UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

Estudio del aprendizaje por refuerzo aplicado a los videojuegos

Trabajo Fin de Grado

Grado en Tecnologías Interactivas

AUTOR/A: Tortosa Zango, Daniel

Tutor/a: Palanca Cámara, Javier

CURSO ACADÉMICO: 2022/2023

Tabla de contenido

Índice de ilustraciones

Índice de tablas

Agradecimientos

En este capítulo, quiero expresar mi sincero agradecimiento a todas aquellas personas e instituciones que han contribuido de manera significativa en la realización de este Trabajo de Fin de Grado.

En primer lugar, me gustaría agradecer a la Universidad Politécnica de Valencia por brindarme la oportunidad de cursar el grado en Tecnologías Interactivas. A lo largo de mi formación académica, he recibido una educación sólida y de calidad que me ha permitido adquirir los conocimientos necesarios para llevar a cabo este trabajo. Agradezco a todos los profesores y personal involucrado en mi educación por su dedicación y compromiso.

En segundo lugar, quiero expresar mi más profundo agradecimiento a Javier, mi tutor en este trabajo. Su experiencia, conocimientos y orientación han sido invaluables para mí. Desde el inicio, su apoyo constante y sus comentarios constructivos han sido cruciales para el avance y la mejora continua de este proyecto. Estoy agradecido por su disposición a escuchar mis ideas, brindar orientación experta y ofrecer soluciones cuando me enfrenté a desafíos. Ha sido un privilegio trabajar con alguien tan dedicado y apasionado por la materia.

También quiero extender mi gratitud a mi familia por su inquebrantable apoyo a lo largo de mi carrera universitaria. Agradezco profundamente su compresión y paciencia durante este proceso. Sin su respaldo y confianza en mí, no habría sido posible alcanzar mis metas académicas. Estoy eternamente agradecido por el amor y el apoyo incondicional que me han brindado.

Por último, pero no menos importante, deseo agradecer a mis amigos. Su compañía, ánimo y amistad han sido fundamentales para mantenerme motivado y equilibrar mi vida durante este desafiante periodo de estudio. Agradezco su apoyo, compresión y momentos compartidos, que han sido un alivio bienvenido en momentos de estrés. Su presencia ha sido un recordatorio constante de la importancia de tener un equilibrio entre el trabajo y la vida persona

Resumen

En este proyecto se abordará el estudio de la Inteligencia Artificial, en concreto el campo del Aprendizaje por Refuerzo, para generar modelos que aprendan a jugar a determinados videojuegos clásicos en 2D. Para ello se realizará un amplio estudio del campo del Aprendizaje por Refuerzo y del Aprendizaje por Refuerzo Profundo y de su estado del arte para investigar los diferentes algoritmos que se pueden aplicar a este ámbito y realizar un estudio de los mismos.

Se diseñará un conjunto de experimentos y se implementará un conjunto de algoritmos seleccionados con el fin de realizar un estudio de los mismos en función de los resultados de aprendizaje que obtengan.

Resum

En aquest projecte s'abordarà l'estudi de la Intel·ligència Artificial, en concret el camp de l'Aprenentatge per Reforç, per a generar models que aprenguen a jugar a determinats videojocs clàssics en 2D. Per a això es realitzarà un ampli estudi del camp de l'Aprenentatge per Reforç i de l'Aprenentatge per Reforç Profund i del seu estat de l'art per a investigar els diferents algorismes que es poden aplicar a aquest àmbit i realitzar un estudi d'aquests.

Es dissenyarà un conjunt d'experiments i s'implementarà un conjunt d'algorismes seleccionats amb la finalitat de realitzar un estudi dels mateixos en funció dels resultats d'aprenentatge que obtinguen.

Abstract

This Project Will address the study of Artificial Intelligence, specifically the field of Reinforcement Learning, to generate models that learn to play certain classic 2D video games. To achieve this, an extensive study of the field of Reinforcement Learning and Deep Reinforcement Learning, as well as their state of the art, will be conducted to investigate the different algorithms that can be applied in this domain and conduct a study of them.

A set of experiments will be designed, and a selected set of algorithms will be implemented to study them based on the learning results they achieve.

Palabras clave

Inteligencia artificial; aprendizaje por refuerzo; q-learning; deep learning; videojuegos; redes neuronales; entrenamiento de agentes; gymnasium; acciones; recompenasas; experimentación; agente-jugador; hiperparámetros; Mountain Car; CartPole; Acrobot;

1. Introducción:

1.1 Introducción

El objetivo principal de este Trabajo de Fin de Grado es adquirir conocimientos sobre el campo del Aprendizaje por Refuerzo (Reinforcement Learning) y desarrollar un modelo de Inteligencia Artificial capaz de aprender a jugar un juego utilizando la biblioteca Gymnasium, basada en OpenAI Gym. A lo largo de este proyecto, se explorarán diversos algoritmos y escenarios con el propósito de entrenar a un agente en cada uno de ellos, evaluando su efectividad a través de representaciones gráficas.

1.2 Motivación

Desde mi infancia, siempre he sentido una gran pasión por los videojuegos y la tecnología. El mundo de los juegos me ha proporcionado horas de entretenimiento y desafíos emocionantes. Sin embargo, a medida que fui aprendiendo más sobre la inteligencia artificial y el aprendizaje automático, me di cuenta de que había un vasto potencial para combinar estos dos campos y explorar cómo los agentes de inteligencia pueden aprender a jugar y mejorar en los juegos de manera similar a los seres humanos.

Uno de los aspectos más emocionantes del Aprendizaje por Refuerzo aplicado a los juegos es la posibilidad de que un agente pueda superar a los jugadores humanos en rendimiento y habilidad. La idea de que un agente de inteligencia artificial pueda aprender de forma autónoma y alcanzar niveles de rendimiento que están más allá de las capacidades humanas tradicionales es realmente emocionante y despierta mi curiosidad y motivación para explorar esta área en mi TFG.

2. Objetivos

El objetivo general de este trabajo es, desarrollar a un agente de Inteligencia Artificial basado en el Aprendizaje por refuerzo para aprender a jugar un juego de la biblioteca Gymnasium. El objetivo es aplicar diferentes algoritmos de Aprendizaje por refuerzo y evaluar el desempeño del agente en diferentes escenarios de juego.

Objetivos para conseguir:

- Estudiar los fundamentos teóricos del aprendizaje por refuerzo y familiarizarse con los algoritmos más comunes.
- Explorar la biblioteca Gymnasium y seleccionar los entornos de juego adecuados para el desarrollo del agente.
- Implementar y entrenar el agente utilizando los algoritmos seleccionados.
- Evaluar y comparar el desempeño del agente en términos de recompensas obtenidas y mejora de rendimiento.
- Documentar los resultados obtenidos, incluyendo los algoritmos implementados, los escenarios de juego utilizados y las conclusiones del estudio.

3. Planificación

Para el desarrollo de este proyecto se ha utilizado la metodología SCRUM, esta es una metodología ágil ampliamente utilizada en la gestión de proyectos. Se basa en un enfoque iterativo e incremental, donde los proyectos se dividen en ciclos llamados “sprints”.

Los roles que hay en esta metodología son:

- Product Owner: Responsable de definir los requisitos y prioridades del proyecto.
- Scrum Master: Actúa como facilitador del equipo de desarrollo, asegurándose de que se sigan las prácticas y principios de Scrum.
- Equipo de Desarrollo: Está compuesto por profesionales encargados de desarrollar el producto o el proyecto.

Hay distintos elementos necesarios para seguir la metodología SCRUM que son los siguientes:

- Product Backlog: Es una lista de todas las funcionalidades, características y requisitos que se deben desarrollar. Se mantiene actualizada y priorizada por el Product Owner.
- Sprint Backlog: Es una lista de tareas específicas que se seleccionan del Product Backlog.

También hay distintos eventos a seguir:

- Sprint Planning: Es una reunión donde se seleccionan las tareas del Product Backlog para el Sprint Backlog y se define el objetivo del sprint.
- Sprint Review: Es una reunión al final de cada sprint donde se presenta el incremento desarrollado y se recopila el feedback del Product Owner y los stakeholders.

En nuestro caso, el Scrum Master y el Equipo de Desarrollo soy yo, el dueño de este trabajo de final de grado y el Product Owner es el tutor de este trabajo, Javier Palanca.

