
GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

# ENTORNO DE DESARROLLO BASADO EN TÉCNICAS DE APRENDIZAJE PROFUNDO POR REFUERZO

Autor: Jaime Fúster de la Fuente

Director: Miguel Ángel Sanz Bobi

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

ENTORNO DE DESARROLLO BASADO EN TÉCNICAS DE APRENDIZAJE PROFUNDO POR REFUERZO

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2018/2019 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Jaime Fúster de la Fuente Fecha: 11/ 07/ 19

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Miguel Ángel Sanz Bobi Fecha: 12/ 07/ 2019

**AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO**

***1º. Declaración de la autoría y acreditación de la misma.***

El autor D. JAIME FÚSTER DE LA FUENTE

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: ENTORNO DE DESARROLLO BASADO EN TÉCNICAS DE APRENDIZAJE PROFUNDO POR REFUERZO que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

***2º. Objeto y fines de la cesión.***

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

***3º. Condiciones de la cesión y acceso***

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

***4º. Derechos del autor.***

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

***5º. Deberes del autor.***

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e

intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

***6º. Fines y funcionamiento del Repositorio Institucional.***

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 12 de Julio de 2019

**ACEPTA** JAIME FÜSTER DE LA FUENTE

Fdo.....

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:


GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

# ENTORNO DE DESARROLLO BASADO EN TÉCNICAS DE APRENDIZAJE PROFUNDO POR REFUERZO

Autor: Jaime Fúster de la Fuente

Director: Miguel Ángel Sanz Bobi

Madrid

# ENTORNO DE DESARROLLO BASADO EN TÉCNICAS DE APRENDIZAJE PROFUNDO POR REFUERZO

**Autor: Fúster de la Fuente, Jaime.**
Director: Sanz Bobi, Miguel Ángel.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

## RESUMEN DEL PROYECTO

Diseño e implementación de una plataforma de desarrollo basada en técnicas de Aprendizaje por Refuerzo que permita al usuario final plantear diferentes escenarios o problemas con diferentes entornos de aprendizaje y diferentes técnicas para aprender a vencer los mismos.

**Palabras clave**: Python, Inteligencia Artificial, Aprendizaje por Refuerzo, OpenAI Gym, Q-Learning, Deep Q-Learning, TensorFlow, Red Neuronal

### 1. Introducción

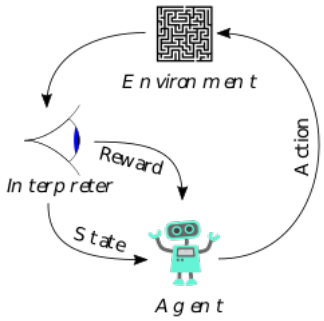



*Ilustración 1 – Encuadre típico de un escenario de RL*

El **Aprendizaje por Refuerzo**, también conocido como "*Reinforcement Learning*" o RL, es una técnica de **Aprendizaje Automático**, "*Machine Learning*" o ML, que busca que un agente aprenda a realizar una tarea de forma óptima en base a los estímulos y refuerzos obtenidos al realizar una serie de acciones durante el proceso de entrenamiento.

De esta manera, una aplicación inteligente puede desarrollar unas habilidades muy importantes para ganar un juego observando, en muchas jugadas de este, lo que ocurre cuando se ejecutan determinadas acciones, así como las recompensas obtenidas.

La motivación para el desarrollo del proyecto está fundamentada en la combinación de las diferentes tecnologías y técnicas disponibles para el análisis y entrenamiento de entornos en RL, con el fin de crear una plataforma que aúne distintas técnicas y metodologías de análisis de entornos para aprendizaje por refuerzo, destacando las diferentes características, así como deduciendo las ventajas e inconvenientes inherentes, y que surjan de su implementación.

## 2. Definición del proyecto

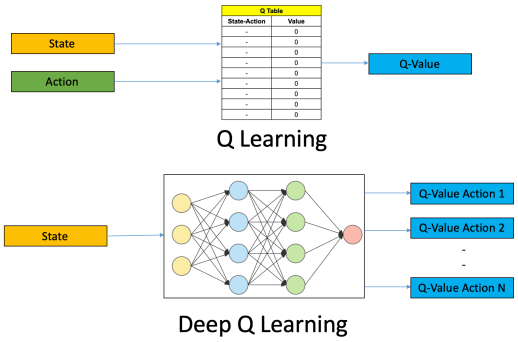



*Ilustración 2 – Q-Learning y Deep Q-Learning*

El objetivo de este trabajo es desarrollar una plataforma que permita plantear diferentes escenarios (juegos y problemas reales) con diferentes entornos de aprendizaje y diferentes técnicas para evaluar las mismas. La plataforma deberá ser intuitiva y presentar un conjunto variado de entornos y problemas que el usuario final pueda estudiar fácilmente.

Los entornos de desarrollo más sencillos se han analizado con *Q-Learning Simple*. Entonrnos más complejos, como los juegos ATARI, que deben tener en cuenta un amplio rango de acciones y estados posibles, han requerido del uso de una red neuronal para la implementación del algoritmo *Deep Q-Learning*.

Para el desarrollo del trabajo se ha empleado Python v3.6 como principal lenguaje de programación, haciendo uso de los recursos disponibles sobre aprendizaje por refuerzo, en especial de las librerías o *toolkits* OpenAI Gym [1], que proporciona las herramientas y entornos que se estudian y Tensorflow [2] [3] [4], para el diseño e implementación de redes neuronales.

## 3. Descripción y diseño de la Plataforma

Como objetivos principales a la hora de desarrollar el proyecto, se han buscado cumplir:

1. Interfaz intuitiva y fácil de ejecutar e instalar.
2. Disponibilidad de múltiples entornos, de carácter variado.
3. Ejecución de experimentos, con información detallada en tiempo real.

La plataforma diseñada ofrece al usuario la posibilidad de entrenar a un agente en los siguientes entornos de la librería OpenAI Gym:

**Entornos Classic Control**: problemas de control que comprenden desde entornos muy sencillos (juegos de texto con pocas variables) a entornos 2D y 3D más complejos:

- Taxi-v2
- FrozenLake-v0 y FrozenLake8x8-v0
- CartPole
- MountainCar

**Entornos de la consola Atari-2600**, sacados de la librería OpenAI Gym Retro [5] [6]:

- Space Invaders
- Ms. Pac-Man
- Breakout
- Pong

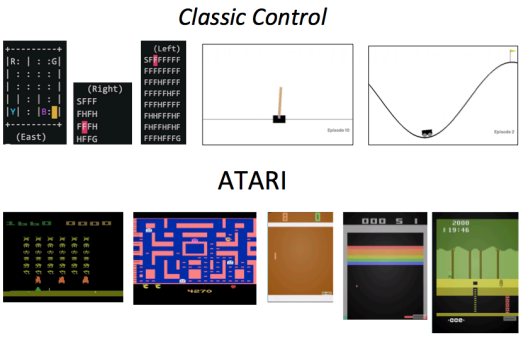



*Ilustración 2 – Conjunto de entornos implementados*

## 4. Resultados

La plataforma diseñada se muestra en la Ilustración 2. Se ha buscado la mayor claridad y fluidez de uso para el usuario final. El diseño dispone de varias pantallas por las que el usuario puede navegar entre los distintos menús de selección de entornos y características.

A través de la plataforma, el usuario puede elegir y entrenar un agente de RL en cualquiera los entornos que se han listado en el apartado anterior. La interfaz de usuario le permite:

1. Seleccionar el entorno, o juego en el que entrenar a un agente.
2. Editar los parámetros e hiperparámetros del entrenamiento:
    a. Número de episodios de entrenamiento y/o test.
    b. Valor de diversos hiperparámetros de entrenamiento en RL.
3. Recibir información útil en tiempo real sobre el proceso de entrenamiento.

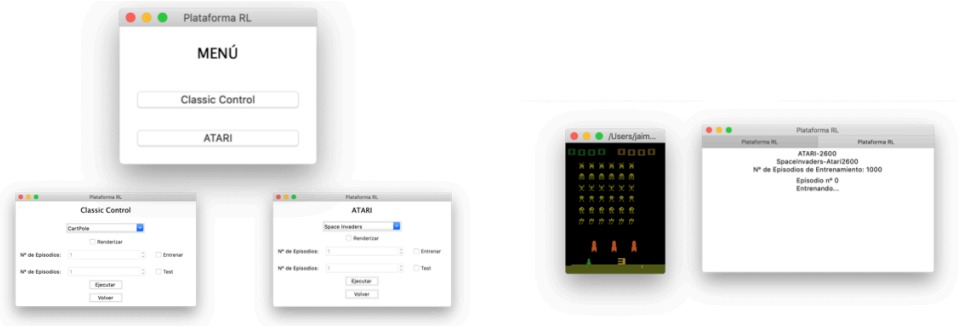



*Ilustración 3 – Ejemplo de ejecución de la plataforma*

## 5. Referencias


[1] OpenAI, «OpenAI Gym,» [En línea]. Available: https://gym.openai.com/.

[2] TensorFlow, «TensorFlow Guide - Graphs,» [En línea]. Available: https://www.tensorflow.org/guide/graphs.

[3] M. Abadi, «TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,» [En línea]. Available: https://arxiv.org/abs/1603.04467.

[4] Google LLC, «TensorFlow,» [En línea]. Available: https://www.tensorflow.org/.

[5] «Repositorio OpenAI Gym Retro,» [En línea]. Available: https://github.com/openai/retro/tree/develop.

[6] «OpenAI Gym Retro,» [En línea]. Available: https://openai.com/blog/gym-retro/.

# DEVELOPMENT ENVIRONMENT BASED ON DEEP REINFORCEMENT LEARNING TECHNIQUES

**Author: Fúster de la Fuente, Jaime.**
Supervisor: Sanz Bobi, Miguel Ángel.
Collaborating Entity: Universidad Pontificia Comillas

## ABSTRACT

Design and implementation of a development platform based on Reinforcement Learning techniques that allows the end user to approach different scenarios or problems with different learning environments and different techniques to learn how to beat them.

**Keywords**: Python, Artificial Inteligence, Reinforcement Learning, OpenAI Gym, Q-Learning, Deep Q-Learning, TensorFlow, Neural Network

## 1. Introduction

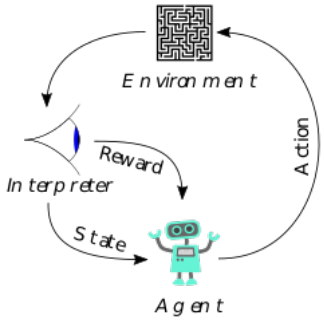


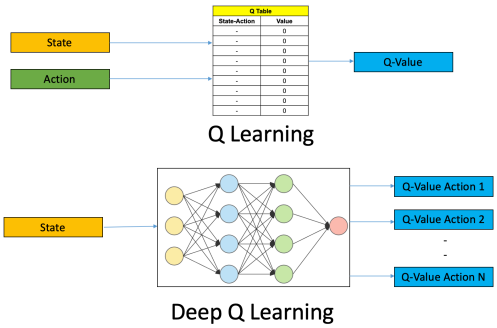

*Illustration 1 - Typical framing of a RL scenario*

Reinforcement Learning or RL, is a technique of Machine Learning or ML, that aims for an agent to learn to perform a task optimally based on the stimuli and reinforcements obtained by performing a series of actions during a training process.

In this way, an intelligent application can develop very important skills to win a game by observing, in many steps of the game, the rewards it obtains and what happens when certain actions are executed.

The motivation for the development of this project is based on the combination of the different technologies and techniques available for the analysis and training of environments in RL, in order to create a platform that brings together different techniques and methodologies of analysis of environments for Reinforcement Learning, highlighting the different characteristics, as well as deducing the advantages and disadvantages inherent, and arising from its implementation.

## 2. Project Definition




*Ilustration 2 – Q-Learning and Deep Q-Learning*

The aim of this work is to develop a platform that provides different scenarios (games and real-life problems) with different learning environments and different techniques to evaluate them. The platform should be intuitive and have a varied set of environments and problems that the end user can easily study.

The simplest development environments have been analyzed with "naive" Q-Learning. More complex environments, such as ATARI videogames, which must take into account a wide range of possible actions and states, have required the use of a neural network for the implementation of the Deep Q-Learning algorithm.

For the development of the platform, the main programming language employed was Python v3.6, in order to make use of the resources available on Reinforcement Learning, particularly the OpenAI Gym toolkit [1], which provided the tools and environments implemented and TensorFlow [2] [3] [4], for the design and implementation of the neural networks.

## 3. Description and Design of the Platform

The following objectives have been pursued when developing the project:

1. Intuitive interface and easy to execute and install.
2. Availability of multiple environments, of varied character.
3. Execution of experiments, with detailed information in real time.

The designed platform offers the user the possibility of training an agent in the following environments available in the OpenAI Gym library:

**Classic Control Environments**: control problems that range from very simple (text games with few variables) to more complex 2D and 3D environments:

- Taxi-v2
- FrozenLake-v0 y FrozenLake8x8-v0
- CartPole
- MountainCar

**Atari-2600 environments**, taken from the OpenAI Gym Retro library [5] [6]:

- Space Invaders
- Ms. Pac-Man
- Breakout
- Pong

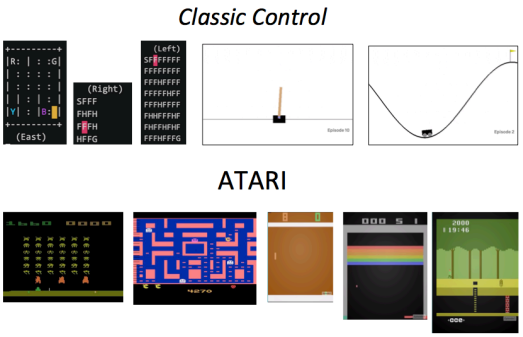



*Illustration 2 - Set of Implemented Environments*

## 4. Results

The platform designed is depicted in Illustration 2. Clarity and fluidity of use have been sought in favour of the end user. The design has several screens through which the user can navigate between the different selection menus of environments and characteristics.

Through the platform, the user can choose and train an RL agent in any of the environments listed in the previous section. The user interface allows him to:

1. Select the environment, or game in which to train an agent.
2. Edit training parameters and hyperparameters:
    a. Number of training and/or test episodes.
    b. Value of different training hyperparameters in RL.
3. Receive useful information in real time about the training process.


