ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

APLICACIÓN DE TÉCNICAS DE APRENDIZAJE POR REFUERZO EN ENTORNOS GYM

Autor: Pilar Alvargonzález Boulet

Director: Miguel Ángel Sanz Bobi

Madrid

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

APLICACIÓN DE TÉCNICAS DE APRENDIZAJE POR REFUERZO EN ENTORNOS GYM en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2019/20 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.: Pilar Alvargonzález Boulet Fecha: 21/08/ 2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Miguel Ángel Sanz Bobi Fecha: 22/08/2020

Agradecimientos

A mi familia, amigos y mi tutor.

APLICACIÓN DE TÉCNICAS DE APRENDIZAJE POR REFUERZO EN ENTORNOS GYM

Autor: Alvargonzalez Boulet, Pilar.
Director: Sanz Bobi, Miguel Ángel.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El objetivo principal de este proyecto es crear una plataforma desde la cual se pueda aplicar fácilmente algoritmos de aprendizaje por refuerzos una serie de entornos diferentes de Gym OpenAI. Una vez desarrollada la aplicación, se utilizará para llevar a cabo un análisis de los efectos de la parametrización.

1. Introducción.

El aprendizaje por refuerzo es una rama del aprendizaje automático que busca mejorar el desempeño de un agente a la hora de realizar una actividad determinada por medio de un análisis de las recompensas que se obtienen para cada acción ejecutada. Uno de los algoritmos utilizados en este campo es el Deep Q-Learning o DQN, basado en asignar un valor numérico a cada par de acción y estado, de manera que cuanto mayor sea este, más cerca estará el agente de su objetivo. Además, en él se hace uso de las redes neuronales para facilitar el cálculo de estos valores ante la existencia de numerosos estados y/o acciones.

2. Definición del Proyecto.

Hoy en día se pueden encontrar diversos experimentos sobre la aplicación de algoritmos de aprendizaje por refuerzo a los entornos de Gym. Sin embargo, todos ellos están formados simplemente por una serie de códigos complejos, individuales y con unos parámetros definidos. La aplicación desarrollada en este proyecto simplifica enormemente el trabajo del usuario ya permite acceder a varios juegos desde una misma plataforma, modificar los parámetros a su gusto de manera sencilla e incluso comparar entre sí los resultados obtenidos. De esta manera, esta herramienta no sólo agiliza la implantación del algoritmo para diferentes casos, sino que además puede ser utilizada por cualquier persona con unos conocimientos básicos y sin tener que llevar a cabo un estudio exhaustivo para comprender el código utilizado.

Para conseguir el objetivo, se llevará a cabo la implementación del algoritmo Deep Q-Learning o DQN para cinco entornos diferentes de Gym, mediante el lenguaje de programación Python y se desarrollará la interfaz gráfica de la aplicación con este mismo lenguaje. Una vez creada la plataforma, se realizarán diversas ejecuciones para poner a prueba su funcionamiento y a su vez analizar el comportamiento del algoritmo según ciertas variaciones en sus parámetros.

3. Descripción del programa.

El programa creado está formado por siete archivos de Python, tal y como se puede ver en la Figura 1. El primero se denomina ‘Programa’ y es el que ejecuta la aplicación en sí. A continuación, en ‘Interfaz’ se encuentra programada toda la parte de la interfaz gráfica, así

como las llamadas a las funciones del algoritmo. Por último, los cinco restantes corresponden a la implementación del algoritmo DQN para cada entorno.

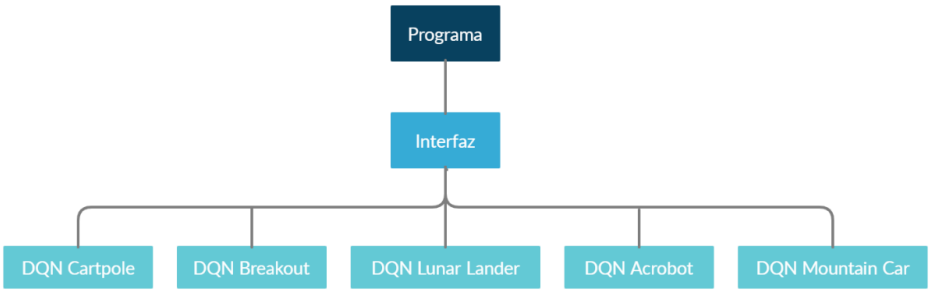


Figura 1: Estructura general de la aplicación

4. Resultados.

Por un lado, se muestra en las Figuras 2 la ventana principal de la aplicación creada, que cumple con todas las características deseadas.

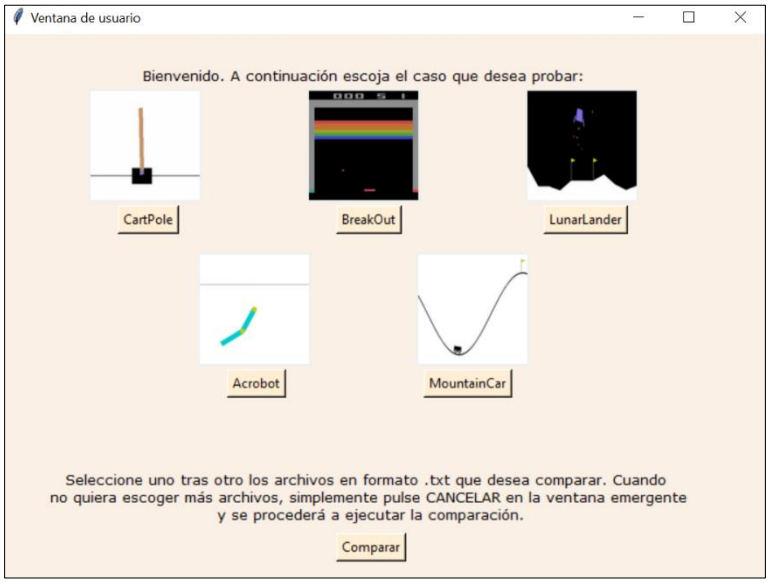


Figura 2: Ventana principal de la aplicación

Además, una vez desarrollado el programa, se analiza el desempeño del algoritmo en los diferentes entornos según los parámetros escogidos. A continuación, se muestran dos de los

resultados obtenidos para una ejecución de 1000 episodios con sus respectivos parámetros en los Acrobot (Figura 4) y LunarLander (Figura 3)

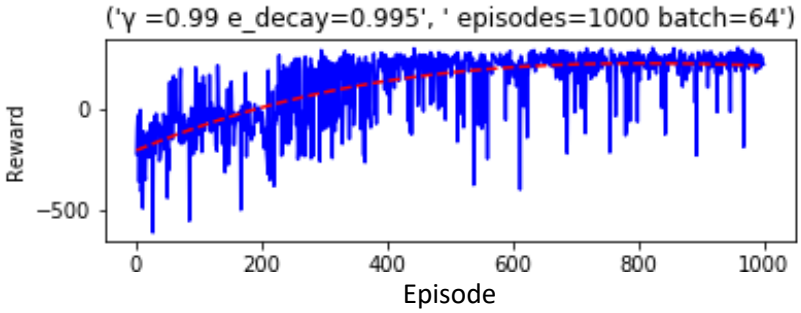


Figura 3: Resultado ejecución de 1000 episodios de Lunar Lander

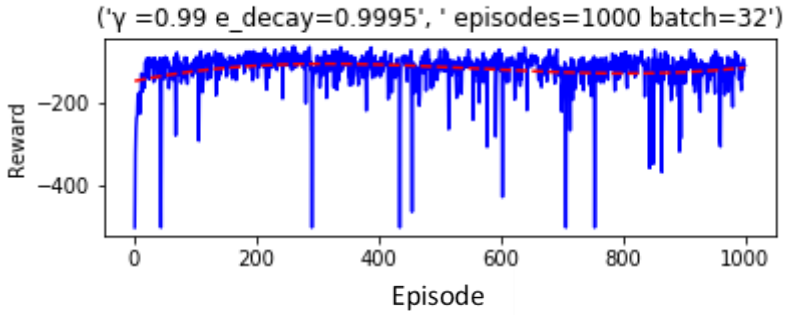


Figura 4: Resultado ejecución de 1000 episodios de Acrobot

5. Conclusiones

Se ha conseguido desarrollar la aplicación deseada, que tal y cómo se esperaba, resulta enormemente útil para aplicar el algoritmo DQN a una serie de entornos de Gym y sencilla de utilizar. Los resultados obtenidos de las ejecuciones han demostrado que el algoritmo cumple con su función, obteniendo un desempeño muy bueno en varios casos. Además, se han podido observar las notables diferencias según los valores otorgados a las variables, siendo capaces así de distinguir los parámetros que garantizan un mejor comportamiento del agente.

6. Referencias

[1] Sutton ,Richard S. and Barto Andrew G. (2014) *Reinforcement Learning: an introduction*, Cambridge, Massachusetts, The MIT Press.

[2] Caparrini, Fernando (2019). Fernando Sancho Caparrini. http://www.cs.us.es/~fsancho/?e=109.

[3] Anónimo. *Exploration Vs. Exploitation - Learning The Optimal Reinforcement Learning Policy*, Deeplizardz, https://deeplizard.com/learn/video/mo96Nqlo1L8

REINFORCEMENT LEARNING TECHNIQUES APPLIES TO GYM ENVIRONMENTS

Author: Alvargonzález Boulet, Pilar.
Supervisor: Sanz Bobi, Miguel Ángel.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

The aim of this project is to create a platform from which reinforcement learning algorithms can be easily applied to several Gym OpenAI environments. Furthermore, when the program is finished, it will be used to analyse the effects of the parametrization on the algorithm performance.

1. Introduction.

Reinforcement learning is an area of machine learning which aims to improve the performance of an agent when carrying out a specific activity by analysing the rewards obtained for each action performed. An algorithm that is commonly used in this field is the Deep Q-Learning or DQN. It works by assigning a numerical value to each pair of state and action, so that the higher this value is, the closer the agent will be to its objective. In addition, it uses deep neural networks to facilitate the calculation of these values when the environment has a wide variety of states or actions.

2. Definition of the project.

Nowadays several experiments where reinforcement learning algorithms are applied to Gym environments can be found. However, all of them contain complex codes focused on individual cases and with already defined parameters. The program created in this project eases the user's work since it allows to run different environments from the same platform, to modify effortlessly the parameters as desired and to compare the results obtained. Therefore, this tool is very useful not only to accelerate the application of the algorithm to different cases, but also it can be easily used by anyone with a basic knowledge on the field, without having to exhaustively analyse the code used.

In order to achieve the purpose of this project, we will implement the DQN algorithm in five different Gym environments, using Python as programming language. We will also use this language to develop the graphic interface of the platform. Once the program is finished, it will be used to run several cases in order to verify the proper functioning and to analyse the performance of the algorithm under different conditions.

3. Description of the program.

The program consists of seven Python files, as it is shown in Figure 1. The first one, which is called 'Programa', runs the whole app. Next, in 'Interfaz', the graphical interface is created and the algorithm functions are called. Finally, the remaining five files correspond to the implementation of the DQN algorithm for each environment.

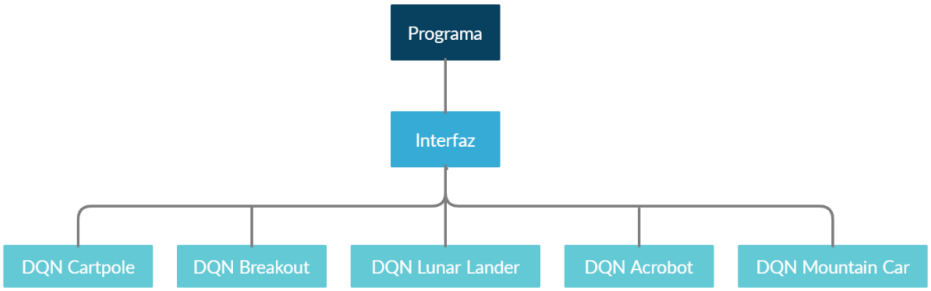


Figure 1: General structure of the program.

4. Results

In first place, Figure 2 shows the main window of the app developed, containing all the desired features.

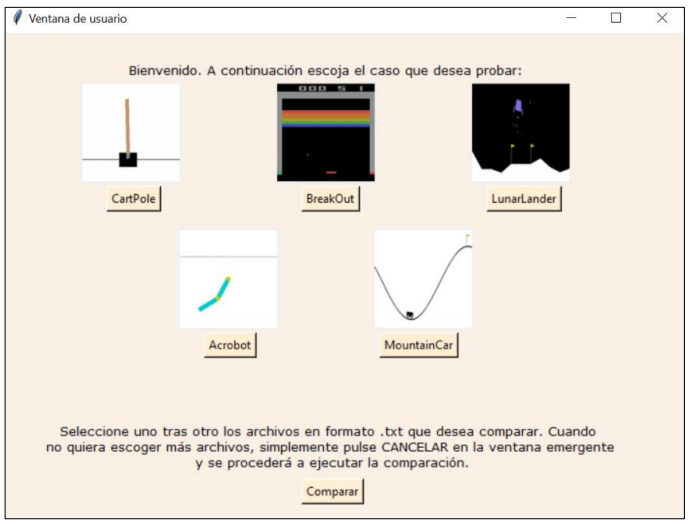


Figure 2: Main window of the platform.

Furthermore, once the program was finished, we have analysed the performance of the algorithm in different environments according to some chosen parameters. Below are shown

two of the results obtained for a run of 1000 episodes their respective parameters in Acrobot (Figure 4) and LunarLander (Figure 3)

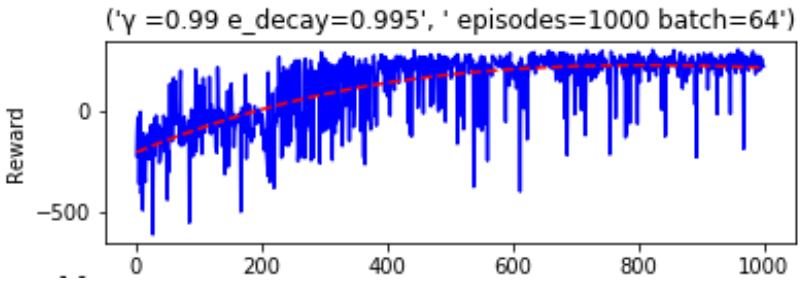


Figure 3: results of running 1000 episodes in Lunar Lander.

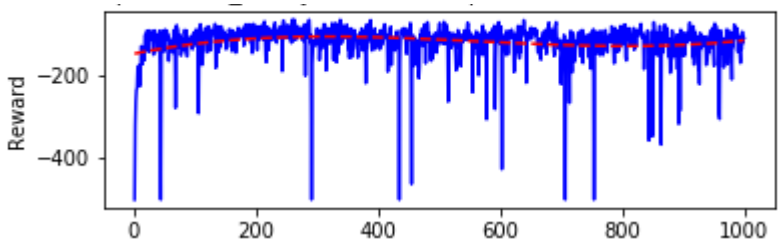


Figure 4: results of running 1000 episodes in Acrobot.

5. Conclusions

To sum up, the desired program has been developed, which is extremely useful for applying a reinforcement learning algorithm to Gym environments, as well as easy to use. The results obtained have generally shown good performance, highlighting some environments over others. In addition, we have noticed the important differences in the performance according to the values assigned to the variables. As a result of this, we have been able to distinguish the parameters that guarantee a better behaviour of the agent.

6. References

[1] Sutton ,Richard S. and Barto Andrew G. (2014) *Reinforcement Learning: an introduction*, Cambridge, Massachusetts, The MIT Press.

[2] Caparrini, Fernando (2019). Fernando Sancho Caparrini. http://www.cs.us.es/~fsancho/?e=109.

[3] Anónimo. *Exploration Vs. Exploitation - Learning The Optimal Reinforcement Learning Policy*, Deeplizardz, https://deeplizard.com/learn/video/mo96Nqlo1L8

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

APLICACIÓN DE TÉCNICAS DE APRENDIZAJE POR REFUERZO EN ENTORNOS GYM

Autor: Pilar Alvargonzález Boulet

Director: Miguel Ángel Sanz Bobi

Madrid

Índice de contenidos.

Índice de figuras.

Capítulo 1. Introducción

1.1 Aprendizaje por refuerzo

El ser humano siempre ha buscado la automatización de las tareas con el fin de optimizar sus resultados. Tras la aparición de la inteligencia artificial, no tarda llegar el aprendizaje automático, una parte de este campo dedicada a generar en un sistema informático la capacidad de aprender, es decir, desarrollar habilidades que no tenía inicialmente. Con ese fin, se analizan una serie de datos para reconocer patrones entre ellos y generalizar comportamientos.

El aprendizaje por refuerzo es una rama del aprendizaje automático que se centra en averiguar qué acciones se deben llevar a cabo para mejorar la destreza a la hora de realizar una actividad determinada. Para ello, se sigue el siguiente proceso:

- Existe un agente, que dispone de un conjunto finito de acciones (A) para interactuar con el entorno, y un conjunto finito de estados (S) en los que se puede encontrar.
- En cada instante de tiempo, t, el agente se encuentra en un estado concreto s_t, y ejecuta una acción determinada a_t.
- Para cada acción ejecutada, el entorno devuelve un nuevo estado s_{t+1} y una recompensa r_{t+1}. A esta recompensa se le otorga un valor numérico, siendo mejor cuanto mayor sea. Gracias a esto, el agente puede descubrir qué acciones le generan un mayor beneficio en cada caso, y así ir determinando el comportamiento adecuado según el estado en el que se encuentre.

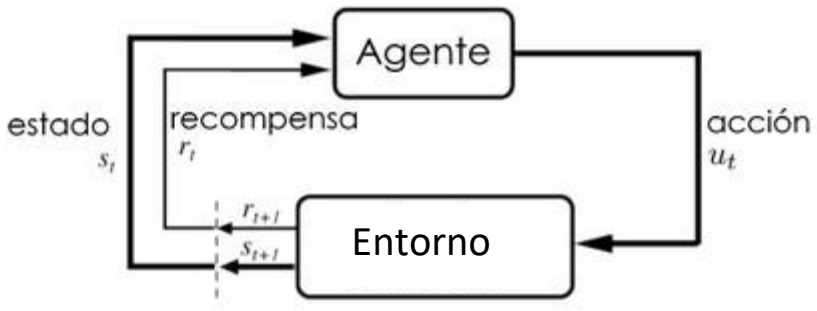


Figura 1: Diagrama aprendizaje por refuerzo

1.2 El algoritmo Q-learning

Uno de los múltiples algoritmos utilizados en el aprendizaje por refuerzo es el Q-learning, introducido en 1989 por Watkins. Esta técnica se basa en asignar un valor numérico a cada par estado y acción $Q(s, a)$, de manera que cuanto mayor sea este, más cerca estará el agente de conseguir su objetivo. Dicho valor no tiene en cuenta solamente la recompensa inmediata que se obtiene del entorno, sino que también se ve influenciado por las recompensas futuras (recompensas de nuevos estados que se alcanzan al ejecutar la acción *a* en el entorno *s*). De esta manera:

- En un estado determinado, el agente debe aprender a aplicar las acciones que generan mejores resultados y evitar las que producen resultados menos deseables.
- Si en un estado todas las acciones posibles generan resultados negativos, se debe evitar llegar a ese estado. Por lo tanto, no ejecutar acciones desde otros estados que nos lleven a este. Contrariamente, si se encuentra un estado altamente beneficioso, el agente debe intentar llegar hasta él.

Inicialmente, todos los valores de Q para los pares estado-acción se encuentran a 0. Conforme progresa la experiencia vivida por el agente a través de las acciones realizadas sobre el entorno y estados observados como consecuencia de esas acciones, se van sobrescribiendo sus valores. Así, se va confeccionando una tabla o matriz que contiene los valores de Q para cada posible combinación de estado y acción tras un proceso de aprendizaje. El agente recurre a esta tabla cuando quiere saber cuál es la mejor acción que puede realizar desde un estado determinado.

1.2.1 Ecuación del algoritmo

Para estimar matemáticamente los valores Q, se utilizaría la siguiente ecuación:

$$Q(s_t, a_t) = \ r(s_t, a_t) + \gamma \ max_{a_{t+1}}(s_{t+1}, a_{t+1})$$

Ecuación 1

Cuyas componentes son:

- $Q(s_t, a_t)$: valor de Q para el par de acción y estado en el instante actual t.
- $r(s_t, a_t)$: recompensa obtenida para el par de acción y estado en el instante actual t.
- γ : factor de descuento, determina el nivel de influencia de las recompensas futuras en el cálculo de Q.
- $max_{a_{t+1}}(s_{t+1}, a_{t+1})$: valor máximo correspondiente al mejor resultado que se puede obtener desde el próximo estado que se alcanza (instante t + 1) ejecutando la acción más beneficiosa para este.

Es decir, el valor Q para un par determinado es igual a la recompensa inmediata que se recibe del entorno, más una proporción γ del mejor valor que se puede obtener desde el siguiente estado que se alcanza.

Sin embargo, para evitar un cambio brusco en los valores de Q, y por tanto, conseguir que estos se vayan actualizando de forma gradual, se introduce un factor de aprendizaje α, dando lugar a la siguiente ecuación:

$$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha \left(r(s_t, a_t) + \gamma \, max_{a_{t+1}}(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)$$

Ecuación 2

De esta manera, *Q'* es el nuevo valor actualizado y *Q* el valor que existía anteriormente. El parámetro γ, también conocido como factor de descuento, es un valor comprendido entre 0 y 1, y establece la influencia de las recompensas futuras sobre el valor del par analizado. Por último, el parámetro α o factor de aprendizaje, determina el efecto de los nuevos valores de *Q* sobre los que existían anteriormente.

Q-learning se considera un algoritmo 'off policy', ya que la función Q(s,a) se va formando a base de realizar diferentes acciones, sin necesidad de seguir una política concreta. Por otro lado, existe el algoritmo SARSA (state – action – reward – state - action), considerado 'on policy' puesto que la función Q(s,a) se forma en base a las acciones realizadas siguiendo una determinada política.

1.2.2 Exploración vs explotación

Cuando el agente tiene que decidir qué acción llevar a cabo, puede seguir dos estrategias: exploración o explotación.

- Exploración consiste en ejecutar las acciones de manera aleatoria.
- Explotación consiste en seleccionar las acciones en base a la información que ya se conoce, es decir, escoger la acción con el mayor valor de Q.

Inicialmente, el agente desconoce totalmente el entorno, por lo que sus primeras acciones son escogidas mediante exploración. Según va descubriendo y almacenando los resultados de cada par acción-estado, va desarrollando la capacidad de reconocer situaciones ya experimentadas y explotar la información que posee para ejecutar la acción más beneficiosa.

Una de las principales políticas que se utilizan para controlar este proceso, se denomina ε - greedy. Se comienza con un valor de ε generalmente igual a 1, y a medida que se ejecutan acciones, se va multiplicando este valor por un ε _decay entre 0 y 1, de manera que ε va decreciendo. Para decidir si el agente realiza exploración o explotación, se genera un número aleatorio entre 0 y 1 y se compara con ε. Si el número aleatorio es menor que ε, se lleva a cabo una exploración del entorno, mientras que, si es mayor, se explota el conocimiento almacenado. De esta manera, se consigue que, cuantas más acciones se hayan ejecutado, menor sea el valor de ε y, por tanto, el agente tienda más a explotar la información recaudada en vez de explorar nuevas posibilidades.

1.2.3 Implantación de redes neuronales profundas.

El método del algoritmo Q-learning funciona muy bien para entornos simples, con una tabla de valores Q de tamaño moderado. Sin embargo, en entornos más complejos, estas tablas de valores adquieren dimensiones desmesuradas debido a la combinación de abundantes estados y/o acciones realizables, resultando inviable su utilización.