	Nombre	Duración	Inicio	Fin
1	Sprint 0 (Gestión del proyecto)	15 días	01/03/2023	15/03/2023
2	Sprint 1	30 días	16/03/2023	14/04/2023
3	Sprint Backlog 1	1 día	16/03/2023	16/03/2023
4	Estudio teórico aprendizaje por refuerzo	7 días	17/03/2023	23/03/2023
5	Estudio algoritmos asociados	21 días	24/03/2023	13/04/2023
6	Sprint Review 1	1 día	14/04/2023	14/04/2023
7	Sprint 2	35 días	15/04/2023	19/05/2023
8	Sprint Backlog 2	1 día	15/04/2023	15/04/2023
9	Exploración de la biblioteca Gymnasium	9 días	16/04/2023	24/04/2023
10	Selección de los entornos de juego	6 días	25/04/2023	30/04/2023
11	Estudio entornos de juego seleccionados	18 días	01/05/2023	18/05/2023
12	Sprint Review 2	1 día	19/05/2023	19/05/2023
13	Sprint 3	42 días	20/05/2023	30/06/2023
14	Sprint Backlog 3	1 día	20/05/2023	20/05/2023
15	Implementación y entrenamiento del algoritmo 1 en los escenarios seleccionados	8 días	21/05/2023	28/05/2023
16	Implementación y entrenamiento del algoritmo 2 en los escenarios seleccionados	12 días	29/05/2023	09/06/2023
17	Implementación y entrenamiento del algoritmo 3 en los escenarios seleccionados	20 días	10/06/2023	29/06/2023
18	Sprint Review 3	1 día	30/06/2023	30/06/2023
19	Sprint 4	31 días	01/07/2023	31/07/2023
20	Sprint Backlog 4	1 día	01/07/2023	01/07/2023
21	Evaluación del desempeño del agente con los algoritmos utilizados en el primer escenario.	8 días	02/07/2023	09/07/2023
22	Evaluación del desempeño del agente con los algoritmos utilizados en el segundo escenario.	10 días	10/07/2023	19/07/2023
23	Evaluación del desempeño del agente con los algoritmos utilizados en el tercer escenario.	11 días	20/07/2023	30/07/2023
24	Sprint Review 4	1 día	31/07/2023	31/07/2023
25	Sprint 5 - Final	36 días	01/08/2023	05/09/2023
26	Sprint Backlog 5 - Final	1 día	01/08/2023	01/08/2023
27	Documentación de los resultados obtenidos	14 días	02/08/2023	15/08/2023
28	Redacción final	20 días	16/08/2023	04/09/2023
29	Sprint Review 5 - Final	1 día	05/09/2023	05/09/2023

TABLA 1 - PLANIFICACIÓN DEL PROYECTO

En la Tabla 1 - Planificación del proyecto muestra la planificación del proyecto utilizando la metodología SCRUM, donde se ha organizado el tiempo de ejecución del proyecto desde el 01/03/2023 hasta el 05/09/2023 en sprints. Esta planificación detallada proporciona una estructura clara y un seguimiento adecuado del progreso del proyecto a lo largo de su desarrollo.

4. Marco teórico

La inteligencia artificial ha desempeñado un papel significativo en la evolución de los videojuegos, ya sea en el desarrollo de personajes no jugadores inteligentes, la mejora de la experiencia del jugador o la creación de entornos virtuales más realistas.

Este capítulo busca proporcionar una visión general de los diferentes enfoques, técnicas y algoritmos utilizados en la aplicación de la inteligencia artificial a los videojuegos.

4.1 ¿Qué es el aprendizaje por refuerzo?

En este capítulo explicaremos en que consiste el aprendizaje por refuerzo y qué estudia este subcampo del Machine Learning y sus aplicaciones. [1]

En el aprendizaje tradicional del Machine Learning encontramos 2 grandes áreas:

- Aprendizaje supervisado: Los datos introducidos para el entrenamiento incluyen la solución de cada ejemplo, esta solución es llamada por “etiquetas”. Un ejemplo sería entrenar una IA capaz de clasificar a partir de fotos si se trata de un perro o un gato, para ello primeramente se entrenaría con una librería de fotos de perros y gatos clasificados previamente.
- Aprendizaje no supervisado: A diferencia del aprendizaje supervisado, este no incluye “etiquetas”, por lo que el objetivo del algoritmo es que sea capaz de clasificar o descifrar la información por sí solo. Un ejemplo muy común sería un detector de anomalías, es decir, en producción industrial el estudio de anomalías podría decir cuando es necesario realizar un mantenimiento de la maquinaria, así conseguiríamos reducir costes por mantenimientos no necesarios.

En el aprendizaje por refuerzo, a diferencia del aprendizaje supervisado, no tenemos una “etiqueta de salida” y tampoco son de tipo no supervisado, ya que no intenta clasificar grupos teniendo en cuenta alguna distancia entre muestras.

En los modelos clásicos se busca “minimizar la función de coste”, reducir el error, en cambio en el aprendizaje por refuerzo se intenta “maximizar la recompensa”.

4.2 Elementos del RL

En primer lugar, definiremos algunos términos para poder comprender mejor los algoritmos aplicados.

- Agente: Modelo al que queremos entrenar y queremos que aprenda a tomar decisiones.
- Ambiente: Entorno en el que el agente interactuará y se moverá. El agente contendrá limitaciones y reglas para cada momento.
- Acción: Conjunto de acciones posibles que puede tomar el agente.
- Estado: Indicadores del ambiente de cómo están los elementos que lo componen en ese momento.
- Recompensa: Después de cada acción tomada por el Agente, podremos obtener un premio o una penalización, consiguiendo así orientar al Agente para saber si lo está haciendo bien o mal.

El aprendizaje por refuerzo intentará hacer que el agente aprenda con un esquema de “recompensa y castigo”. El funcionamiento de este es que el agente recibe un estado inicial y toma una acción con la que influye e interviene en el ambiente y este le devuelve un estado y una recompensa. Si la recompensa es positiva estaremos reforzando el

comportamiento para el futuro. En cambio, si la recompensa es negativa la estaremos penalizando, para que el agente actúe de manera distinta.

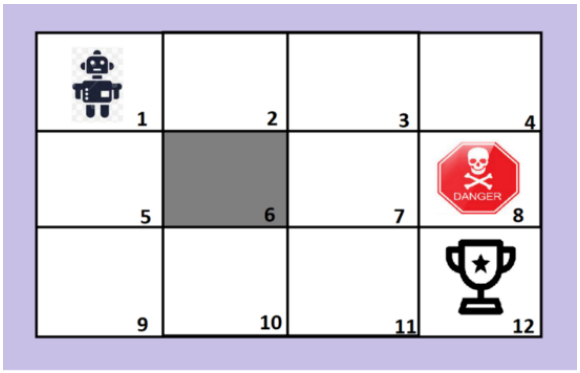

ILUSTRACIÓN 1 - EJEMPLO DE REINFORCEMENT LEARNING

En la Ilustración 1 - Ejemplo de Reinforcement Learning encontramos un claro ejemplo de Reinforcement Learning, que es el laberinto, en el que el estado inicial del agente es en la casilla número 1, si este fuera hacia la derecha hasta llegar a la casilla número 4 y luego hacia abajo obtendría una recompensa negativa por estar en la casilla número 8 y en un futuro intentaría no hacer estas acciones, pero en cambio, si el agente fuera hacia abajo hasta la casilla número 9 y luego hacia la derecha hasta la casilla número 12 obtendría una recompensa positiva, por lo que el robot tendería a hacer estas acciones mientras consiguiera una recompensa positiva.

4.3 Proceso de Decisión de Markov

A continuación, explicaremos los “Procesos de Decisión de Markov” [2] ya que es una forma de representar matemáticamente el conjunto de elementos en el que se apoya el Reinforcement Learning.

Un proceso de decisión de Markov es un proceso sin memoria y aleatorio, en otras palabras, una secuencia de estados aleatorios que posee la propiedad de Marvkov.

Los Procesos de Decisión de Markov son la forma idealizada matemáticamente del problema de aprendizaje por refuerzo.

La propiedad de Markov nos muestra que el futuro es independiente del pasado, dado el presente, lo cual se presentaría en la siguiente formula:

$$\mathbb{P}\left[S_{t+1} \middle| S_t\right] = \mathbb{P}\left[S_{t+1} \middle| S_1, \ldots, S_t\right]$$

ILUSTRACIÓN 2 - ECUACIÓN PROCESOS DE DECISIÓN DE MARKOV

En la Ilustración 2 - Ecuación Procesos de Decisión de Markov, el estado actual es representado por “St”, y la información de su estado es independiente de los estados anteriores, “|S1,…,St|”.

$$\begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & & \\ P_{n1} & & P_{nn} \end{bmatrix}$$

ILUSTRACIÓN 3 - ECUACIÓN MATRIZ DE TRANSICIÓN DE ESTADOS

En la Ilustración 3 - Ecuación Matriz de transición de estados explicamos otro componente del proceso de Decisión de Markov, esta nos muestra cual sería la probabilidad de transición desde un estado S a un estado S.