*Ilustration 4 - Example of the execution of the platform*

## 5. Referencias


[1] OpenAI, «OpenAI Gym,» [En línea]. Available: https://gym.openai.com/.

[2] TensorFlow, «TensorFlow Guide - Graphs,» [En línea]. Available: https://www.tensorflow.org/guide/graphs.

[3] M. Abadi, «TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,» [En línea]. Available: https://arxiv.org/abs/1603.04467.

[4] Google LLC, «TensorFlow,» [En línea]. Available: https://www.tensorflow.org/.

[5] «Repositorio OpenAI Gym Retro,» [En línea]. Available: https://github.com/openai/retro/tree/develop.

[6] «OpenAI Gym Retro,» [En línea]. Available: https://openai.com/blog/gym-retro/.

# *Índice de la memoria*

# *Índice de figuras*

# *Índice de cuadros*

# *Índice de tablas*

# Capítulo 1. INTRODUCCIÓN

El contenido de este capítulo presenta el contexto en que se ha desarrollado el proyecto e ilustrar los objetivos marcados, así como una breve introducción a los conceptos y herramientas que se han utilizado para su implementación.

## *1.1 MOTIVACIÓN DEL PROYECTO*

El **Aprendizaje por Refuerzo**, también conocido como "*Reinforcement Learning*" o RL, es una técnica de **Aprendizaje Automático**, "*Machine Learning*" o ML, que busca que un agente aprenda a realizar una tarea de forma óptima en base a los estímulos y refuerzos obtenidos al realizar una serie de acciones durante el proceso de entrenamiento.

De esta manera, una aplicación inteligente puede desarrollar unas habilidades muy importantes para ganar un juego observando, en muchas jugadas de este, lo que ocurre cuando se ejecutan determinadas acciones y las recompensas obtenidas. Lo mismo que para tomar las mejores decisiones estratégicas en un negocio viendo la evolución e interacción del mismo con su entorno.

La motivación para el desarrollo del proyecto está fundamentada en la combinación de las diferentes tecnologías y técnicas disponibles para el análisis y entrenamiento de entornos en RL (que se describirán en el presente informe), con el fin de crear una plataforma que aúne distintas técnicas y metodologías de análisis de entornos para aprendizaje por refuerzo, destacando las diferentes características, así como deduciendo las ventajas e inconvenientes inherentes, y que surjan de su implementación.

## *1.2 OBJETIVO*

El objetivo de este trabajo es desarrollar una plataforma que permita plantear diferentes escenarios (juegos y problemas reales) con diferentes entornos de aprendizaje y diferentes técnicas para evaluar las mismas. La plataforma deberá ser intuitiva y presentar un conjunto variado de entornos y problemas que el usuario final pueda estudiar fácilmente.

Para el desarrollo del trabajo se ha empleado Python v3.6 como principal lenguaje de programación, haciendo uso de los recursos disponibles sobre aprendizaje por refuerzo y otras técnicas de aprendizaje automático.

Al no requerir una gran potencia de computación, los entornos de desarrollo más sencillos se han analizado con *Q-Learning Simple*. Sin embargo, entornos más complejos, como los juegos ATARI, que deben tener en cuenta un amplio rango de acciones y estados posibles, han requerido del uso de una red neuronal para la implementación del algoritmo *Deep Q-Learning*.

En los capítulos siguientes se explicarán en mayor profundidad las distintas herramientas y algoritmos empleados.

### 1.2.1 OBJETIVOS DEL PROYECTO

A continuación, se presenta un listado de los principales objetivos a cumplir en el desarrollo del proyecto.

1. **Desarrollo de la plataforma**

   El proyecto busca desarrollar una plataforma con una interfaz intuitiva, mediante la cual el usuario final pueda analizar y escoger entre una variedad de entornos de problemas de aprendizaje por refuerzo. Debe ser fácil de ejecutar e instalar.

2. **Ejecución de experimentos**

   En la investigación de algoritmos de Aprendizaje por Refuerzo, la ejecución de experimentos es un elemento crucial, ya que permite poner a prueba los diferentes algoritmos en cada entorno. Así, variando los diferentes parámetros, se consigue comprobar sus efectos en el entrenamiento.

   Los usuarios finales de la plataforma deben invertir el tiempo mínimo para la preparación y ejecución de estos experimentos. Por ello, esta debe ser sencilla e intuitiva.

3. **Múltiples entornos**

   Deberá ser posible realizar experimentos en diferentes entornos, y el usuario deberá poder cambiar fácilmente entre ellos. Estos entornos deben por lo tanto ser variados, mostrando al usuario distintos enfoques y aplicaciones del aprendizaje por refuerzo.

4. **Análisis de resultados de los algoritmos y entornos implementados**

   Al ejecutar los algoritmos, el usuario debe ser consciente del efecto que el valor de cada hiperparámetro ejerce sobre el mismo. Los hiperparámetros son aquellos parámetros de los algoritmos de aprendizaje por refuerzo que modifican el proceso de aprendizaje y explotación del mismo.

   Uno de los objetivos de la experimentación será, por lo tanto, el análisis de los resultados obtenidos, y la búsqueda de los valores de los hiperparámetros que permitan al algoritmo aprender de manera efectiva.

# Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

En este capítulo se describen las tecnologías, algoritmos y herramientas, que se han utilizado a lo largo del proyecto.

## *2.1 APRENDIZAJE POR REFUERZO*

Como se ha explicado en la Introducción a este documento, el Aprendizaje por Refuerzo es un área del Aprendizaje Automático que busca que un agente aprenda a realizar una tarea de forma óptima en base a los estímulos y refuerzos obtenidos al realizar una serie de acciones durante el proceso de entrenamiento. Distingue entre tres elementos principales:

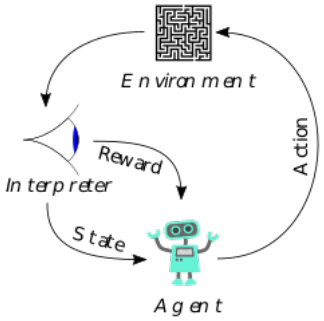



*Ilustración 1: Encuadre típico de un escenario de RL*

- **Entorno** del problema que se quiere resolver. Puede ser un modelo o incluso el problema real, en el que se pueden realizar acciones para interactuar con él.
- El **Intérprete**, que observa el entorno e interpreta el estado en que se encuentra, calculando una recompensa en base a lo cercano que se encuentra al objetivo.
- **Agente**, u objeto del entrenamiento. Recibe el estado y recompensa y decide las acciones a tomar en el entorno.

### 2.1.1 Q-LEARNING

*Q-Learning* es un algoritmo enfocado al Aprendizaje por Refuerzo. La "Q" en *Q-Learning* proviene de la palabra inglesa "quality", esto es porque, en esencia, *Q-Learning* estima la "calidad" de una acción determinada en un estado dado. Así, el agente escoge la próxima acción a tomar basándose en la calidad de la misma.

La función *Q-learning*, $Q(s_t, a_t)$ es una función que representa el valor de recompensa futura descontada de tomar la acción $a_t$ desde el estado $s_t$ en un instante *t*. En otras palabras, representa la mejor puntuación posible al final del juego tras realizar a acción $a_t$ en el estado $s_t$.

Esta función se calcula a partir de la ecuación de Bellman:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

*Ilustración 2: Ecuación de Bellman*

Donde $\boldsymbol{r_t}$ es la recompensa obtenida al pasar al estado $\boldsymbol{s_{t+1}}$ desde el estado actual $\boldsymbol{s_t}$ mediante la acción $\boldsymbol{a_t}$, $\boldsymbol{\alpha}$ es el "*learning rate*" o índice de entrenamiento ($0 < \alpha \leq 1$), $\boldsymbol{\gamma}$ es el factor de descuento ($0 < \gamma \leq 1$).

$\boldsymbol{\alpha}$ y $\boldsymbol{\gamma}$ son los denominados hiperparámetros. En Machine Learning, un hiperparámetro es un parámetro cuyo valor se establece antes de que el proceso de aprendizaje comience. Optimizando el valor de estos parámetros se consigue mejorar el aprendizaje del agente.

A partir de los valores obtenidos con la ecuación anterior durante cada iteración de entrenamiento, se va construyendo la tabla "Q-Learning", que es consultada en cada estado por el agente para elegir su próxima acción:

$$a_{t+1} = argmax(Q(s, a))$$

### 2.1.2 DEEP Q-LEARNING

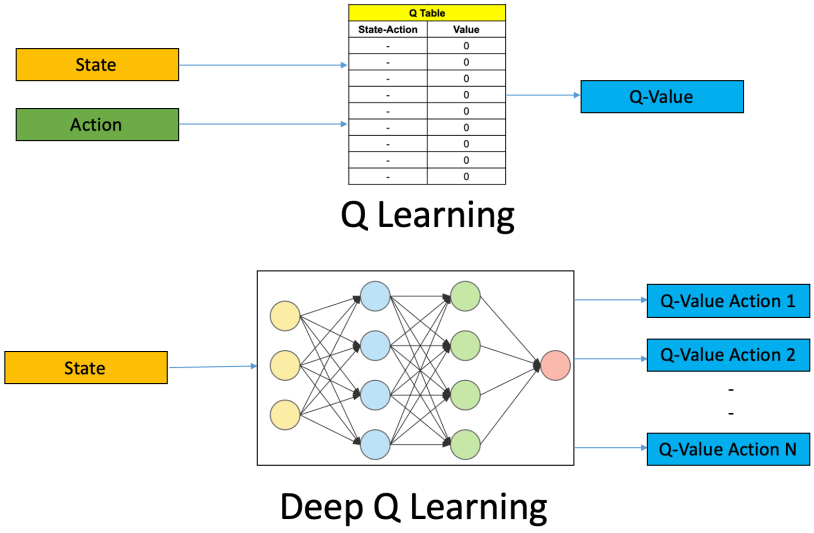



*Ilustración 3: Comparación entre Q-Learning y Deep Q-Learning [1]*

Cuando el problema o entorno tiene estados y acciones más complejos la potencia de procesamiento y tabla "Q-learning" requeridas son mucho mayores. Esto se soluciona mediante la implantación de redes neuronales o Deep Q-Networks (DQN). Para este proyecto, estas DQN se han implementado en el algoritmo de entrenamiento de los juegos Atari.

Deep Q-learning, utiliza una red neuronal para aproximar la función *Q-learning*. El estado se proporciona como entrada y el valor Q de todas las acciones posibles se genera como salida (ver la Ilustración 3 para una comparación entre *Deep Q-Learning* y *Q-Learning*). La red neuronal es, pues, el agente que aprende a asignar los pares acción-estado a recompensas.

Como todas las redes neuronales, la DQN utiliza coeficientes para aproximar la función que relaciona las entradas con las salidas, y su aprendizaje consiste en encontrar los coeficientes correctos, o pesos, ajustando iterativamente esos pesos a lo largo de gradientes que prometen menos errores.

En los entornos más complicados, como los juegos de la Atari-2600 estudiados en este proyecto, se puede emplear redes convolucionales para analizar la imagen del juego y reconocer el estado de un agente [2]; por ejemplo, en *"Space Invaders"*, cómo de cerca están los enemigos, número de vidas, o cercanía de un proyectil al agente.

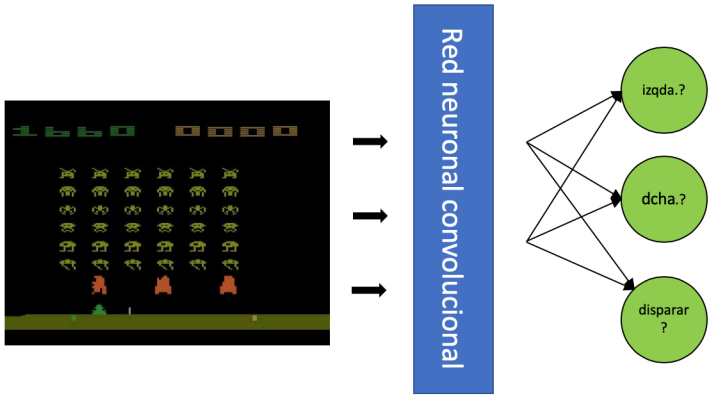



*Ilustración 4: Redes Convolucionales y Deep Q-Learning*

Así, tal y como se observa en la Ilustración 4, dada una imagen que representa un estado, una red convolucional puede clasificar las acciones posibles en ese estado. Por ejemplo, puede predecir que desplazarse hacia la derecha devolverá 5 puntos, saltar 7 y correr hacia la izquierda ninguno.

## *2.2 PYTHON V3.6*

Con el fin de aprovechar el amplio rango de librerías y toolkits que existen destinados a facilitar el estudio y desarrollo de entornos de Inteligencia Artificial y Reinforcement Learning, la programación de la plataforma se ha realizado en Python [3], en su versión más actualizada (3.6).

El uso de Python como lenguaje de programación es cada vez más común para sistemas aplicados a la Inteligencia Artificial. Esto es debido a un conjunto de cualidades que lo hacen idóneo para este tipo de entornos. Entre estas, destacan como principales:

1. Legibilidad de código
2. Multiplataforma
3. Orientado a objetos
4. Comunidad extensa y activa
5. Amplio catálogo y accesibilidad a librerías y toolkits

En primer lugar, la filosofía de diseño detrás de Python hace énfasis en la **legibilidad de código** y simpleza sintáctica. Esto hace que sea óptimo para la programación e implementación de algoritmos complejos.

Por otro lado, Python es un lenguaje **multiplataforma**, ya que dispone de implementaciones en la mayoría de los sistemas operativos. Esto aporta una gran flexibilidad en el desarrollo y exportación a otros entornos. Además, al ser **orientado a objetos** y tener una sintaxis simple, Python permite el desarrollo de aplicaciones de RL con facilidad.

Por otra parte, el **gran número de usuarios activos** hace que se disponga de extensa documentación sobre el lenguaje, lo que favorece la resolución de problemas y dudas.

Por último, la principal ventaja de Python es el **amplio catálogo de librerías y toolkits** que ofrece.

- OpenAI Gym: Toolkit para la explotación de entornos de Aprendizaje por Refuerzo que se explica en detalle en la subsección siguiente (2.3).
- TensorFlow: Librería para el diseño y explotación de redes neuronales. (ver subsección (2.5)
- NumPy: Librería nativa de Python para el cálculo vectorial y matricial.