Para solucionar este problema, la compañía DeepMind pleanteó en 2014 la implementación de redes neuronales profundas para aproximar los valores Q, en lugar de utilizar las tablas mencionadas anteriormente. De esta manera se ve mejorado el desempeño del algoritmo, desarrollando así la técnica del Deep Q-Learning o DQN.

Las redes neuronales son una manera eficaz de aproximar funciones no lineales y están compuestas por una serie de capas conectadas entre sí que permiten obtener unas salidas a partir de unos datos de entrada, tal y como se puede observar en la Figura 2. Estos datos iniciales se reciben en la primera capa o 'capa de entrada', a continuación, pasan por una serie de capas ocultas donde se procesa la información para finalmente, acabar en una capa de salida. En el caso del Deep Q-learning, los datos de entrada son los componentes de los estados y las salidas son los valores de Q que se obtienen para las diferentes acciones que se pueden ejecutar. De esta manera, teniendo la información del estado en el que se encuentra el agente, la red neuronal aproxima el valor de Q que genera cada acción.

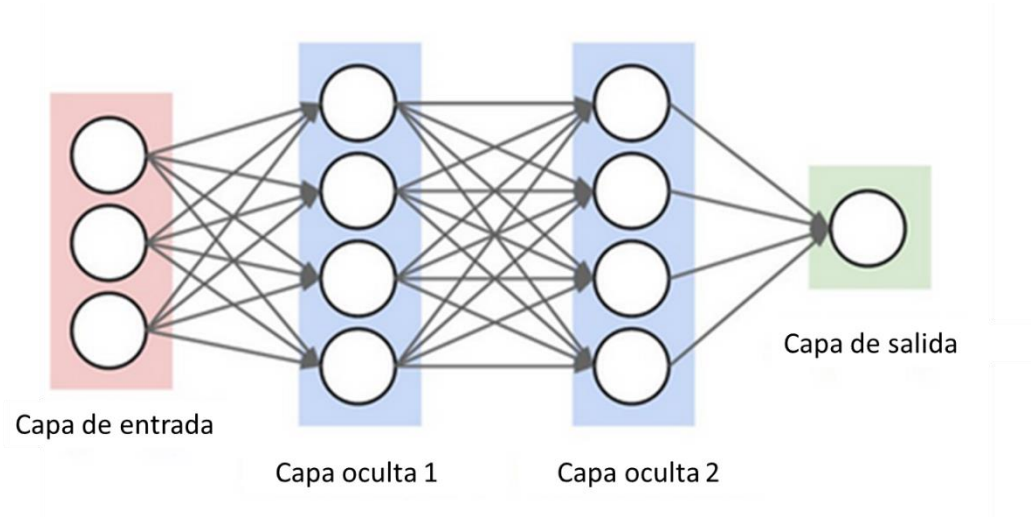


Figura 2: Diagrama estructura redes neuronales.

1.3 Motivación.

Una herramienta útil para desarrollar y probar los algoritmos de aprendizaje por refuerzo, es la plataforma Gym de OpenAI. Mediante sus juegos, se puede analizar el desempeño de estos algoritmos en entornos de mayor y menor complejidad.

El propósito de este proyecto es crear una plataforma desde la cual poder acceder rápidamente a diferentes juegos de Gym y cambiar fácilmente los parámetros de los algoritmos utilizados. Esto agilizaría no solo la aplicación del algoritmo, sino que también facilitaría la comparación de su eficiencia según los distintos parámetros y entornos. Todo ello sin olvidar que aumentaría enormemente la accesibilidad para el usuario.

Capítulo 2. Definición del trabajo.

2.1 Justificación.

Actualmente se puede encontrar en internet una gran variedad de algoritmos Deep Q-Learning aplicados a diferentes entornos Gym. Sin embargo, todos los códigos están desarrollados de manera individual y no se ha planteado una agrupación de estos para que el usuario pueda acceder fácilmente a diferentes tipos de entornos. Además, las variables del algoritmo están asignadas a un valor fijo, por lo que, cada vez que se quieran cambiar estos valores es necesario modificar el código, lo cual ralentiza el proceso y requiere un conocimiento previo de dicho código.

Vistos estos inconvenientes de los métodos existentes, resultaría tremendamente útil una aplicación que englobe varios entornos diferentes y permita al usuario parametrizar el algoritmo simplemente introduciendo los valores deseados. Por un lado, facilitaría el acceso a diferentes juegos desde un mismo menú. Asimismo, se podrían modificar cómodamente los valores de las variables sin tener que alterar los códigos existentes. Además, esta aplicación incluiría un sistema de comparación que permitiría contrastar los resultados obtenidos. Todo esto, sin olvidar que esta aplicación mejoraría enormemente la experiencia interactiva entre el usuario y los algoritmos.

2.2 Objetivos.

En este proyecto se pretende desarrollar una aplicación desde la cual se puedan aplicar algoritmos de aprendizaje por refuerzo a diferentes entornos de Gym. Para ello, se utilizará el algoritmo Deep Q-Learning, también conocido como DQN. Una vez desarrollada la plataforma, se pretende llevar a cabo un análisis y comparación de los resultados obtenidos según la parametrización. Además, en este proyecto se desea cumplir con alguno de los Objetivos de Desarrollo Sostenible establecidos por la ONU en el año 2015.

Los objetivos a cumplir se pueden agrupar de la siguiente manera:

Objetivos generales:

- Creación de una aplicación que permita aplicar el algoritmo DQN a diferentes entornos Gym.
- Comparación de resultados obtenidos en función de la parametrización.

Objetivos parciales:

- Implementación del algoritmo en los diferentes entornos.
- Desarrollo de la interfaz de la aplicación.
- Ejecución de los entornos variando los parámetros para analizar el desempeño del algoritmo.
- Cumplimiento de objetivos ODS de la ONU.

2.3 Metodología.

Para llevar a cabo este proyecto, se sigue una metodología expuesta en la *Figura 3*. En primer lugar, se realiza un estudio y comprensión del funcionamiento del algoritmo Q-Learning y su variante con redes neuronales. A continuación, utilizando el lenguaje de programación Python, se va llevando a cabo implementación del algoritmo en los diferentes entornos de Gym. Para ello, se debe de prestar atención a las características de cada entorno determinado, con el fin de llevar a cabo una correcta adaptación del algoritmo en cada caso. Paralelamente, también utilizando Python, se desarrolla la interfaz de la aplicación, en la cual se engloban a las funciones del algoritmo aplicado a cada entorno. Por último, una vez que todo funciona correctamente, se analizan los resultados con el fin de obtener unas conclusiones sobre el desempeño del algoritmo y los efectos de la parametrización.

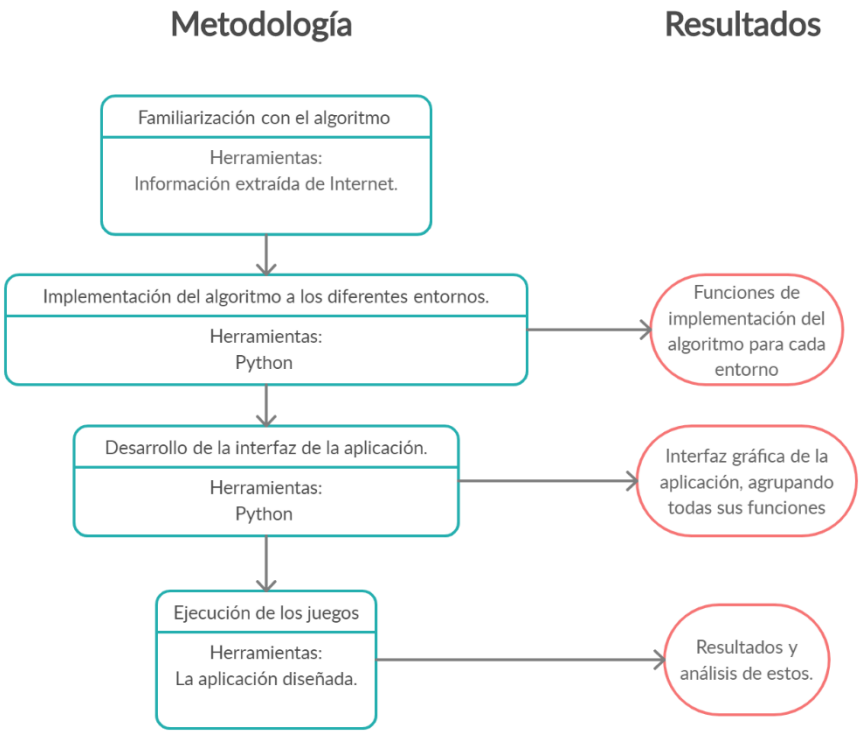


Figura 3: Esquema de la metodología.

Capítulo 3. Desarrollo de la aplicación.

3.1 Herramienta.

Para desarrollar la aplicación deseada, se utiliza un paquete de Python denominado Tkinter. Python ofrece varias opciones diferentes para diseñar interfaces gráficas de usuario, no obstante, Tkinter suele ser la más habitual ya que se encuentra dentro de su librería estándar y además permite crear de manera rápida y asequible una GUI. Es cierto que no sirve para crear interfaces excesivamente modernas, pues tiene un diseño bastante básico, sin embargo, cumple perfectamente con lo necesario para el desarrollo de la aplicación deseada.

El elemento fundamental de una interfaz gráfica en Tkinter es la "ventana". Las ventanas son la base sobre la cual se añaden todos los demás elementos de la GUI. Estos otros elementos son en su mayoría unas herramientas interactivas denominadas "widgets". Además, para agrupar y organizar los widgets dentro de la ventana, se utiliza un "marco".

Gracias a la interfaz gráfica, se ofrece una buena presentación de las opciones del programa para que el usuario pueda interactuar con el de una manera más intuitiva.

3.2 Estructura de la aplicación.

La primera ventana de la aplicación tiene dos utilidades distintas, tal y como se puede ver en la Figura 4. Por un lado, permite al usuario escoger entre los cinco casos posibles: Cartpole, Breakout, Lunar Lander, Acrobot y Mountain Car. Por otro lado, se ofrece también la opción de comparar casos ya guardados.

Al pulsar el botón de "Comparar", se van abriendo nuevas ventanas desde las cuales se permite al usuario seleccionar de uno en uno los casos deseados (Figura 5). Es importante destacar que, tal y como se menciona en la propia aplicación, los archivos seleccionados para comparar deben de ser de tipo '.txt',. Una vez escogidos todos los archivos que se quieren comparar, aparece un nuevo gráfico (Figura 6) en el que se superponen los distintos casos. Gracias a esta función de la aplicación, el usuario puede

observar las diferencias en el desempeño de casos con diferentes parámetros, y así descubrir qué valores resultan mejores o peores. Además, se da la opción de guardar el nuevo gráfico con la comparación.

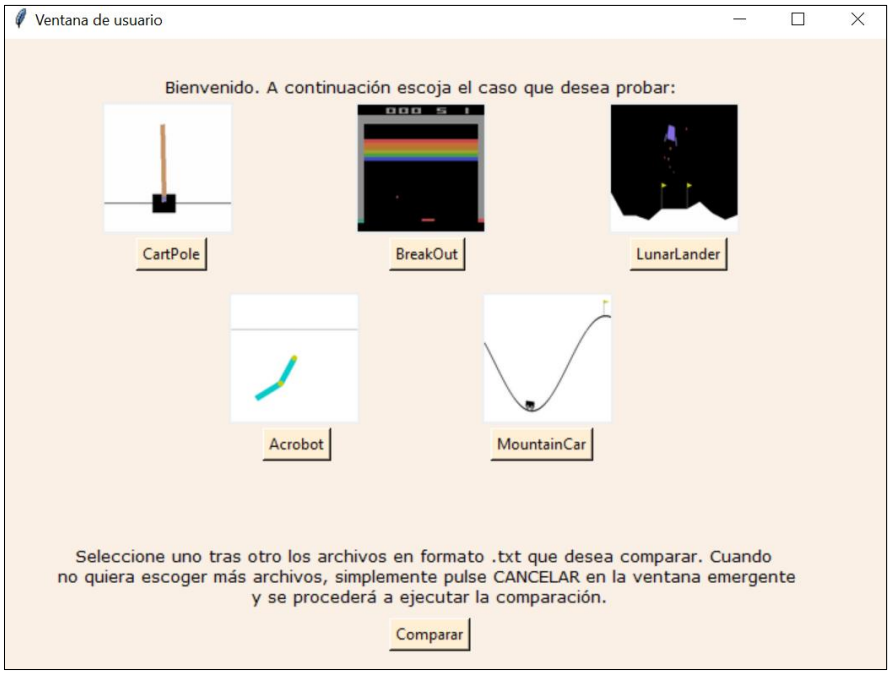


Figura 4: Ventana principal apliación.

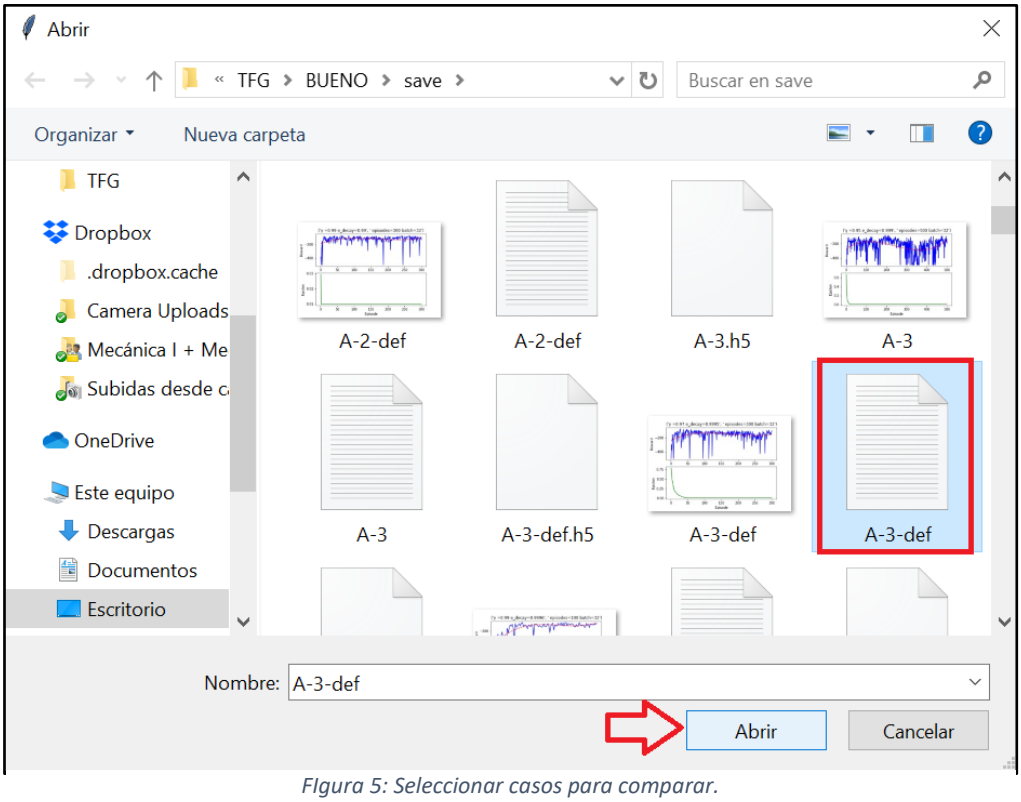


FIgura 5: Seleccionar casos para comparar.

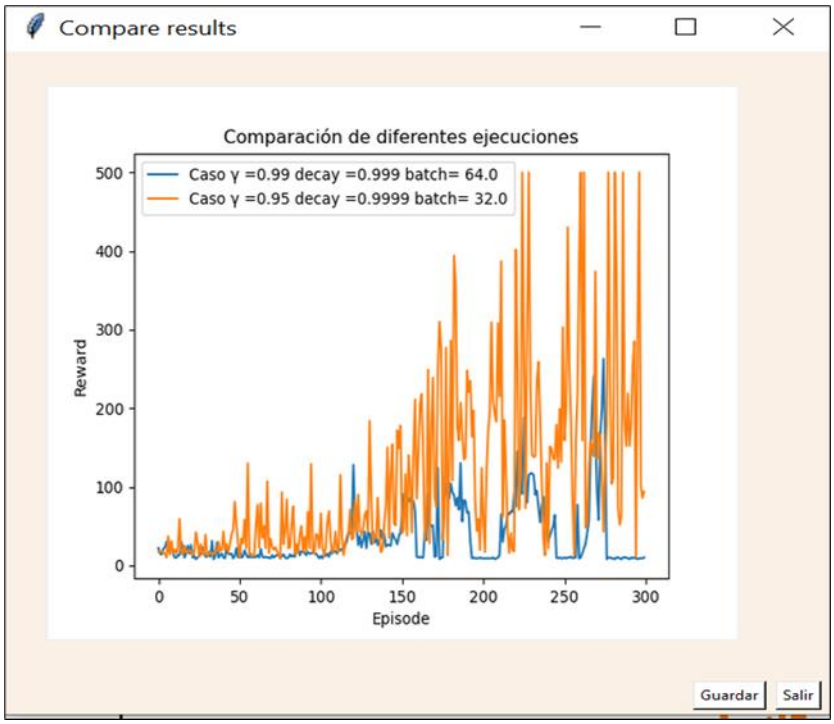


Figura 6: Comparación de casos.

Al pulsar cualquiera de los cinco casos posibles, se abre una nueva ventana con las siguientes opciones a escoger:

- Ayuda: se explica en qué consiste el juego y cuál es su funcionamiento, también se describen los parámetros que se deben introducir. (Figura 7)

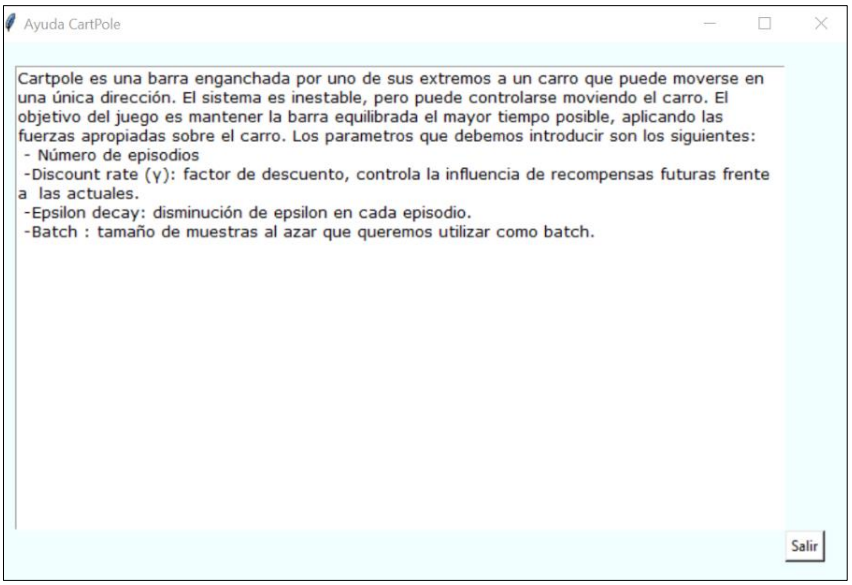


Figura 7: Ventana 'Ayuda' de la aplicación.

- Detalles: se describen los diferentes componentes del entorno (acciones, recompensas, estados). (Figura 8)

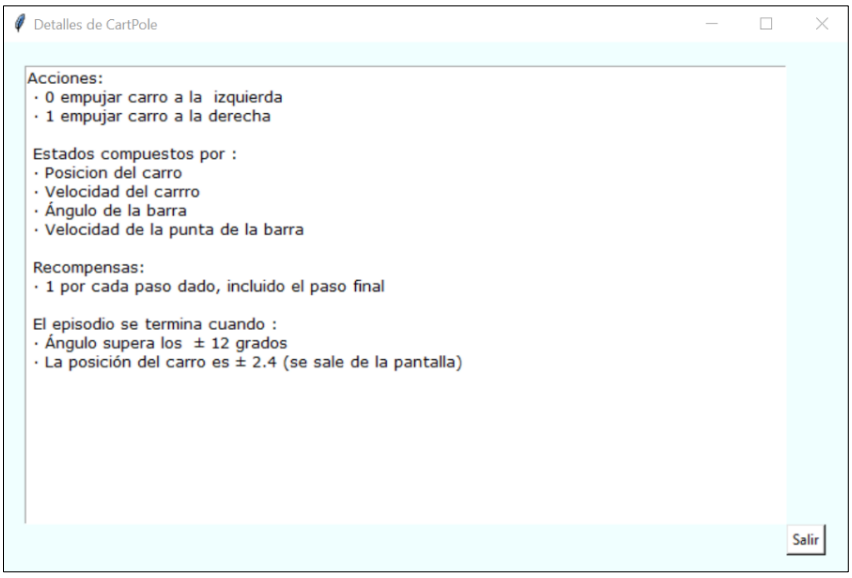


Figura 8: Ventana 'Detalles' de la aplicación.

- Parámetros: en esta opción es donde se da comienzo al juego. Para ello, se debe introducir primero los parámetros requeridos y a continuación pulsar "intro" en el teclado del ordenador. Además, se ofrece la opción de simplemente entrenar o de utilizar un modelo ya entrenado para jugar. También se incluye un botón de "Ayuda", que contiene la explicación del juego mencionada anteriormente. Por último, se puede escoger entre mostrar la simulación o no. (Figura 9)

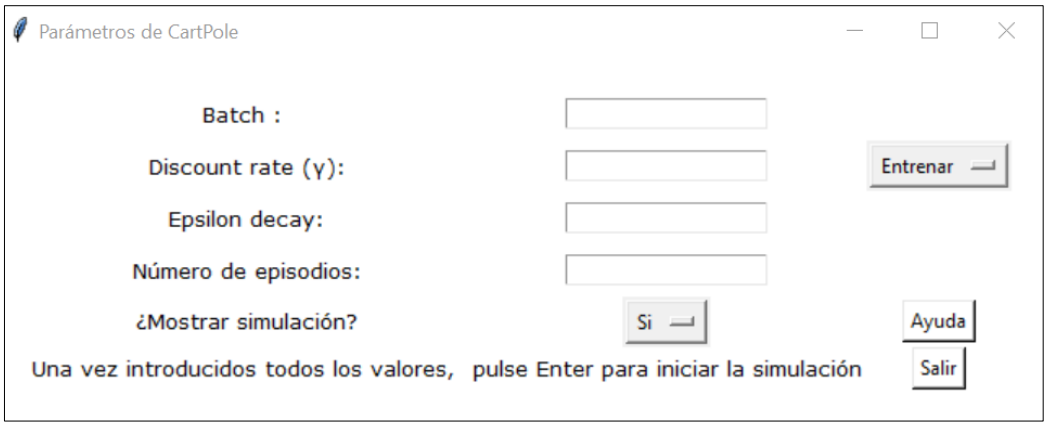


Figura 9: Ventana 'Parámetros' aplicación.

Los parámetros requeridos son los siguientes:

- Batch: tamaño de la muestra de experiencias guardadas que se utiliza para entrenar la red neuronal.
- Discount rate: factor de descuento (γ), valor comprendido entre 0 y 1 que establece la proporción en la que influyen las recompensas futuras en el cómputo de los valores de Q.
- Epsilon decay: ε_decay, proporción de épsilon (entre 0 y 1) que se mantiene tras cada acción realizada. Cuanto menor sea este valor, más rápido decrecerá ε y por tanto el agente llevará a cabo una menor exploración del entorno.
- Número de episodios: número de veces que se quiere repetir el juego. Cuantos más episodios, más conocimiento irá adquiriendo el agente sobre los efectos que tienen las acciones en el entorno para los diferentes estados.