Un ejemplo de esta Matriz sería en un juego de fútbol en el que el jugador va a chutar un penalti, en este pasan por muchos estados, empezando por el que el jugador chute la pelota, seguidamente la pelota puede ir fuera, pegarle al palo, o que vaya en dirección a la portería y seguidamente que sea gol o no, ya sea porque el portero pare la pelota o porque el agente falle el penalti.

Los Procesos de Recompensa de Markov, se pueden ver como procesos de Markov normales con valores que juzgan si un estado es positivo o no, este contendría los siguientes elementos:

- S es una lista de estados a los que puede acceder
- P es una matriz que conforman las transiciones de un estado a otro
- R la recompensa del estado en el que estamos
- γ varía entre 0 y 1 y es el valor de descuento

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

ILUSTRACIÓN 4 - ECUACIÓN FUNCIÓN DE RETORNO

Seguidamente, tenemos la Ilustración 4 - Ecuación Función de Retorno, que es el total de recompensa multiplicado por el valor de descuento en cada paso de tiempo. Si el valor de descuento es 0, solo hay que tener en cuenta la primera recompensa, por otro

lado, si el valor de descuento es 1, se tendrían en cuenta todas las recompensas por igual.

El descuento es fundamental usarlo porque evita bucles infinitos en procesos de Markov cíclicos.

$$v(s) = \mathbb{E}\left[G_t \middle| S_t = s \right]$$

ILUSTRACIÓN 5 - ECUACIÓN FUNCIÓN DE VALOR

Finalmente tenemos la Ilustración 5 - Ecuación Función de Valor.

Esta nos muestra cual es la recompensa esperada desde ese estado hasta el final de la secuencia.

4.4 Q-Learning

El algoritmo de Q-Learning [3] es una forma sencilla para que los agentes aprendan a actuar de manera óptima en situaciones markovianas controladas. Es uno de los modelos más usados en Reinforcement Learning.

Q-Learning es una forma de aprendizaje por refuerzo sin modelos y también puede ser visto como un método de programación dinámica asíncrona.

El aprendizaje procede de la siguiente manera: Un agente intenta una acción en un estado particular y evalúa sus consecuencias en términos de la recompensa o penalización inmediata que recibe y su estimación del valor del estado al que se lleva. Al intentar todas las acciones en todos los estados repetidamente, aprende cuales son las mejores acciones a llevar a cabo según el estado o cadena de estados de los que se parte.

Los elementos que contiene este algoritmo son:

- Políticas: Una tabla (con tamaño n-dimensiones) en el que le indicará al agente que acción tomar en cada estado.
- Acciones: Diversas elecciones que puede hacer el agente en cada estado.
- Recompensas: Si sumamos o restamos puntaje con la acción elegida.
- Comportamiento del agente: Es decir si tomará las acciones con más recompensa o irá tomando otro tipo de acciones a lo largo de las iteraciones, como, por ejemplo, cada 10 acciones harán una aleatoria.

El objetivo principal de este modelo es que a través de simulaciones iremos “rellenando” la tabla de Políticas, de manera que las decisiones que vaya tomando nuestro agente vayan obteniendo mayor recompensa a la vez que avanzamos y no nos quedamos estancados, pudiendo así cumplir el objetivo que deseamos alcanzar.

A la política normalmente se le llama “Q” y para saber cómo ir completando la tabla de políticas se utiliza la ecuación de Bellman.

4.4.1 Ecuación de Bellman

A continuación, se presenta la Ecuación de Bellman, que será necesaria para rellenar la tabla de políticas del algoritmo Q-Learning.

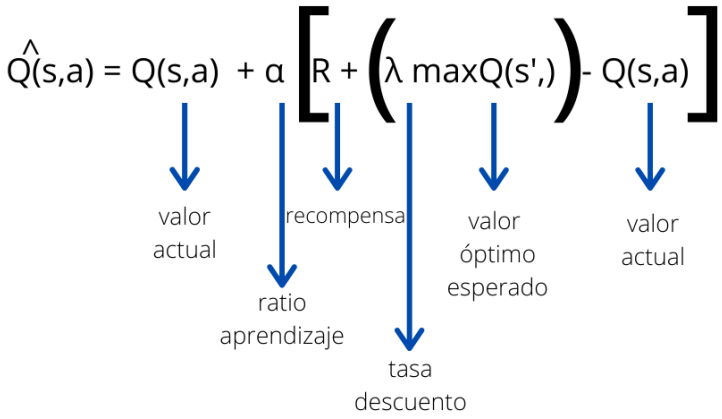

ILUSTRACIÓN 6 - ECUACIÓN DE BELLMAN

Seguidamente encontramos la Ilustración 6 - Ecuación de Bellman, esta ecuación también se denomina como "la ecuación de programación dinámica", nombrada en honor a su descubridor Richard Bellman. Esta ecuación explica como hay que ir actualizando la tabla de Políticas, y se calcula en base al valor actual sumado a la futura recompensa que se conseguiría por tomar esa acción en concreto.

Hay dos variables que afectan de manera directa a esta ecuación:

- El ratio de aprendizaje, ya que regula "la velocidad" en la que el agente aprende.
- La tasa de descuento, porque tiene en cuenta la recompensa a corto o largo plazo.

4.4.2 Q-Learning: conceptos básicos

El Q-Learning es un algoritmo de aprendizaje por refuerzo que permite a un agente aprender a tomar decisiones óptimas en un entorno desconocido basándose en las recompensas recibidas por sus acciones. A continuación, se presentan los conceptos básicos del Q-Learning [1]:

1. Agente: Es la entidad que aprende a través del algoritmo Q-Learning. El agente interactúa con el entorno, toma decisiones y aprende a maximizar las recompensas obtenidas.
2. Entorno: Es el contexto o mundo en el que el agente opera y toma decisiones. Puede ser cualquier sistema o situación donde el agente interactúe y reciba retroalimentación en forma de recompensas.
3. Estado: Es una representación del entorno en un momento específico. Puede incluir información relevante para la toma de decisiones, como la posición, las condiciones ambientales o cualquier otro factor relevante para el problema.
4. Acción: Es la elección que realiza el agente en un estado determinado. Puede ser cualquier acción posible en el entorno, como moverse a una dirección específica, realizar una acción particular, etc.
5. Recompensa: Es una señal de retroalimentación que el agente recibe del entorno después de realizar una acción en un estado dado. La recompensa

puede ser positiva, negativa o neutra y sirve para evaluar la calidad de las acciones tomadas por el agente.

6. Q-Valor: Es un valor numérico que representa la utilidad esperada de una acción en un estado específico. El Q-Valor se utiliza para determinar qué acción tomar en un estado determinado y se actualiza mediante la iteración del algoritmo Q-Learning.
7. Tabla Q: Es una estructura de datos que almacena los Q-Valores para todos los pares estado-acción posibles. La tabla Q se inicializa de manera arbitraria y se actualiza a medida que el agente explora y aprende del entorno.
8. Política de selección de acciones: Es una estrategia que el agente utiliza para elegir la mejor acción en un estado dado. Puede ser determinista, siempre seleccionando la acción con el Q-Valor más alto, o probabilística, considerando la exploración y explotación del entorno.
9. Actualización del Q-Valor: Es el proceso mediante el cual se actualizan los Q-Valores de la tabla Q después de que el agente realiza una acción y recibe una recompensa. La actualización se basa en la diferencia entre la recompensa obtenida y la estimación actual del Q-Valor para esa acción y estado.

Estos conceptos básicos son fundamentales para comprender el funcionamiento del algoritmo Q-Learning y su aplicación en diversos problemas de aprendizaje por refuerzo.

4.4.3 Representación del espacio de estados y acciones

En el algoritmo Q-Learning, la representación del espacio de estados y acciones es crucial para el aprendizaje y toma de decisiones del agente. A continuación, se presentan algunas consideraciones sobre cómo representar el espacio de estados y acciones en Q-Learning:

1. Espacio de estados [15]:
    - Discreto: Si el espacio es discreto, se puede utilizar una representación directa mediante una estructura de datos, como una tabla, donde cada estado se mapea a una entrada en la tabla. Cada entrada de la tabla contiene los Q-Valores correspondientes a las acciones posibles en ese estado.
    - Continuo: Si el espacio de estados es continuo, se requiere una técnica de discretización para convertirlo en un estado discreto. Esto puede lograrse dividiendo el espacio continuo en una cuadrícula o utilizando técnicas de agrupación para agrupar estados similares. Una vez discretizado, se puede utilizar una representación similar a la del espacio de estados discreto.
    - Funciones de aproximación: En algunos casos, especialmente cuando el espacio de estados es grande o continuo, se utilizan funciones de aproximación, como redes neuronales, para estimar los Q-Valores en lugar de utilizar una representación tabular. Estas funciones de aproximación toman como entrada el estado y devuelven una estimación del Q-Valor para cada acción posible.
2. Espacio de acciones [16]:
    - Discreto: Si el espacio de acciones es discreto, se puede utilizar una representación directa mediante una lista o un conjunto de acciones posibles. Cada acción se asocia con un Q-Valor en la tabla Q o en la función de aproximación.