- Matplotlib: Librería para la generación y análisis de gráficos. Matplotlib es una extensión de NumPy.
- Tkinter: Binding de la librería gráfica Tcl/Tk. Se considera un estándar para el diseño de GUIs en Python [4].

Como IDE, se ha escogido PyCharm v.2019.1.2, por su carácter open-source y facilidad de uso [5].

## 2.3 OPENAI GYM

OpenAI Gym [6] es un toolkit que ofrece un amplio conjunto de herramientas para el desarrollo e investigación del aprendizaje por refuerzo. Aporta una creciente colección de entornos que pueden emplearse fácilmente para el desarrollo de algoritmos de Deep Reinforcement Learning.

Consiste en una interfaz open-source que facilita la ejecución de tareas de aprendizaje por refuerzo mediante la estandarización de la definición de los distintos entornos.

La lista de entornos disponible actualmente es muy completa y de complejidad muy variada. Los entornos estudiados en este proyecto se explican a continuación.

### 2.3.1 CLASSIC CONTROL

Los entornos de la categoría "Classic Control" son problemas de control que comprenden desde entornos muy sencillos (juegos de texto con pocas variables) a entornos 2D y 3D más complejos.

### *2.3.1.1 Taxi-v2*

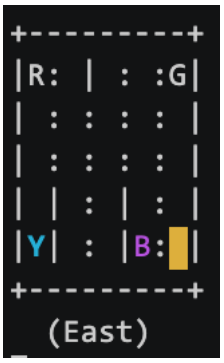



*Ilustración 5: Taxi-v2*

*Taxi-v2* es juego de texto sencillo que consiste en controlar un taxi para recoger a un pasajero y llevarlo a su destino. Hay 4 ubicaciones (etiquetadas con letras diferentes), el jugador recibe +20 puntos por una devolución exitosa y pierde 1 punto por cada paso de tiempo que toma. También hay una penalización de 10 puntos por las acciones ilegales de recogida y devolución.

### *2.3.1.2 FrozenLake-v0 y FrozenLake8x8-v0*


*Ilustración 6: FrozenLake-v0 (izqda.) y FrozenLake8x8-v0 (dcha.)*

Juego sencillo de texto en el que el agente controla el movimiento de un personaje en un lago congelado, representado con una cuadricula. Existen tres tipos de baldosas en la cuadrícula:

1. Casilla de salida: marcada con una *"S"*
2. Casilla de meta: marcada con una *"G"*
3. Baldosas "congeladas": marcadas con la letra *"F"*
4. Baldosas "agujero": representadas con la letra *"H"*

Tanto las baldosas de salida *"S"*, y meta *"G"*, como las marcadas con una *"F"* son transitables. Las casillas representadas con la letra *"H"* hacen que el agente caiga al agua.

Al tratarse de un lago congelado, la dirección del movimiento del agente es incierta y solo depende parcialmente de la dirección elegida. El agente es recompensado por encontrar un camino transitable a una casilla de meta.

La diferencia entre FrozenLake y FrozenLake8x8 es la dimensión de la cuadrícula (3x4 para FrozenLake y 8x8 para FrozenLake8x8).

#### *2.3.1.3 CartPole*

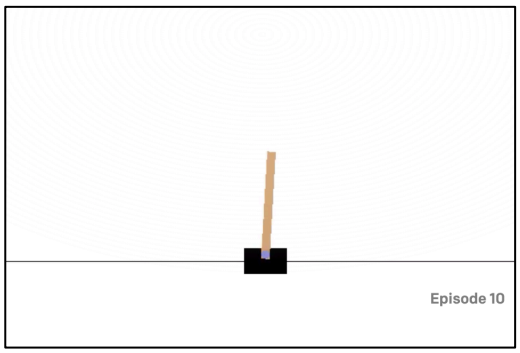



*Ilustración 7: Cartpole-v1*

En *CartPole*, un poste unido mediante una articulación no accionada a un carrito, que se mueve a lo largo de una pista sin fricción. El sistema es controlado aplicando una fuerza de +1 o -1 al carrito. La articulación comienza en posición vertical, y el objetivo es evitar que se caiga.

Una recompensa de +1 es proporcionada por cada paso de tiempo que el poste permanece en posición vertical. El episodio termina cuando el poste está a más de 12 grados del eje vertical, o el carro se mueve más de 2.4 unidades desde el centro.

#### *2.3.1.4 MountainCar*

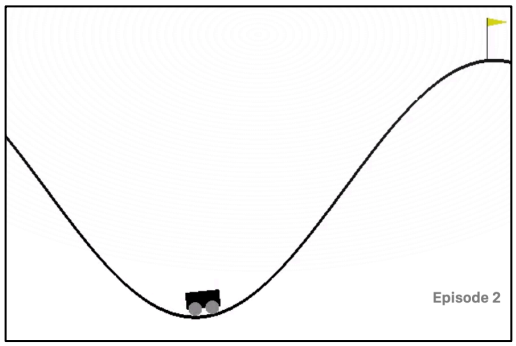



*Ilustración 8: MountaincCar-v0*

En *MountainCar*, un automóvil se sitúa en una pista unidimensional, colocada entre dos colinas.

El objetivo es subir la colina de la derecha. El motor del automóvil no es lo suficientemente potente como para escalar la montaña en una sola pasada por lo que la única manera de tener éxito es conducir hacia adelante y hacia atrás, y ganar momento para aumentar el impulso.

## *2.4 OPENAI RETRO*

OpenAI Gym Retro [7] [8] es una librería derivada de OpenAI Gym, que se utiliza para el análisis de videojuegos "retro" de un amplio catálogo de consolas que incluye Sega Genesis, Atari o NES, entre muchas otras.

En este proyecto se estudian varios entornos de la consola Atari 2600, en concreto los que se presentan a continuación.

### 2.4.1 SPACE INVADERS

*Ilustración 9: Space Invaders - Atari 2600*

Clásico juego de la consola Atari2600 en el que el jugador controla un cañón que puede disparar hacia arriba y moverse de izquierda a derecha. El objetivo del juego es destruir a los extraterrestres enemigos, que se acercan cada vez más rápido al jugador a medida que este los elimina, y maximizar la puntuación.

En este entorno, la observación es una imagen RGB de la pantalla que es una matriz de forma (210, 160, 3). Cada acción se realiza repetidamente durante una duración de cuadros *kk*, donde *kk* se muestrea uniformemente a partir de {2, 3, 4}{2,3,4}.

### 2.4.2 MS. PAC-MAN


*Ilustración 10: Ms. Pac-Man - Atari 2600*

En este mítico juego de arcade, el jugador debe ir comiendo puntos a la vez que esquiva a fantasmas (el contacto con uno hace que Ms. Pac-Man pierda una vida). Existen puntos

especiales, denominados *"power-pellets"*, que hacen que los fantasmas se vuelvan azules y les permite ser comidos a cambio de puntos extra. A medida que el número de rondas aumenta, la velocidad de los enemigos también, y el efecto de los *"power-pellets"* generalmente disminuye. El principal objetivo del agente, por tanto, es maximizar la puntuación obtenida en el menor tiempo posible.

Al igual que en "*Space Invaders"*, la observación consiste en una imagen RGB de la pantalla, de forma (210, 160, 3).

### 2.4.3 PONG

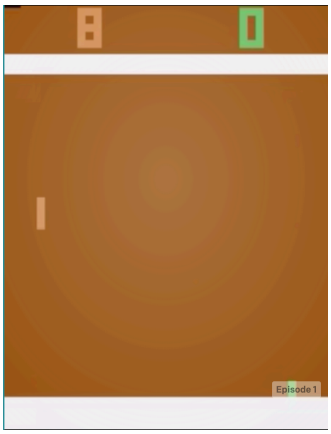


*Ilustración 11: Pong - Atari 2600*

El agente controla una pala, moviéndola verticalmente a través del lado izquierdo o derecho de la pantalla. El objetivo es ganar al oponente, a la vez que marcar el máximo de puntos posible.

Al igual que en "*Space Invaders" y "Ms. Pac-Man"*, la observación consiste en una imagen RGB de la pantalla, de forma (210, 160, 3).

### 2.4.4 BREAKOUT


*Ilustración 12: Breakout - Atari 2600*

En *"Breakout",* el agente debe maximizar la puntuación obtenida mientras intenta romper una pared de ladrillos con una pelota, sin perder esta en su rebote.

El juego comienza con ocho filas de ladrillos, con cada dos filas de un color diferente (amarillo, verde, naranja y rojo). El jugador tiene tres turnos para intentar despejar dos pantallas de ladrillos. Los ladrillos amarillos aportan un punto cada uno, los verdes tres puntos, los ladrillos naranjas dan cinco puntos y los rojos siete puntos cada uno.

La paleta que controla el agente se encoge hasta la mitad de su tamaño después de que la pelota haya atravesado la fila roja y golpeado la pared superior. La velocidad de la bola aumenta a intervalos específicos: después de cuatro golpes, después de doce golpes y después de hacer contacto con las filas naranja y roja.

## 2.5 TENSORFLOW

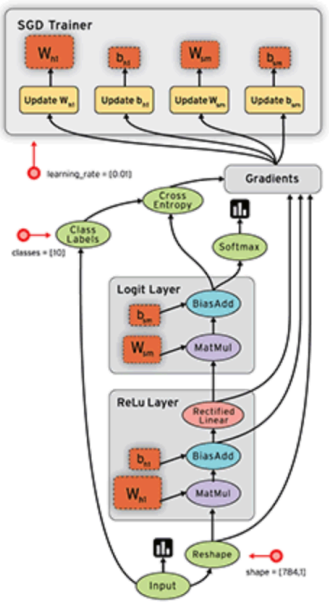



*Ilustración 13: Ejemplo grafo de una Redn Neuronal en TensorFlow [9]*

La librería open-source TensorFlow [10] [11] permite el diseño explotación e implementación de redes neuronales. La implementación de estas redes neuronales está basada en un sistema de grafos definidos mediante operaciones matemáticas, que se aplican a tensores de múltiples dimensiones (ver Ilustración 13 para un esquema de ejemplo detallado).

### 2.5.1 ESTRUCTURA BÁSICA DE UN PROGRAMA TENSORFLOW

Se puede considerar que la estructura de los programas TensorFlow tiene dos secciones o procesos discretos:

1. Construcción del grafo computacional (**tf.Graph**).
2. Ejecución del grafo computacional (**tf.Session**)

En la presente subsección se pretende ahondar en la estructura de ambos apartados, con el fin de aclarar para el lector la estructura básica de un programa TensorFlow.

## *Grafos (tf.Graph)*

Como se ha comentado al inicio de esta sección, en TensorFlow, los grafos computacionales corresponden a una serie de operaciones matemáticas de TensorFlow distribuidas en un grafo. Este grafo está compuesto por dos tipos de objetos:

- **tf.Operation** (o "ops")
- **tf.Tensor**

Las operaciones (**tf.Operation** ) describen cálculos que consumen y producen tensores, y corresponden a los nodos del grafo.

Los tensores (**tf.Tensor**) corresponden a los bordes del grafo. La mayoría de funciones TensorFlow devuelven tf.Tensors, y estos representan los valores que fluirán a través del grafo.

Es importante denotar que los tf.Tensor no tienen valores de por sí, ya que tan sólo sirven de representantes, o *"handles"* de los elementos en el grafo computacional.

## *Sesión (tf.Session)*

tf.Session se utiliza para evaluar tensores. Para ello, el usuario instancia un objeto tf.Session, informalmente conocido como sesión, que encapsula el estado del tiempo de ejecución de TensorFlow y ejecuta las operaciones (tf.Operation).

A modo de ilustración para el lector, la guía oficial de TensorFlow [12] propone la siguiente analogía:

*"Si un grafo tf.Graph es como un archivo de Python ".py", la sesión tf.Session corresponde su ejecutable."* [12]

### 2.5.2 CARACTERÍSTICAS

Las principales características que están consolidando a TensorFlow como estándar para el desarrollo de aplicaciones de Aprendizaje por Refuerzo y que hacen que sea idóneo para las mismas son las siguientes [13]:

- **Open-source y gratuita**, lo que facilita su aplicación y distribución a diferentes sistemas.
- **Número de usuarios** y **documentación disponible**. Aparte de la extensa documentación compartida por su amplia comunidad de usuarios activa, TensorFlow ofrece multitud de tutoriales y casos de estudio para el aprendizaje de sus usuarios y resolución de dudas.
- **Facilidad de uso y modelado**. TensorFlow ofrece múltiples niveles de abstracción para que el usuario final pueda elegir el que mejor se adapte a sus necesidades.
- **Potencia y eficiencia en procesamiento**. TensorFlow permite al usuario construir y entrenar modelos de última generación sin sacrificar velocidad ni rendimiento. Prestaciones como la API "*Functional Keras"* y la "*Model Subclassing API"* para la creación de topologías complejas dan al usuario flexibilidad y control sobre sus modelos.

  Además, proporciona soporte de características avanzadas como utilización de múltiples GPU o ejecución distribuida.
- **Soporte robusto en múltiples sistemas operativos y plataformas**, como entornos móviles (con la librería *TensorFlow Lite*) o Javascript (con *TensorFlow.js*).

En el proyecto se ha utilizado para la implementación del algoritmo Deep Q-Learning, necesario para el entrenamiento de agentes en entornos complejos como Atari o CartPole.

# Capítulo 3. PLATAFORMA

En el presente capítulo se presentan los siguientes aspectos de la plataforma software desarrollada:

1. Una guía de instalación para el correcto funcionamiento de la plataforma.
2. Descripción detallada del proceso de diseño e implementación de la plataforma.
3. Manual de usuario de la plataforma.

Todos estos aspectos se describen en detalle en las secciones a continuación.

## *3.1 GUÍA DE INSTALACIÓN*

Una vez descargada la carpeta del proyecto, el correcto funcionamiento de la plataforma diseñada precisa los siguientes requerimientos de instalación básicos:

- Python v.3.6 o superior [3]
- OpenAI Gym [6]
- TensorFlow [11]
- Scikit-image [14]

A continuación, se indica cómo instalar cada paquete.

### 3.1.1 PYTHON

Para instalar la última versión de Python, visite la página web oficial: https://www.python.org/downloads/, y siga las instrucciones allí indicadas.