3.3 Estructura del código.

Para la elaboración de la interfaz deseada, se comienza creando una ventana principal, que a su vez contiene un marco, dentro del cual se encuentran los widgets. En este caso, los widgets utilizados son de tipo “button” o botón para seleccionar las opciones disponibles y de tipo “label” o etiqueta para incluir ciertas líneas de texto explicativo. Al pulsar cada botón, se llama a una función que abre una nueva ventana. De esta manera, tal y como se puede ver en la *Figura 10*, la interfaz está compuesta por seis funciones principales: los cinco casos y la comparación.

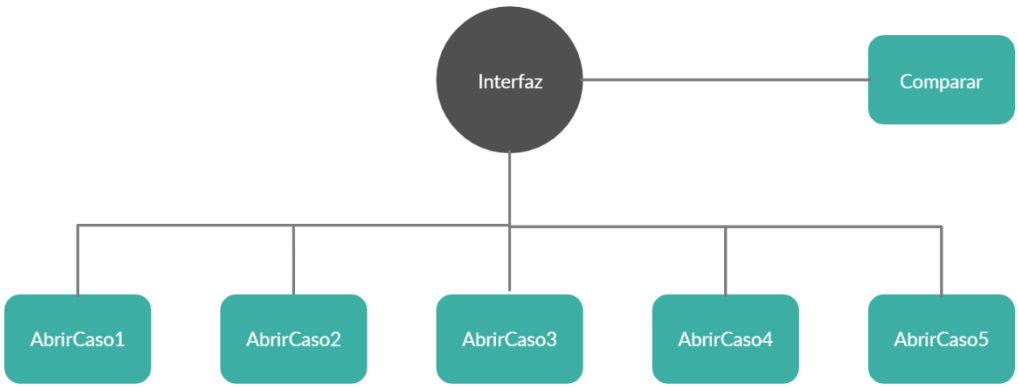


Figura 10: Diagrama estructura general del código de la interfaz

La función de comparación recibe los datos de ciertos archivos seleccionados por el usuario para comparar y elabora una gráfica superponiendo los casos. Además, contiene un nuevo botón “Guardar”, que llama a la función Guardar imagen, permitiendo al usuario guardar la gráfica obtenida en el directorio que desee.

Las otras cinco funciones poseen todas la misma estructura, tres botones correspondientes a las funciones: Parámetros, Ayuda y Detalles. La función Ayuda presenta únicamente un widget de tipo “Text” en el cual se incluyen ciertas explicaciones sobre el funcionamiento del juego. De igual modo, la función Detalles contiene un widget de tipo “Text” para presentar las características principales del entorno, tales como acciones, componentes de los estados, recompensas. La función parámetros incorpora dos nuevos tipos de widgets, “Entry” para permitir al usuario introducir texto (en este caso los valores de los parámetros) y “StringVar” para permitir al usuario escoger entre varias opciones predeterminadas. Mediante una acción del teclado, se activa la función Guardar, en la cual se recogen los datos de los widgets anteriores y se transforman en variables que se utilizan como entradas para la función DQN correspondiente del entorno. También se incluye la función Guardar Imagen, para permitir al usuario guardar las gráficas generadas una vez el juego finaliza todos sus episodios. Todo esto puede verse mejor en el esquema de la *Figura 11*.

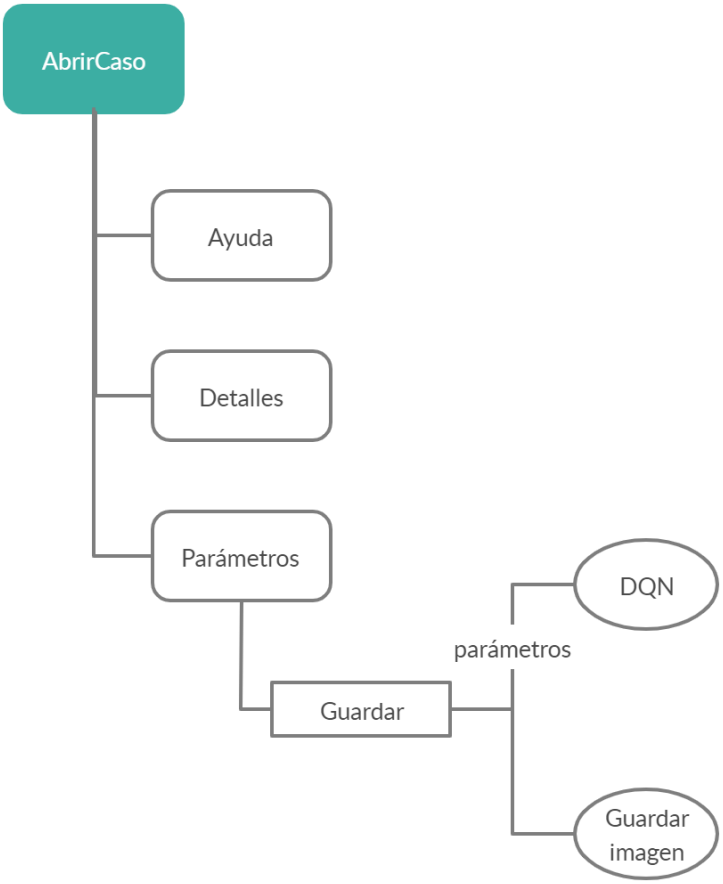


Figura 11: Diagrama estructura del código de cada función 'AbrirCaso'.

La aplicación cuenta con 7 archivos de Python, tal y como se muestra en la *Figura 12*: el de mayor jerarquía, se denomina 'Programa' y desde él se ejecuta 'Interfaz'. A su vez, desde 'Interfaz' se llama a cada función de implementación del algoritmo DQN según el caso escogido por el usuario.

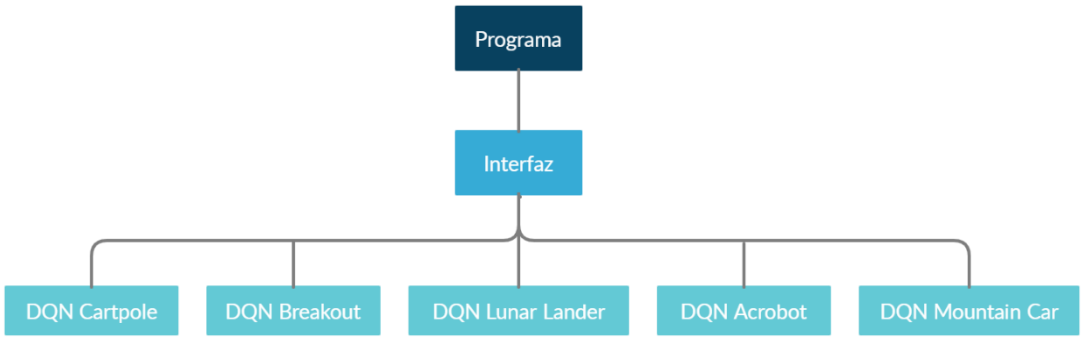


Figura 12: Diagrama estructura archivos del programa

De esta manera, para iniciar la aplicación, basta con ejecutar el archivo de Python denominado ‘Programa’. Para su funcionamiento correcto, es necesario tener instalado un editor de Python y los siguientes paquetes de Python:

- Gym (Version complete)
- Keras
- TensorFlow

Además, en el archivo de la aplicación se encuentra la carpeta ‘save’ para guardar los casos ejecutados y la carpeta ‘imágenes programa’, que almacena las imágenes utilizadas en la interfaz.

Capítulo 4. Implementación del algoritmo DQN para los diferentes casos

Mediante la aplicación desarrollada en este proyecto, se pretende aplicar el algoritmo DQN a cinco entornos diferentes de gym. Debido a las diferentes características y nivel de complejidad de los entornos, la implementación del algoritmo sufrirá ciertas variaciones según se trate de un caso u otro.

4.1 Cartpole.

4.1.1 Definición y características del entorno.

Cartpole es un juego perteneciente a la familia de "Classic Control" y, tal y cómo se muestra en la *Figura 13*, está formado por una barra unida mediante una articulación a un carro, el cual solamente puede moverse horizontalmente. El objetivo del juego es mantener la barra en posición vertical el mayor tiempo posible. El sistema se encuentra inicialmente en equilibrio, pero es inestable, por lo que el carro debe de moverse a izquierda o derecha para controlarlo.

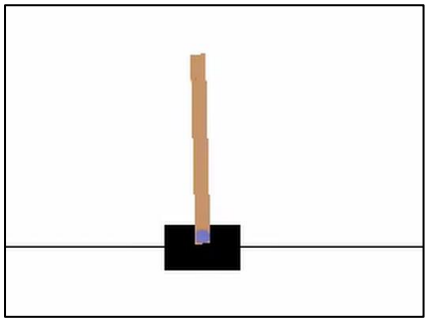


Figura 13: Imagen Cartpole

Este entorno cuenta con las siguientes características:

Acciones.

- Empujar el carro a la izquierda.
- Empujar el carro a la derecha.

Componentes de los estados.

- Posición en eje x del carro.
- Velocidad en x del carro.
- Ángulo de la barra con la vertical.
- Velocidad de la punta de la barra.

Recompensas.

- +1 por cada paso dado, incluido el paso final

Motivos de finalización del episodio.

- La barra sobrepasa un ángulo de ±15 grados con la vertical.
- La posición del carro es de ±2.4 unidades con respecto al centro (se sale de la pantalla).

La máxima recompensa que se puede obtener son 300 puntos, ya que es el número máximo de pasos fijado.

4.1.2 Estructura del código

El código cuenta con una función principal denominada ‘main’, la cual recibe como entradas los parámetros recogidos por la aplicación. Desde ella, se extraen los componentes de entorno, se crea el agente perteneciente a la clase DQNAgent y se ejecuta cada uno de los episodios.

La clase DQNAgent cuenta con una serie de funciones y métodos que implementan todas las partes necesarias para que el algoritmo funcione correctamente, tal y como se puede ver en la *Figura 14*. En primer lugar, se encuentra el constructor donde se inicializan todos los parámetros importantes a tener en cuenta para el desarrollo del algoritmo.

A continuación, se encuentra la función ‘build_model’, que crea la red neuronal de convolución. Para ello, se toman como entradas de la red los componentes de los estados y, tras pasar una serie de capas ocultas, se obtiene como salida las “puntuaciones” correspondientes para las acciones izquierda o derecha. Se utiliza un factor de aprendizaje de 0.001. No obstante, las redes neuronales no almacenan

debidamente la información, ya que la sobrescriben con las nuevas experiencias. Es por ello que se necesita una función que facilite el guardado de datos en la memoria para cada paso que se da. Se utilizará para ello un método denominado ‘remember’ que almacena para cada ejecución de una acción la siguiente información: estado, acción, recompensa, siguiente estado, y un valor booleano que indica si se trata de un estado terminal o no.

Para entrenar la red neuronal con las experiencias almacenadas en la memoria, se añade un método denominado ‘replay’. En este método, se comienza por seleccionar una muestra aleatoria de elementos almacenados en la memoria, del tamaño determinado por el parámetro *batch size.* A continuación, se hace uso de una adaptación en Python de la ecuación de Q-learning para entrenar la red neuronal. Para ello, se llama a otra función denominada ‘get attribute’, que extrae los datos de dicha muestra aleatoria.

También es necesaria la existencia de una función que controle la toma de decisiones del agente a la hora de actuar. Esta función se denomina ‘act’ y en ella se sigue la estrategia mencionada anteriormente, que consiste en comparar el valor de épsilon con un número aleatorio. Si épsilon es mayor, la función devuelve simplemente una acción al azar del conjunto de acciones disponibles. Por el contrario, si épsilon es menor, se hace una predicción mediante la red neuronal de la recompensa obtenida para cada una de las acciones y se devuelve aquella que tenga el valor más alto.

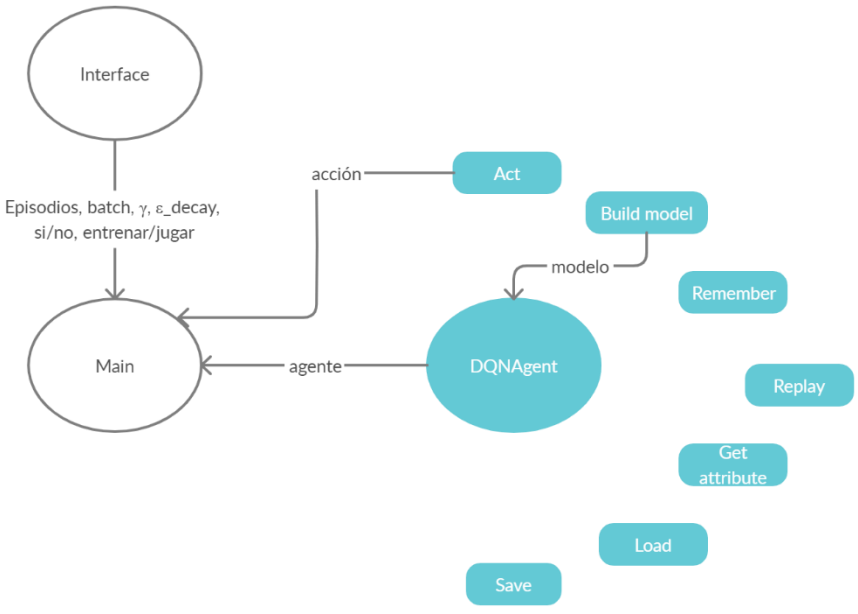


Figura 14: Diagrama estructura código Cartpole

Por último, se añaden los métodos ‘save’ y ‘load’, cuya función consiste respectivamente en guardar o cargar los modelos entrenados. La operación de guardado se ejecuta automáticamente, no obstante, el modelo guardado por defecto se va sobrescribiendo con cada partida. Para guardar definitivamente un modelo concreto, el usuario debe de seleccionar esta opción desde la ventana pertinente de guardado de la interfaz. Por el contrario, la operación de cargar un modelo ya existente sólo tiene lugar cuando la función principal recibe como uno de los parámetros de la interfaz la cadena de caracteres ‘Jugar’. De esta manera, no se va creando un modelo propio según las experiencias del agente, sino que se hace uso de la información ya recogida en el modelo escogido.

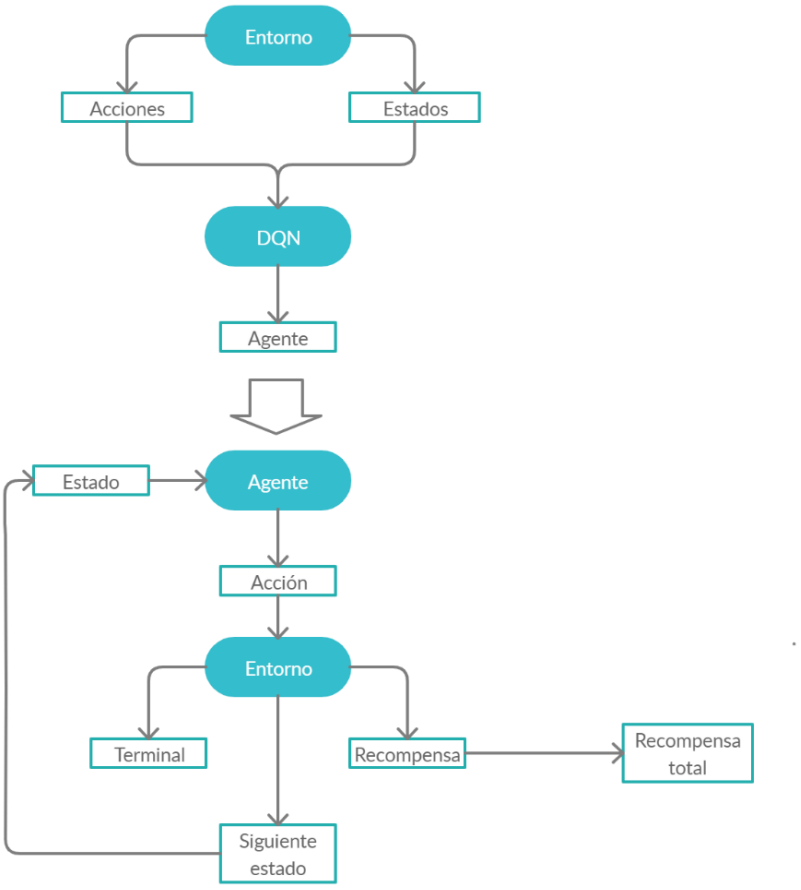


Figura 15: Diagrama estructura código de la función ‘main’ para Cartpole

El funcionamiento de la función principal o ‘main’, mostrado en la *Figura 15*, es el siguiente: para cada paso, se comienza extrayendo la acción que ejecutará el agente para el estado determinado en que encuentra. A continuación, se aplica esta acción al entorno

y se obtienen los resultados pertinentes (recompensa, estado futuro y acción terminal o no) que posteriormente se almacenan en la memoria del agente. Por último, se actualiza el estado al estado futuro y se repite este proceso hasta llegar a un estado terminal o al límite establecido de duración. Además, para cada episodio se muestra en la pantalla el valor de la recompensa obtenida, así como el épsilon actual.

4.2 Breakout.

4.2.1 Definición y características del entorno.

Breakout es un juego perteneciente a la rama de “Atari”. Tal y como se puede ver en la *Figura 16*, este consiste en una capa de ladrillos que se encuentra en la parte superior de la pantalla y que deben de ser destruidos para cumplir con el objetivo. Para ello, se utiliza una pelota cuya trayectoria se controla mediante sus rebotes en una barra móvil y en las paredes. Esta barra puede moverse únicamente a izquierda y derecha. Además, en cada episodio el agente cuenta con 5 vidas, las cuales se van consumiendo cada vez que la barra no alcanza la pelota y por tanto esta se escapa por la parte inferior de la pantalla.

Figura 16: Imagen Breakout

El entorno cuenta con las siguientes características:

Acciones

- No hacer nada.
- Mover la barra hacia la derecha.
- Mover la barra hacia la izquierda.
- Disparar la pelota (comienza el juego).

Componentes de los estados.

- Posición de la barra en el eje x.

- Posición de la bola en el eje x.
- Posición de la bola en el eje y.
- Velocidad de la bola en el eje x.
- Velocidad de la bola en el eje y.
- Número de ladrillos restantes.
- Número de vidas restantes.
- Puntuación actual del juego.

Recompensas.

- +1 punto por cada ladrillo roto.

Motivos de finalización del episodio.

- Se acaban las vidas.

4.2.2 Estructura del código.

En este caso, se mantiene una estructura del código similar al anterior, pero con ciertas modificaciones, tal y como se puede observar en la *Figura 17*. Al igual que en Cartpole, el código cuenta con una función principal denominada 'main', desde la cual se extraen los componentes del entorno, se crea el agente de la clase "DQNAgent" y se ejecuta cada episodio.

Los juegos pertenecientes a la familia Atari están compuestos por defecto unas imágenes de 210 x 160 píxeles con una amplia paleta de colores. Esto resulta bastante exigente a nivel computacional, por lo que es necesario aplicar un preprocesamiento. Para ello es necesaria una función denominada 'preprocess' que modifica la observación para convertir su representación RGB a escala de grises y reducirla a 84 x 84 píxeles.

En cuanto a la clase DQNAgent, esta contiene una serie de funciones y métodos que ejecutan las partes necesarias para el correcto funcionamiento del algoritmo Deep Q-learning. En primer lugar, se encuentra el constructor, en el cual se inicializan todos atributos y parámetros requeridos para la implementación del algoritmo. Se crea la red

neuronal, la red neuronal objetivo y se va entrenando la primera para ir optimizándose y acercándose más a la segunda.

Al igual que en el caso anterior, es necesaria una función que cree la red neuronal. Esta se denomina 'build_model' y elabora un modelo cuyas entradas son las componentes de los estados y como salidas se obtienen las "puntuaciones de bondad" de cada acción. Con el fin de entrenar la red neuronal, se crea la función 'build_training_op'. En ella se recibe como entrada los valores de peso de la red neuronal y se obtiene el error actual de la predicción de la red.

Además, esta clase cuenta con una función denominada 'setup_summary', cuyo trabajo consiste en crear un sumario con variables importantes de cada episodio tales como la recompensa total, duración, valor de Q y error.

Para que el agente escoja que acción ejecutar en cada momento, se implementa la función 'get_action'. Su funcionamiento es igual que para el caso anterior: se genera un número aleatorio entre 0 y 1 y se compara con épsilon. Si épsilon resulta ser mayor, se ejecuta una acción al azar y en el caso contrario se escoge la acción con mayor valor de Q para el determinado estado.

Para entrenar el modelo se introduce un método denominado 'train_model'. Para ello, se comienza por extraer una muestra aleatoria de datos de la memoria de tamaño especificado por el parámetro *batch*. A continuación, se extrae la información de cada set de datos de esta muestra (estado, acción, recompensa, siguiente estado y terminal o no) y mediante una adaptación al lenguaje de python de la ecuación del algoritmo Q-learning se lleva a cabo el entrenamiento del modelo.

Otra función interesante que se añade a esta clase es la función 'task'. En ella se lleva a cabo la tarea de almacenamiento en la memoria de los datos, se llama a la función de entramiento, se actualiza la red objetivo, se escribe el sumario y se muestra en pantalla los valores importantes de cada episodio, tales como la duración, la recompensa total o épsilon. Además, esta función devuelve el siguiente estado y la recompensa total.

También se incorpora una función denominada 'get_inital_state' cuya función es procesar y extraer el estado en el que se iniciará el aprendizaje.

Por último, se incluye el método ‘save’, cuya función es guardar automáticamente el modelo una vez finalizado el último episodio. No obstante, el modelo guardado por defecto se va sobrescribiendo con cada nueva partida. Por lo tanto, para guardar definitivamente un modelo, el usuario debe de escoger esta opción en la ventana de guardado y así almacenarlo con el nombre que desee en el directorio escogido. Además, el programa cuenta también con un método denominado ‘load’ cuya función es precisamente cargar modelos ya existentes. Este no se ejecuta automáticamente, sino únicamente cuando la función principal recibe como uno de los parámetros de la interfaz la cadena de caracteres ‘Jugar’. En este modo, el agente no realiza una exploración del entorno partiendo de cero, sino que se recurre directamente a la información ya almacenada en el modelo escogido.