- Continuo: Si el espacio de acciones es continuo, se requiere una técnica de discretización o un enfoque basado en funciones de aproximación para manejar las acciones continuas. Una opción es discretizar el espacio de acciones en un conjunto finito de acciones posibles. Otra opción es utilizar funciones de aproximación para estimar los Q-Valores para diferentes acciones continuas.

La elección de la representación del espacio de estados y acciones depende del problema específico y de las características del entorno. Es importante considerar el equilibrio entre la complejidad de la representación y la eficiencia computacional, así como la capacidad de capturar la información relevante para la toma de decisiones óptimas.

En conclusión, en el algoritmo Q-Learning, el espacio de estados y acciones se puede representar de manera discreta o continua. Para espacios de estados y acciones discretos, se puede utilizar una representación tabular, mientras que para espacios continuos se requiere una técnica de discretización o el uso de funciones de aproximación. La elección de la representación depende del problema y de las necesidades específicas del entorno.

4.4.4 Parámetros de aprendizaje

En el algoritmo Q-Learning, existen varios parámetros de aprendizaje [1] que influyen en el proceso de actualización de los Q-Valores y en la toma de decisiones del agente. A continuación, se describen los parámetros clave y su impacto en el algoritmo:

1. Tasa de aprendizaje (α):
   - La tasa de aprendizaje controla la rapidez con la que se actualizan los Q-Valores en cada iteración. Un valor alto de α significa que los nuevos valores Q se ponderan más en cada actualización, lo que permite un aprendizaje más rápido, pero puede dificultar la convergencia a una solución óptima. Por otro lado, un valor bajo de α significa que los Q-Valores cambian lentamente, lo que puede llevar a un aprendizaje más estable pero más lento.
2. Factor de descuento (γ):
   - El factor de descuento determina la importancia relativa de las recompensas a corto plazo y a largo plazo. Un valor de γ cercano a 0 significa que el agente se enfoca principalmente en las recompensas inmediatas, mientras que un valor cercano a 1 considera tanto las recompensas a corto plazo como las acumuladas a largo plazo. La elección de γ depende del problema y la naturaleza de las recompensas.
3. Parámetro de exploración (ε):
   - El parámetro de exploración controla la proporción de acciones aleatorias que el agente toma en lugar de seguir la política aprendida. Un valor alto de ε significa una mayor exploración, lo que permite al agente descubrir nuevas acciones y estados, pero puede prolongar el tiempo de convergencia. Un valor bajo de ε favorece la explotación de la política aprendida y puede llevar a una convergencia más rápida, pero existe el riesgo de quedarse atrapado en óptimos locales.

Estos son los parámetros principales del algoritmo Q-Learning que afectan directamente el proceso de aprendizaje y la toma de decisiones del agente. La elección adecuada de estos parámetros puede tener un impacto significativo en el rendimiento y la eficiencia

del algoritmo. Es común realizar experimentos y ajustes en los valores de estos parámetros para encontrar la configuración óptima para un problema específico.

Además de estos parámetros, también hay otros factores que pueden influir en el desempeño del Q-Learning, como el tamaño y la complejidad del espacio de estados de acciones, la inicialización de los Q-Valores y los criterios de terminación del algoritmo.

En conclusión, los parámetros explicados anteriormente, afectan la velocidad de aprendizaje, la importancia relativa de las recompensas a corto y largo plazo, y el equilibrio entre la exploración y la explotación. La elección adecuada de estos parámetros es importante para lograr un buen desempeño del algoritmo.

4.5 Deep Q-Network

El algoritmo de Deep Q-Network o Deep Q-Learning [4] combina Q-Learning con una red neuronal profunda. Se sabe que con este algoritmo se ha llegado a mejorar el rendimiento de los agentes en varios juegos y que las redes neuronales son una fantástica manera de aproximar funciones no lineales.

Este algoritmo usa una red neuronal para aproximar la función Q, evitando así utilizar una tabla para representar la misma. En realidad, este algoritmo usa 2 redes neuronales, una principal que es para estimar los valores Q del estado s y las acciones, y la segunda es la red neuronal objetivo, que tiene la misma arquitectura que la principal, pero su objetivo es aproximar valores Q del siguiente estado y siguiente acción. El aprendizaje ocurre en la red principal y no en el objetivo.

$$Q(s, a; \theta) = r + \gamma \max_{a'} Q(s', a'; \theta')$$

ILUSTRACIÓN 7 – ECUACIÓN DE BELLMAN EN DQN

La Ilustración 7 – Ecuación de Bellman en DQN representa la forma actual de esta para poder entrenar una red neuronal, para ello también es necesario una función de coste.

$$L(\theta) = \mathbb{E}\left[(r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta))^2\right]$$

ILUSTRACIÓN 8 – ECUACIÓN FUNCIÓN DE COSTE DQN

La Ilustración 8 – Ecuación Función de coste DQN es la función que utiliza este algoritmo de descenso de gradientes, el cual en la actualidad se ejecuta automáticamente con librerías como TensorFlow o Pytorch.

4.5.1 Tipos de redes neuronales

Existen varios tipos de redes neuronales [17] que se utilizan en diferentes aplicaciones y contextos.

A continuación, se presentan algunos de los tipos más comunes:

- Redes Neuronales Feedforward (FNN): También conocidas como redes neuronales de alimentación directa, son el tipo más básico y común de redes neuronales. La información fluye en una sola dirección. Desde la capa de entrada hacia la capa de salida, sin ciclos o retroalimentación. Son ampliamente utilizadas en tareas de clasificación, reconocimiento de patrones y aproximación de funciones.
- Redes Neuronales Convolucionales (CNN): Estas redes están diseñadas específicamente para el procesamiento de datos en forma de malla, como

imágenes o señales 2D. Utilizan capas convolucionales para extraer características espaciales y patrones locales de los datos, seguidas de capas de agrupación (pooling) para reducir la dimensionalidad. Las CNN son ampliamente utilizadas en tareas de visión por computadora, como reconocimiento de imágenes y detección de objetos.

- Redes Neuronales Recurrentes (RNN): A diferencia de las FNN, las RNN tienen conexiones retroalimentadas, lo que les permite tener una memoria interna y procesar secuencias de datos. Cada neurona en una capa RNN recibe tanto las entradas actuales como la salida de las neuronas anteriores, lo que les permite capturar la dependencia temporal en los datos. Las RNN son adecuadas para tareas como el procesamiento de lenguaje natural, la traducción automática y la generación de texto.
- Redes Neuronales Generativas Adversariales (GAN): Las GAN son un tipo especial de red neuronal compuesta por dos partes: un generador y un discriminador. El generador crea muestras sintéticas que intentan imitar las muestras reales, mientras que el discriminador intenta distinguir entre las muestras reales y las generadas. Estas redes se utilizan en la generación de imágenes, la síntesis de voz y otras tareas de generación de contenido.

Estos son algunos ejemplos de los tipos de redes neuronales más comunes. Cada tipo de red neuronal tiene sus propias características y se adapta mejor a diferentes tipos de datos y problemas específicos.

4.5.2 Arquitectura de la red neuronal

La arquitectura de la red neuronal en el contexto del Deep Q-Network juega un papel fundamental en el proceso de aprendizaje y toma de decisiones.

La arquitectura de la red neuronal en el Deep Q-Network generalmente se basa en una estructura conocida como Red Neuronal Profunda o Red Neuronal Convolucional Profunda, dependiendo del tipo de datos de entrada que se estén utilizando.

En el caso de problemas donde los datos de entrada son imágenes, como en muchos juegos de OpenAI Gym, la arquitectura más comúnmente utilizada es la Red Neuronal Convolucional Profunda. Estas redes son particularmente efectivas para extraer características relevantes de imágenes y permiten un aprendizaje profundo y jerárquico. Las siglas son DCNN (Deep Convolutional Neural Network).

La arquitectura básica de una DCNN en el contexto del Deep Q-Network consta de varias capas convolucionales seguidas de capas de agrupación (pooling) y capas completamente conectadas. Las capas convolucionales se encargan de extraer características espaciales y patrones locales de las imágenes del juego, mientras que las capas de agrupación reducen la dimensionalidad de las características extraídas. Luego, las capas completamente conectadas se encargan de mapear las características extraídas a los valores Q correspondientes a cada acción posible.

La cantidad y el tamaño de las capas convolucionales y completamente conectadas dependen del problema específico y de la complejidad del entorno en el que se está aplicando el Deep Q-Network. En general, a medida que aumenta la complejidad del problema, es común utilizar redes más profundas con un mayor número de capas convolucionales y completamente conectadas.

