medium.com/max/1600/1*Ji5QhY9QXBlpNNLH4qAcNA.png
158: Towards data science types of optimization algorithms used in neural networks and ways to optimize gradient, https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f
159: Keras optimizers, https://keras.io/optimizers/
160: Lecture 6a: Overview of mini- batch gradient descent , http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
161: Intro to optimization in deep learning: Momentum, RMSProp and Adam, https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/
162: Towards data science types of optimization algorithms used in neural networks and ways to optimize gradient descent, https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f
163: Medium How to pick the best learning rate for your machine learning project, https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2
164: Wikipedia discrete time and continuous time, https://en.wikipedia.org/wiki/Discrete_time_and_continuous_time
165: Wikipedia stochastic, https://en.wikipedia.org/wiki/Stochastic
166: Imagen MDP Starcraft, https://s.yimg.com/ny/api/res/1.2/W7wZ_lIUgO1KN7JbXQWh_A--/YXBwaWQ9aGlnaGxhbmRlc

jtzbT0xO3c9ODAw/http://media.zenfs.com/en-us/homerun/rss.gamesbeat/1da926697f5bbb3ffceaf3a610b7e213
167: Imagen MDP Helicóptero, https://image.slidesharecdn.com/howtotakeovertheworldwithartificialintelligencefinal-160402101616/95/how-to-take-over-the-world-with-artificial-intelligence-final-18-638.jpg?cb=1459592225
168: Wikipedia Markov decision process, https://en.wikipedia.org/wiki/Markov_decision_process
169: Wikipedia Markov property, https://en.wikipedia.org/wiki/Markov_property
170: Ruben Fiszel's website: Reinforcement Learning and DQN, learning to play from pixels, https://rubenfiszel.github.io/posts/rl4j/2016-08-24-Reinforcement-Learning-and-DQN.html
171: Josh Greaves Understanding the bellman equations, https://joshgreaves.com/reinforcement-learning/understanding-rl-the-bellman-equations/
172: Wikipedia Q-Learning, https://en.wikipedia.org/wiki/Q-learning
173: Medium freecodecamp, an introduction to Q--Learning: reinforcement learning, https://medium.freecodecamp.org/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc
174: Towards data science: introduction to various reinforcement learning algorithms. Part I (Q-Learning, SARSA, DQN, DDPG), https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287
175: Medium, Simple Reinforcement Learning with Tensorflow Part 4: Deep Q-Networks and Beyond,
176: Medium, RL - DQN Deep Q-Network, https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4
177: Introduction to Deep Q-network, Washington State university, https://eecs.wsu.edu/~taylorm/17_580/Yunshu_DQNpresentation_10102016.pdf
178: Machinelearningmstery, how to check-point deep learning models in keras, https://machinelearningmastery.com/check-point-deep-learning-models-keras/
179: Keras How can I save a keras model, https://keras.io/getting-started/faq/#how-can-i-save-a-keras-model
180: Stackoverflow Avaiable and used system memory in Python, https://stackoverflow.com/questions/11615591/available-and-used-system-memory-in-python
181: Github Yilundu DQN-DDQN Space Invaders, https://github.com/yilundu/DQN-DDQN-on-Space-Invaders/blob/master/space_invaders.py
182: Yilundu DQN-DDQN Space Invaders Replay Buffer, https://github.com/yilundu/DQN-DDQN-on-Space-Invaders/blob/master/replay_buffer.py
183: Github Yilundu DQN-DDQN Space Invaders Deep Q, https://github.com/yilundu/DQN-DDQN-on-Space-Invaders/blob/master/deep_Q.py
184: Github Yanpanlau Keras FlappyBird Q Learn, https://github.com/yanpanlau/Keras-FlappyBird/blob/master/qlearn.py
185: gsurma, github atari convolutional neural network,
186: Github OpenAI baselines, https://github.com/openai/baselines
187: Github awjuliani DeepRL-Agents, https://github.com/awjuliani/DeepRL-Agents
188: Github rohitgirdhar Deep Q Networks dqn Atari, https://github.com/rohitgirdhar/Deep-Q-Networks/blob/master/dqn_atari.py
189: Github Keras-RL examples dqn atari, https://github.com/keras-rl/keras-rl/blob/master/examples/dqn_atari.py
190: Github erilyth Universe-RL-Samples breakout-v0, https://github.com/erilyth/Universe-RL-Samples/blob/master/breakout-v0.py
191: Towardsdatascience, Reinforcement Learning w/Keras + OpenAi: DQNs, https://towardsdatascience.com/reinforcement-learning-w-keras-openai-dqns-1eed3a5338c