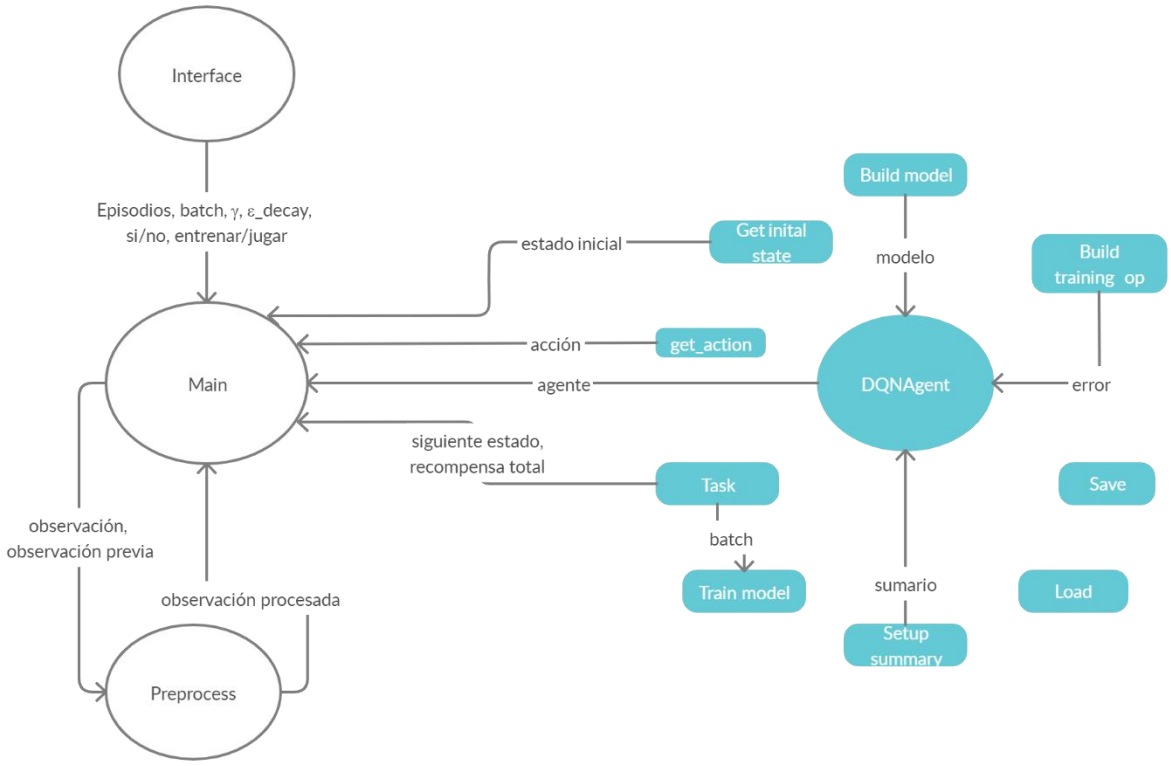


Figura 17: Diagrama estructura código Breakout

Por último, el funcionamiento de la ejecución de cada episodio en la función principal resulta bastante parecido al caso de Cartpole, pero incorporando algunas modificaciones (*Figura 18*). Para mejorar el proceso de aprendizaje, se añade al comienzo de cada episodio un pequeño filtro que considera que las primeras 15 observaciones de la partida son completamente al azar, por lo que no se tienen en cuenta. A causa de esta

incorporación, se necesita la función ‘get_inital_state’ para identificar el estado que se considera como inicial para comenzar con el aprendizaje. Posteriormente, se extrae la acción que ejecutará el agente en el estado determinado. A continuación, se ejecuta esta acción en el entorno para obtener los resultados pertinentes (recompensa, observación y acción terminal o no) y se aplica el preprocesamiento a la observación obtenida. Esta nueva observación ya procesada, se introduce junto con los demás datos en la función ‘task’, obteniendo como salida el nuevo estado y la recompensa total. Por último, se actualiza el valor de la variable estado con el recién obtenido, para partir de este en el siguiente paso. Este proceso (desde que se extrae la acción hasta el final) es un bucle que se repite para cada paso que se da en el episodio hasta llegar a un estado terminal.

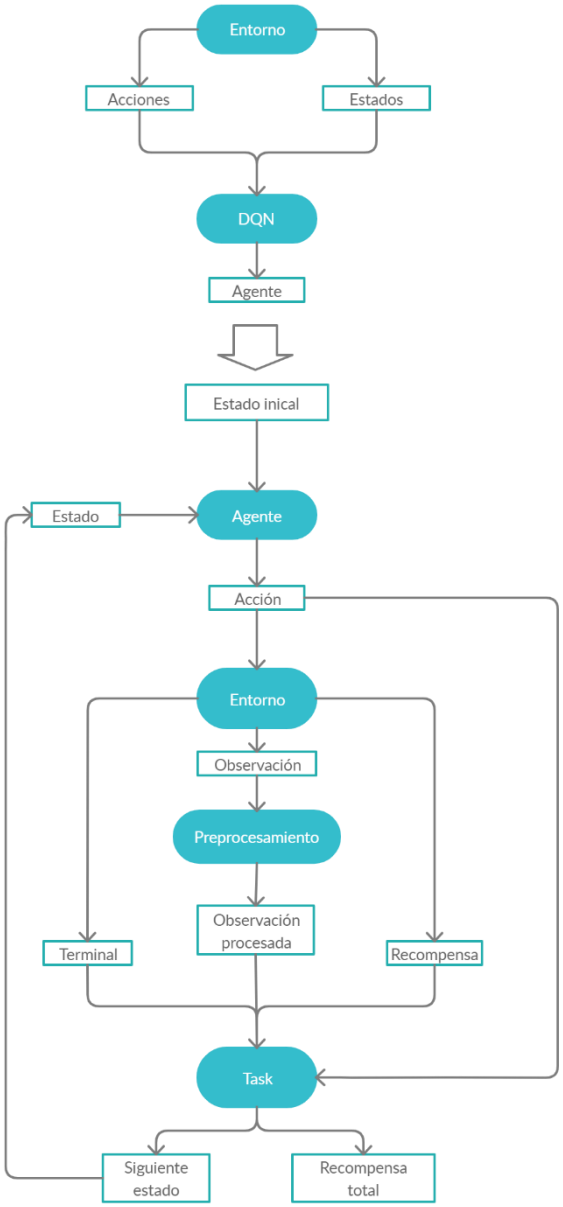


Figura 18: Diagrama estructura código de la función ‘main’ para Breakout

4.3 Lunar Lander.

4.3.1 Definición y características del entorno.

Lunar Lander es un juego perteneciente a la familia de “Box 2D”. Consiste en una nave espacial con tres aceleradores (dos a los lados y uno en la parte inferior) que permiten controlar su dirección. El objetivo del juego es conseguir que la nave aterrice correctamente siguiendo el camino marcado por el espacio entre dos banderas situadas en el suelo, tal y como se puede observar en la *Figura 19.*

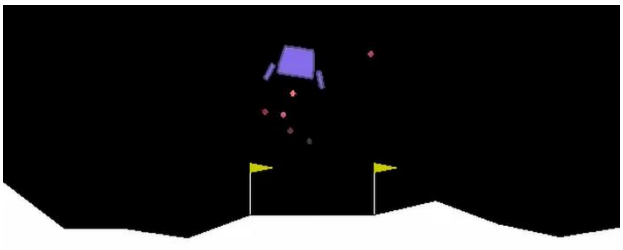


Figura 19: Imagen LunarLander

El entorno cuenta con las siguientes características:

Acciones.

- No hacer nada.
- Activar motor izquierdo.
- Activar motor inferior.
- Activar motor derecho.

Estados compuestos por:

- Posición en x de la nave.
- Posición en y de la nave.
- Velocidad en x de la nave.
- Velocidad en y de la nave.
- Ángulo de la nave con la horizontal.
- Velocidad angular de la nave.
- Contacto con tierra de pata ambas patas (valor booleano igual a 1 en caso de contacto y 0 en caso contrario)

Recompensas.

- +100 puntos si la nave aterriza correctamente.
- -100 puntos si la nave no aterriza correctamente (se estrella).
- -0.3 puntos cada vez que se activa el motor inferior.
- -0.03 puntos cada vez que se activa uno de los motores laterales.
- +10 puntos contacto de cada pata con tierra.

Motivos de finalización del episodio:

- La nave aterriza.
- La nave se estrella.

En este caso no existe un límite de puntuación máxima, pero se considera una victoria a partir de 200 puntos.

4.3.2 Estructura del código,

En este caso se utiliza prácticamente la misma estructura que en Cartpole para realizar la implementación del algoritmo DQN, ver *Figura 14* . Únicamente es necesaria una ligera modificación en cuanto a la red neuronal que se utiliza: debido a la mayor complejidad de este entorno, se aumenta considerablemente el número de nodos que contienen las capas ocultas.

En cuanto al resto del código, se mantienen igual todos los componentes que en el caso inicial, ver *Figura 9*. El programa cuenta con una función principal, desde la cual se extraen los componentes del entorno, se crea un agente de clase DQNAgent y se lleva a cabo la ejecución de cada episodio.

4.4 Acrobot.

4.4.1 Definición y características del entorno.

Acrobot es un juego de gym que pertenece a la familia de “Classic Control” y consiste en un péndulo formado por dos barras unidas por una articulación, tal y como se puede observar en la *Figura 20*. El objetivo del juego es conseguir que la barra exterior supere una altura determinada. Para ello, se ejercen una serie de torques que producen un balanceo del péndulo.

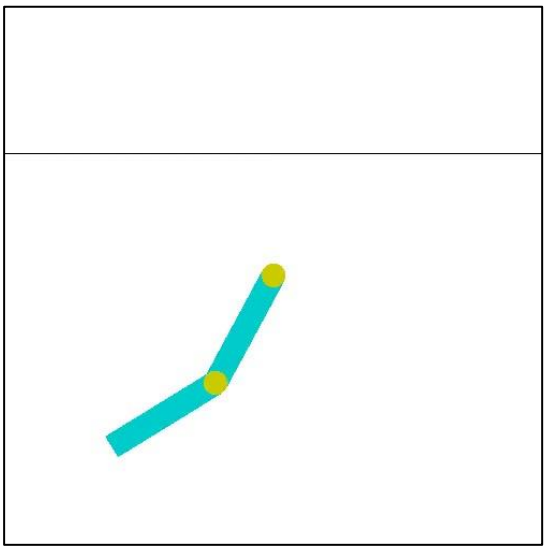


Figura 20: Imagen Acrobot

El entorno cuenta con las siguientes características:

Acciones

- Aplicar torque en sentido horario en la articulación que une ambas partes.
- Aplicar torque en sentido antihorario en la misma articulación.

Estados compuestos por:

- Seno del primer ángulo (barra interior con la normal).
- Coseno del primer ángulo.

- Seno del segundo ángulo (barra exterior con barra interior, por lo que es relativo al primero).
- Coseno del segundo ángulo.
- Velocidad angular de la primera articulación.
- Velocidad angular de la segunda articulación.

Recompensas.

- -1 por cada paso que no sea terminal, es decir, valor negativo del número de pasos que tarda en conseguir el objetivo.

Motivos de finalizaicón del episodio:

- La barra exterior sobrepasa la altura deseada.

4.4.2 Estructura del código.

Para implementar el algoritmo DQN en este caso, se utiliza el mismo código desarrollado para Cartpole, ver *Figuras 14 y 15*. El programa cuenta con una función principal o ‘main’, desde la cual se obtienen los componentes del entorno, se crea un agente de la clase DQNAgent y, a continuación, se ejecuta cada uno de los episodios.

4.5 Mountain Car

4.5.1 Definición y características del entorno.

Mountain car es un juego perteneciente a la familia de “Classic Control” y consiste en un coche que se encuentra atrapado entre dos colinas, tal y como se puede ver en la *Figura 21*. El objetivo del juego es conseguir que el coche llegue a una meta situada en lo alto de la colina derecha, para lo cual, debe de impulsarse mediante pequeños movimientos hacia la izquierda o derecha, ya que el motor no es lo suficientemente potente.

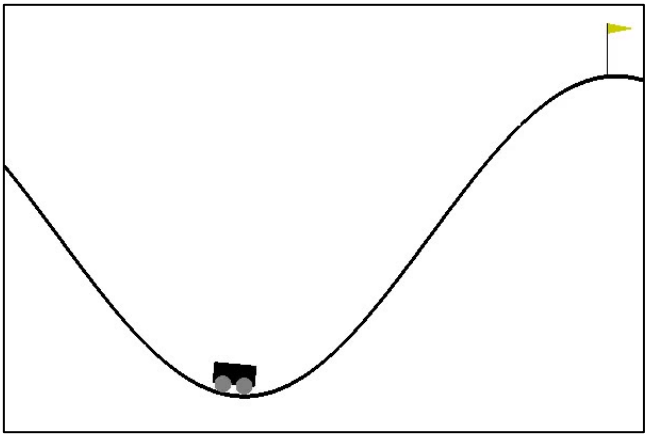


Figura 21: Imagen Mountain Car.

Este entorno cuenta con las siguientes características:

Acciones

- Aplicar una unidad de fuerza en la dirección izquierda del coche.
- No aplicar ninguna fuerza.
- Aplicar una unidad de fuerza en la dirección derecha del coche.

Componentes de los estados.

- Posición del coche en el eje y.
- Velocidad del coche.

Recompensas.

- +100 puntos si el coche llega a la meta al finalizar la partida.

- -200 puntos si el coche no llega a la meta al finalizar la partida.

Motivos de finalización del episodio.

- El coche llega a la meta.

4.5.2 Estructura del código.

Para implementar el algoritmo DQN en Mountain Car, se utiliza un código prácticamente igual al desarrollado para Cartpole. No obstante, este caso requiere una ligera modificación con respecto a la manera de obtener las recompensas, tal y como se puede ver en la *Figura 22*. Esto se debe al sistema por defecto del entorno, que se limita a otorgar una puntuación positiva únicamente cuando el coche alcanza la meta. Por tanto, el agente no recibe una recompensa por cada acción que ejecuta y, en consecuencia, no es capaz de distinguir cuales son más y menos acertadas. Por esta razón se introduce una función denominada 'get_reward', que devuelve una cierta recompensa en función de la altura alcanzada por el coche. Gracias a esta modificación, el agente obtendrá una mayor puntuación cuanto más alto llegue (un máximo de 150 puntos si llega a la meta) y así podrá identificar los movimientos que resultan más beneficiosos.

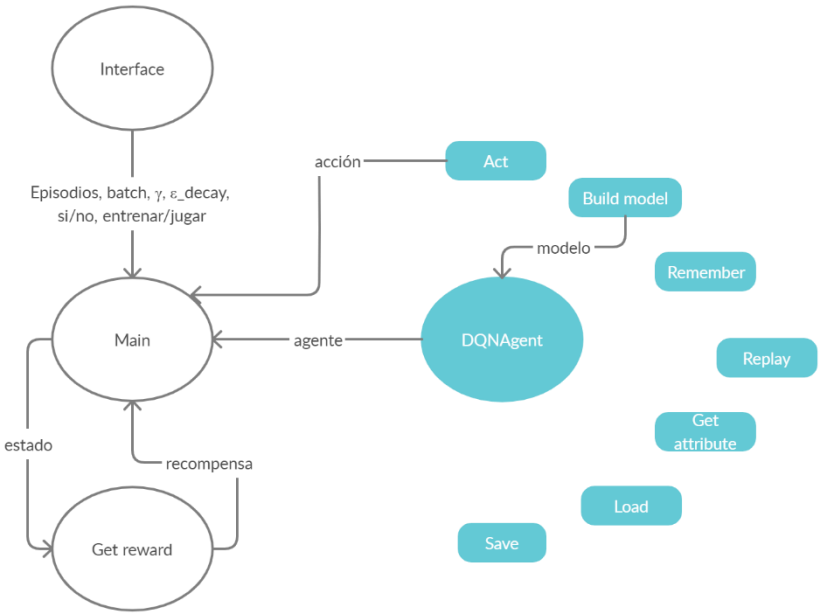


Figura 22: Diagrama estructura código Mountain Car

De esta manera, en la función principal o ‘main’, se obtiene esta nueva recompensa ‘artificial’ mediante la función incorporada, en vez de extraerse del propio entorno, ver *Figura 23*. El resto del proceso se mantiene igual que el diseñado para el caso Cartpole.

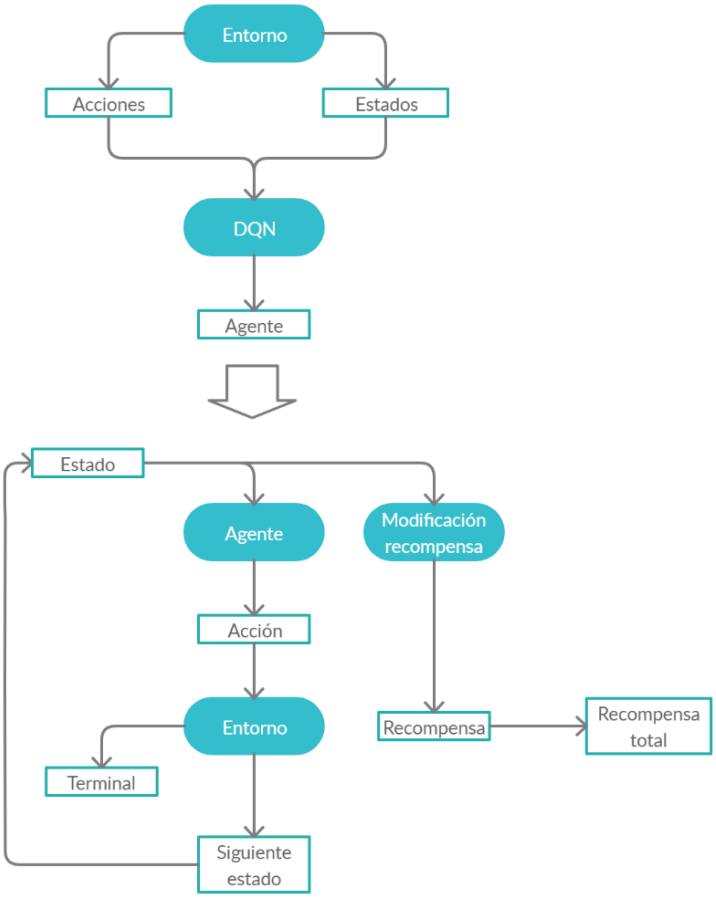


Figura 23: Diagrama estructura código función ‘main’ para Mountain Car

Capítulo 5. Análisis y comparación de resultados de las ejecuciones.

Una vez explicado todo el funcionamiento del programa, se procede a la exposición y análisis de los resultados obtenidos. Para ello, se presentarán las gráficas de la evolución de las recompensas y del valor de ε con cada episodio, seleccionando las más representativas de entre todas las obtenidas. Para observar las diferencias que ocasionan los valores de los parámetros, se expondrán para cada caso los resultados variando el factor de descuento, épsilon decay y el tamaño del batch. Con el objetivo de encontrar la parametrización que da lugar al mejor comportamiento del agente, se tomará como valor principal el factor de descuento y en función de los resultados que se obtengan, se variarán los otros dos factores.

5.1 Resultados Cartpole

Se comienza fijando un batch de 64, un número de episodios igual a 500, un factor de descuento igual a 0.99 y un e_decay igual a 0.99.

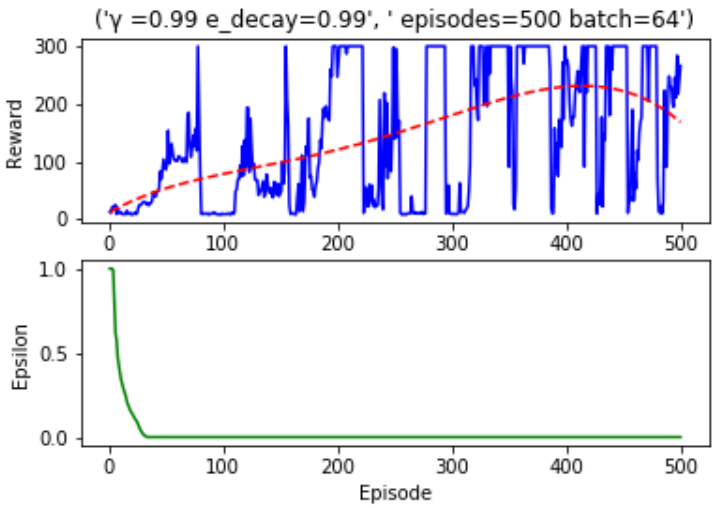


Figura 24: Resultados Cartpole

El resultado obtenido, visible en la Figura 24, a pesar de llegar a la puntuación máxima (300) en varias ocasiones, muestra importantes irregularidades, especialmente en los 300 primeros episodios, por lo que no demuestra una gran estabilidad.

A continuación, se disminuye el factor de descuento a 0.95, manteniendo los demás parámetros iguales.

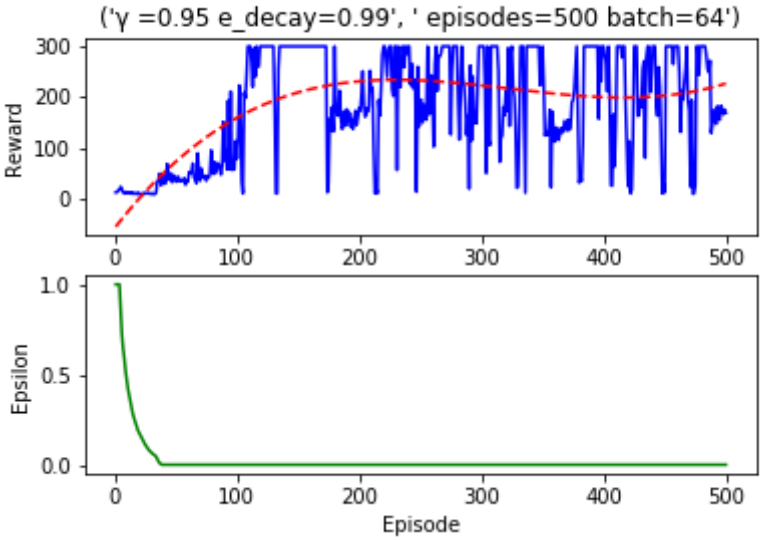


Figura 25 : Resultados Cartpole

Se puede observar en la Figura 25 como el desempeño del agente mejora notablemente con respecto al caso anterior, ya que a partir de los 100 episodios se consigue llegar a la máxima puntuación en numerosas ocasiones. Aun teniendo ciertas caídas en algunos tramos, son más leves y breves que en la Figura 18. Por lo tanto, este modelo presenta una mayor estabilidad.

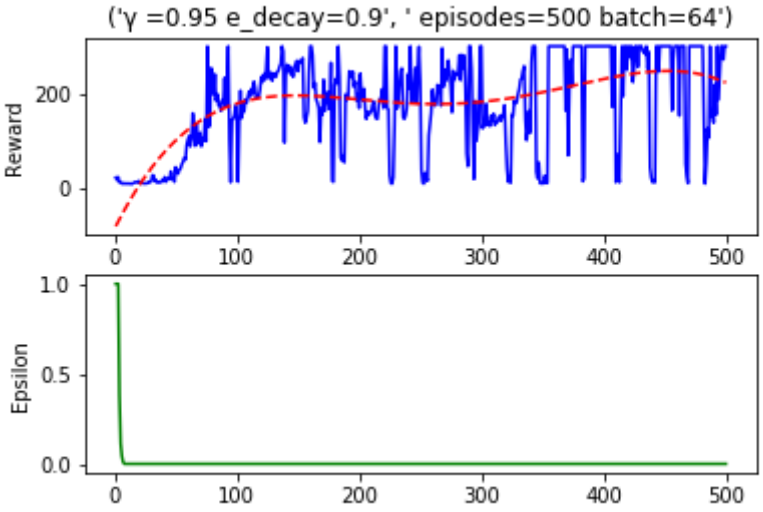


Figura 26: Resultados Cartpole

Probando a disminuir la e_decay a 0.9 mientras se mantienen los demás parámetros, se obtiene un resultado ligeramente peor (Figura 26)

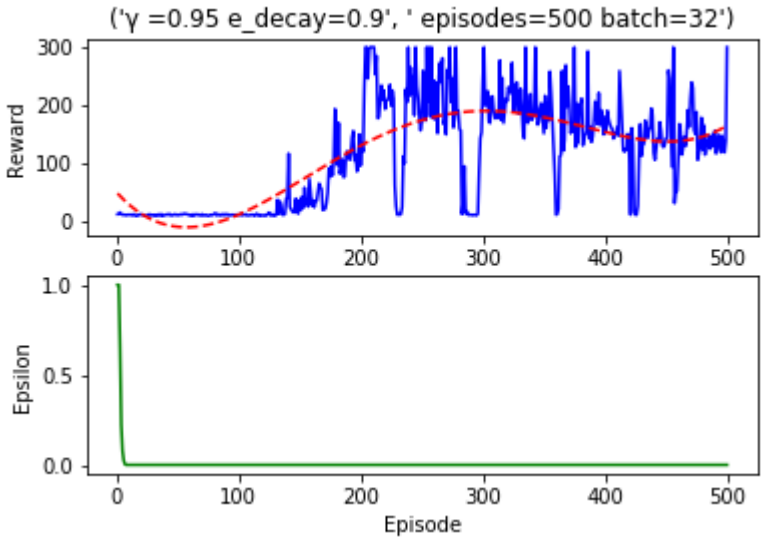


Figura 27: Resultados Cartpole

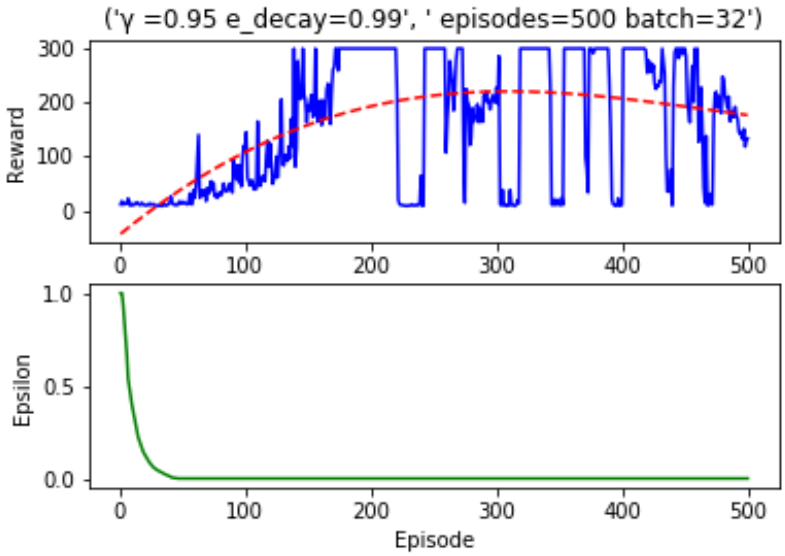


Figura 28: Resultados Cartpole

Tal y como se puede apreciar en las figuras 27 y 28, disminuir el batch a 32 empeora considerablemente el rendimiento del agente, por lo que se demuestra que resulta más efectivo mantenerlo en 64.