Además de la arquitectura básica, existen varias técnicas y mejoras que se pueden aplicar en la arquitectura de la red neuronal para mejorar el rendimiento del Deep Q-Network.

Algunas de estas técnicas incluyen el uso de capas de normalización (como Batch Normalization), capas de regularización (como Dropout), tipo de función de activación (como ReLU) y arquitecturas más avanzadas, como las redes residuales (ResNets) o las redes neuronales recurrentes (RNNs).

Es importante destacar que el diseño de la arquitectura de la red neuronal en el Deep Q-Network es un proceso de prueba y error. Se requiere experimentación y ajuste de los hiperparámetros para encontrar la arquitectura óptima que se adapte al problema específico y permita un aprendizaje eficiente y estable.

En definitiva, la arquitectura de la red neuronal en el Deep Q-Network se basa en una estructura de Red Neuronal Convolucional Profunda (DCNN) en el caso de problemas de imágenes. Esta arquitectura consta de capas convolucionales, capas de agrupación y capas completamente conectadas y se puede mejorar con técnicas adicionales como normalización, regularización y activación no lineal. El diseño de la arquitectura requiere experimentación y ajuste para lograr un rendimiento óptimo en el problema específico abordado.

4.5.3 Técnicas de exploración y explotación

En el Deep Q-Network, la exploración y la explotación son dos estrategias fundamentales para que el agente aprenda y tome decisiones óptimas en un entorno de aprendizaje por refuerzo.

A continuación, se presentan algunas técnicas comunes utilizadas para equilibrar la exploración y la explotación en el Deep Q-Network:

- Epsilon-Greedy: Estrategia básica donde el agente elige la acción óptima (explotación) la mayor parte del tiempo, pero también realiza acciones aleatorias (exploración) con una probabilidad épsilon. El valor de épsilon se puede establecer inicialmente en un valor alto para promover la exploración y luego ir disminuyendo gradualmente a medida que el agente adquiere más conocimiento.
- Softmax: En lugar de elegir la acción óptima o aleatoria según una probabilidad fija, se puede utilizar una función softmax para asignar probabilidades a cada acción. La función softmax utiliza los valores Q como entradas y produce una distribución de probabilidad sobre las acciones posibles. Esto permite que el agente explore diferentes acciones en función de su incertidumbre sobre cual es la mejor opción.
- UCB (Upper Confidence Bound): Esta técnica se basa en el principio de que las acciones que se han explorado menos tienen un mayor potencial de recompensa. Se utiliza una fórmula que considera tanto el valor Q estimado como la incertidumbre (basada en el número de veces que se ha explorado la acción) para seleccionar la acción con la mayor puntuación UCB.
- Experience Replay: Esta técnica consiste en almacenar las experiencias pasadas del agente en un búfer de reproducción (replay buffer) y utilizar un muestreo aleatorio de esas experiencias durante el entrenamiento. Esto permite al agente aprender de forma más eficiente y explorar diferentes acciones en función de experiencias pasadas.

Estas son algunas de las técnicas más utilizadas en el Deep Q-Network para equilibrar la exploración y la explotación. La elección de la estrategia adecuada dependerá del problema específico y puede requerir ajustes y experimentación para encontrar el equilibrio óptimo entre exploración y explotación.

4.6 Gymnasium

A continuación, explicaremos la librería [5] con la que podemos acceder a diversos juegos clásicos a través de "Python" y crear algoritmos para que nuestro agente aprenda a jugar a un juego en concreto.

Esta librería está desarrollada por OpenAI, que es una de las compañías de investigación en Inteligencia Artificial más importantes a nivel mundial, y nos permite simular la interacción entre Agentes y Entornos.

GYMNASIUM contiene ya pre-instalados varios entornos útiles en problemas básico y en áreas como el control, la robótica y videojuegos. También permite crear entronos personalizados.

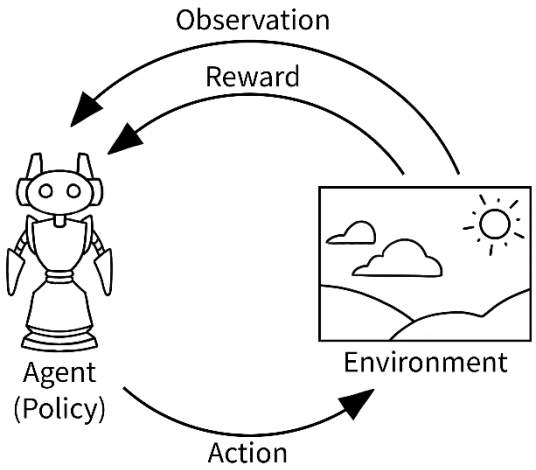

ILUSTRACIÓN 9 – BUCLE ENTRE AGENTE Y AMBIENTE

GYMNASIUM implementa el clásico bucle entre el agente y el ambiente, que viene representado en la Ilustración 9 – Bucle entre Agente y Ambiente.

Algunos de los métodos estándar son:

- Step: Este envía una acción al entorno y devuelve una tupla compuesta por:
  - Observación (ObsType): Este será un elemento del espacio de observación del entorno tras la ejecución de la acción.
  - Recompensa (Float): La cantidad de recompensa al realizar dicha acción.
  - Terminado (Bool): Si se ha alcanzado un estado terminal, es decir, ya no se puede avanzar más.
  - Truncado (Bool): Se cumple esta condición cuando ha pasado el límite de tiempo o cuando el agente se sale físicamente de los límites.
  - Info (Diccionario): Este contiene información útil para la depuración, el aprendizaje y el registro.
- Reset: Este método restablece el entorno a un estado inicial y devuelve la observación inicial y devuelve:
  - Observación (ObsType): Observación del estado inicial.

    - Información (diccionario): Este contiende información complementaria a la observación.
- Render: Este método calcula los fotogramas de representación según lo especificado por el atributo render_mode durante la inicialización del entorno.
    - None: Este el de por defecto y no se calcula ningún renderizado.
    - Humano (None): El entorno se representa continuamente en la pantalla o terminal actual. Normalmente para consumo humano.
    - Lista_rgb (numpy.ndarray): Devuelve un “frame” representando el estado actual del entorno. Un frame es un numpy.ndarray con forma (x,y,3) que representa valores RGB para una imagen de píxeles x por y.
    - Lista_rgb_array (array de numpy.ndarray): Devuelve una lista de “frames” que representan los estados del entrono desde el último reinicio.
    - Ansi (string): Contiene una representación de texto de estilo terminal cada paso de tiempo puede incluir saltos de línea y secuencias de forma ANSI, como por ejemplo para colores.
- Action_space: Este atributo da el formato de acciones válidas.
- Observation_space: Este atributo da el formato de las observaciones válidas.
- Reward_range: Este atributo es una tupla que corresponde a las posibles recompensas mínimas y máximas. El rango predeterminado se establece en (-inf,+inf). Se puede configurar si se desea un rango más estrecho.
- Close: Este método cerrará los entornos.
- Wrapper: Es una clase que permite una transformación modular de los métodos step() y reset().
- ObservationWrapper: Es una “superclass” de contenedores que pueden modificar las observaciones usando los métodos observation(), reset() y step().
- RewardWrapper: Es una “superclass” de wrappers que pueden modificar la recompensa que devuelta desde un step.
- ActionWrapper: Es una “superclass” que pueden modificar la acción antes de enviar una acción al entorno.

5. Desarrollo del proyecto

En este capítulo, se abordará la etapa crucial del desarrollo de nuestro TFG. En esta sección, se detallará el análisis exhaustivo del problema planteado y se presentará el diseño de la solución propuesta, haciendo hincapié en los algoritmos y técnicas de la inteligencia artificial que se utilizarán para lograr los objetivos establecidos y en cómo se han implementado.

5.1 Lenguajes y herramientas utilizadas

Para este trabajo se ha seleccionado como lenguaje de programación el lenguaje Python. Elegimos este por varias razones, entre ellas que permite empezar a trabajar muy rápidamente, no se necesitan grandes entornos de desarrollo y su gestión de paquetes es bastante rápida y sencilla de manejar, por lo que en poco tiempo podemos tener un entorno preparado y configurado para poder trabajar en cualquier ordenador.

Cabe añadir, que durante los últimos años Python ha sido la mejor opción para el aprendizaje automático. Esto se debe a la gran cantidad de recursos disponibles, ya sea en forma de frameworks, librerías de utilidades, documentación o la gran comunidad activa que hay dispuesta a resolver cualquier duda.

El entorno de desarrollo elegido para Python es Anaconda, el cual incluye multitud de librerías básicas para trabajar el aprendizaje automático.