192: Becominghuman Beat Atari with Deep Reinforcement Learning! (Part1: DQN), https://becominghuman.ai/lets-build-an-atari-ai-part-1-dqn-df57e8ff3b26
193: Medium Reinforcement Learning w/Keras+OpenAI: The Basics, https://medium.com/@yashpatel_86510/reinforcement-learning-w-keras-openai-698add10b4eb
194: Deep Reinforcement Learning Course, https://simoninithomas.github.io/Deep_reinforcement_learning_Course/
195: Simonini Thomas github Deep Q Learning with Atari Space Invaders, https://github.com/simoninithomas/Deep_reinforcement_learning_Course/blob/master/Deep%20Q%20Learning/Space%20Invaders/DQN%20Atari%20Space%20Invaders.ipynb
196: Deep Q Learning with Tensorflow and Space Invaders (tutorial), https://www.youtube.com/watch?time_continue=2&v=gCJyVX98KJ4
197: Using Keras and Deep Q-Network to Play FlappyBird, https://yanpanlau.github.io/2016/07/10/FlappyBird-Keras.html
198: Beat Atari with Deep Reinforcement Learning! (Part 2: DQN improvements), https://becominghuman.ai/beat-atari-with-deep-reinforcement-learning-part-2-dqn-improvements-d3563f665a2c
199: Stackoverflow role of flatten in keras, https://stackoverflow.com/questions/43237124/role-of-flatten-in-keras
200: Datascience stackexchange why random sampling from replay for DQN?, https://datascience.stackexchange.com/questions/24921/why-random-sample-from-replay-for-dqn?rq=1
201: Seita`s Place, Frame Skipping and Pre-Processing for Deep Q-Networks on Atari 2600 Games, https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/
202: Tom Schaul, John Quan, Ioannis Antonoglou y David Silver, Prioritized Experience Replay, ,https://arxiv.org/pdf/1511.05952.pdf
203: Jaromiru web, lets make a dqn double learning and prioritized experience replay, https://jaromiru.com/2016/11/07/lets-make-a-dqn-double-learning-and-prioritized-experience-replay/
204: Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, David Silver, Rainbow: combining improvements in deep reinforcement learning, ,https://arxiv.org/pdf/1710.02298.pdf
205: Shehroze Bhatti, Alban Desmaison, Ondrej Miksik, Nantas Nardelli, N. Siddharth, Philip H. S. Torr, Playing Doom with SLAM-Augmented Deep Reinforcement Learning, ,https://arxiv.org/pdf/1612.00380.pdf
206: Captura videojuego Doom, http://vignette2.wikia.nocookie.net/doom/images/b/b3/Imp.png/revision/latest?cb=20050113171050
207: Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C), https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2
208: Quora what is hierachical reinforcement learning?, https://www.quora.com/What-is-hierachical-reinforcement-learning
209: Documentación instalación Anaconda linux, https://docs.anaconda.com/anaconda/install/linux/
210: Requisitos instalación tensorflow, https://www.tensorflow.org/install/#download-a-package
211: Como instalar ficheros .run en ubuntu, https://askubuntu.com/questions/18747/how-do-i-install-run-files
212: Anaconda instalación de tensorflow en anaconda, https://www.anaconda.com/tensorflow-in-anaconda/
213: Web de anaconda, instalación del paquete de cudatoolkit, https://anaconda.org/anaconda/cudatoolkit

214: Web de anaconda, instalación de cuDNN, https://anaconda.org/anaconda/cudnn
215: xxblx, como solucionar pantalla en negro de spyder 3 en linux, http://xxblx.blogspot.com/2016/10/black-screen-in-spyder3-ide-on-gnulinux.html
216: Web cuDNN, https://developer.nvidia.com/cudnn
217: Guía instalación cuda en linux, https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html
218: Solución problema compatibilidad cuDNN y tensorflow, https://devtalk.nvidia.com/default/topic/1043867/failed-to-get-convolution-algorithm-this-is-probably-because-cudnn-failed-to-initialize
219: FAQ de Google Colaboratory, https://research.google.com/colaboratory/faq.html
220: Medium Google Colab Free GPU Tutorial, https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d
221: Github explicacion v0, v4 OpenAI-Gym, https://github.com/openai/gym/issues/1280
222: OpenAI-Gym Installation, https://gym.openai.com/docs/#installation
223: Github OpenAI-Gym Installing Everything, https://github.com/openai/gym#installing-everything
224: Mujoco web, http://www.mujoco.org/index.html
225: OpenAI-Gym documentation, enviroments, https://gym.openai.com/docs/#environments