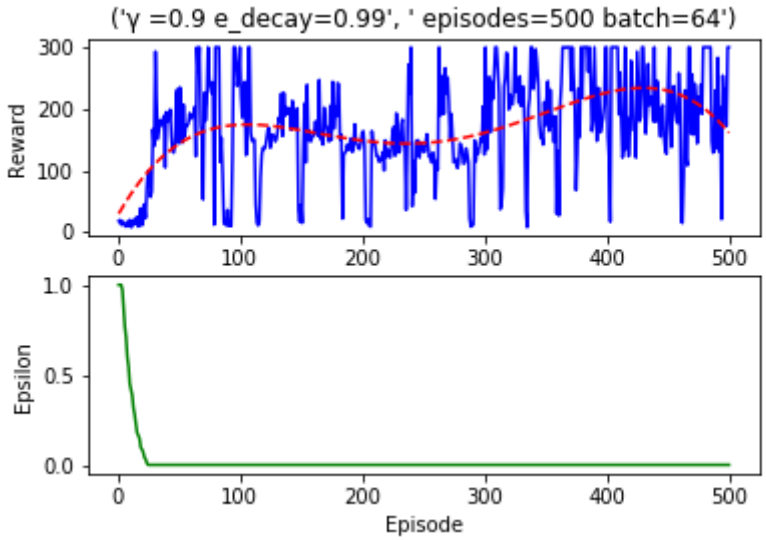


Figura 29: Resultados Cartpole

En la Figura 29 se advierte como disminuir γ a 0.9 da lugar a un comportamiento bastante inestable del agente y alcanzando escasas veces a la máxima puntuación.

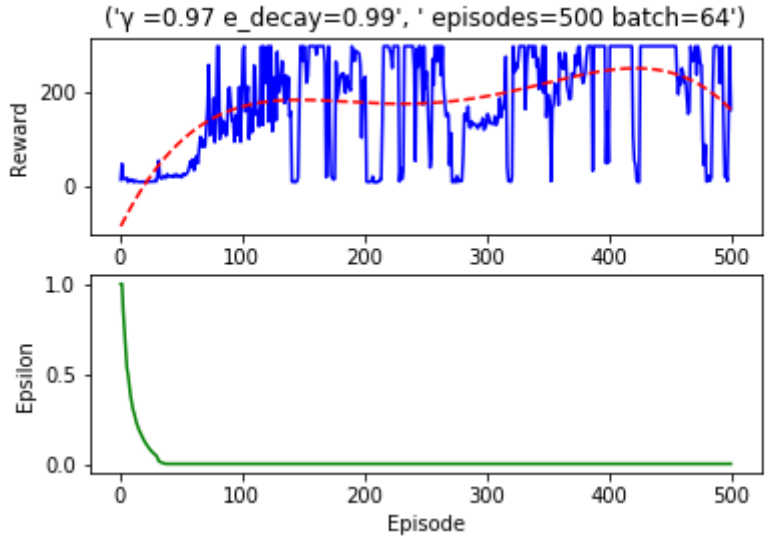


Figura 30: Resultados Cartpole

Por último, se prueba a fijar el valor de γ en 0.97. La Figura 30 demuestra que esta modificación no da un resultado destacable, ya que presenta importantes caídas de puntuación.

Una vez demostrado que el mejor desempeño se consigue con los parámetros γ=0.95, e_decay= 0.99 y batch = 64, se ejecuta este caso para un número mayor de episodios.

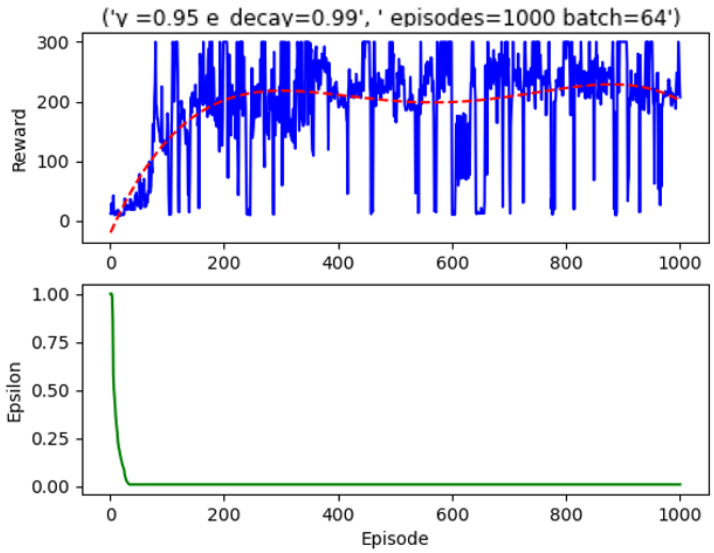


Figura 31: Resultados Cartpole

En la Figura 31 se puede observar como el resultado obtenido para una ejecución de 1000 episodios con los parámetros deseados es considerablemente positivo. El agente consigue llegar por primera vez a la máxima puntuación alrededor de los 100 episodios. De ahí en adelante, pese a algunas alteraciones, muestra una tendencia bastante constante en torno a altas puntuaciones, obteniendo la máxima recompensa en varios episodios.

Tras numerosas ejecuciones del juego, se llega a la conclusión de que la implementación del algoritmo ha sido exitosa. Con los debidos parámetros, el agente presenta un buen rendimiento. Cabe destacar que los mejores resultados se han obtenido para los siguientes valores de las variables: γ = 0.95, e_decay = 0.99 y batch = 64.

5.2 Resultados Breakout

Al tratarse de un entorno con una complejidad muy elevada, son necesarios muchos episodios y horas de ejecución para conseguir ver una mejoría. Se empieza fijando un batch de 32, un factor de descuento igual a 0.95 y un e_decay igual a 0.9995.

Se comienza a ejecutar casos con 5000 episodios, sin conseguir buenos resultados.

A continuación, se decide aumentar el número de episodios hasta 15000.

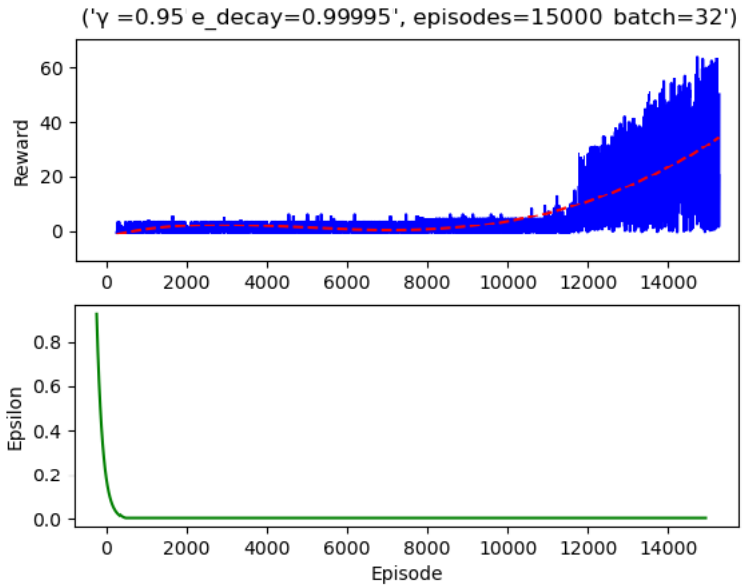


Figura 32: Resultados Breakout

En la Figura 32 se puede observar cómo el agente comienza a mejorar su destreza alrededor de los 12000 episodios, lo cual resulta bastante tardío.

A continuación, se decide reducir la decadencia de épsílon a 0.9995 para comprobar si se observa alguna mejoría.

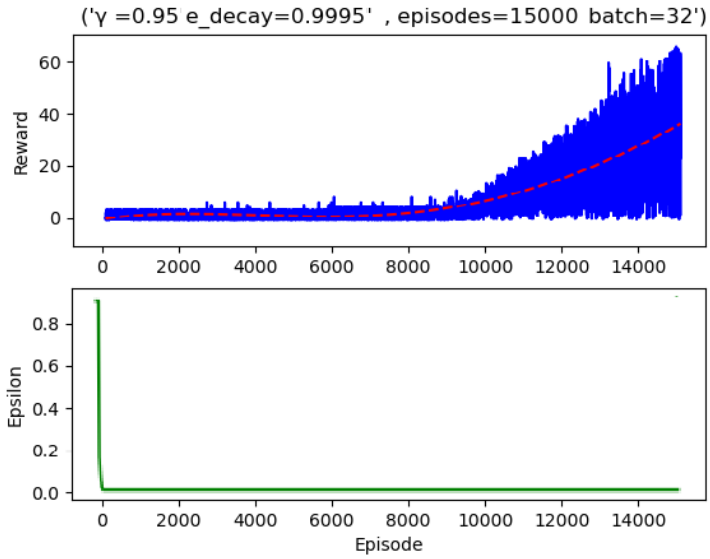


Figura 33: Resultados Breakout

Tal y como se puede observar en la Figura 33, el resultado obtenido tiene un crecimiento más temprano (se comienzan a visualizar los cambios en torno al episodio 10000) y más uniforme.

Resulta positivo descubrir que el agente consigue mejorar su destreza en este entorno. Sin embargo, necesita un número muy elevado de episodios para demostrar este comportamiento. Además, demuestra un crecimiento bastante uniforme, aunque con grandes variaciones en la puntuación. Todo ello se debe a la elevada complejidad del entorno.

A pesar de no demostrar un rendimiento destacable, se consigue el objetivo de mejorar el comportamiento del agente en el juego.

5.3 Resultados de Lunar Lander

Se comienza fijando el valor del parámetro γ a 0.99 y 64 como tamaño del batch, mientras se varía ligeramente el valor de decadencia de épsilon.

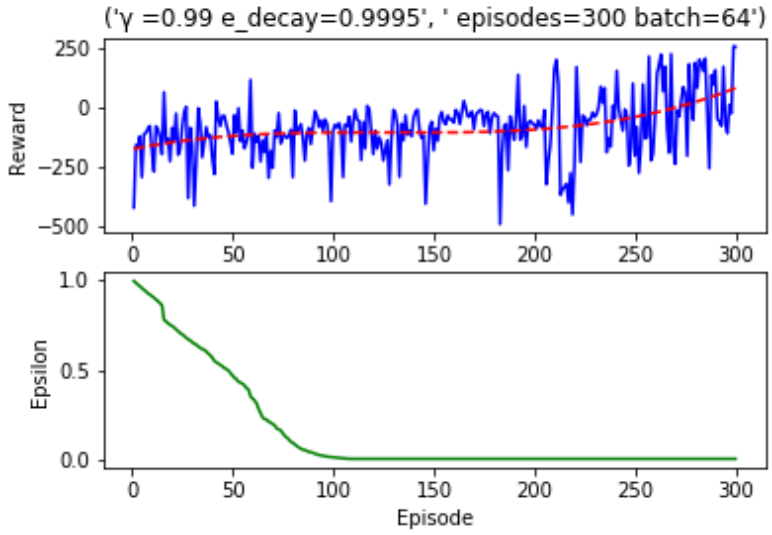


Figura 34: Resultados LunarLander

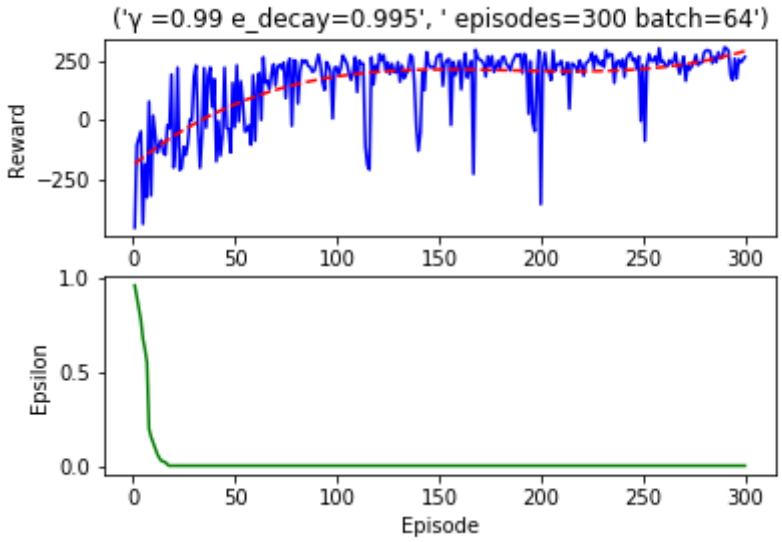


Figura 35: Resultados LunarLander

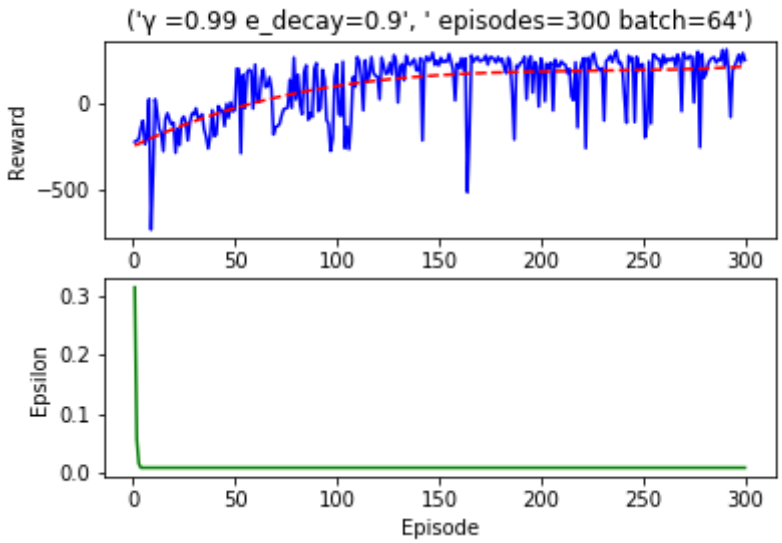


Figura 36: Resultados LunarLander

La Figura 34 muestra como una decadencia de épsilon demasiado alta da lugar a puntuaciones más bien bajas, mientras que al situarse en 0.995 o 0.9 (Figuras 35 y 36 respectivamente) se obtiene un mayor número de recompensas altas, especialmente en el primer caso, donde se obtiene un desempeño muy bueno.

Una vez localizada esta parametrización que da lugar a un comportamiento del agente tan eficaz, se prueba a modificar el tamaño del batch por si este mejorase más aún los resultados.

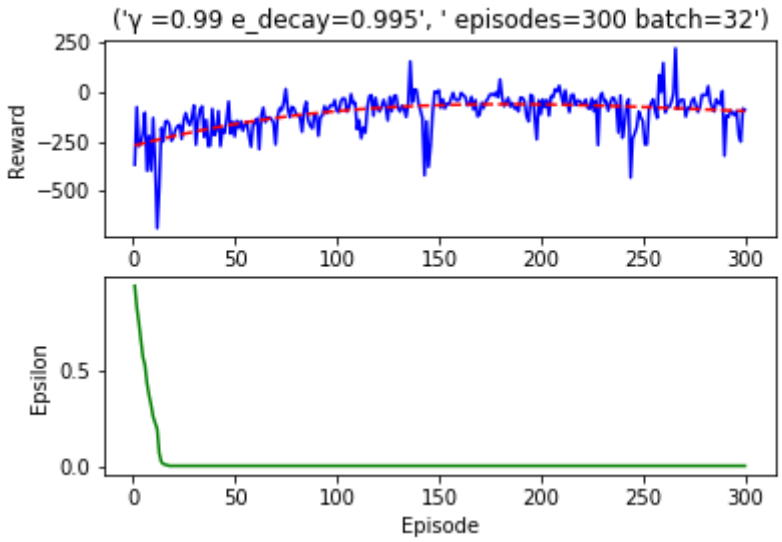


Figura 37: Resultados LunarLander

No obstante, tal y como se puede observar en la Figura 37, esta modificación da lugar a recompensas notablemente inferiores.

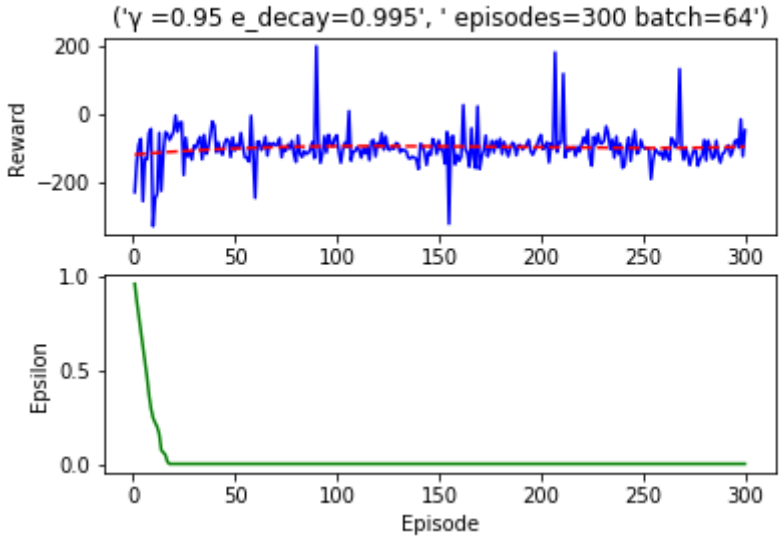


Figura 38: Resultados LunarLander

A continuación, se prueba a establecer un valor de γ igual a 0.95, obteniendo un resultado bastante pobre pues las recompensas prácticamente no superan el 0, salvo algún episodio particular (Figura 38).

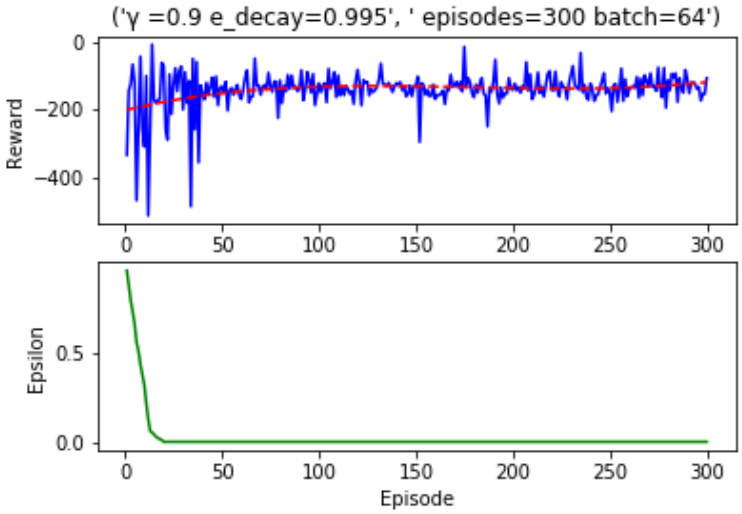


Figura 39: Resultados LunarLander

Bajando el valor de γ a 0.9, el resultado obtenido es incluso peor, pues todas las puntuaciones se encuentran por debajo de 0, tal y como se puede ver en la Figura 39.

Considerando la tendencia a mejores resultados cuanto mayor sea el factor de descuento, se prueba a incrementar este a 0.999, con el fin de determinar si empeoraría el desempeño del agente con un valor tan alto, o por el contrario se convertiría en la parametrización óptima.

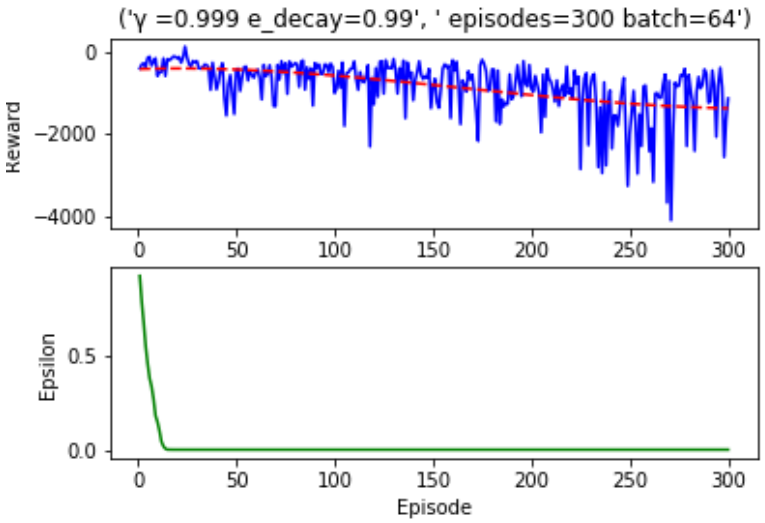


Figura 40: Resultados LunarLander

La Figura 40 demuestra que esta modificación da lugar a unos resultados nefastos, por lo que se asume la Figura 35 como el caso más idóneo.

Para finalizar, se ejecuta para un número mayor de episodios el caso que ha demostrado un mejor desempeño (γ= 0.99, e_decay= 0.995 y batch= 64). Con esto se pretende analizar su comportamiento a largo plazo.

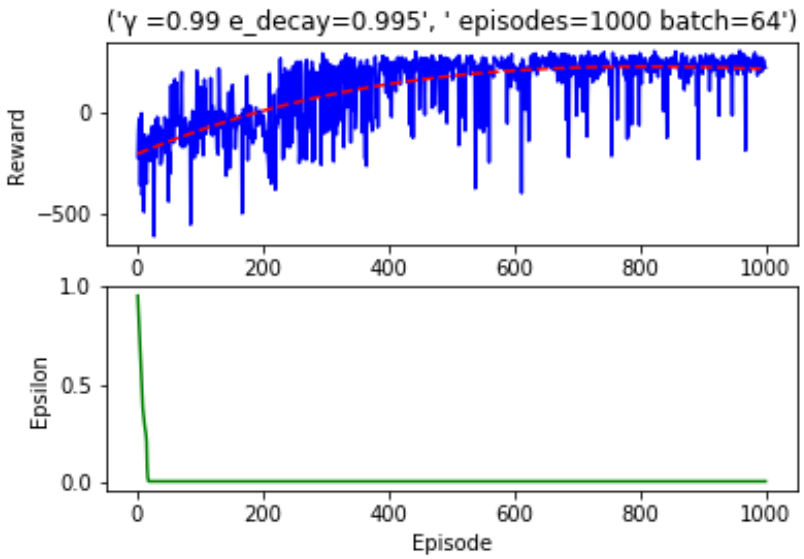


Figura 41: Resultados LunarLander

La Figura 41 muestra un buen rendimiento del agente, pues a pesar de tener un comienzo más irregular, a partir de los 400 episodios las puntuaciones se mantienen notablemente constantes en torno a los 250 puntos.