### 3.1.2 INSTALACIÓN DE LIBRERÍAS

Desde la línea de comandos:

#### *3.1.2.1 OpenAI Gym*

Para instalar Gym, simplemente introduzca:

```
pip install gym
```

#### *3.1.2.2 TensorFlow*

Para instalar la última versión de TensorFlow escriba el comando:

```
pip install tensorflow
```

#### *3.1.2.3 Scikit-image*

Por último, para la instalación de la librería de procesamiento de imágenes "scikit-image":

```
pip install scikit-image
```

### 3.1.3 ENTORNOS ATARI-2600

Para la instalación de los entornos Atari-2600 utilizados en la librería OpenAI Gym Retro, sitúese en la carpeta del proyecto e introduzca el siguiente comando:

```
python -m retro.import .
```

Para que el comando funcione, es importante no excluir el espacio entre `retro.import` y el último punto.

## 3.2 DISEÑO E IMPLEMENTACIÓN

En esta sección se describe el conjunto de métodos y soluciones que se han utilizado en el diseño de la plataforma e implementación de los distintos algoritmos.

Se divide en tres subsecciones las que se desarrollan los principales procesos llevados a cabo en diseño de la plataforma:

1. La interfaz gráfica de usuario (*Graphical User Interface* o *GUI*).
2. Implementación de los algoritmos, donde se distinguen:
    a. Implementación en entornos Classic Control
    b. Implementación en entornos Atari-2600

En la primera subsección se describe la estructura de la interfaz gráfica de usuario, así como detalles sobre su diseño y aplicación.

La segunda subsección pretende presentar al lector las características y particularidades inherentes a cada tipo de entorno, con el fin de ahondar en la justificación y explicación del proceso de diseño y planteamiento de los distintos enfoques aplicados.

### 3.2.1 INTERFAZ GRÁFICA DE USUARIO (GUI)

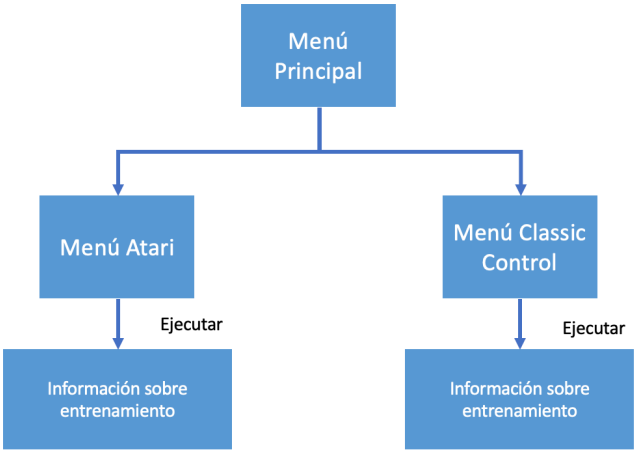



*Ilustración 14: Diagrama de Navegación*

La interfaz de usuario (*Graphical User Interface* o *GUI*) del proyecto se ha diseñado y programado con Tkinter, un binding de la librería gráfica Tcl/Tk nativo de Python v.3.6 [4].

Se ha buscado la mayor claridad y fluidez de uso para el usuario final. El diseño dispone de varias pantallas por las que el usuario puede navegar entre los distintos menús de selección de entornos y características.

La Ilustración 14 muestra un diagrama de navegación de la plataforma simple a modo de ilustración para el lector. Como podrá ver, el programa está compuesto por las siguientes pantallas:

1. Menú Principal
2. Menús de selección de entornos. Los cuales:
    a. Selección de entornos Atari.
    b. Selección de entornos Classic Control.

El Menú principal proporciona acceso al resto de pantallas mediante diversos botones. En las pantallas, o menús de selección de entornos, se proporciona al usuario las herramientas y campos necesarios para customizar el proceso de entrenamiento de un agente, cuya información se muestra en las pantallas de "Información Sobre Entrenamiento".

A lo largo de esta sección se explicará brevemente el funcionamiento y composición de cada pantalla.

### *3.2.1.1 Menú Principal*

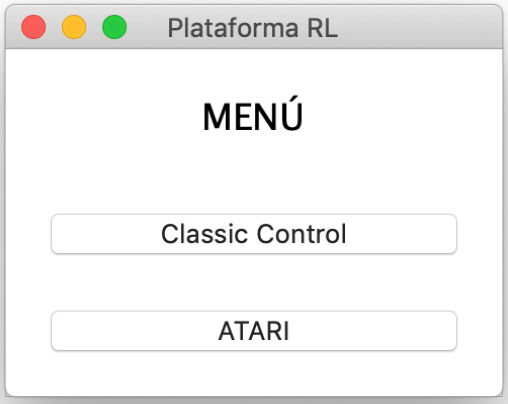



*Ilustración 15: Menú Principal*

La Ilustración 15 muestra la estructura de la primera pantalla, el Menú Principal, que como el lector podrá apreciar, es deliberadamente sencilla.

Se ha buscado cumplir dos principales cualidades en su diseño:

1. Inmediatez en la comprensión de su funcionamiento.
2. Fácil acceso a las funcionalidades principales.

La inmediatez del diseño desde la primera pantalla es esencial para facilitar una interfaz intuitiva y fácil de entender. El usuario debe ser capaz de comprender la estructura y funcionamiento básico de la plataforma desde el instante en que abre la aplicación, y sin ningún tipo de ayuda externa.

Así mismo, se ha considerado crucial que el usuario final sea capaz de llegar a las funcionalidades principales con el mínimo número de obstáculos en el camino, lo que justifica la maquetación y implementación minimalista de esta pantalla.

### *3.2.1.2 Menús de selección de entornos Classic Control y ATARI*


*Ilustración 16: Menús Classic Control y ATARI*

Como muestra la Ilustración 16, el diseño de los menús de selección de los entornos es idéntico para cada categoría. Se ha optado por la separación entre ambos tipos de entornos con el fin de facilitar la incorporación de nuevas características o funcionalidades particulares a cada entorno que se discutirán en la conclusión a esta memoria.

Cuando el usuario elija los parámetros que desea aplicar y el entorno que quiera entrenar, pulsará el botón “Ejecutar”, que, como se ha indicado en la Ilustración 14, le llevará a la ventana de “Información sobre el entrenamiento”. Aparecerá también una renderización del entorno si el usuario ha marcado la casilla “Renderizar”. Un ejemplo de este caso se representa en la Ilustración 17.


*Ilustración 17: Pantalla de "Información de Entrenamiento"*

La sección 3.3, en este capítulo explica de forma detallada el modo de uso de la plataforma, y funcionalidades de la misma, por lo que esta subsección no ahondará en su funcionamiento.

## IMPLEMENTACIÓN DE LOS ALGORITMOS

### *3.2.1.3 Classic Control*

En la presente sección se describen los criterios, así como metodologías aplicadas al diseñar e implementar los distintos algoritmos empleados en los entornos *Classic Control* (2.3.1). Estos incluyen juegos de texto como *"Taxi-v2"* o *"FrozenLake"*, así como entornos y problemas más complejos como *"MountainCar"* o *"CartPole.*

La disparidad y variedad en la complejidad de estos tipos de problemas requiere la utilización de implementaciones diferentes, que se detallan y justifican en las subsecciones a continuación.

### *3.2.1.3.1 Q-Learning – Implementación básica*

El procedimiento básico de implementación del algoritmo Q-Learning se explica a continuación. La sencillez de esta implementación es óptima para entornos menos complicados como *"Taxi-v2"* o *"FrozenLake"*. El Cuadro 1 y el Cuadro 2, presentan el código en Python.

El Cuadro 1 muestra un ejemplo de implementación de Q-Learning en el entorno *"Taxi-v2"*. Como el lector podrá apreciar, el primer paso consiste en instanciar el entorno, o *"environment"* mediante el comando `env = gym.make('Taxi-v2')`.

```
env = gym.make('Taxi-v2')
agente = Agente(env)

print("###### APRENDIENDO #####")
reward_total = 0.0
for i in range(ql.EPISODIOS_APRENDIZAJE):
   agente.aprender()
   if i % int(ql.EPISODIOS_APRENDIZAJE * 0.1) == 0:
      print("*", end=" ")
print("[DONE]")
print("N° Episodios      : {}".format(ql.EPISODIOS_APRENDIZAJE))
print("Tabla Q       :{}".format(agente.qtable))

print("###### TEST #####")
reward_total = 0.0
for i in range(ql.EPISODIOS_TEST):
   if (i + 1) == ql.EPISODIOS_TEST:
      reward_total += agente.test(True)
   else:
      reward_total += agente.test()
   if i % int(ql.EPISODIOS_APRENDIZAJE * 0.1) == 0:
      print("*", end=" ")
print("[DONE]")
print("N° Episodios      : {}".format(ql.EPISODIOS_TEST))
print("Premio Total  : {}".format(reward_total))
print("Media Premio: {:.2f}".format(reward_total / ql.EPISODIOS_TEST))
env.close()
```

*Cuadro 1: Ejemplo de implementación del algoritmo Q-Learning en Python*

A continuación, con el entorno creado, se construye una instancia de la clase *Agente* (ver Cuadro 2), que contiene como atributos el entorno y la tabla Q, y los siguientes métodos responsables de la implementación del algoritmo:

- `aprender(self)`: cuya función es ejercer el entrenamiento del agente durante 1 episodio. Como se puede apreciar en el Cuadro 2:
    1. Con `state = self.env.reset()` se obtiene el estado inicial.
    2. El entrenamiento se realiza en un bucle for, donde se avanza de estado en estado y que dura hasta el número máximo de pasos determinados por el usuario o hasta que el entorno devuelve `done = True` al calcular un nuevo estado mediante el comando:

```
sig_state, reward, done, info = self.env.step(action)
```

Por cada estado se calcula el valor Q aplicando la ecuación de Bellman (Ilustración 2) y se actualiza la tabla Q:

```
self.qtable[state, action] = (1-ALFA)*self.qtable[state,action] +
        ALFA*(reward + GAMMA*np.max(self.qtable[sig_state]))
```

Si el entorno devuelve `done = True`, el episodio ha finalizado y el método devuelve la recompensa obtenida.

En caso contrario, el bucle continúa, y el agente pasa al siguiente estado (`state = sig_state`), obtenido al ejecutar una nueva acción.

- `test(self, render = False)`: para probar el agente ya entrenado. Si se desea renderizar el resultado, basta con llamar al método con el argumento opcional `render = True`.

  El procedimiento es básicamente el mismo que el llevado a cabo en el método anterior, pero en este caso no se actualiza la tabla Q, ya que la utilizamos como guía para saber qué acción tomar, es decir:

```
action = np.argmax(self.qtable[state])
```

La acción por realizar se elige dependiendo de cuál es el valor máximo de la tabla Q para el estado actual.

```
class Agente:

   def __init__(self, env):
      self.env = env
      self.qtable = np.zeros((self.env.observation_space.n,
self.env.action_space.n))

   def aprender(self):
      # 1 episodio

      state = self.env.reset()

      for step in range(MAX_STEPS):
         action = self.env.action space.sample()
         sig state, reward, done, info = self.env.step(action)
         # actualizo tabla Q con la ecuacion de Bellman
         self.qtable[state, action] = (1-ALFA)*self.qtable[state,action]
+ ALFA * (reward + GAMMA *
            np.max(self.qtable[sig_state]))
         if done:
            return reward
         else:
            state = sig_state

   def test(self, render = False):
      state = self.env.reset()
      for step in range(MAX_STEPS):
         action = np.argmax(self.qtable[state])
         if render:
            self.env.render()
         sig state, reward, done, info = self.env.step(action)
         if done:
            if render:
               print(reward)
            return reward
         else:
            state = sig_state
```

*Cuadro 2: Clase Agente – Implementación de Q-Learning en Python*

#### *3.2.1.3.2 Q-Learning – Implementación Avanzada*

Entornos como *"MountainCar"* o *"CartPole"* necesitan una aplicación del algoritmo Q-Learning más avanzada. Esto se debe a que, al funcionar en tiempo real, el número de estados y pasos obtenidos en su ejecución es más complejo.

Así, en esta sección se explicará la implementación de Q-Learning para *"MountainCar"* y *"CartPole"* en dos procesos principales:

1. Una etapa de preprocesamiento.
2. La implementación del propio algoritmo Q-Learning.

En la etapa de preprocesamiento, la principal medida será la **discretización del espacio de observaciones**, esto es, del conjunto de estados y acciones obtenido en cada instante del transcurso del entrenamiento (ver sección 2.1.1).

Por otro lado, durante la descripción de la implementación de Q-Learning, se describirán las diferencias con respecto a la implementación de carácter más sencillo presentada en la sección anterior, además de los métodos y técnicas empleados en cada entorno. En caso de que dichos métodos que se aplican difieran entre los entornos, también se describirán las diferencias.

### *Preprocesamiento: Discretización de Observaciones*

| *Nº* | **Observación** | **Min** | **Máx** |
|---|---|---|---|
| *0* | Posición del Carro | -1.2 | 0.6 |
| *1* | Velocidad del Carro | -0.07 | 0.07 |

*Tabla 1: Estructura de una observación en "MountainCar" [6]*

Como viene representado en la Tabla 1, en el caso de ***"MountainCar"***, el espacio de observaciones ("*observation space*" en inglés) está formado por:

1. La posición del carro.
2. La velocidad del carro.

Como muestra la Tabla 1, el valor atribuido a la posición del carro en un instante determinado está comprendida en el rango [-1.2, 0.6]. Del mismo modo, la velocidad de este pertenece al rango [-0.07, 0.07].

Estamos, por tanto, en un espacio de observaciones continuo, donde existen muchos pares estado-acción, por lo que es necesario discretizar el espacio de alguna manera. El procedimiento escogido consiste en disminuir el número de estados a un número, `n_states`, que se ha fijado a 40.

Se ha creado así, el método `obs_to_state(self,env,obs)`, que recibe como argumentos el entorno (`env`), y la observación (`obs`), obtenida al pasar a un nuevo estado (`obs, reward, done = self.env.step(action)`).