Algunas de las bibliotecas que hemos utilizado son Numpy, Matplotlib, OS, OpenAI Gymnasium, TensorFlow, ImageIO y OpenCV:

- Numpy [6]: Es una biblioteca fundamental para el cálculo científico y también sirve para crear y manipular arrays multidimensionales eficientes en términos de rendimiento y memoria.
- Matplotlib [7]: Esta biblioteca permite crear una amplia variedad de gráficos, como de líneas o de barras, también ofrece un control completo sobre la apariencia en los gráficos, como modificar los colores, tipos de línea, títulos, etiquetas de ejes, leyendas y estilos de fuente. Por último, también dispone de un soporte para diferentes formatos de salidas para guardar los gráficos, como imágenes (PNG, JPEG, GIF), formatos vectoriales (PDF, SVG, EPS) y formatos interactivos (HTML).
- OS [8]: Esta biblioteca permite manipular archivos y directorios, es decir, proporciona las funciones de crear, eliminar, copiar, mover y renombrar archivos y directorios.
- Gymnasium [5] (basada en Open AI GYM): Es una librería diseñada para facilitar la experimentación y el desarrollo de algoritmos de aprendizaje por refuerzo. Contiene una amplia colección de entornos de prueba y una gran API para poder interactuar con ellos. En este trabajo nos centraremos especialmente en los juegos de Classic Control.
- Torch [9]: Torch, o también conocido como PyTorch, es una biblioteca de código abierto muy popular y diseñada específicamente para el desarrollo de aplicaciones de aprendizaje profundo, incluyendo redes neuronales artificales.
- ImageIO [10]: ImageIO es una biblioteca de Python que proporciona una interfaz fácil de usar para leer y escribir una variedad de formatos de imágenes y videos.
- OpenCV [11]: Esta biblioteca proporciona una amplia gama de herramientas y funciones para trabajar con imágenes y vídeos.

5.2 Experimento 1: Algoritmo con acciones aleatorias

En este apartado vamos a ilustrar la importancia del aprendizaje utilizando en primer lugar una aproximación naíf con un algoritmo bastante simple. En este algoritmo el agente, en un determinado número de episodios, tomará acciones aleatorias y se guardará las recompensas de cada episodio. Estas recompensas posteriormente serán representadas en un gráfico.

Con esta aproximación, apreciaremos la importancia de usar un algoritmo de aprendizaje para que un agente aprenda a jugar a un juego, en vez de realizar acciones aleatorias y sin tener en cuenta para nada las observaciones del entorno.

Con la librería Gymnasium podemos utilizar el método llamado action_space.sample(), que selecciona una acción aleatoria. En la sección 5.2.1 Pseudocódigo algoritmo con acciones aleatorias, se muestra el pseudocódigo del algoritmo que se ha implementado para esta aproximación.

5.2.1 Pseudocódigo algoritmo con acciones aleatorias

```
entorno = gym.make(NombreDelEntorno-VersiónDelEntorno ') #creamos el entorno

lista_recompensas = [] #creamos la lista para las recompensas de los episodios

for i in range (episodios):

        Inicializamos el estado

        Mientras no termine el episodio:

                Hacemos una acción aleatoria

        Una vez finalizado el episodio añadimos la recompensa a la lista creada

Cerramos el entorno

Generamos gráfica con las recompensas conseguidas
```

Finalmente, para generar la gráfica con las recompensas conseguidas, guardaremos la lista de recompensas obtenidas en un fichero de texto, que después recorreremos y, con “matplotlib”, crearemos la gráfica de la recompensa acumulada.

5.3 Experimento 2: Algoritmo Q-Learning

El aprendizaje por refuerzo es una rama del aprendizaje automático que se centra en desarrollar algoritmos capaces de tomar decisiones óptimas de entorno dinámicos y desconocidos. Un algoritmo de aprendizaje por refuerzo popular y ampliamente utilizado es el Q-Learning, que permite a un agente aprender a tomar decisiones en un entorno basado en recompensas.

Al finalizar este capítulo, se espera tener una comprensión clara de Q-Learning y cómo implementarlo en diversos escenarios de aprendizaje por refuerzo.

5.3.1 Estructura y flujo general del algoritmo Q-Learning

En este apartado, se va a presentar la estructura y flujo general del algoritmo Q-Learning. Este se puede describir en los siguientes pasos:

4. Creamos el entorno en el que el agente realizará el aprendizaje por refuerzo.
5. Definimos parámetros e hiperparámetros:
    - Tasa de aprendizaje (25psil)
    - Factor de descuento (gamma)
    - Exploración inicial (épsilon)
    - Decaimiento de la exploración (épsilon_decay)
    - Valor mínimo de exploración (épsilon_min)
6. Inicializar la tabla Q. Inicializamos la tabla Q vacía como un diccionario. Esta tabla se utilizará para almacenar los valores de Q para cada estado-acción
7. Definimos la función de política épsilon-greedy
8. Bucle de entrenamiento:
    - Este bucle de entrenamiento ejecutará el algoritmo de Q-Learning durante un número específico de episodios
    - Para cada episodio, se reiniciará el entorno y se inicializará el estado actual
    - En cada paso del bucle, se selecciona una acción utilizando la función épsilon-greedy
    - Después de tomar la acción se obtiene los valores del siguiente estado, la recompensa y se verifica si el episodio ha terminado
    - Se actualiza la tabla Q utilizando la ecuación de Q-Learning
    - Se actualiza el estado al siguiente estado
    - Se repite este proceso hasta que el episodio termine
9. Reducción de la exploración (épsilon): Después de cada episodio, se reduce la exploración (épsilon) utilizando el decaimiento de la exploración definido previamente.
10. Finalmente, se cierra el entorno

En definitiva, el flujo general del algoritmo Q-Learning implica la inicialización de los Q-Valores, la selección de acciones, la ejecución de las acciones en el entorno, la actualización de los Q-Valores y la iteración hasta que se cumpla el criterio de terminación establecido. Este proceso permite al agente aprender una política óptima para tomar decisiones en un entorno basado en recompensas.

5.3.2 Inicialización de los valores Q

La inicialización de los valores Q en el algoritmo Q-Learning es un paso importante para comenzar el proceso de aprendizaje del agente. Los valores Q representan las estimaciones iniciales de la recompensa acumulada esperada para cada par estado-acción.

Existen varias estrategias comunes para la inicialización de los valores Q en Q-Learning. A continuación, se presentan dos enfoques ampliamente utilizados:

1. Inicialización a cero:
    - En este enfoque, todos los valores Q se inicializan a cero al comienzo del proceso de aprendizaje. Esta estrategia supone que inicialmente no se tiene información sobre la calidad de las acciones en cada estado y, por lo tanto, se parte de una estimación neutral. A medida que el agente interactúa con el entorno y actualiza los valores Q en función de las recompensas recibidas, se irán ajustando y refinando.
2. Inicialización aleatoria:
    - En este enfoque, los valores Q se inicializan con valores aleatorios en un rango determinado. Esto permite introducir cierta variabilidad en las

estimaciones iniciales y explorar diferentes opciones desde el principio. Al elegir valores aleatorios, se evita el sesgo inicial hacia ciertas acciones o estados y se fomenta una mayor exploración del espacio de acciones y estados.

Si se tiene alguna información inicial sobre la calidad de las acciones, puede ser beneficioso utilizar una inicialización aleatoria para evitar el sesgo inicial. Sin embargo, si no se dispone de ninguna información, la inicialización a cero es una opción válida y neutral.

En nuestro caso utilizaremos la inicialización a cero, creando una tabla Q vacía.

5.3.3 Implementación de la actualización de los valores Q

La actualización de los valores Q en el algoritmo Q-Learning se realiza a medida que el agente interactúa con el entorno y recibe recompensas. Esta actualización se basa en la regla de actualización de Q-Learning, A continuación, se presenta la implementación que se ha realizado en este trabajo de la actualización de los valores Q en Q-learning:

1. Inicialización del episodio:
    - En cada episodio el entorno se reinicia y se obtiene el estado inicial.
2. Selección de acciones:
    - En el bucle interno el agente selecciona una acción utilizando la política épsilon-greedy.
3. Transición de estado:
    - El agente toma la acción seleccionada y obtiene el siguiente estado, la recompensa y si el episodio ha finalizado.
4. Calcula el nuevo valor Q para ese estado.
    - $Q(s, a) = (1 - \alpha) * Q(s, a) + \alpha * (r + \gamma * \max(Q(s', a')))$
    - Q(s, a) es el valor Q del estado-acción actual
    - α (26psil) es la tasa de aprendizaje
    - r es la recompensa obtenida al tomar la acción
    - γ (gamma) es el factor de descuento
    - max(Q(s', a')) es el valor Q máximo obtenido para todas las posibles acciones en el siguiente estado
5. Reducción de la exploración
    - La tasa de exploración (épsilon) se reduce con el tiempo para disminuir gradualmente la exploración aleatoria y permitir que el agente confíe más en sus valores Q aprendidos.