Tras el análisis de todos los resultados se demuestra que el algoritmo da lugar a un buen desempeño del agente en este entorno. Para los debidos valores de los parámetros, se ha conseguido obtener puntuaciones muy altas y con una notable estabilidad. Cabe destacar que el mejor rendimiento se ha obtenido con lo sparám.etros γ= 0.99, e_decay= 0.995 y batch= 64.

5.4. Resultados Acrobot

Se comienza la ejecución estableciendo un γ igual a 0.99 para 300 episodios y con un batch de tamaño 32. La decadencia de ε se probará como 0.9995 y 0.99.

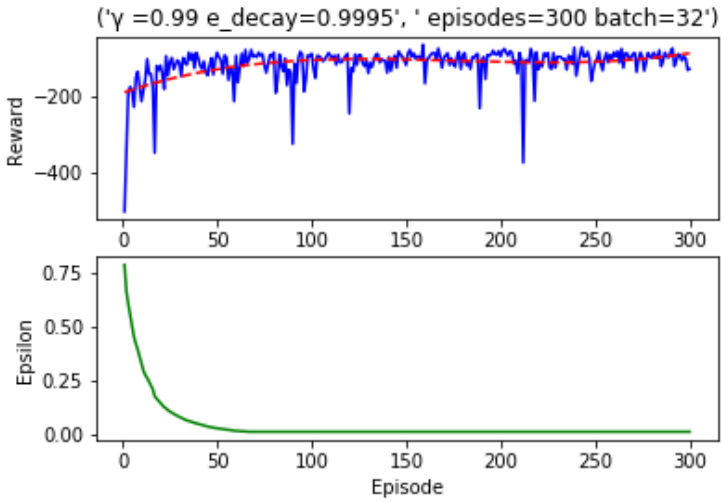


Figura 42: Resultados Acrobot

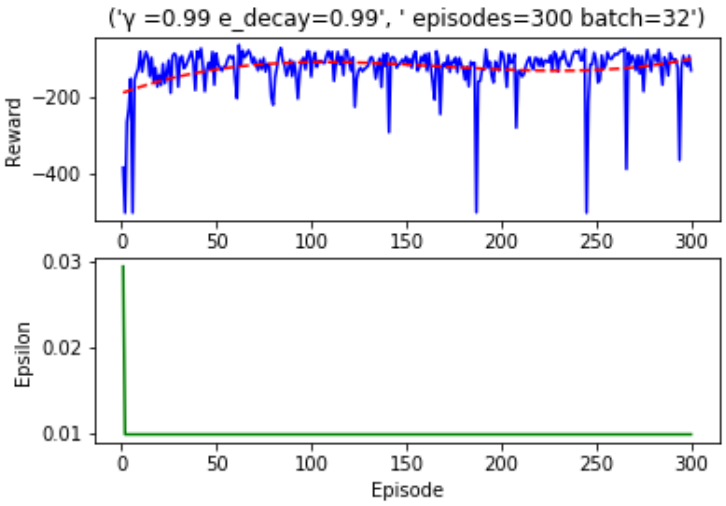


Figura 43: Resultados Acrobot

Tal y como se puede ver en las Figuras 42 y 43, en ambos casos se observa un buen desempeño, especialmente en el correspondiente a e_decay = 0.9995, Figura 36 .

A continuación, se disminuye el valor de γ a 0.95 y 0.9, manteniendo los demás parámetros igual.

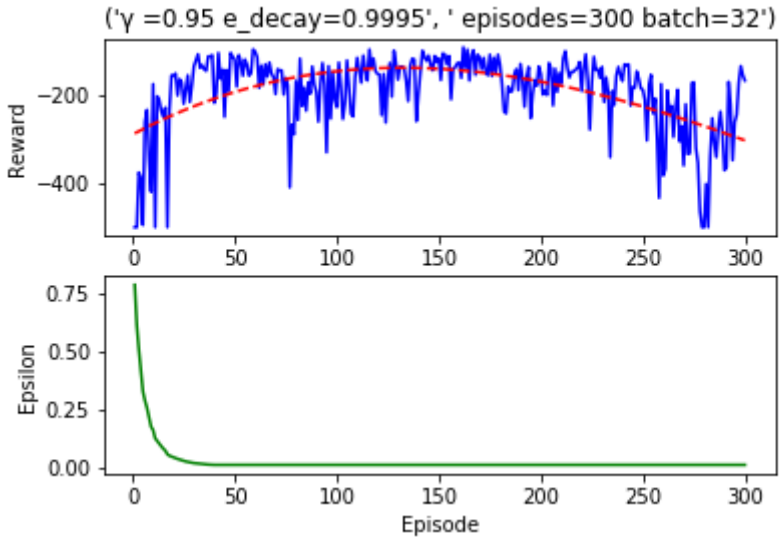


Figura 44: Resultados Acrobot

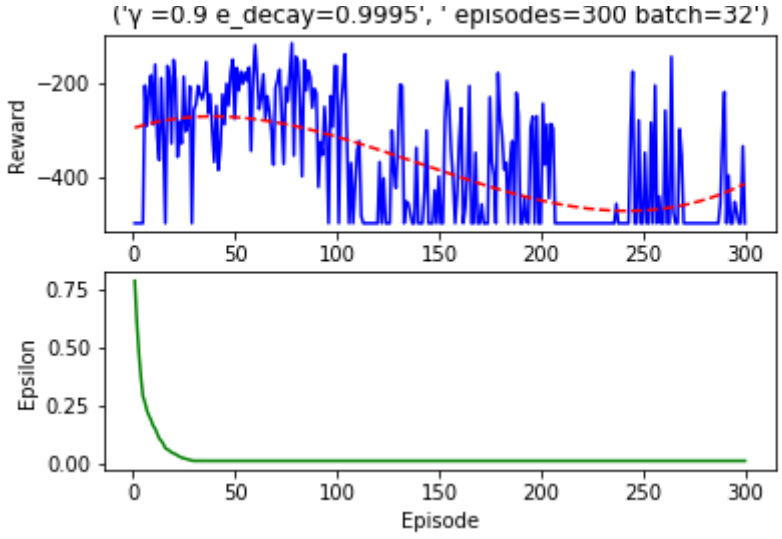


Figura 45: Resultados Acrobot

De acuerdo con las figuras 44 y 45, se observa una mayor inestabilidad en ambos casos, especialmente pronunciada para el γ menor (0.9).

Seguidamente, para el caso de γ = 0.95 (el mejor hasta el momento), se prueba a aumentar el tamaño del batch a 64, con el objetivo de comprobar si esta modificación supondría alguna mejora.

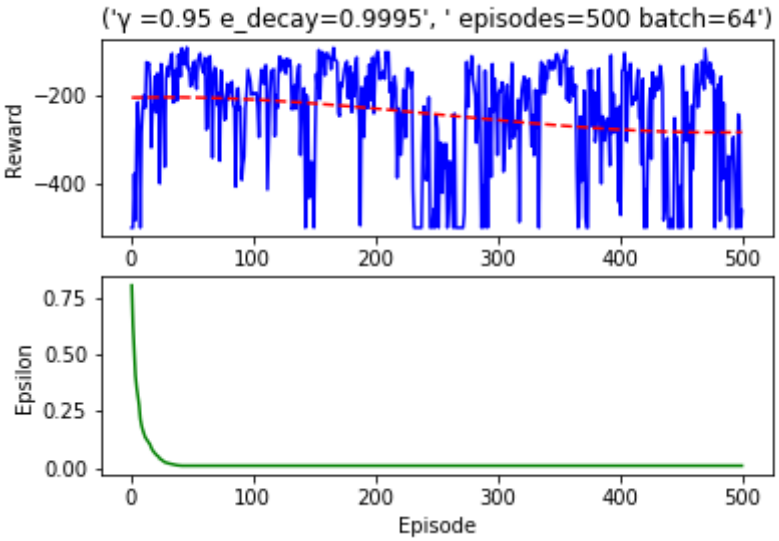


Figura 46: Resultados Acrobot

Tal y como nos demuestra la Figura 46, la modificación llevada a cabo no tiene ningún éxito. Por tanto, se asume que los mejores resultados están vinculados a mayores valores de γ.

A continuación, se prueba a fijar el parámetro γ en un valor intermedio entre 0.95 y 0.99.

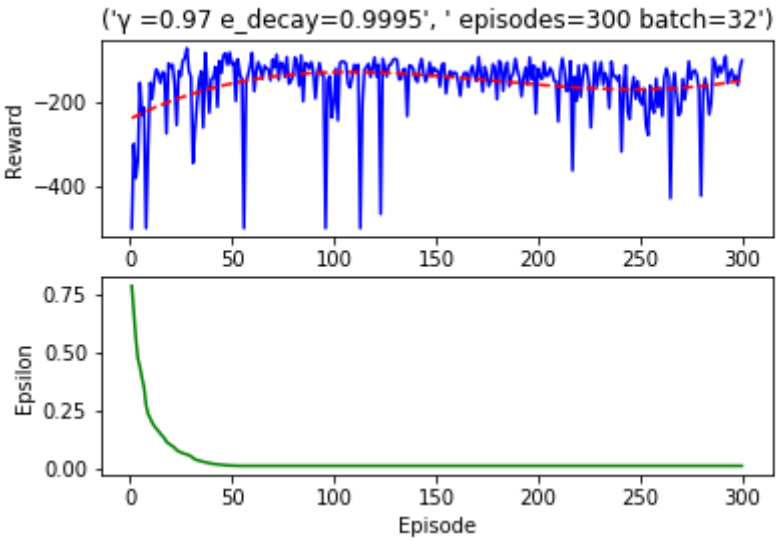


Figura 47: Resultados Acrobot

En la Figura 47 se advierte una notable mejoría con respecto a los resultados anteriores. No obstante, no llega a ser equiparable al mejor caso registrado hasta el momento, Figura 36, lo cual demuestra de nuevo que se obtienen mejores resultados cuando más alto sea el valor del factor de descuento.

Por último, se prueba a aumentar más aún el valor de γ, fijándolo en 0.999, con la incertidumbre de si el resultado obtenido será mejor aún o comenzará de nuevo a empeorar.

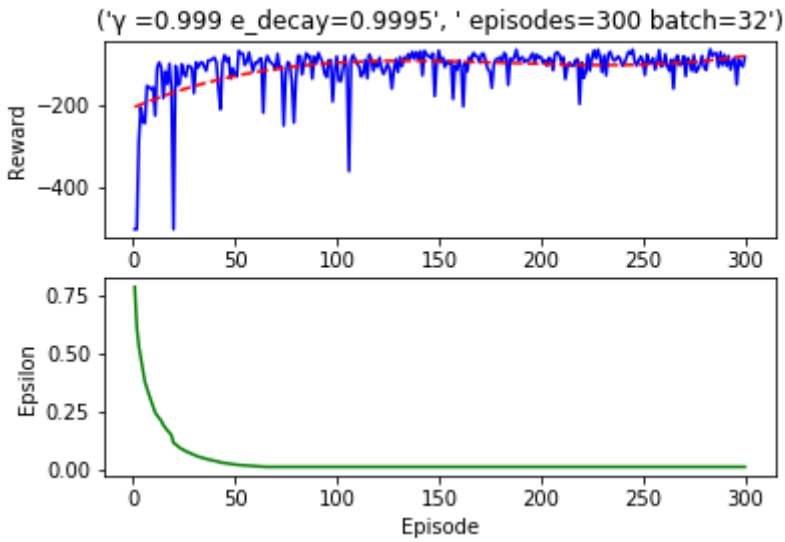


Figura 48: Resultados Acrobot

La Figura 48 muestra que esta ligera modificación optimiza aún más el desempeño del agente, estableciéndose como el mejor caso obtenido.

Una vez encontrados los dos mejores casos, se aumenta el número de episodios con el objetivo de comparar el rendimiento de ambos a largo plazo.

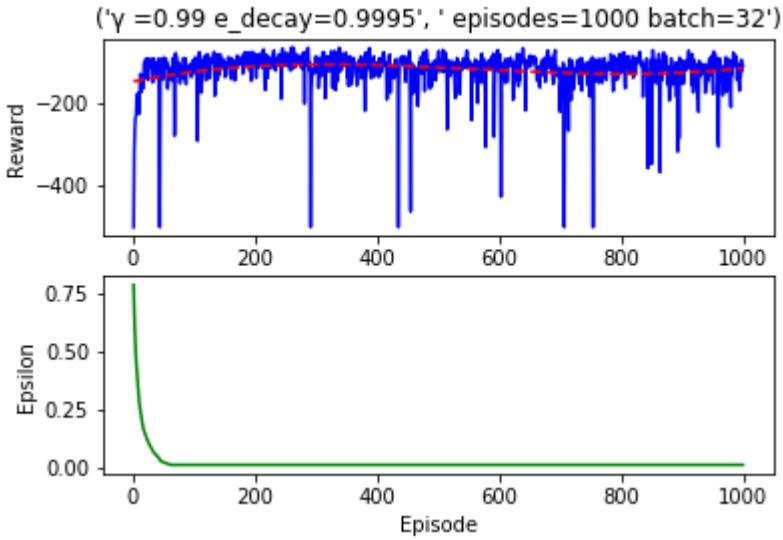


Figura 49: Resultados Acrobot

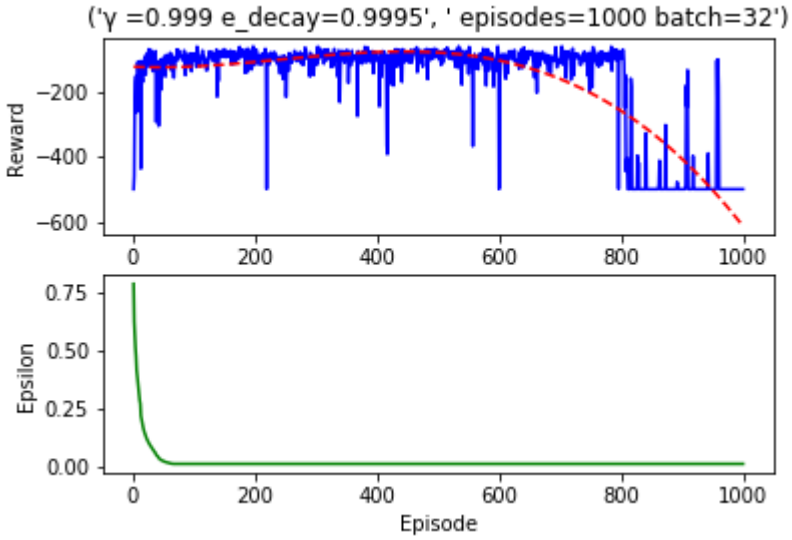


Figura 50: Resultados Acrobot

Observando los resultados, se descubre que a largo plazo un factor de descuento de 0.999 no da lugar a buenos resultados, ya que, a pesar de empezar con un buen desempeño, termina desestabilizándose por completo y dando lugar a recompensas muy bajas (Figura 50), por lo que no demuestra una buena estabilidad. Sin embargo, con un valor de γ igual a 0.99 se obtiene un desempeño muy bueno (Figura 49) ya que se

mantienen prácticamente constantes las recompensas altas una vez que el agente se hace al entorno (lo cual ocurre en pocos episodios).

En resumen, la aplicación del algoritmo a este entorno ha proporcionado unos resultados muy positivos. Con los debidos parámetros, el agente ha demostrado un excelente rendimiento y gran estabilidad. Tras numerosas ejecuciones del entorno, se ha llegado a la conclusión de que los parámetros que dan lugar a mejores resultados son: γ = 0.99, e_decay = 0.9995 y batch = 32.

5.5 Resultados de Mountain Car

Se comienza la ejecución con lo siguientes valores: γ= 0.99, e_decay= 0.9995 y tamaño de batch igual a 32.

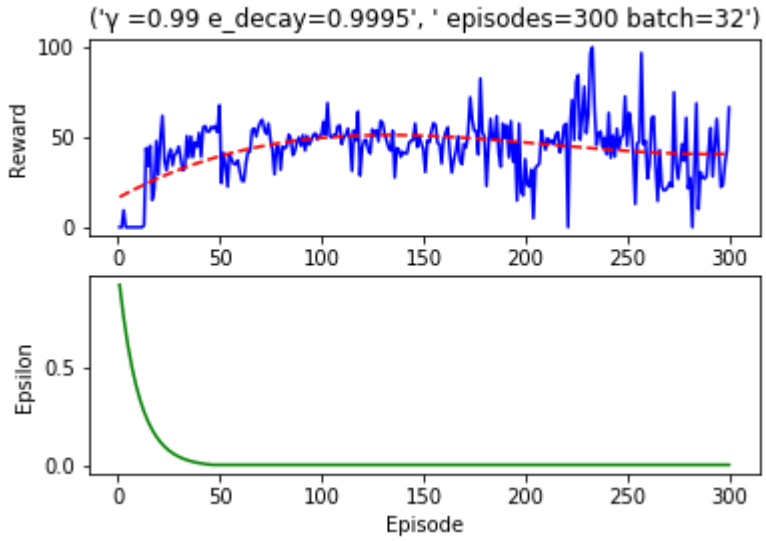


Figura 51: Resultados MountainCar

Tal y como se puede observar en la Figura 51, los resultados no son muy positivos, pues el agente no consigue llegar a la meta en ningún caso y las puntuaciones son bastante bajas en general. Se decide entonces cambiar el factor de descuento a 0.95.

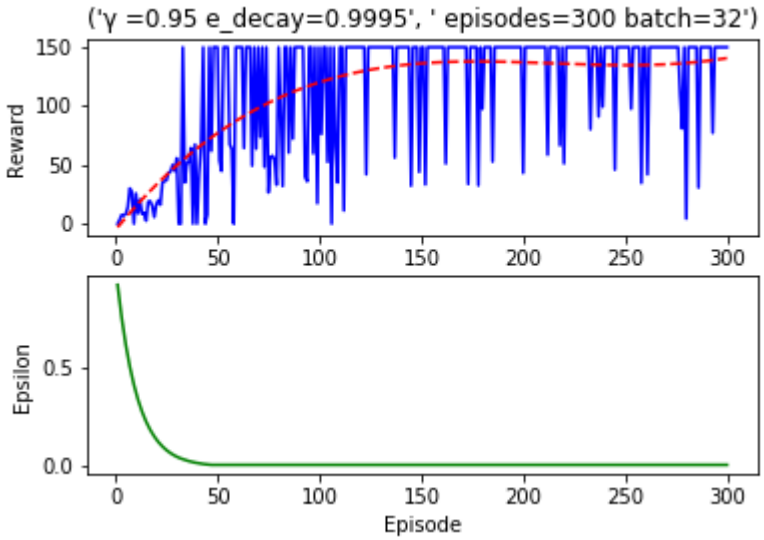


Figura 52: Resultados MountainCar

En la Figura 52 se advierte una mejora muy notable con esta nueva modificación, pues alrededor del episodio número 50 el carrito ya comienza a llegar a la meta y a partir del 100 esta tendencia se estabiliza bastante.

Debido a este buen resultado, se decide probar a ampliar el tamaño del batch a 64 para explorar la opción de una posible mejoría.

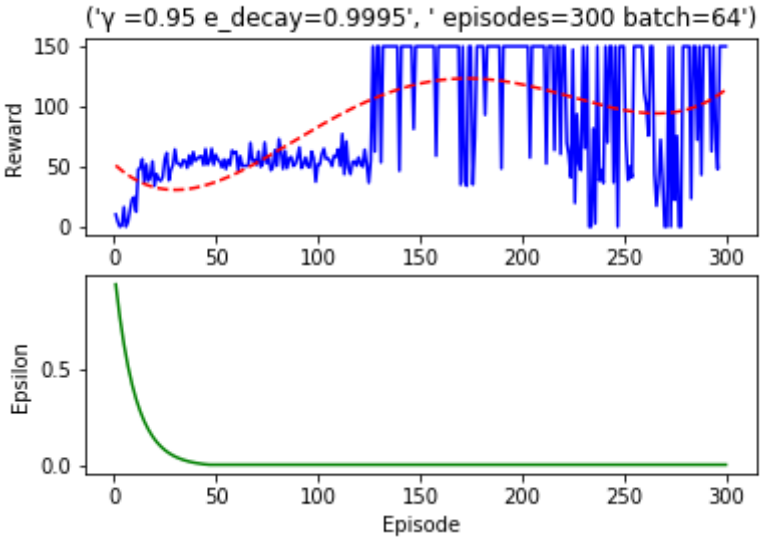


Figura 53: Resultados MountainCar

Sin embargo, tal y como se demuestra en la Figura 53, esta modificación no resulta beneficiosa ya que no sólo retrasa la llegada del carrito a la meta, si no que además muestra un comportamiento más inestable. Por lo tanto, se continúa con un tamaño del batch igual a 32.

Vista la mejoría al reducir el factor de descuento, se disminuye en este caso a 0.9.

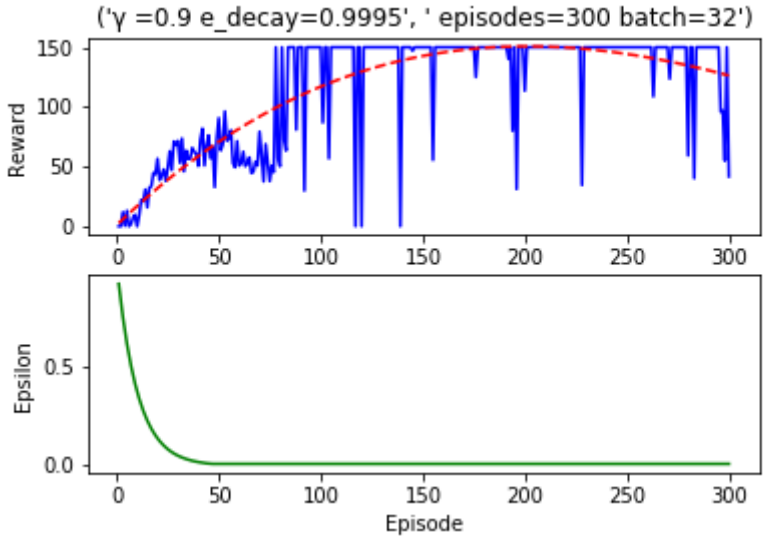


Figura 54: Resultados MountainCar

La Figura 54 demuestra que esta medida da lugar a un resultado muy bueno, ya que, a pesar de que el agente tarda algo más en llegar al objetivo que en otros casos, una vez alcanzado se mantiene bastante constante y con muchas menos caídas.