Este método se presenta en el Cuadro 3, donde, como el lector podrá observar, se hace uso de los atributos del entorno (`env`) `observation_space.low`, y `observation_space.high` (definidos en la librería gym [6]), que aportan los límites inferior y superior respectivamente del espacio de observación. Si volvemos a la Tabla 1, el lector verá que:

```
env_low = env.observation_space.low = [-1.2, -0.07]
```

y,

```
env_high = env.observation_space.high = [0.6, 0.07]
```

Así, para discretizar el número de estados posibles a un número fijo, `n_states`, basta con calcular el nuevo estado $s = (a, b)$ a partir de la observación actual `obs`:

```
a = int((obs[0] - env_low[0])/env_dx[0])
```

```
b = int((obs[1] - env_low[1])/env_dx[1])
```

Donde `obs[0]` es la posición del carro en el instante actual, `obs[1]` es su velocidad, y `env_dx` es el factor calculado como:

$$env_{dx} = \frac{env_{high} - env_{low}}{n_{states}}$$

```
def obs_to_state(self,env, obs):
    """ Convierte observación a estado """
    env_low = self.env.observation_space.low
    env_high = self.env.observation_space.high
    env_dx = (env_high - env_low) / n_states
    a = int((obs[0] - env_low[0])/env_dx[0])
    b = int((obs[1] - env_low[1])/env_dx[1])
    return a, b
```

*Cuadro 3: Q-Learning en MountainCar - obs_to_state*

| *Nº* | **Observación** | **Min** | **Máx** |
|---|---|---|---|
| *0* | Posición del Carro | -2.4 | 2.4 |
| *1* | Velocidad del Carro | -Infinito | Infinito |
| *2* | Ángulo del Palo | -41.8 º | 41.8º |
| *3* | Velocidad del Palo en la punta | -Infinito | Infinito |

*Tabla 2: Estructura de una observación en "CartPole" [6]*

La técnica utilizada en ***"CartPole"*** (Cuadro 4), es similar a la de *"MountainCar"*, pero difiere en que es algo más compleja. Se ha optado por implementar métodos diferentes debido a que la estructura de las observaciones y acciones en cada entorno es diferente.

Tal y como se muestra en las TablasTabla 1 yTabla 2, las observaciones en *"CartPole"* son más detalladas que las de *"MountainCar"*, ya que *"CartPole"* tiene en cuenta:

1. Posición del carro
2. Velocidad del carro
3. Ángulo del palo
4. Velocidad del palo en la punta

Las dos primeras observaciones son parecidas a las de *"MountainCar"*, aunque difieren en rango. La posición del carro en *"CartPole"* está comprendida entre -4.8 y 4.8, por lo que el rango es mayor al de *"MountainCar"*. Además, el rango de velocidades, tanto del carro como de la punta del palo en *"CartPole"* es infinito. Esto marca un problema a resolver a la hora de plantear el método de discretización.

| ***Nº*** | **Acción** |
|---|---|
| *0* | Empujar carro a la derecha |
| *1* | Empujar carro a la izquierda |

*Tabla 3: Espacio de acciones - CartPole [6]*

Por otra parte, el espacio de acciones de *"CartPole"* ilustrado en la Tabla 3, muestra que las acciones sólo afectan al carrito que mueve el palo. Esto quiere decir que la velocidad de punta en el palo (observación 3 en la Tabla 2) puede ser ignorada.

Por ello, el método `discretize(self, obs)` (Cuadro 4) empleado para *"CartPole"* limita la velocidad del carro a al rango [-0.5, 0.5], pero ignora la velocidad de punta en el palo.

```
def discretize(self, obs):
    upper_bounds = [self.env.observation_space.high[0], 0.5,
self.env.observation_space.high[2], math.radians(50)]
    lower_bounds = [self.env.observation_space.low[0], -0.5,
self.env.observation_space.low[2], -math.radians(50)]
    ratios = [(obs[i] + abs(lower_bounds[i])) / (upper_bounds[i] - lower_bounds[i]) for i
in range(len(obs))]
    new_obs = [int(round((self.buckets[i] - 1) * ratios[i])) for i in range(len(obs))]
    new_obs = [min(self.buckets[i] - 1, max(0, new_obs[i])) for i in range(len(obs))]
    return tuple(new_obs)
```

*Cuadro 4: Q-Learning en CartPole - discretización*

## *Implementación de Q-Learning*

```
scores = deque(maxlen=100)
    for episode in range(total_episodes):
        epsilon = max(min_epsilon, min(1, 1.0 - math.log10((t + 1) / 25)))
        alpha = max(min_aplha, min(1.0, 1.0 - math.log10((episode + 1) / 25)))
        obs = env.reset()
        step = 0
        done = False
        if episode % 1000 == 0:
            print("Episodio n°", episode, "/10000")
        for step in range(max_steps):
            a, b = obs_to_state(env, obs)
            exp_exp_tradeoff = random.uniform(0,1)

            if exp_exp_tradeoff > epsilon:
                np.argmax(qtable[a][b])
            else:
            # Exploración (acción aleatoria)
                action = env.action_space.sample()

            obs, reward, done, info = env.step(action)

            a_, b_ = obs_to_state(env, obs)
            qtable[a][b][action] = qtable[a][b][action] + alpha * (reward +gamma*
                np.max(qtable[a_][b_]) - qtable[a][b][action])

            if done:
                break
```

*Cuadro 5: Q-Learning - "MountainCar"*

El Cuadro 5 muestra el código de aplicación del algoritmo *Q-Learning* para el entorno *"CartPole-v0"*. La implementación para *"MountainCar-v0"* solo difiere en el uso de la función `discretize(self, obs)`, descrita anteriormente en esta sección.

Se han incorporado dos nuevos enfoques con el fin de optimizar el resultado obtenido. Estos son:

1. El método, o política "Epsilon-greedy"
2. Optimización del factor de aprendizaje $\alpha$ durante el entrenamiento.

La finalidad de ambos métodos, que se explicará en mayor detalle a lo largo de esta sección, se fundamenta en obtener un entrenamiento más robusto y eficaz, con el fin de que el agente aprenda de forma eficiente.

## *Política "Epsilon-greedy"*

Aprovechando la notable magnitud y complicación de los estados y acciones de *"CartPole-v0"* y *"MountainCar-v0"* con respecto a entornos más simples como *"Taxi-v2"* y *"FrozenLake-v0"*, se ha buscado mejorar el algoritmo tratado en la sección 3.2.1.3.1 con la técnica denominada "Epsilon-greedy".

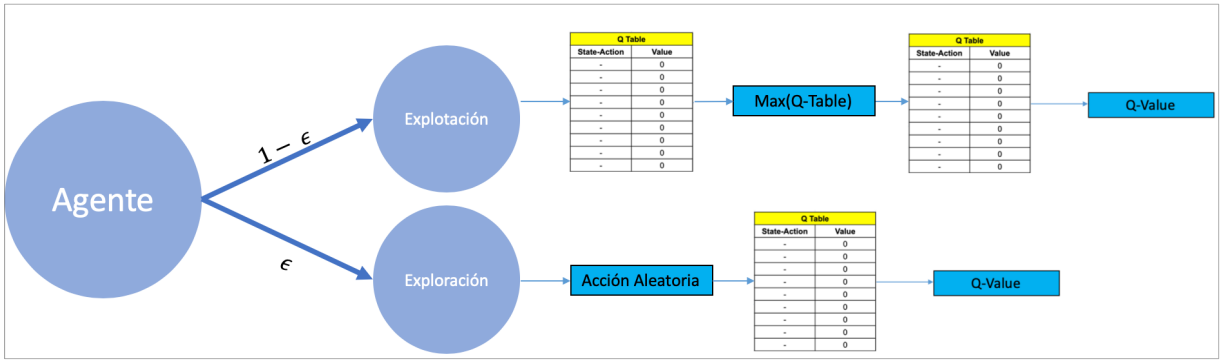



*Ilustración 18: Política "Epsilon-greedy" aplicada a Q-Learning*

**"Epsilon-greedy"** es un método comúnmente aplicado al Aprendizaje por Refuerzo, que busca incrementar el factor o habilidad de exploración durante el entrenamiento de un

agente. Cuando se entrena a un agente de Aprendizaje por Refuerzo, es muy importante que éste no se limite a seguir estrictamente la política aprendida mientras computa la tabla Q, de lo contrario, tiene muchas probabilidades de quedarse atrapado en una solución no óptima al poder existir muchas otras soluciones aún no exploradas.

Además, la tabla Q es inicialmente poco fiable, por lo que al inicio del entrenamiento la exploración se convierte en un elemento crucial: el agente necesita explorar hasta que esta sea óptima.

Es conveniente, pues, poner en regla un criterio o parámetro que obligue al agente a **explorar** acciones nuevas con cierta probabilidad o frecuencia. Este problema entre exploración de nuevos estados y explotación de la tabla Q, es el que pretende solucionar la política "Epsilon-greedy".

Para lograrlo, "Epsilon-greedy" define el término $\epsilon$, que, cuando se decide una acción a tomar, indica la probabilidad que lleva al agente a tomar una acción aleatoria:

$$P_{explotación} = 1 - \epsilon$$

$$P_{exploración} = \epsilon$$

Así, durante el entrenamiento se toma un número aleatorio entre 0 y 1, se compara con épsilon y, dependiendo de si este es menor o mayor, el agente toma una acción aleatoria o sigue la política dictada por la función Q (Ilustración 18).

Por otro lado, tal y como se ha explicado antes, la exploración es muy importante al inicio del entrenamiento. No obstante, a medida que el aprendizaje progresa y la política (o tabla Q) se va optimizando, el factor de exploración disminuye en importancia. Es apropiado, por lo tanto, reducir el parámetro $\epsilon$ paulatinamente hasta que se considere que la tabla Q es óptima y, en consecuencia, no sea necesario explorar más. En la práctica, épsilon se ha calculado de la siguiente manera:

$$\epsilon = \max\left(\epsilon_{mín}, \frac{1.0 - \log_{10}(t+1)}{25}\right)$$

Donde $t$ es el "tick" o instante de tiempo de la acción, y $\epsilon_{mín}$ el valor mínimo de épsilon, determinado por el programador.

### *Optimización del factor de aprendizaje durante el entrenamiento*

Análogamente, uno de los principales retos a la hora de entrenar redes neuronales es equilibrar la calidad de la solución final con el tiempo de entrenamiento necesario para llegar a ésta. El factor de aprendizaje o "learning rate" (2.1.1) es el hiperparámetro más importante para optimizar este equilibrio.

De la ecuación de Bellman (representada en la Ilustración 2), se puede fácilmente deducir que $\alpha$, es decir, el factor de aprendizaje, representa la medida en que los valores Q se actualizan en cada iteración.

A modo de ilustración, puede considerarse que dependiendo de si es pequeño o elevado, un factor de aprendizaje puede tener diferentes personalidades:

- Un factor de aprendizaje pequeño conlleva un proceso de aprendizaje cauteloso, es decir, hace que la red se ajuste lenta y cuidadosamente.
- Los factores de aprendizaje elevados son arriesgados ya que permiten aprendizajes muy rápidos pero tienen el riesgo de saltarse óptimos mejores que los que encuentran.

Cuando se entrena a un agente con Aprendizaje por Refuerzo, uno de los objetivos principales es que éste aprenda de forma rápida y precisa al mismo tiempo. Una eficaz solución a este problema consiste en ajustar el ritmo de aprendizaje durante el entrenamiento de alto a bajo, ralentizándolo a medida que se acerca a una solución óptima.

Así, como aparece en el Cuadro 5, el "learning rate" se reduce de forma similar al método empleado en "Epsilon-greedy" para disminuir gradualmente el valor de $\epsilon$:

$$\alpha = \max\left(\alpha_{\min}, \frac{1.0 - \log_{10}(t+1)}{25}\right)$$

Donde $\alpha_{\min}$ es el factor de aprendizaje mínimo, y $t$ es el "tick", o instante de tiempo en el momento del cálculo.

### *3.2.1.4 ATARI*

La implementación de entornos Atari requiere de técnicas aún más elaboradas de preprocesamiento, además de la aplicación del algoritmo Deep Q-Learning descrito en la sección 2.1.2.

Para empezar, el concepto del espacio de observaciones de los juegos Atari es completamente distinto a las observaciones presentes en los entornos Classic Control. Como se ha explicado en la sección 2.4, las observaciones en los entornos Atari consisten en una imagen RGB de la pantalla que constituye una matriz de forma (210, 160, 3).

Por otro lado, el espacio de acciones o "*action space"* propio de estos entornos es también particular, ya que cada acción se realiza repetidamente durante una duración de frames o fotogramas *kk*, donde *kk* se muestrea uniformemente a partir de {2, 3, 4} {2,3,4}.

Por todo esto, es necesario distinguir varios pasos en la integración del Aprendizaje por Refuerzo Profundo, o "*Deep Reinforcement Learning"* en estos entornos:

1. Preprocesamiento
    a. Reducir carga computacional y submuestreo
    b. Frame-skipping/stacking
2. Deep Q-Learning

El primer paso pretende reducir la carga de procesamiento, así como facilitar y preparar el espacio de observaciones para su integración en el algoritmo Deep Q-Learning.

El segundo paso implementa el algoritmo DQL en la plataforma mediante el diseño y aplicación de la Deep Q-Network, y aprovechando los resultados obtenidos del preprocesamiento.

La presente sección ahonda en el planteamiento de estas soluciones, así como la implementación de los distintos métodos en el diseño de la plataforma.