En definitiva, la actualización de los valores Q en Q-Learning se basa en la regla de actualización que utiliza la diferencia temporal entre el valor Q actual y el valor Q esperado. La actualización se realiza después de que el agente tome una acción y observe la recompensa y el siguiente estado. Esta implementación puede adaptarse según las necesidades y la complejidad del problema particular. En la 5.3.4 Pseudocódigo del algoritmo Q-Learning se muestra el pseudocódigo del algoritmo implementado en este trabajo.

5.3.4 Pseudocódigo del algoritmo Q-Learning

```
Inicializar la tabla Q como un diccionario vacío
Repetir hasta que se cumpla un criterio de terminación:
    Reiniciar el entorno
    Mientras no se haya cumplido el criterio de terminación:
        Selecciona una acción utilizando la política épsilon-greedy
        Toma la acción seleccionada y obtiene el siguiente estado, la recompensa y si se ha cumplido el criterio de terminación
        Calcula el nuevo valor Q con la ecuación de Q-Learning
        Actualiza la tabla Q
        Se actualiza el estado actual al siguiente estado
    Una vez se ha cumplido el criterio de terminación se reduce el valor de exploración
Generamos gráfica con las recompensas conseguidas
```

El algoritmo Q-Learning utiliza una tabla de Q-Valores para estimar la utilidad esperada de cada acción en cada estado. En cada iteración, se toma una acción, se observa la recompensa y el nuevo estado, y se actualiza el Q-Valor correspondiente utilizando la fórmula de actualización Q-Learning. El proceso continúa hasta que se cumpla un criterio de terminación, como alcanzar un número máximo de iteraciones.

5.4 Experimento 3: Deep Q-Network

En el ámbito del aprendizaje por refuerzo, el algoritmo Deep Q-Network ha surgido como una técnica poderosa y versátil para permitir que los agentes aprendan a tomar decisiones óptimas en entornos complejos. Este apartado se centra en explorar en profundidad el concepto y la implementación del algoritmo Deep Q-Network.

El Deep Q-Network combina dos enfoques fundamentales: el algoritmo de Q-Network, que se basa en una tabla de valores para tomar decisiones, y las redes neuronales profundas, que tienen la capacidad de aprender características y patrones complejos. Al combinar estas dos técnicas, el Deep Q-Network permite a los agentes aprender automáticamente a través de la experiencia, sin necesidad de un conocimiento previo sobre el entorno.

A continuación, nos adentraremos en el diseño del algoritmo Deep Q-Network. Se explorarán aspectos fundamentales como la representación del estado y las acciones, la estructura de la red neuronal utilizada y los mecanismos de entrenamiento y la actualización de los valores Q.

Una vez comprendido el diseño del algoritmo, nos centraremos en su implementación práctica. Se discutirán consideraciones prácticas, como la selección de hiperparámetros y técnicas de regularización, para lograr un modelo de Deep Q-Network robusto y eficiente.

Con el algoritmo implementado, se llevarán a cabo experimentos para evaluar su desempeño.

5.4.1 Proceso de entrenamiento y actualización de la Deep Q-Network versión 1

El proceso de entrenamiento y actualización de los Q-Valores en el Deep Q-Network se basa en la iteración de los siguientes pasos:

1. Inicialización de la red neuronal: Se establece la arquitectura de la red neuronal, que será utilizada para estimar los Q-Valores para cada estado y acción. Esta red tiene tres capas completamente contectadas con activación ReLU en las dos primeras capas y una capa de salida.
2. Definimos el agente DQN:
    - Se inicializan los hiperparámetros:
        - Tasa de aprendizaje
        - Factor de descuento
        - Épsilon
        - Factor de decaimiento de épsilon
    - Se define la función de perdida.
3. Inicializamos el entorno el buffer de experiencias que se utilizará para almacenar transiciones experiencia-recompensa.
4. Entrenamiento del agente, ejecutado durante un número específico de episodios:
    - Se reinicia el entorno y se obtiene el estado inicial
    - Se selecciona la acción con la política épsilon-greedy
    - Ejecutamos la acción y obtenemos el siguiente estado, la recompensa y si ha cumplido el criterio de terminación
    - Almacenamos la transición (estado actual, acción, recompensa, siguiente estado y criterio de terminación) se almacena en el buffer de experiencias
    - Entrenamiento del agente:
        - El agente realiza un paso de entrenamiento utilizando una muestra aleatoria del buffer de experiencias. Calcula el valor Q predicho y se ajusta utilizando la función de pérdida (MSELoss) y el optimizador (Adam)
    - Al final de cada episodio se reduce el valor de épsilon para disminuir la exploración y favorecer la explotación de la política aprendida.
5. Cerramos el entorno.

Este proceso de entrenamiento y actualización de los pesos de la red se repite iterativamente para mejorar la estimación de los pesos de la red neuronal y permitir que el agente aprenda una política óptima en el entorno.

5.4.2 Pseudocódigo algoritmo Deep Q-Network versión 1

Definimos los hiperparámetros

Creamos el entorno, la red neuronal, el agente, y el buffer de experiencias vacío.

Bucle de entrenamiento:

    Reiniciamos el entorno y obtenemos el estado inicial

    Mientras no se cumpla el criterio de terminación:

        Seleccionamos la acción mediante política épsilon-greedy

        Ejecutamos la acción y obtenemos el siguiente estado, la recompensa y si cumple el criterio de terminación

        Añadimos esto al buffer de experiencias

        Calculamos Q-Valor predicho y ajustamos utilizando la función de perdida MSE

        Actualizamos los pesos del modelo

    Reducimos el valor de épsilon

Cerramos el entorno

Generamos gráfica con las recompensas conseguidas

5.4.3 Proceso de entrenamiento y actualización de la Deep Q-Network versión 2

El proceso de entrenamiento y actualización de los Q-Valores en el Deep Q-Network se basa en la iteración de los siguientes pasos:

1. Inicialización de la red neuronal: Se establece la arquitectura de la red neuronal, que será utilizada para estimar los Q-Valores para cada estado y acción. Esta red tiene cuatro capas completamente contectadas con activación ReLU en las tres primeras capas y una capa de salida.
2. Definimos el agente DQN:
    - Se inicializan los hiperparámetros:
        - Tasa de aprendizaje
        - Factor de descuento
        - Épsilon
        - Factor de decaimiento de épsilon
    - Se define la función de perdida.
3. Inicializamos el entorno el buffer de experiencias que se utilizará para almacenar transiciones experiencia-recompensa.
4. Entrenamiento del agente, ejecutado durante un número específico de episodios:
    - Se reinicia el entorno y se obtiene el estado inicial
    - Se define un número de episodios que hasta que este no se ha superado, el agente utilizará una exploración guiada. Esto ayuda a acumular experiencias iniciales para el buffer de experiencias

- Una vez superado el número de episodios predefinido para la exploración guiada, se selecciona la acción con la política épsilon-greedy
- Ejecutamos la acción y obtenemos el siguiente estado, la recompensa y si ha cumplido el criterio de terminación
- Almacenamos la transición (estado actual, acción, recompensa, siguiente estado y criterio de terminación) se almacena en el buffer de experiencias
- Entrenamiento del agente:
    - El agente realiza un paso de entrenamiento utilizando una muestra aleatoria del buffer de experiencias. Calcula el valor Q predicho y se ajusta utilizando la función de pérdida (MSELoss) y el optimizador (Adam)
- Al final de cada episodio se reduce el valor de épsilon para disminuir la exploración y favorecer la explotación de la política aprendida.

5. Cerramos el entorno.

5.4.4 Pseudocódigo algoritmo Deep Q-Network versión 2

Definimos los hiperparámetros

Creamos el entorno, la red neuronal, el agente, y el buffer de experiencias vacío.

Bucle de entrenamiento:

    Reiniciamos el entorno y obtenemos el estado inicial

    Durante un número definido de episodios el agente utilizará una exploración guiada

    Una vez superado el número de episodios para la exploración guiada seguirá el siguiente algoritmo

    Mientras no se cumpla el criterio de terminación:

        Seleccionamos la acción mediante política épsilon-greedy

        Ejecutamos la acción y obtenemos el siguiente estado, la recompensa y si cumple el criterio de terminación

        Añadimos esto al buffer de experiencias

        Calculamos Q-Valor predicho y ajustamos utilizando la función de perdida MSE

        Actualizamos los pesos del modelo

    Reducimos el valor de épsilon

Cerramos el entorno

Generamos gráfica con las recompensas conseguidas

Esta variante del algoritmo Deep Q-Network guarda similitudes con la mencionada previamente. No obstante, se distingue por incorporar una fase inicial de exploración guiada. Esta elección se basa en la consideración de que, en el contexto del primer algoritmo, en uno de los escenarios que se ha utilizado para entrenar el agente, este no logró alcanzar los resultados esperados, lo que motivó la búsqueda de un enfoque alternativo, como la exploración guiada.