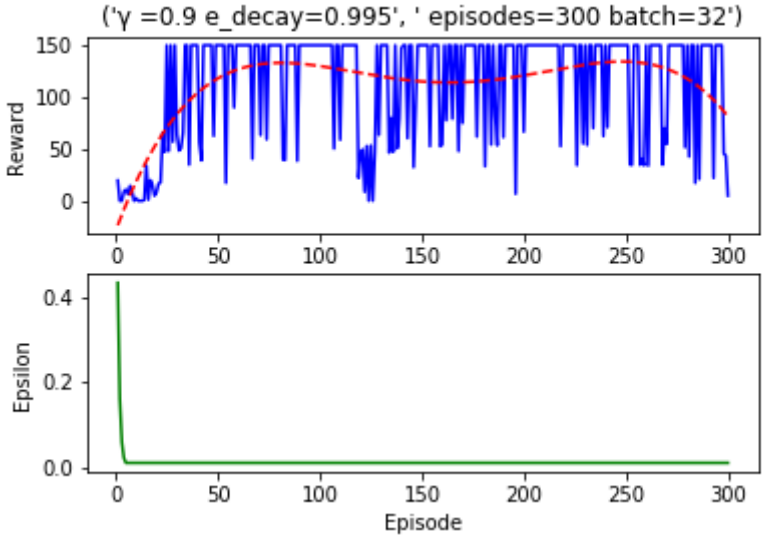


Figura 55: Resultados MountainCar

Al disminuir el valor de decadencia de épsilon a 0.995, se puede observar como el agente llega antes al objetivo pero su comportamiento posterior resulta más inestable (Figura 55) que para el caso anterior.

A continuación, se decide explorar el comportamiento del agente para valores de γ entre los vistos previamente (0.97 y 0.93).

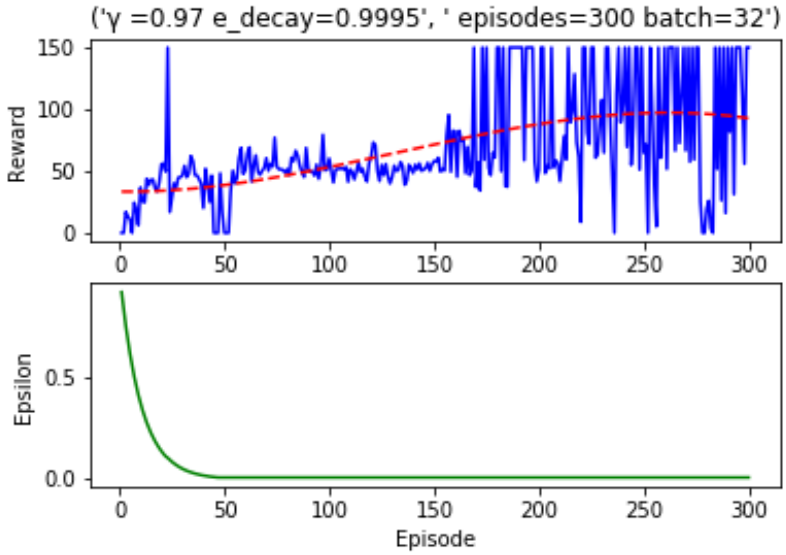


Figura 56: Resultados MountainCar

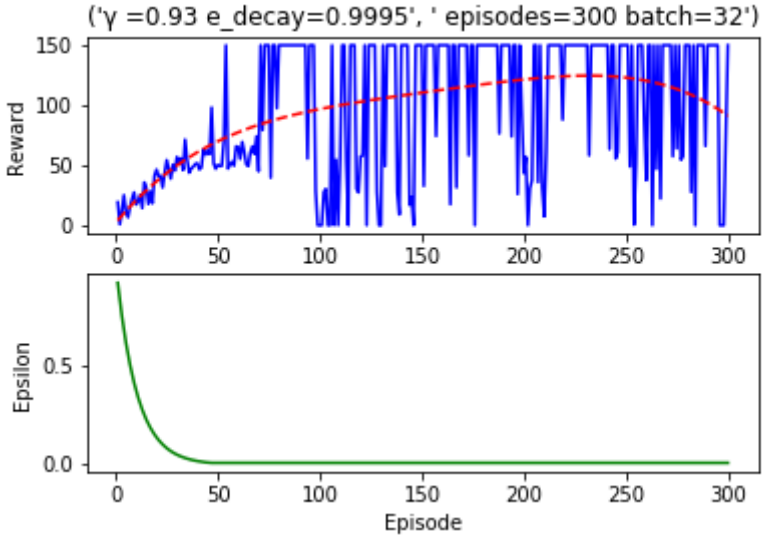


Figura 57: Resultados MountainCar

Los resultados obtenidos no son especialmente buenos, no obstante, se confirma la tendencia a conseguir un mejor comportamiento del agente con valores del factor de descuento menores (Figura 57 frente a Figura 56).

Por último, y basándose en dicha tendencia, se decide probar un valor de γ inferior incluso al mejor caso registrado (0.9).

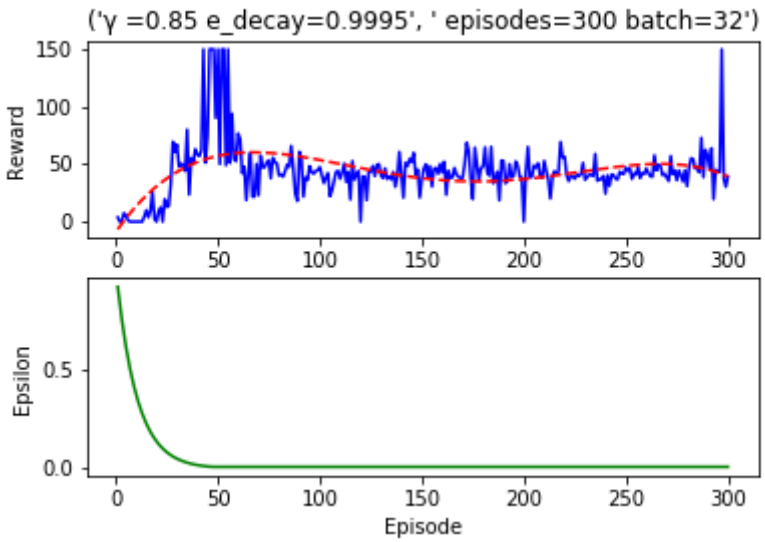


Figura 58: Resultados MountainCar

Sin embargo, según muestra la Figura 58, esta bajada del factor de descuento da lugar a un resultado pésimo, por lo que se asume que el mejor valor para dicho parámetro es 0.9.

Una vez diferenciado el caso más óptimo, se aplica su parametrización a un número mayor de episodios, con el fin de observar su comportamiento a largo plazo.

El resultado obtenido es bastante satisfactorio (Figura 59). El agente empieza a alcanzar la máxima puntuación alrededor de los 100 episodios. A partir de ahí, pese a ciertas desestabilizaciones, la tendencia de las recompensas se mantiene muy alta, llegando en numerosas ocasiones al objetivo.

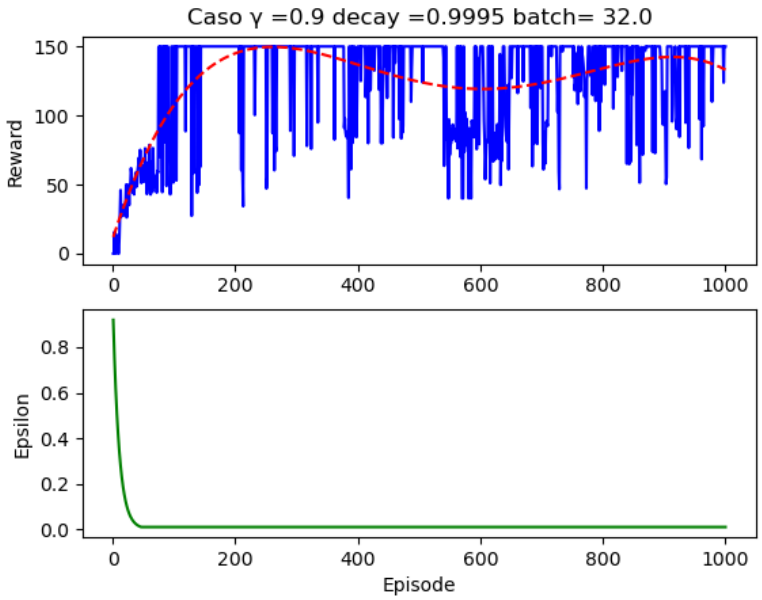


Figura 59: Resultados MountainCar

Tras numerosas ejecuciones del juego, se llega a la conclusión de que la implementación del algoritmo ha sido exitosa. Con los debidos parámetros, el agente consigue llegar a la meta en múltiples ocasiones. Los mejores resultados se han obtenido para los siguientes valores de las variables: $\gamma = 0.9$, e_decay = 0.9995 y batch = 32.

Capítulo 6. Conclusiones y futuros trabajos.

Una vez finalizado el proyecto, se exponen las conclusiones derivadas del mismo, así como un planteamiento de trabajos futuros.

6.1 Conclusiones.

En primer lugar, se ha conseguido desarrollar exitosamente una plataforma en la cual el usuario puede aplicar fácilmente el algoritmo Deep Q-Learning a una serie de videojuegos de OpenAI Gym. Para ello, este programa cuenta con una ventana desde la cual se realiza la parametrización, así como otras elecciones tales como la opción de visualizar el entorno o no, o la opción de hacer un entrenamiento del modelo o simplemente probar uno ya entrenado previamente. Además, la aplicación cuenta con todo tipo de detalles sobre el funcionamiento de cada juego, así como la explicación de cada parámetro que se debe de introducir para dar comienzo a la partida. Adicionalmente, el programa contiene una opción para facilitar la comparación de las gráficas de resultados deseadas. Todo ello agiliza enormemente la experimentación y análisis del comportamiento de este algoritmo de aprendizaje por refuerzo.

Por otro lado, se utiliza dicha aplicación para estudiar el desempeño del Deep Q-Learning adjudicándose diferentes valores a los parámetros. Debido a las grandes diferencias de características de unos entornos a otros, se encuentran distintos valores óptimos para cada caso.

En cuanto a los resultados obtenidos, se consigue alcanzar un desempeño muy bueno en cuatro de los casos. A pesar de ciertas variaciones habituales en la utilización de este algoritmo, se logra llegar a las puntuaciones máximas en pocos episodios y estas se mantienen prácticamente constantes. En definitiva, el agente consigue aprender a dominar el entorno para maximizar las recompensas en cada jugada. En el caso de Breakout, el agente consigue mejorar su destreza en el juego, aunque no presenta un rendimiento tan bueno como los demás debido a la gran complejidad del entorno. No obstante, sí que se cumple con el objetivo principal del algoritmo que es mejorar la destreza del agente en el juego.

Además, el proyecto está involucrado con los Objetivos de Desarrollo Sostenible de la ONU. Esto se debe a que el aprendizaje por refuerzo, al igual que muchas otras áreas del Machine Learning,, colabora en la mejora de las nuevas tecnologías, por medio del análisis de datos y reconocimiento de patrones. Aunque en este este proyecto este orientado a unos videojuegos, toda herramienta útil para el aprendizaje por refuerzo contribuye al aumento de conocimiento en este campo, que tiene diversas aplicaciones relacionadas con los ODS, tales como fomentar el crecimiento económico sostenible, el desarrollo de infraestructuras resilientes, combatir el cambio climático y muchos otros.

6.2 Trabajos futuros

Una vez realizado todo este trabajo en torno al algoritmo Deep Q-learning, resultaría muy interesante ampliar la aplicación creada para incluir otros algoritmos del aprendizaje por refuerzo tales como el Double Deep Q- Learning, Dueling Deep Q-Learning o Noisy Deep Q-Learning. Esto además propiciaría que las comparaciones de resultados no se llevasen a cabo simplemente en cuestión de parametrización, sino también entre el comportamiento de los diferentes algoritmos para un caso concreto.

Por otro lado, se podrían incorporar a la aplicación más entornos. Gym cuenta con una gran variedad de entornos diferentes, por lo que se podrían incluir casos tanto de familias ya utilizadas (Classic Control, Box2D y Atari), como de familias más complejas que no han sido incluidas en este proyecto, tales como MuJoCo Y Robotics.

Por último, se podrían aumentar las opciones de parametrización, agregando por ejemplo la variable ‘learning rate’ a la ventana de parámetros, en vez de mantener un valor fijo e inamovible determinado en el código del programa.

Capítulo 7. Bibliografía.

Watkins, C.J.C.H. (1989), *Learning from Delayed Rewards*, Cambridge University [fecha de consulta: octubre 2019]

Sutton ,Richard S. and Barto Andrew G. (2014) *Reinforcement Learning: an introduction*, Cambridge, Massachusetts, The MIT Press. [última fecha de consulta: octubre 2019]

Caparrini, Fernando (2019). Fernando Sancho Caparrini. http://www.cs.us.es/~fsancho/?e=109. [última fecha de consulta: noviembre 2020]

Surma, Greg (2018). *Deep Q-learning with Keras and Gym*, Towards data science. https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288. [última fecha de consulta: octubre 2019]

López, Rubén. (2015) *Q-Learning: Aprendizaje automático por refuerzo.* https://rubenlopezg.wordpress.com/2015/05/12/q-learning-aprendizaje-automatico-por-refuerzo/ [última fecha de consulta: octubre 2019]

Anónimo. *Exploration Vs. Exploitation - Learning The Optimal Reinforcement Learning Policy*, Deeplizardz, https://deeplizard.com/learn/video/mo96Nqlo1L8 [última fecha de consulta: enero 2020]

Britz, Denny (2014). *Exploration vs Exploitation*, Medium, https://medium.com/@dennybritz/exploration-vs-exploitation-f46af4cf62fe [última fecha de consulta: octubre 2019]

Amos, David (2020) *Python GUI Programming With TKinter*, Real Python, https://realpython.com/python-gui-tkinter/#building-your-first-python-gui-application-with-tkinter, [última fecha de consulta: febrero 2020]

Roth, Tom (2018) *Let's build a DQN: basics.* https://tomroth.com.au/dqn-basics/ [última fecha de consulta: noviembre 2019]

Karagiannakos, Sergios (2018) *Deep Q Learning and Deep Q Networks.* AI Summer. https://theaisummer.com/Deep_Q_Learning/ [última fecha de consulta: noviembre 2019]

Guzmán (2019) *Programación dinámica con Q-learning.* Planeta ChatBot https://planetachatbot.com/programaci%C3%B3n-din%C3%A1mica-con-q-learning-9e1474312287 [última fecha de consulta: noviembre 2019]

Verma, Shiva (2019). *Solving reinforcement learning classic control problems*, Towards Data Science, https://towardsdatascience.com/solving-reinforcement-learning-classic-control-problems-openaigym-1b50413265dd [última fecha de consulta: mayo 2020]

Verma, Shiva (2019). *Train your lunar lander*, Towards Data Science, https://towardsdatascience.com/solving-lunar-lander-openaigym-reinforcement-learning-785675066197 [última fecha de consulta: mayo 2020]

Grisby, Jake (2018). Advanced DQNs: Playing Pac-man with Deep Reinforcement Learning. Towards Data Science, https://towardsdatascience.com/advanced-dqns-playing-pac-man-with-deep-reinforcement-learning-3ffbd99e0814 [última fecha de consulta: marzo 2020]

Mazasl (2017) *Master*. Gitbhub https://github.com/MasazI/dqn-gym/blob/master/dqn.py [última fecha de consulta: marzo 2020]

Berges, Vicent. Rao Priyanka. Pryzant Reid. Reiforcement learning for Atari Breakout. Tandford University. https://cs.stanford.edu/~rpryzant/data/rl/paper.pdf [última fecha de consulta: marzo 2020]

Ecoffet, Adriend Lucas (2017) *Beat Atari with reinforcement learning!* Medium, https://becominghuman.ai/lets-build-an-atari-ai-part-1-dqn-df57e8ff3b26 [última fecha de consulta: marzo 2020]

Brownlee Jason (2019) *How to Control the Stability of Training Neural Networks With the Batch Size.* Machine Learning Mastery. https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/ [última fecha de consulta: junio 2020]

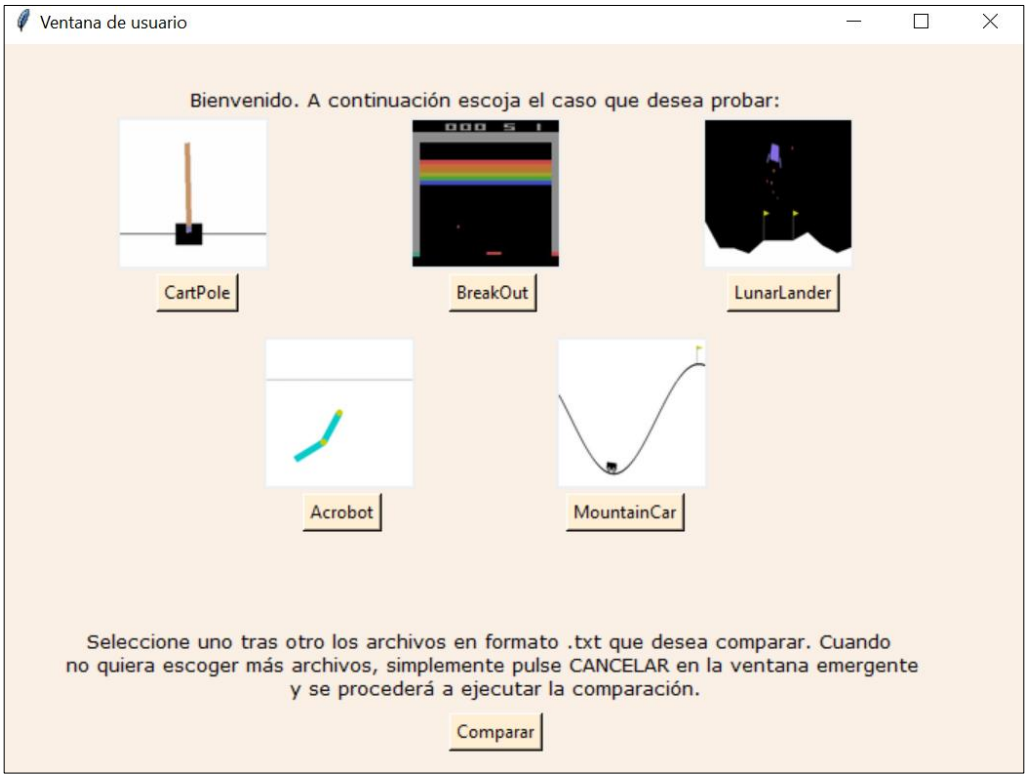
Gupta Ashish (2019) *AI Learning to land a Rocket(Lunar Lander) | Reinforcement Learning.* Toward Data Science. https://towardsdatascience.com/ai-learning-to-land-a-rocket-reinforcement-learning-84d61f97d055 [última fecha de consulta: junio 2020]

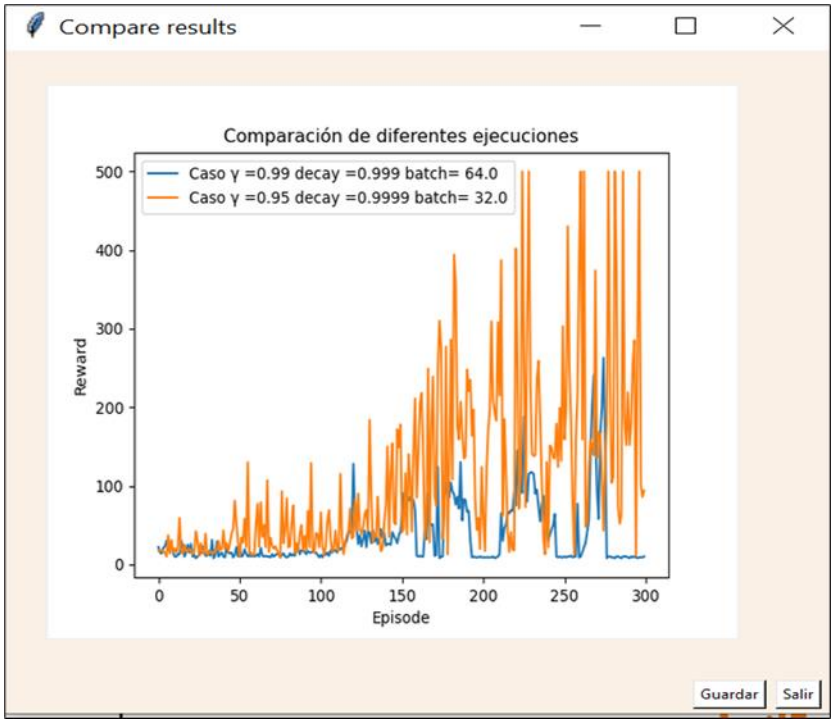
Wojciech, Mormul (2018) *Deep Q-Learning.* Github https://github.com/WojciechMormul/rl-dqn [última fecha de consulta: marzo 2020]

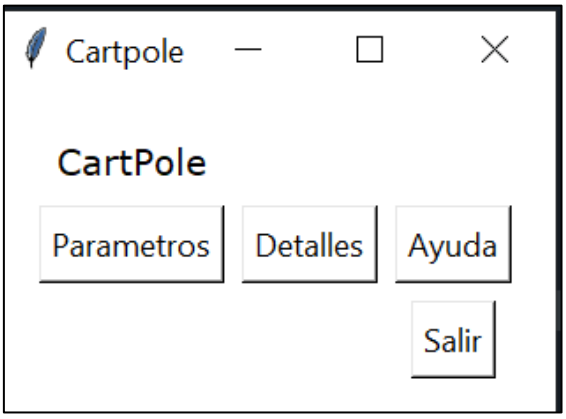
Ossanolik (2019) *Gym environments*. Github https://github.com/openai/gym/blob/master/docs/environments.md [última fecha de consulta: mayo 2020]