### *3.2.1.4.1 Preprocesamiento y Submuestreo*

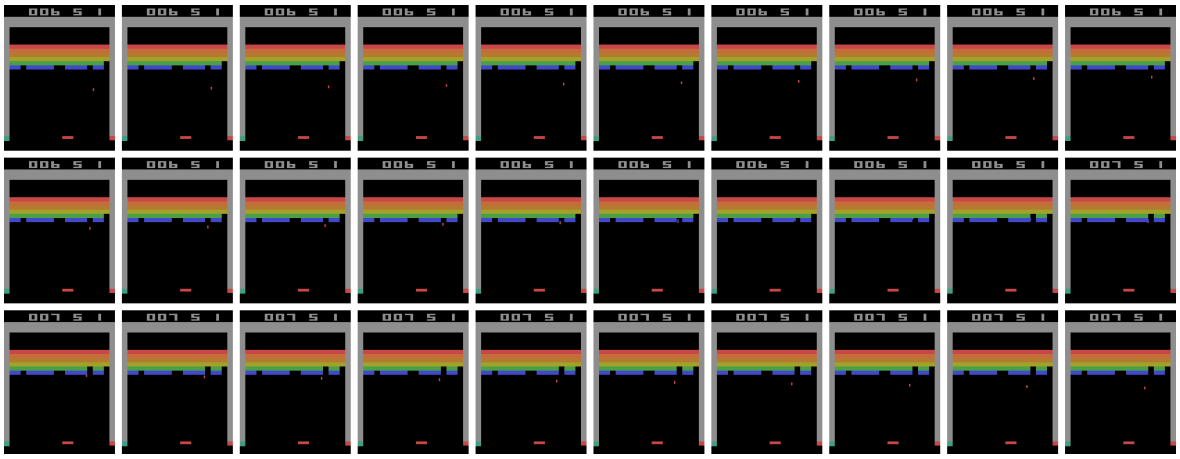


*Ilustración 19: Muestra de 30 fotogramas de Breakout [14]*

Al observar la ejecución fotograma a fotograma de un juego Atari-2600 (ver Ilustración 19), podemos ver que existen muchos fotogramas repetidos o redundantes, que no añaden información útil.

Esto perjudica la aplicación de algoritmos de *Deep Q-Learning*, al aumentar la dimensión y complejidad de los estados, y consecuentemente de la *Deep Q-Network* [15]. Es necesario, por lo tanto, realizar un procesamiento previo para así poder reducir la complejidad de los estados y el tiempo de cálculo necesario para el entrenamiento. Este proceso tiene dos pasos esenciales:

1. **Reducir, o acortar** la imagen a un tamaño fácilmente procesable.
2. Convertir la imagen a **escala de grises**, para ignorar la información de color y reducir la carga de procesamiento.

El resultado de ambos pasos se muestra en la Ilustración 20.

Para recortar la imagen hay que tener en cuenta qué elementos queremos conservar y cuáles pueden ser eliminados. En el caso de "*Breakout*" (ver Ilustración 19), la puntuación en la parte superior de la pantalla de juego puede ser un elemento interesante para omitir. Por un lado, reduciría drásticamente el número de estados, y su dimensión, pero, por otro lado, se ignoraría información importante, como el número de vidas restantes. Por esto

último, y para ayudar al usuario a interpretar el entrenamiento del agente, se ha optado por conservarla.

Un elemento que sí se puede eliminar con seguridad es la parte inferior de la pantalla, debajo del jugador, ya que no aporta ninguna información útil.

Para convertir imágenes RGB a escala de grises, basta con aplicar la función *rgb2gray* de la librería "*skimage*" [16].

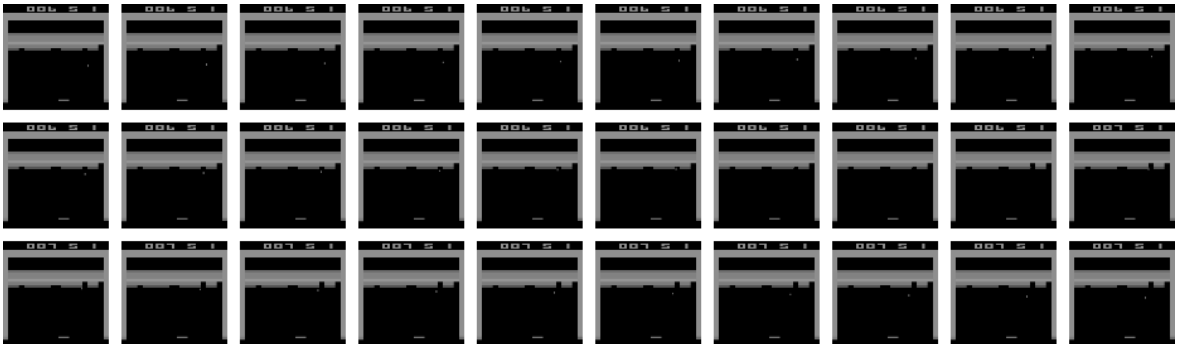


*Ilustración 20: Muestra de 30 fotogramas de Breakout preprocesados [14]*

Por suerte, todos los entornos Atari-2600 tienen la misma estructura de imagen (ver sección 2.4), lo que facilita la generalización de este preprocesamiento. A continuación, en el Cuadro 6 se muestra cómo se ha implementado el proceso en Python.

```
def preprocess_frame(frame):
    # Escala de grises
    gray = rgb2gray(frame)

    # Acortar pantalla (quitar la parte inferior del jugador)
    # Crop [Up: Down, Left: right]
    cropped_frame = gray[8:-12, 4:-12]

    # Normalizar valores de los pixeles
    normalized_frame = cropped_frame / 255.0

    # redimensionamos
    preprocessed_frame = transform.resize(normalized_frame, [110, 84])

    return preprocessed_frame
```

*Cuadro 6: Preprocesamiento de frames en Python*

Como se puede apreciar, se ha definido la función `preprocess_frame(frame)`, que recibe un fotograma (o *frame)* y devuelve la misma imagen en escala de grises, recortada (para eliminar la parte inferior de la pantalla), normalizada y redimensionada de 210 x 160 a 110 x 84 píxeles.

Para la asignación de los estados que serán introducidos en la *Q-Network*, y con el fin de simplificar el contenido y número de estados, es necesario **apilar fotogramas** [14].

En el caso de *"Breakout"* (2.4.4), por ejemplo, un solo fotograma no permite saber si la pelota está subiendo o bajando. De este modo, si asignamos a cada estado, $s_1$ por ejemplo, un número determinado de fotogramas $(x_1, x_2, x_3, x_4)$ consecutivos (4 en nuestro caso), podremos otorgar al agente una noción de movimiento.

$$s_1 = (x_1, x_2, x_3, x_4)$$

Por otro lado, los fotogramas apilados no deben ser inmediatamente consecutivos, pues como se ha comentado en el inicio de esta sección, existen muchos fotogramas idénticos o redundantes (ver Ilustración 19). Así, como se muestra en la Ilustración 21, la mejor opción es aplicar un parámetro de salto, que determine el número de fotogramas a omitir antes de guardar la imagen en un estado (en este caso 4). Como es lógico, este proceso también reduce el tiempo de entrenamiento y carga de procesamiento del algoritmo.

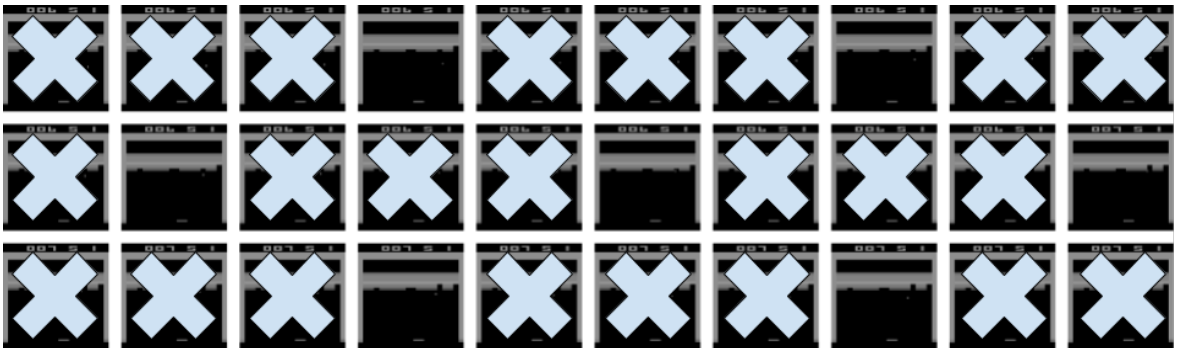


*Ilustración 21: Frame-skipping en Breakout [14]*

```
def stack_frames(stacked_frames, state, is_new_episode, stack_size):
    # Preprocesamos frame
    frame = preprocess_frame(state)

    if is_new_episode:
        # Vaciamos stacked_frames
        stacked_frames = deque([np.zeros((110, 84), dtype=np.int) for i in
range(stack_size)], maxlen=4)

        # Nuevo episodio -> copiamos la misma frame 4 veces
        stacked_frames.append(frame)
        stacked_frames.append(frame)
        stacked_frames.append(frame)
        stacked_frames.append(frame)

        # Stack frames
        stacked_state = np.stack(stacked_frames, axis=2)

    else:
        # Concatenamos frame a deque -> elimina automáticamente la más antigua
        stacked_frames.append(frame)

        stacked_state = np.stack(stacked_frames, axis=2)

    return stacked_state, stacked_frames
```

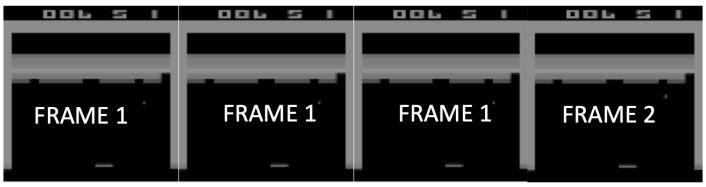
*Cuadro 7: Implementación del algoritmo de frame-stacking*

Como muestra el Cuadro 7, en la implementación de este proceso en Python, se ha creado la función:
`stack_frames(stacked_frames,state,is_new_episode,stack_size)`.
Ésta recibe, como muestra su prototipo:

- `stacked_frames:` es una la pila de fotogramas o *frames*. Denominada *deque* y al principio inicializada a cero
- `state:` representa el fotograma actual en el juego.
- `is_new_episode:` booleano que indica si el episodio es nuevo.

- `stack_size`: tamaño de la pila, en nuestro caso su valor es 4.


ESTADO INICIAL = < X1, X1, X1, X1 >




SIG_ESTADO = < X1, X1, X1, X2 >

*Ilustración 22: Apilamiento de frames*

El proceso funciona de la siguiente manera:

- Al comienzo del episodio, se vacía la pila de *frames* (`stacked_frames`), se rellena con el primer *frame* copiado cuatro veces y se apila en un nuevo estado (`stacked_state`).
- Si el episodio no es nuevo, basta con introducir el nuevo *frame* en la pila, eliminando así el más antiguo, y creando un nuevo `stacked_state`.

#### *3.2.1.4.2 Implementación del Algoritmo DQL*

La presente subsección explica el proceso de diseño e integración del algoritmo *Deep Q-Learning* en los entornos Atari-2600 de OpenAI Gym Retro. Consta de dos procesos principales, que se desarrollan en detalle más adelante:

1. Diseño de la red neuronal, es decir de la **Deep Q-Network**.
2. Implementación del método **"Experience replay"**, que añade robustez al algoritmo y entrenamiento del agente.

En los apartados que siguen se trata de explicar y dar una idea al lector del proceso de diseño y funcionamiento de ambos procesos.

## *Deep Q-Network*


*Ilustración 23: Diseño de la Red Neuronal*

La red neuronal implementada está basada en la presentada en el artículo de DeepMind *"Human level control through deep reinforcement learning"* [15] que es una referencia importante dentro del mundo del aprendizaje por refuerzo. La estructura del algoritmo diseñado, representada en la Ilustración 23, es la siguiente:

1. Mediante el proceso descrito en la subsección anterior, se obtiene el estado actual como una **pila de cuatro *frames*** relativamente consecutivos.
2. Esta pila se hace pasar por tres **redes convolucionales**, las cuales detectan los elementos y partes más importantes de la imagen mediante la convolución de esta con una serie de matrices bidimensionales de pesos, denominadas filtros o kernels.
3. A continuación, se aplican dos **FC** (*Fully-Connected Layer),* responsables de mapear o clasificar los resultados obtenidos en la capa anterior, obteniendo un **Q-valor para cada acción**.

La implementación en Python se muestra en el Cuadro 8.

```
class DQNetwork:
    def __init__(self, state_size, action_size, learning_rate, name='DQNetwork'):
        self.state_size = state_size
        self.action_size = action_size
        self.learning_rate = learning_rate

        with tf.variable_scope(name):
            # Creo placeholders
            self.inputs_ = tf.placeholder(tf.float32, [None, *state_size], name="inputs")
            self.actions_ = tf.placeholder(tf.float32, [None, self.action_size], name="actions_")

            # target_Q = R(s,a) + ymax Qhat(s', a')
            self.target_Q = tf.placeholder(tf.float32, [None], name="target")

            """
            Pasamos por lera convnet:
            CNN
            ELU
            """
            # Input is 110x84x4
            self.conv1 = tf.layers.conv2d(inputs=self.inputs_,
                                          filters=32,
                                          kernel_size=[8, 8],
                                          strides=[4, 4],
                                          padding="VALID",
                                          kernel_initializer=tf.contrib.layers.xavier_initializer_conv2d(),
                                          name="conv1")

            self.conv1_out = tf.nn.elu(self.conv1, name="conv1_out")

            """
            2ª convnet:
            CNN
            ELU
            """
            self.conv2 = tf.layers.conv2d(inputs=self.conv1_out,
                                          filters=64,
                                          kernel_size=[4, 4],
                                          strides=[2, 2],
                                          padding="VALID",
                                          kernel_initializer=tf.contrib.layers.xavier_initializer_conv2d(),
                                          name="conv2")

            self.conv2_out = tf.nn.elu(self.conv2, name="conv2_out")

            """
            3era convnet:
            CNN
            ELU
            """
            self.conv3 = tf.layers.conv2d(inputs=self.conv2_out,
                                          filters=64,
                                          kernel_size=[3, 3],
                                          strides=[2, 2],
                                          padding="VALID",
                                          kernel_initializer=tf.contrib.layers.xavier_initializer_conv2d(),
                                          name="conv3")

            self.conv3_out = tf.nn.elu(self.conv3, name="conv3_out")

            self.flatten = tf.contrib.layers.flatten(self.conv3_out)

            self.fc = tf.layers.dense(inputs=self.flatten,
                                      units=512,
                                      activation=tf.nn.elu,
                                      kernel_initializer=tf.contrib.layers.xavier_initializer(),
                                      name="fc1")

            self.output = tf.layers.dense(inputs=self.fc,
                                          kernel_initializer=tf.contrib.layers.xavier_initializer(),
                                          units=self.action_size,
                                          activation=None)

            # Q -> valor previsto.
            self.Q = tf.reduce_sum(tf.multiply(self.output, self.actions_))

            # loss = diferencia entre predicted Q_values y Q_target -> Sum(Q_target - Q)^2
            self.loss = tf.reduce_mean(tf.square(self.target_Q - self.Q))

            self.optimizer = tf.train.AdamOptimizer(self.learning_rate).minimize(self.loss)

# Reset graph
tf.reset_default_graph()

# Instanciamos la DQNetwork
DQNetwork = DQNetwork(state_size, action_size, learning_rate
```

*Cuadro 8: Implementación de DQL*

## *Experience Replay*

```python
class Memory():
    def __init__(self, max_size):
        self.buffer = deque(maxlen=max_size)

    def add(self, experience):
        self.buffer.append(experience)

    def sample(self, batch_size):
        buffer_size = len(self.buffer)
        index = np.random.choice(np.arange(buffer_size),
                                 size=batch_size,
                                 replace=False)

        return [self.buffer[i] for i in index]
```

*Cuadro 9: Implementación de Experience Replay en Python*

*"Experience Replay"* es una técnica muy potente para el entrenamiento de agentes con Deep Q-Networks.

En esencia, se basa en que un agente puede aprender a realizar una tarea de manera más robusta si se almacenan las experiencias del mismo agente y muestrean aleatoriamente batches de ellas para entrenar la red. De esta manera, manteniendo al azar las experiencias, se evitaría que la red aprenda únicamente de lo que está haciendo en ese momento dentro del entorno, y le permitiría aprender de una muestra más diversa de experiencias anteriores. Cada experiencia se guarda como una tupla $(estado, acción, recompensa, siguiente\ estado)$.

Como se muestra en la implementación presentada en el Cuadro 9, se ha creado la clase `Memory` que define dos métodos `add(self, experience)`, para añadir una nueva experiencia a la memoria (un buffer de tipo deque), y `sample(self, batch_size)` que crea un batch de experiencias aleatorias.

## 3.3 MANUAL DE USUARIO

El funcionamiento de la plataforma es muy sencillo. Desde la pantalla de inicio, o menú principal (Ilustración 15), el usuario puede acceder al menú de selección de entornos haciendo clic en los botones "Atari" o "Classic Control". Ya en el menú de selección (Ilustración 16), configurar el entrenamiento de distintos entornos de Aprendizaje por Refuerzo. Si desea entrenar un agente, el usuario puede:

1. Seleccionar el entorno mediante el cuadro combinado situado en la parte superior de la pantalla (Ilustración 24).

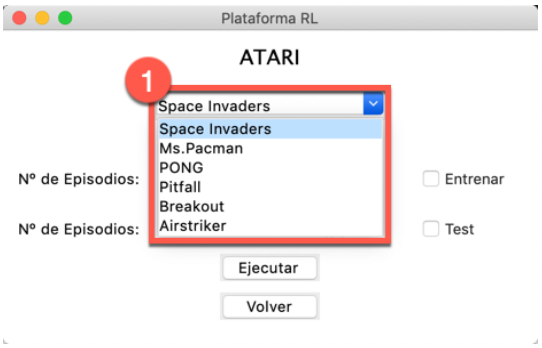



*Ilustración 24: Paso 1: Selección de entorno*

2. Una vez escogido el entorno, puede marcar varias opciones (Ilustración 25):
   - Renderizar[1] el entorno, marcando la casilla "Renderizar".
   - Entrenar al agente: que le permitirá establecer el número de episodios de entrenamiento.
   - Hacer una prueba o "Test" del modelo entrenado.

[1] Renderización, (del inglés *rendering*) es un anglicismo usado en informática para referirse al proceso de generar una imagen visible e inteligible para el ser humano, a partir de información digital.


*Ilustración 25: Paso 2: Opciones de Entrenamiento*

3. Ejecutar el entorno, haciendo clic en el botón "Ejecutar" (Ilustración 26)


*Ilustración 26: Paso 3: Ejecutar Entrenamiento*

Estos pasos, llevarán al usuario a la pantalla de "Información de Entrenamiento", que proporcionará información útil sobre el desarrollo de la ejecución del algoritmo. Esta pantalla se presenta en la Ilustración 17.

Para volver al menú principal basta con pulsar el botón "Volver" (ver Ilustración 27).

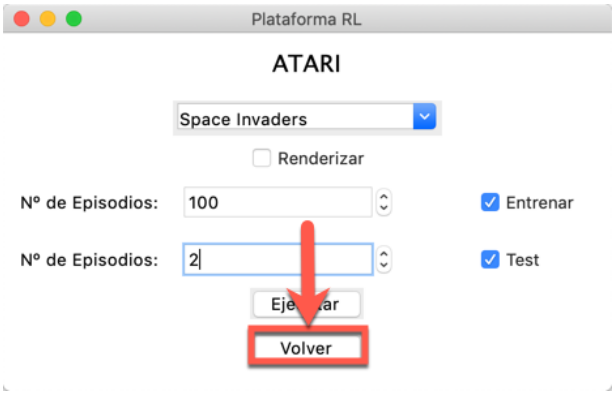



*Ilustración 27: Volver al Menú Principal*

# Capítulo 4. ANÁLISIS

En este capítulo se presenta un análisis de los resultados obtenidos de la aplicación de los distintos algoritmos a cada entorno. En él se van a explicar:

1. Metodología y criterios del análisis.
2. Análisis de experimentos en los distintos entornos, divididos en las categorías:
    - Classic Control
    - Atari 2600

## *4.1 METODOLOGÍA*

La presente sección expone y justifica los distintos criterios implementados a la hora de ejecutar los experimentos, así como los parámetros medidos.

Debido a las diferencias entre las implementaciones de los distintos tipos de entornos explicadas en los capítulos anteriores, es necesario dividir el análisis en las categorías siguientes:

- Classic Control: Que comprende los juegos "MountainCar", "CartPole", "FrozenLake" y "Taxi".
- ATARI: que comprende los juegos "Space Invaders", "Ms Pacman" y "Breakout

A fin de poder comparar eficazmente los entornos con algoritmos similares o idénticos, se han aplicado parámetros parecidos. Los parámetros empleados en cada categoría se presentan en la Tabla 4, Tabla 5 y Tabla 7. Estos parámetros, así como el análisis implementado dependen de los algoritmos integrados y de las características de cada tipo de entorno que se han explicado a lo largo de este documento.

Así mismo, propiedades diferentes entre entornos requieren el análisis de diferentes variables tras los experimentos.

Debido a su sencillez y velocidad de entrenamiento, para los entornos *"Classic Control"* se ha optado por evaluar dos parámetros:

- Nº de episodios mínimo para llegar a una solución.
- Recompensa/puntuación por episodio.

Para los juegos Atari se ha buscado analizar tres parámetros:

- Pérdida o error Q (*"Q Loss"*): la diferencia entre el máximo valor Q posible para el siguiente estado (valor Q objetivo o $Q_{target}$) y el valor Q estimado:

$$Loss = \left(Q_{target} - Q_{predicted}\right)^2$$

  Donde: $Q_{target} = R + \gamma \max_a \hat{Q}(s', a)$

- Recompensa/puntuación por episodio.
- Tiempo de entrenamiento.

## 4.2 CLASSIC CONTROL

En esta sección se presenta el análisis de los entornos "Classic Control", explicando e ilustrando mediante experimentos las diferencias entre cada entorno y los algoritmos empleados (descritos en la sección 3.2). Debido a su relativa sencillez con respecto a los entornos Atari, un entrenamiento eficaz en estos entornos lleva menos tiempo y requiere menos capacidad de procesamiento.

### 4.2.1 MOUNTAINCAR Y CARTPOLE

| *Entorno* | Nº Episodios | Max steps | $\alpha$ | $\alpha_{mín}$ | $\gamma$ |
|---|---|---|---|---|---|
| *MountainCar* | 100 000 | 100 | 1.0 | 0.1 | 1.0 |
| *CartPole* | 10 000 | 100 | 1.0 | 0.1 | 1.0 |

*Tabla 4: Parámetros de Análisis - Classic Control: MountainCar y CartPole*

El entrenamiento en los entornos "*CartPole-v0*" y *"MountainCar-v0"* durante 10.000 episodios con las características detalladas en la Tabla 4, da los siguientes resultados.


*Ilustración 28: CartPole - Puntuación por Episodio*

En ***"CartPole"***, la puntuación obtenida es directamente proporcional al tiempo de supervivencia. El objetivo del juego es, por lo tanto, maximizar la puntuación.

La Ilustración 28 muestra que el agente converge hacia la puntuación máxima obtenible (200 puntos), llegando a una solución óptima a partir del episodio 255. Este proceso se representa en mayor detalle en la Ilustración 29, donde se puede apreciar un crecimiento exponencial, y prácticamente constante a partir del episodio 115, hasta llegar a la solución óptima.


*Ilustración 29: CartPole - Optimización de la Puntuación*

En ***"MountainCar"*** se descuenta un punto al agente por cada *"tick"*, o instante de ejecución. El objetivo del juego es acercarse a una puntuación de cero puntos.

La Ilustración 30 muestra que la puntuación obtenida a lo largo de 100 000 episodios parece converger a -150 de forma exponencial similar a la obtenida en el experimento con *"CartPole"*. Observando la figura de la derecha, se puede apreciar que el agente aprende a resolver el entorno alrededor del episodio 50. El número de episodios de aprendizaje requerido para llegar a una solución apropiada es mayor al de *"CartPole"*. Esto puede ser debido al método de discretización implementado.

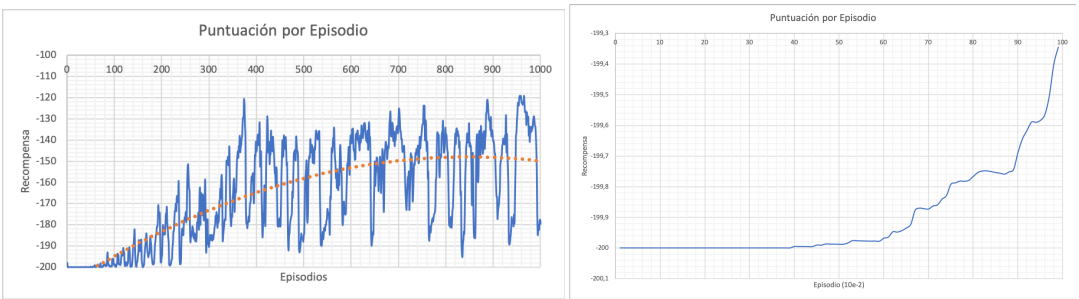



*Ilustración 30: MountainCar-v0 - Puntuación por Episodio*

### 4.2.2 FROZENLAKE Y TAXI

| *Entorno* | Nº Episodios | Max steps | $\alpha$ | $\gamma$ |
|---|---|---|---|---|
| *Taxi-v2* | 10000 | 50 | 0.1 | 0.99 |

*Tabla 5: Parámetros de Análisis - Classic Control: Taxi*

El algoritmo *"Q-Learning simple"* también conocido como *"Naive Q-Learning"* (sección 3.2.1.3.1) implementado en "FrozenLake" y "Taxi" construye la tabla Q tomando acciones aleatorias, y actualizando los valores medidos a lo largo del entrenamiento.

No tiene sentido, por lo tanto, analizar la puntuación obtenida en cada episodio del entrenamiento, puesto que ésta es siempre aleatoria y no es representativa del proceso de aprendizaje. Tampoco es posible establecer un número de episodios de aprendizaje mínimo para la obtención de una solución óptima. Es por esto que en estos entornos, se ha optado por evaluar los resultados obtenidos en la fase de prueba o "test" de la política (o tabla Q) obtenida del entrenamiento.

A continuación, se explican los experimentos realizados y cómo se han analizado.

#### *4.2.2.1 Taxi-v2*

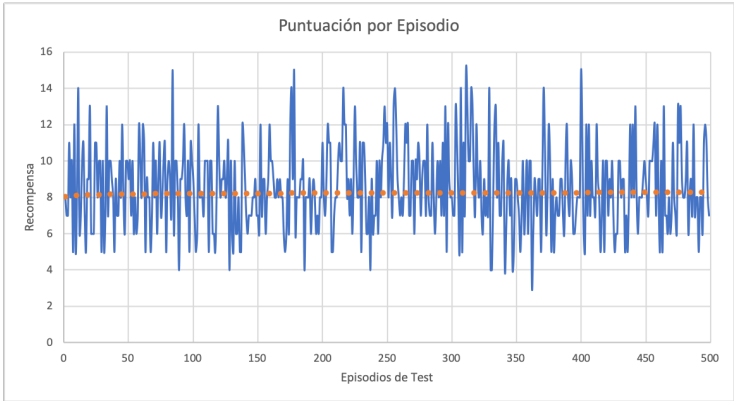



*Ilustración 31: Taxi-v2 - Puntuación por Episodio de Prueba*

En *"Taxi-v2"*, cada acción descuenta 1 punto, y se obtiene +20 puntos por llevar al pasajero a su destino. La recogida o entrega errónea de un pasajero descuenta 10 puntos. El episodio termina cuando se entrega al pasajero.

Un estudio de las puntuaciones obtenidas durante 500 episodios cuando el agente ha entrenado 100 000 episodios (Ilustración 31) muestra que el agente ha aprendido satisfactoriamente a resolver el entorno. Como muestra la figura, la puntuación media a lo largo de los 500 episodios es de 8.61 puntos, y nunca negativa, lo que indica que el agente ha resuelto el entorno el 100% de los intentos, por lo que ha aprendido satisfactoriamente. Esta puntuación podría mejorarse con más episodios de entrenamiento.

#### *4.2.2.2 FrozenLake-v0 y FrozenLake8x8-v0*

En los entornos *"FrozenLake"*, la puntuación obtenida depende sólo de si el agente ha llegado o no a la casilla de meta. Se recibe 1 punto si se alcanza la meta y 0 en caso contrario.

Teniendo esto en cuenta, un estudio de la media de puntos obtenida en la fase de "test", indica que el agente aprende a resolver el entorno eficazmente. Los resultados obtenidos se muestran en la Tabla 6, donde la media de puntos, 0.71 en *"FrozenLake-v0"* y 0.69 en *"FrozenLake8x8"* obtenida indica que el entorno se ha resuelto en la mayoría de los episodios.