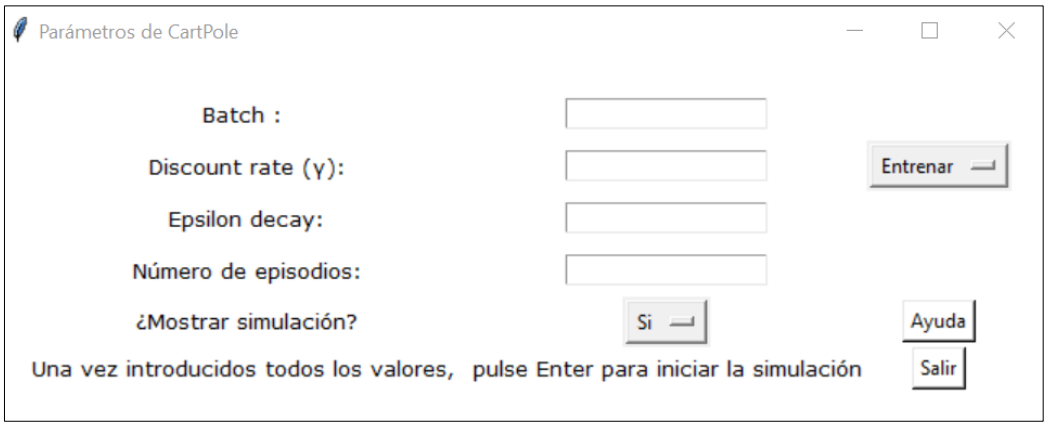
ANEXO A

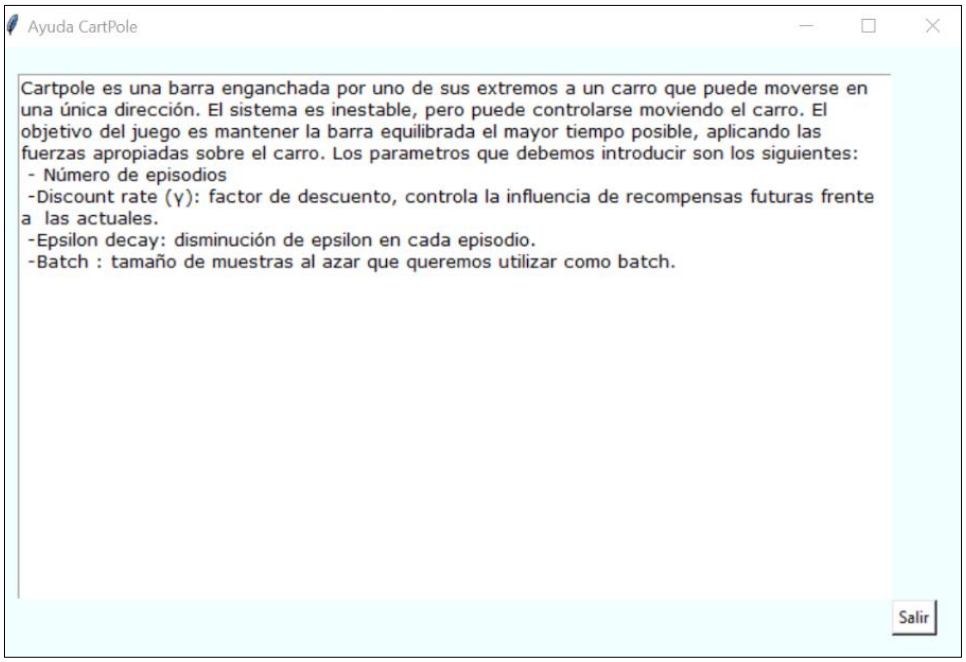
1. Imágenes de la aplicación

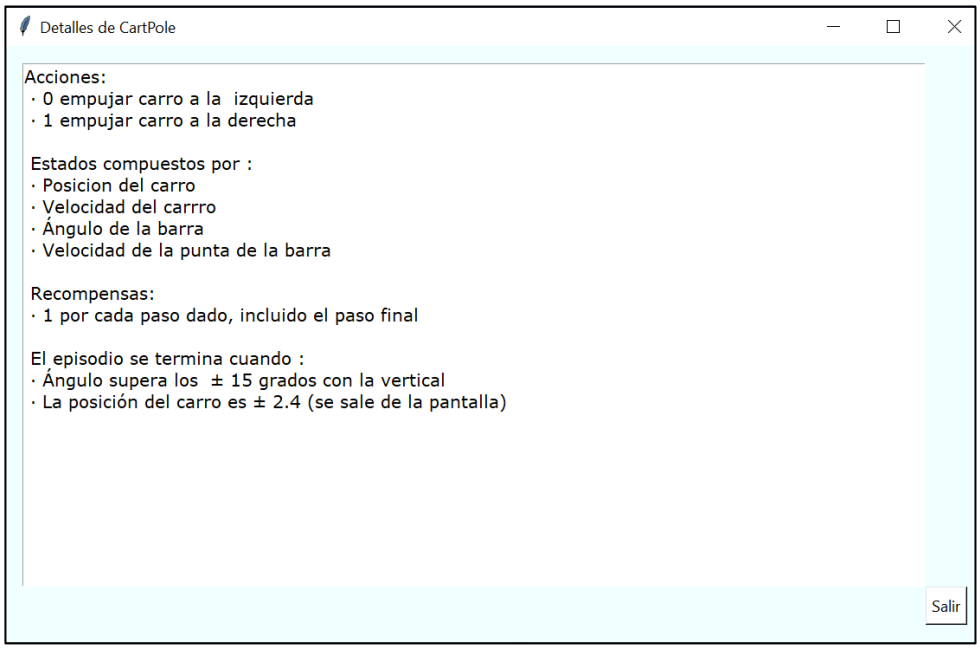


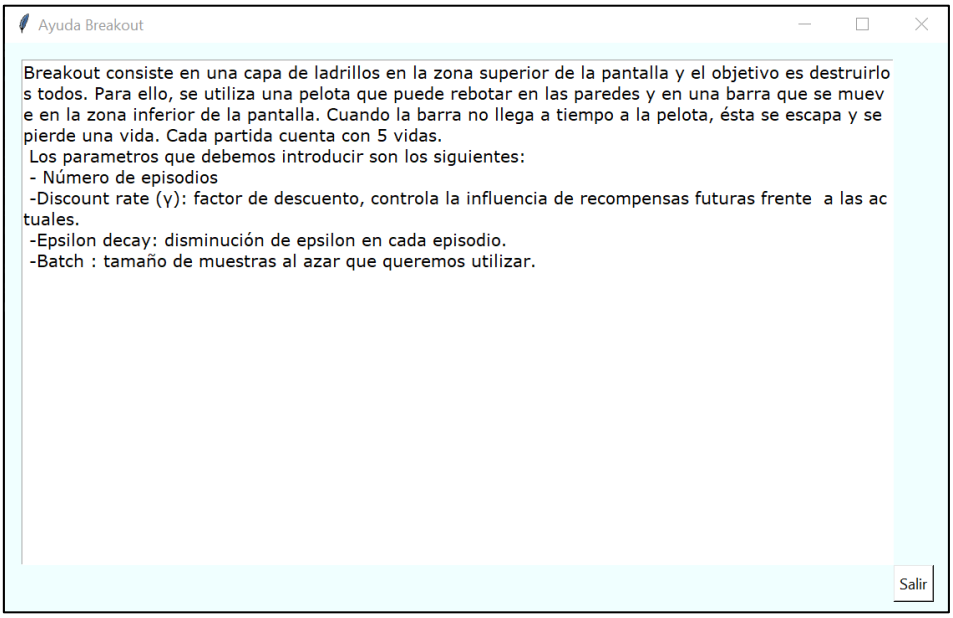


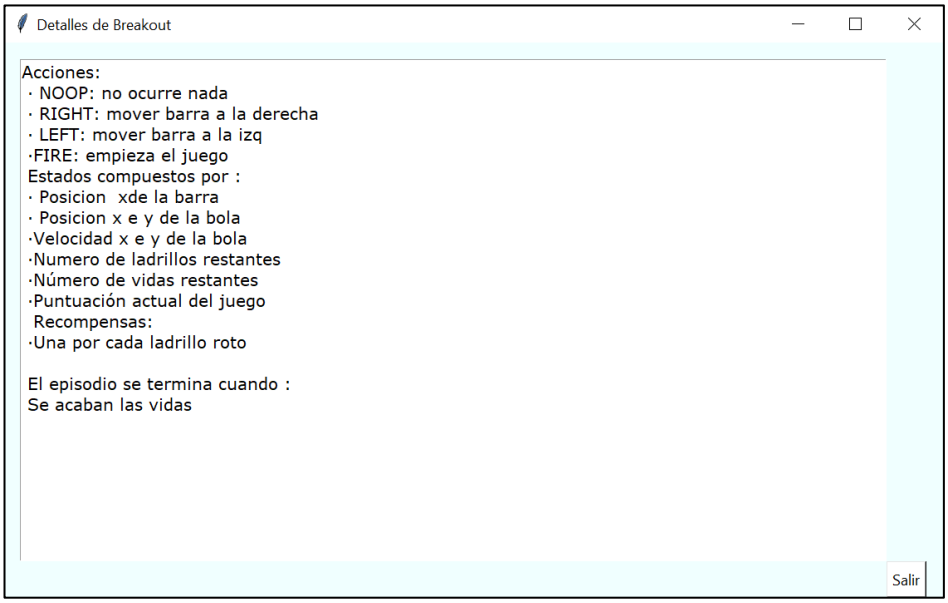

Cartpole
CartPole
Parametros
Detalles
Ayuda
Salir

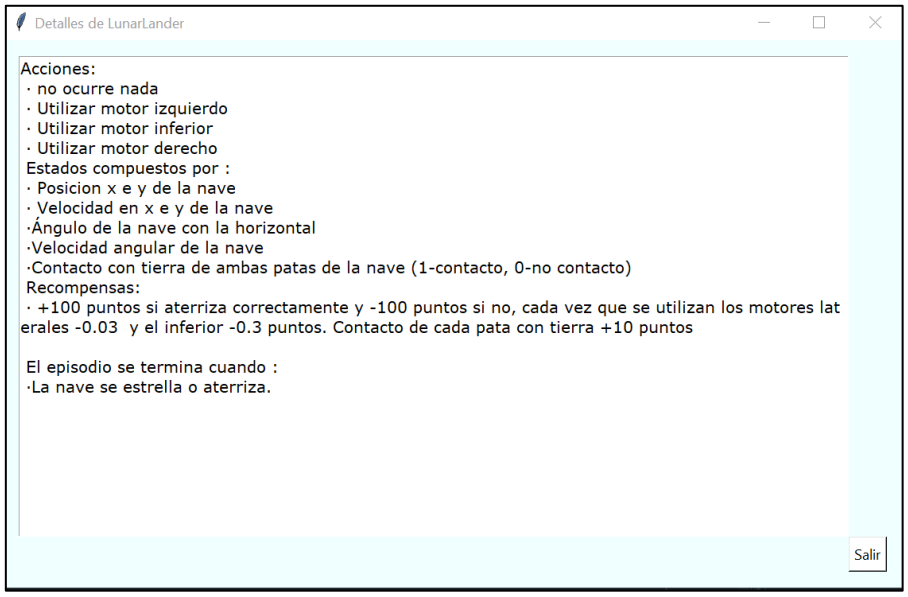

Parámetros de CartPole
Batch :
Discount rate (γ):
Epsilon decay:
Número de episodios:
¿Mostrar simulación?
Si
Entrenar
Ayuda
Una vez introducidos todos los valores, pulse Enter para iniciar la simulación
Salir

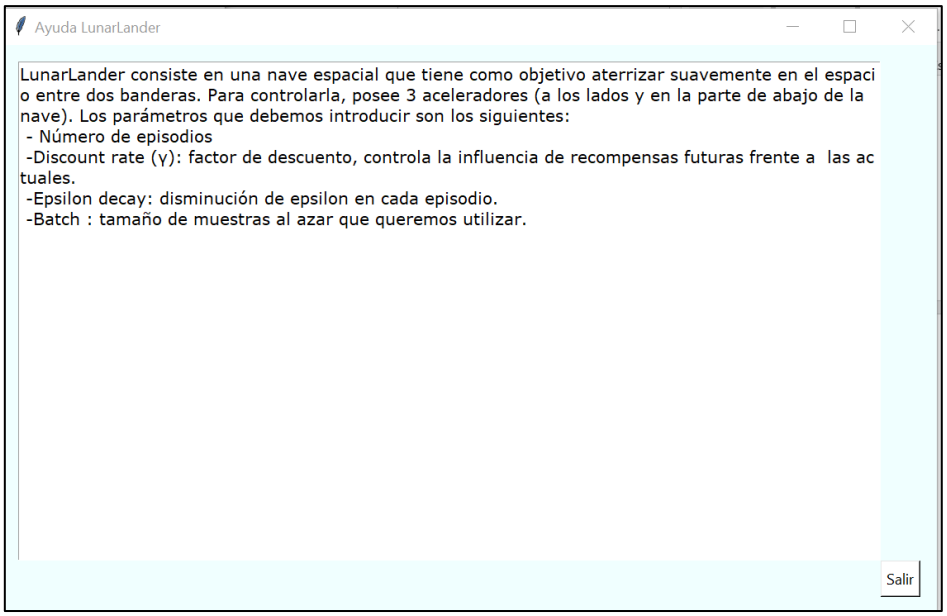

Ayuda CartPole
Cartpole es una barra enganchada por uno de sus extremos a un carro que puede moverse en una única dirección. El sistema es inestable, pero puede controlarse moviendo el carro. El objetivo del juego es mantener la barra equilibrada el mayor tiempo posible, aplicando las fuerzas apropiadas sobre el carro. Los parametros que debemos introducir son los siguientes:
- Número de episodios
-Discount rate (γ): factor de descuento, controla la influencia de recompensas futuras frente a las actuales.
-Epsilon decay: disminución de epsilon en cada episodio.
-Batch : tamaño de muestras al azar que queremos utilizar como batch.
Salir

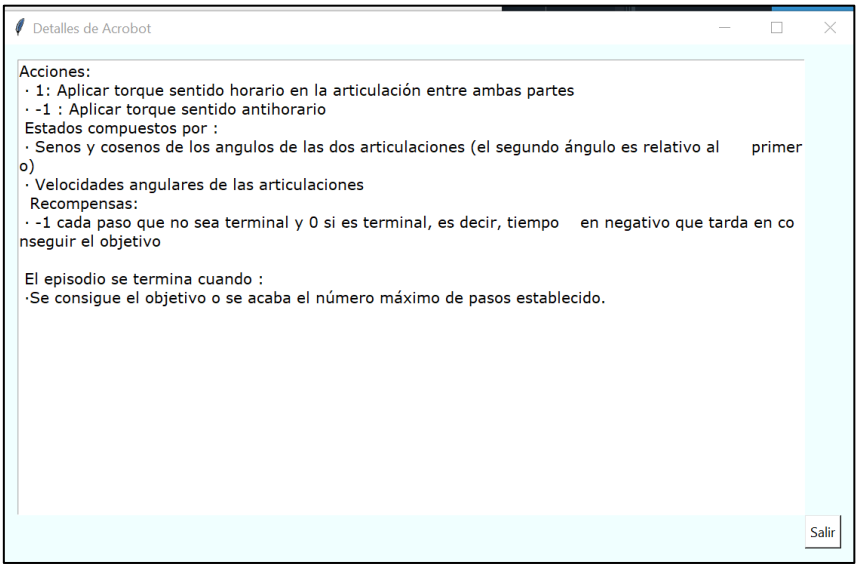

Detalles de CartPole
Acciones:
· 0 empujar carro a la izquierda
· 1 empujar carro a la derecha
Estados compuestos por :
· Posicion del carro
· Velocidad del carrro
· Ángulo de la barra
· Velocidad de la punta de la barra
Recompensas:
· 1 por cada paso dado, incluido el paso final
El episodio se termina cuando :
· Ángulo supera los ± 15 grados con la vertical
· La posición del carro es ± 2.4 (se sale de la pantalla)
Salir

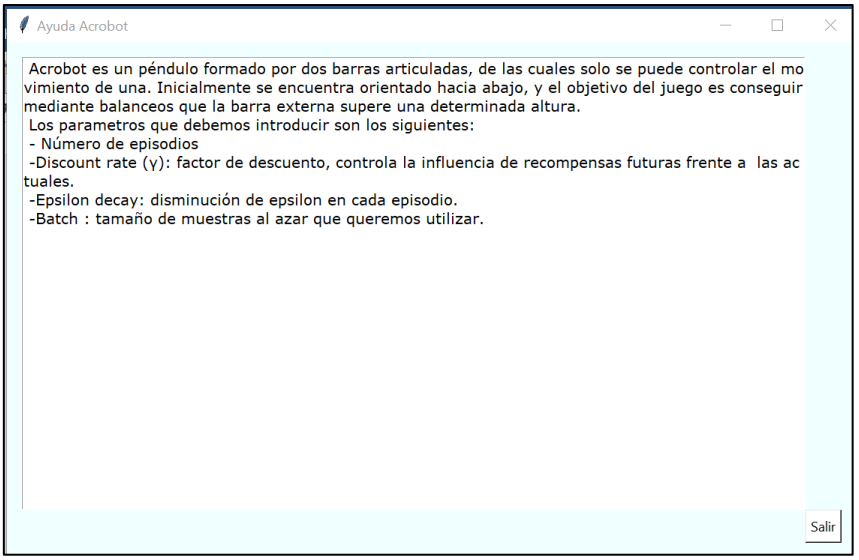

Ayuda Breakout
Breakout consiste en una capa de ladrillos en la zona superior de la pantalla y el objetivo es destruirlo
s todos. Para ello, se utiliza una pelota que puede rebotar en las paredes y en una barra que se muev
e en la zona inferior de la pantalla. Cuando la barra no llega a tiempo a la pelota, ésta se escapa y se
pierde una vida. Cada partida cuenta con 5 vidas.
Los parametros que debemos introducir son los siguientes:
- Número de episodios
-Discount rate (γ): factor de descuento, controla la influencia de recompensas futuras frente a las ac
tuales.
-Epsilon decay: disminución de epsilon en cada episodio.
-Batch : tamaño de muestras al azar que queremos utilizar.
Salir

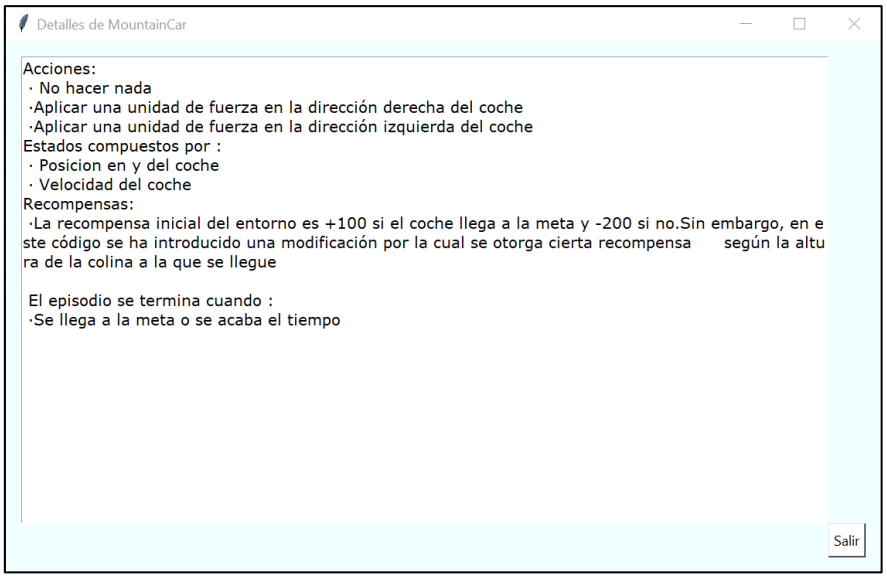

Detalles de Breakout
Acciones:
· NOOP: no ocurre nada
· RIGHT: mover barra a la derecha
· LEFT: mover barra a la izq
·FIRE: empieza el juego
Estados compuestos por :
· Posicion xde la barra
· Posicion x e y de la bola
·Velocidad x e y de la bola
·Numero de ladrillos restantes
·Número de vidas restantes
·Puntuación actual del juego
Recompensas:
·Una por cada ladrillo roto
El episodio se termina cuando :
Se acaban las vidas
Salir

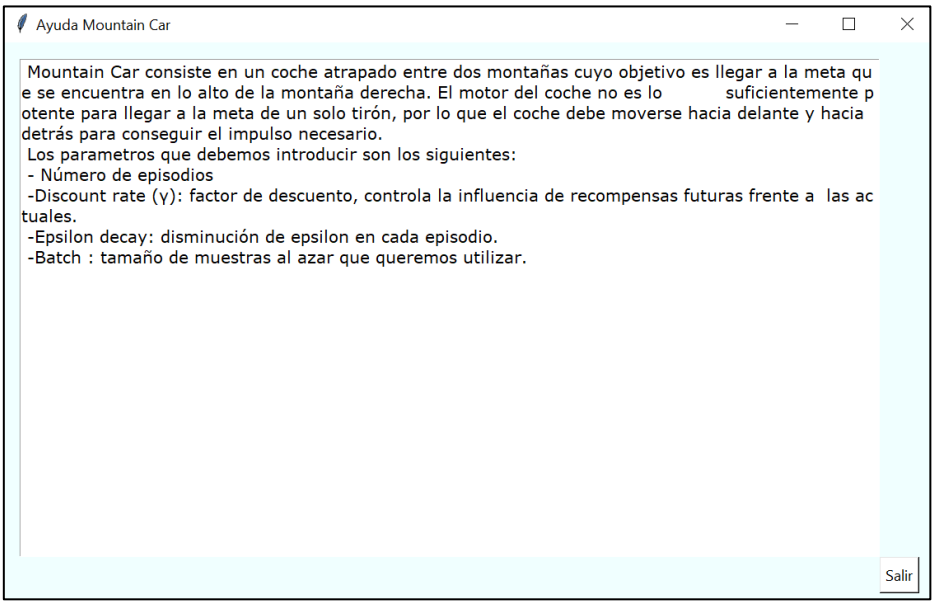

Detalles de LunarLander
Acciones:
· no ocurre nada
· Utilizar motor izquierdo
· Utilizar motor inferior
· Utilizar motor derecho
Estados compuestos por :
· Posicion x e y de la nave
· Velocidad en x e y de la nave
·Ángulo de la nave con la horizontal
·Velocidad angular de la nave
·Contacto con tierra de ambas patas de la nave (1-contacto, 0-no contacto)
Recompensas:
· +100 puntos si aterriza correctamente y -100 puntos si no, cada vez que se utilizan los motores lat
erales -0.03 y el inferior -0.3 puntos. Contacto de cada pata con tierra +10 puntos
El episodio se termina cuando :
·La nave se estrella o aterriza.
Salir


Ayuda LunarLander
LunarLander consiste en una nave espacial que tiene como objetivo aterrizar suavemente en el espaci
o entre dos banderas. Para controlarla, posee 3 aceleradores (a los lados y en la parte de abajo de la
nave). Los parámetros que debemos introducir son los siguientes:
- Número de episodios
-Discount rate (γ): factor de descuento, controla la influencia de recompensas futuras frente a las ac
tuales.
-Epsilon decay: disminución de epsilon en cada episodio.
-Batch : tamaño de muestras al azar que queremos utilizar.
Salir


Detalles de Acrobot
Acciones:
· 1: Aplicar torque sentido horario en la articulación entre ambas partes
· -1 : Aplicar torque sentido antihorario
Estados compuestos por :
· Senos y cosenos de los angulos de las dos articulaciones (el segundo ángulo es relativo al primer
o)
· Velocidades angulares de las articulaciones
Recompensas:
· -1 cada paso que no sea terminal y 0 si es terminal, es decir, tiempo en negativo que tarda en co
nseguir el objetivo
El episodio se termina cuando :
·Se consigue el objetivo o se acaba el número máximo de pasos establecido.
Salir


Ayuda Acrobot
Acrobot es un péndulo formado por dos barras articuladas, de las cuales solo se puede controlar el mo
vimiento de una. Inicialmente se encuentra orientado hacia abajo, y el objetivo del juego es conseguir
mediante balanceos que la barra externa supere una determinada altura.
Los parametros que debemos introducir son los siguientes:
- Número de episodios
-Discount rate (γ): factor de descuento, controla la influencia de recompensas futuras frente a las ac
tuales.
-Epsilon decay: disminución de epsilon en cada episodio.
-Batch : tamaño de muestras al azar que queremos utilizar.
Salir


Detalles de MountainCar
Acciones:
· No hacer nada
·Aplicar una unidad de fuerza en la dirección derecha del coche
·Aplicar una unidad de fuerza en la dirección izquierda del coche
Estados compuestos por :
· Posicion en y del coche
· Velocidad del coche
Recompensas:
·La recompensa inicial del entorno es +100 si el coche llega a la meta y -200 si no.Sin embargo, en e
ste código se ha introducido una modificación por la cual se otorga cierta recompensa según la altu
ra de la colina a la que se llegue
El episodio se termina cuando :
·Se llega a la meta o se acaba el tiempo
Salir


Ayuda Mountain Car
Mountain Car consiste en un coche atrapado entre dos montañas cuyo objetivo es llegar a la meta qu
e se encuentra en lo alto de la montaña derecha. El motor del coche no es lo suficientemente p
otente para llegar a la meta de un solo tirón, por lo que el coche debe moverse hacia delante y hacia
detrás para conseguir el impulso necesario.
Los parametros que debemos introducir son los siguientes:
- Número de episodios
-Discount rate (γ): factor de descuento, controla la influencia de recompensas futuras frente a las ac
tuales.
-Epsilon decay: disminución de epsilon en cada episodio.
-Batch : tamaño de muestras al azar que queremos utilizar.
Salir