Por otro lado, es apreciable que, al ser un entorno más complicado, *"FrozenLake8x8"*, ha requerido el doble de episodios de entrenamiento para obtener un rendimiento similar a *"FrozenLake-v0"*.

| *Entorno* | Nº Episodios | Max steps | $\alpha$ | $\gamma$ | Puntuación Media en test |
|---|---|---|---|---|---|
| *FrozenLake-v0* | 1.000.000 | 50 | 0.1 | 0.99 | 0.71 |
| *FrozenLake8x8* | 2.000.000 | 100 | 0.1 | 0.99 | 0.69 |

*Tabla 6: FrozenLake - Parámetros y Resultados*

## 4.3 ATARI

| *Entorno* | Nº Episodios | Max Steps | Stack Size | Batch Size | $\alpha$ | $\epsilon_0$ | $\epsilon$ Decay Rate | $\gamma$ |
|---|---|---|---|---|---|---|---|---|
| *Atari-2600* | 100 | 5000 | 4 | 64 | 0.00025 | 1.0 | 0.00001 | 0.9 |

*Tabla 7: Parámetros de Análisis - Atari 2600*

El tiempo de entrenamiento en los entornos ATARI es un factor importante. El empleo de redes neuronales para la traducción de observaciones a nivel de *"frame"* a estados y acciones interpretables por un agente hace que el proceso de entrenamiento requiera una gran labor computacional.

El entrenamiento en estos entornos viene, por otro lado, condicionado por la duración media de una partida en cada juego. Así como un entorno Classic Control finaliza en cuanto el agente resuelve, vence o pierde el juego; entornos como "Space Invaders", "Breakout" o "Ms Pac-Man" proporcionan varios intentos al jugador, lo que hace que la duración de un episodio determinado pueda resultar especialmente larga.

Otros entornos, como "Pong" no tienen límite de tiempo o vidas, pero se parecen más a los entornos Classic Control, en que establecen un umbral, o límite de puntos (21 en "Pong") para que la partida se dé por ganada o perdida.

Es por todo esto que el tiempo de entrenamiento de los entornos Atari-2600 es especialmente largo. Como muestran las subsecciones a continuación, la media por entorno para 100 episodios es de 10 horas, lo que hace que 1000 episodios, un número mínimo para que el agente aprenda eficazmente a resolver un entorno, requiera una media de 100 horas por agente.

Por ello a la hora de realizar los experimentos en los entornos Atari, no se ha buscado entrenar al agente hasta que este sea capaz de llegar a una solución completa. El objetivo de este proyecto es el desarrollo de la plataforma, y esto requeriría capacidad de procesamiento de la que no se dispone.

Se ha optado por entrenar 100 episodios en cada entorno y comparar los distintos resultados obtenidos, analizando y comentando sus características.

### 4.3.1 SPACE INVADERS

El "Space Invaders" requiere un tiempo de entrenamiento para 100 episodios de 16 horas y 14 minutos. La puntuación por episodio y pérdida medidas han sido representadas en la Ilustración 32.

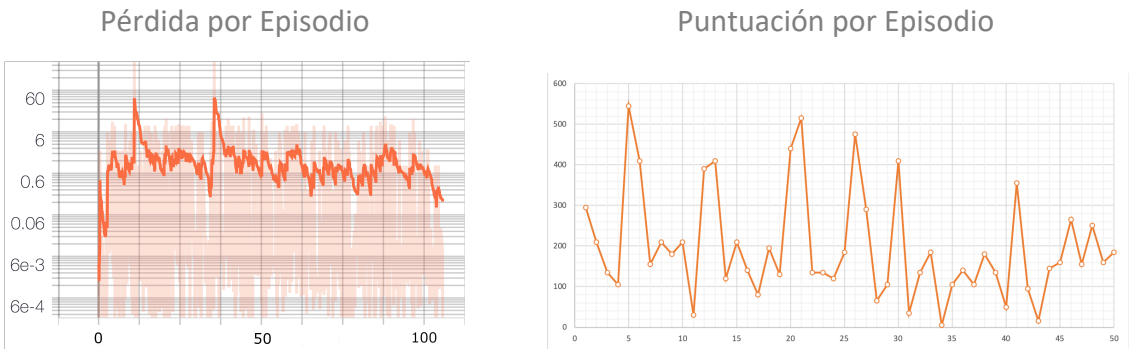



*Ilustración 32: Space Invaders - Pérdida media (izqda.) y Recompensa (dcha.) por episodio*

Observando las gráficas, a lo largo del proceso de aprendizaje, el valor Q-Loss parece converger a un mínimo local. Esto indica que la política del agente se va perfeccionando, y acercando a una solución concreta. Aún así, la alta variabilidad entre episodios apunta a que haría falta más tiempo de aprendizaje.

Por otro lado, la puntuación no parece converger hacia ningún valor específico. Esto confirma la teoría expresada anteriormente: el agente requiere más tiempo de entrenamiento para llegar a una a solución óptima que dé resultados constantes.

### 4.3.2 MS. PAC-MAN

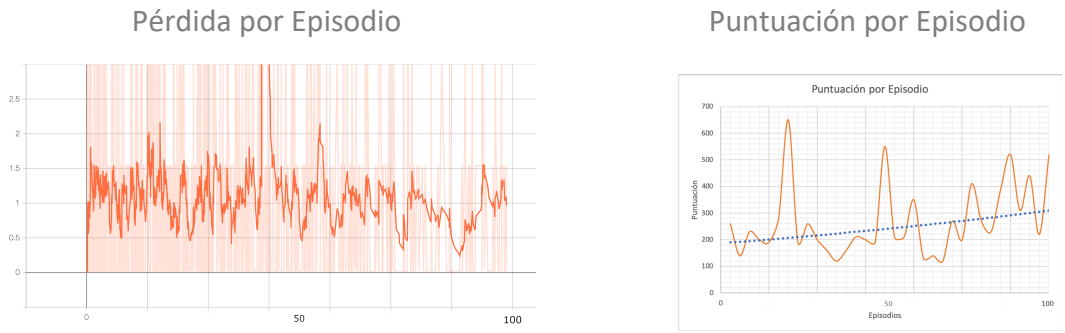



*Ilustración 33: Ms. Pac-Man - Pérdida media (izqda.) y Recompensa por episodio (dcha.)*

Los resultados obtenidos en el entrenamiento de "Ms. Pac-Man" (Ilustración 33) indican lo mismo que en "Space Invaders". El tiempo de entrenamiento requerido para 100 episodios es de 20 horas y 13 minutos. Como se puede apreciar las ilustraciones, la pérdida por episodio parece centrarse entorno a 1.0, mientras que la puntuación obtenida crece exponencialmente con el tiempo de aprendizaje. Se requeriría, no obstante, de más tiempo de entreno y capacidad de procesamiento para llegar a resultados más representativos.

### 4.3.3 BREAKOUT

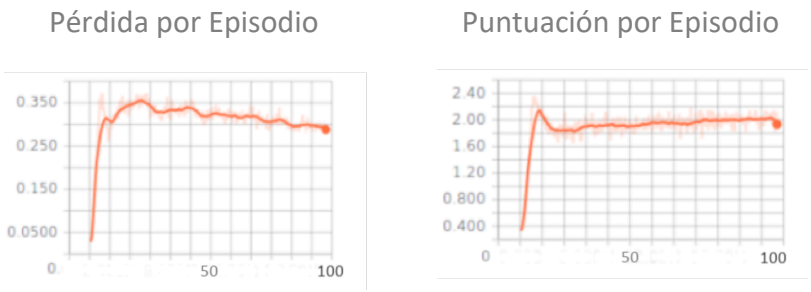



*Ilustración 34: Breakout - Pérdida media (izqda.) y Recompensa por episodio (dcha.)*

En "Breakout", el jugador dispone de 5 vidas, y obtiene puntos por cada ladrillo que elimina. El agente tardó 4h 13 min para entrenar 100 episodios, obteniéndose los resultados mostrados en la Ilustración 34.

El análisis de la pérdida media muestra que a medida que el agente aprende, la diferencia entre $Q_{target}$ (el valor máximo esperado para el siguiente estado) y $Q_{predicted}$ (el valor Q estimado en el momento del cálculo), se minimiza y parece converger a un mínimo local entorno al 0.3, indicando un entrenamiento eficiente, con poca variación entre episodios.

Por otro lado, la recompensa media por episodio parece converger entorno a 2.00 puntos. La puntuación máxima posible en el entorno es de 864, lo que indica que, en la mayoría de los casos, el agente no consigue atravesar más allá de las dos primeras filas de ladrillos. Requeriría, por tanto, más entrenamiento.

# Capítulo 5. CONCLUSIONES

En este capítulo se discuten las principales conclusiones extraídas del desarrollo del proyecto.

Tal y como se ha detallado en el desarrollo de la presente memoria, el objetivo de este proyecto es el diseño e implementación de una plataforma o entorno de desarrollo para el entrenamiento de agentes de Aprendizaje por Refuerzo que cumpliese con las características explicadas en el Capítulo 1.

Así, tanto el diseño como la implementación del proyecto han ido enfocados a cumplir con dichos requisitos, cuya consecución se detalla a continuación como conclusión.

## *5.1 DESARROLLO DE LA PLATAFORMA*

Como objetivos principales a la hora de desarrollar el proyecto, se pretendía cumplir:

1. Interfaz intuitiva y fácil de ejecutar e instalar.
2. Disponibilidad de múltiples entornos, de carácter variado.
3. Ejecución de experimentos, con información detallada en tiempo real.

El uso de Python como lenguaje de programación permite que la ejecución de experimentos sea sencilla y facilite al usuario final la investigación de cada entorno sin requerir conocimientos de programación avanzados.

Por otro lado, la integración de múltiples entornos ha sido facilitada e implementada gracias al uso de las librerías OpenAI Gym y OpenAI Gym Retro.

Además, el diseño de la plataforma permite al usuario final estudiar cada entorno y algoritmo rápidamente y de forma sencilla tras su instalación. Se ha proporcionado en esta memoria una guía de instalación para facilitar este mismo proceso.

## 5.2 ANÁLISIS Y EJECUCIÓN DE EXPERIMENTOS

La presente memoria ha buscado también proporcionar al lector un análisis de experimentos realizados, a fin de mostrar la eficacidad y funcionamiento de los algoritmos implementados en la plataforma.

Este análisis se ha desarrollado en el Capítulo 4. , mostrando los requerimientos de cada algoritmo implementado, así como ventajas y desventajas inherentes a la hora de entrenar a un agente de forma satisfactoria.

Los entornos "Classic Control" se han conseguido entrenar de manera eficaz, debido a la relativa sencillez de los problemas. Por otro lado, tal y como se explica en la sección 4.3, la implementación de redes neuronales, así como el alto tiempo de ejecución necesario y complejidad de los juegos Atari requiere una capacidad de procesamiento y CPU mucho mayor. La ejecución de los experimentos en estos entornos se ha centrado, por lo tanto, en mostrar el comportamiento típico del entrenamiento en los mismos, aunque no se llegue a una solución definitiva.

## 5.3 DESARROLLO FUTURO

Como último apartado de esta conclusión, se ofrece una descripción de varias líneas de desarrollo futuro que han surgido como posibilidades a la hora de mejorar la plataforma diseñada.

### 5.3.1 EJECUCIÓN DISTRIBUIDA

Con el fin de evitar la carga de procesamiento impuesta por los entornos Atari-2600, se propone la implementación de un sistema distribuido, mediante el cual el usuario pueda repartir la cantidad de procesado entre otros sistemas, o escoger el sistema donde ejecutar los entornos.

Esto permitiría también la implementación de entornos más complejos.

### 5.3.2 EDICIÓN Y CREACIÓN DE ALGORITMOS

Otra línea de desarrollo propuesta es la integración de una funcionalidad que permita al usuario editar los algoritmos implementados, o crear nuevos algoritmos desde la plataforma.

La implementación sería intuitiva, y permitiría al usuario crear, editar y probar sus propios algoritmos sin la necesidad de conocimientos en programación.

### 5.3.3 IMPLEMENTACIÓN DE ANÁLISIS GRÁFICO DETALLADO

Por último, la plataforma podría proporcionar al usuario un análisis gráfico en tiempo real del aprendizaje. Se podría buscar implementar directamente en la plataforma herramientas de análisis en línea como TensorBoard [17] que permite el análisis online de entornos Deep Q-Learning, y que reduciría la carga de procesamiento, pero requeriría la conexión a internet de la plataforma.

# Capítulo 6. BIBLIOGRAFÍA

[1] A. Choudhary, «A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python,» [En línea]. Available: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/.

[2] S. Wiki, «A Beginner's Guide to Deep Reinforcement Learning,» [En línea]. Available: https://skymind.ai/wiki/deep-reinforcement-learning.

[3] Python Software Foundation, «Python Web Page,» [En línea]. Available: https://www.python.org/.

[4] P. S. Foundation, «Tkinter — Python interface to Tcl/Tk,» [En línea]. Available: https://docs.python.org/2/library/tkinter.html.

[5] JetBrains s.r.o, «PyCharm,» [En línea]. Available: https://www.jetbrains.com/pycharm/.

[6] OpenAI, «OpenAI Gym,» [En línea]. Available: https://gym.openai.com/.

[7] «OpenAI Gym Retro,» [En línea]. Available: https://openai.com/blog/gym-retro/.

[8] «Repositorio OpenAI Gym Retro,» [En línea]. Available: https://github.com/openai/retro/tree/develop.

[9] TensorFlow, «TensorFlow Guide - Graphs,» [En línea]. Available: https://www.tensorflow.org/guide/graphs.

[10] M. Abadi, «TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,» [En línea]. Available: https://arxiv.org/abs/1603.04467.

[11] Google LLC, «TensorFlow,» [En línea]. Available: https://www.tensorflow.org/.

[12] G. B. Team, «TensoFlow Guide - Introduction,» [En línea]. Available: https://www.tensorflow.org/guide/low_level_intro.

[13] TensorFlow, «Why TensorFlow?,» [En línea]. Available: https://www.tensorflow.org/about.

[14] «skimage API,» [En línea]. Available: https://scikit-image.org/docs/dev/api/skimage.html.

[15] D. Seita, «Frame Skipping and Pre-Processing for Deep Q-Networks on Atari 2600 Games,» [En línea]. Available: https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness y M. G. Bellemare, «Human-level control through deep reinforcement learning,» 2015.

[17] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang y W. Zaremba, «OpenAI Gym,» 2016. [En línea]. Available: https://arxiv.org/abs/1606.01540.