En este capítulo se ha presentado el desarrollo realizado en este trabajo, el cual se ha centrado en tres implementaciones (desde una naíf, completamente aleatoria, hasta una basada en redes neuronales) de algoritmos de aprendizaje por refuerzo orientados a videojuegos. Además de describir los algoritmos, se han implementado para cada uno de los entornos que se van a utilizar en los experimentos del siguiente capítulo. Para cada uno de los experimentos se han realizado pequeños ajustes que preparan el modelo a las características de cada uno de los mismos.

6. Experimentos

El capítulo de experimentos es una etapa crucial en la evaluación y validación del modelo propuesto. En este capítulo, se llevan a cabo una serie de experimentos con el objetivo de probar la efectividad y el rendimiento del sistema desarrollado en el contexto de los videojuegos.

La finalidad en esta sección es proporcionar una evaluación objetiva y rigurosa de la solución propuesta, analizando su desempeño en diferentes situaciones y escenarios de juego. A través de los experimentos, se busca responder respuestas como: ¿cuál es la tasa de éxito en la consecución de los objetivos del juego?

Durante la ejecución de los experimentos, se registrarán y analizarán los resultados obtenidos, incluyendo mediciones de rendimiento del aprendizaje y análisis de posibles limitaciones o problemas encontrados. Estos resultados se presentarán de manera clara y concisa, utilizando gráficas, tablas y estadísticas relevantes que permitan una interpretación adecuada.

6.1 ESCENARIOS

En este apartado, presentaremos los escenarios en los que se han realizado las pruebas en el contexto de nuestro proyecto de inteligencia artificial aplicada a los videojuegos. Los escenarios de prueba desempeñan un papel fundamental en la evaluación del rendimiento y la eficacia de nuestro sistema, ya que nos permite analizar cómo se comporta el agente en entornos específicos y cómo se adaptan a nuestras soluciones a diferentes desafíos. Los primeros entornos que se presentarán no son juegos sino entornos cinemáticos más simples que nos permitirán explorar la eficacia de los algoritmos planteados para finalmente utilizar un entorno que sí corresponde con un conocido videojuego donde se ha aplicado el trabajo desarrollado a lo largo de este proyecto.

6.1.1 Escenario 1: Cartpole

El primer escenario donde realizaremos pruebas de los modelos presentados se trata del juego Cartpole. [12]

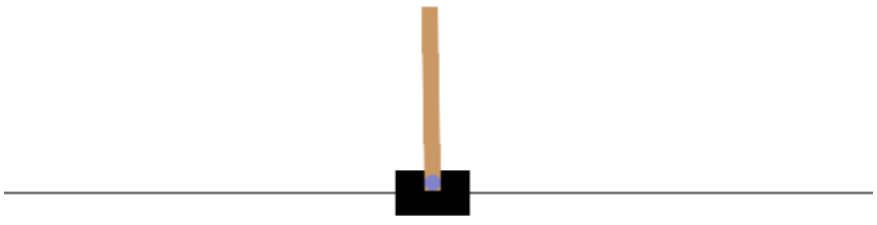

ILUSTRACIÓN 10 – CARTPOLE

En la Ilustración 10 – Cartpole vemos una foto representativa del juego Cartpole.

Cartpole es uno de los problemas más conocidos de la Inteligencia Artificial. Fue introducido por primera vez por Andrew G. Barto.

La descripción del juego es la siguiente:

Un poste está unido por una articulación no accionada a un carro, que se mueve a lo largo de una pista sin fricción. El péndulo se coloca en posición vertical sobre el carro y el objetivo es equilibrar el poste aplicando fuerzas en la dirección izquierda y derecha del carro.

TABLA 2 – OBSERVACIÓN DEL ENTORNO CARTPOLE

Num	Observación	Min	Max
0	Posición del carro	-4.8	4.8
1	Velocidad del carro	-Inf	Inf
2	Ángulo del poste	-24 º	24 º
3	Velocidad del poste	-Inf	Inf

En la Tabla 2 – Observación del entorno Cartpole encontramos una representación estructurada de las diferentes observaciones o estados que se pueden obtener del entorno del juego Cartpole.

6.1.1.1 Acciones

El apartado de acciones del agente en un escenario de inteligencia artificial es fundamental para comprender como el agente interactúa y toma decisiones en su entorno. En este apartado, se mencionarán las diferentes acciones posibles.

Estas acciones son:

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

6.1.1.2 Recompensa

El apartado de recompensa en un contexto de inteligencia artificial juega un papel fundamental en el aprendizaje y el desempeño del agente. La recompensa es una señal que el agente recibe del entorno como retroalimentación por sus acciones y decisiones tomadas.

El objetivo del agente en este escenario es mantener el poste en posición vertical el mayor tiempo posible, se otorga una recompensa de “+1” por cada paso dado y el umbral de recompensa es 500 para la versión 1 del cartpole.

6.1.1.3 Estado inicial

El estado inicial proporciona la configuración inicial del entorno y define las condiciones iniciales en las que el agente se encuentra al comenzar su tarea. Este estado puede incluir información relevante sobre la posición, la velocidad, las restricciones o cualquier otro factor que afecte la toma de decisiones del agente.

En el caso del escenario cartpole, a todas las observaciones se les asigna un valor aleatorio uniforme entre el rango de “-0.05” y “0.05”.

6.1.1.4 Estado final

El estado final del agente en un escenario de inteligencia artificial marca el término de su interacción con el entorno. Una definición clara y adecuada del estado final permite determinar si el agente ha alcanzado con éxito su objetivo y proporciona una medida objetiva de su eficacia. Además, el estado final puede utilizarse para proporcionar retroalimentación al agente, ayudándole a aprender y mejorar su comportamiento en futuras interacciones.

El estado final en el escenario cartpole es alcanzado en las siguientes situaciones:

- El ángulo de la varilla es mayor que [-12º, 12º]
- La posición del carro (el centro del carro llega al borde la pantalla) es superior a [-2.4, 2.4]
- La duración del episodio es mayor que 500 para la versión 1 del cartpole y superior a 200 para la versión 0.

6.1.2 Acrobot

El segundo escenario donde realizaremos pruebas de los modelos presentados se trata del juego Acrobot. [13]

ILUSTRACIÓN 11 – ACROBOT

En la Ilustración 11 – Acrobot vemos una foto representativa del juego Cartpole.

La descripción del juego es la siguiente:

En Acrobot, el jugador controla un sistema de doble péndulo con el objetivo de hacer que este se balancee hacia arriba y alcance una posición vertical estable. El juego presenta desafíos en términos de control y coordinación, ya que el jugador debe aprender a aplicar las fuerzas adecuadas en los momentos precisos para lograr el equilibrio deseado.

TABLA 3 – OBSERVACIÓN DEL ENTORNO ACROBOT

Num	Observación	Min	Max
0	Coseno del theta1	-1	1
1	Seno del theta1	-1	1
2	Coseno del theta2	-1	1
3	Seno del theta2	-1	1
4	Velocidad angular del theta1	~ -12.567 (-4 * pi)	~ 12.567 (4 * pi)
5	Velocidad angular del theta2	~ -28.274 (-9 * pi)	~ 28.274 (9 * pi)

Donde:

- Theta1: Es el ángulo de la primera articulación, donde un ángulo de 0 indica que el primer eslabón apunta directamente hacia abajo.
- Theta2: Es relativo al ángulo del primer enlace del primer enlace. Un ángulo de 0 corresponde a tener el mismo ángulo entre los dos enlaces.
- Las velocidades angulares de theta1 y theta3 están limitadas a ±4π y ±9π rad/s respectivamente.

En la Tabla 3 – Observación del entorno Acrobot encontramos una representación estructurada de las diferentes observaciones o estados que se pueden obtener del entorno del juego Cartpole.

6.1.2.1 Acciones

En este apartado, se explorará en detalle las diferentes acciones disponibles para los jugadores.

Hay 3 acciones determinadas discretas y representa el par aplicado en la unión accionada entre los dos eslabones, son las siguientes:

- Aplicar torque -1 a la junta accionada.
- Aplicar torque 0 a la junta accionada.
- Aplicar torque 1 a la junta accionada.

Hay que destacar que la unidad torque se mide en Newton por metro (N * m) en el sistema internacional de las unidades. El torque es una medida de la tendencia de una fuerza para girar un objeto alrededor de un punto o un eje.

6.1.2.2 Recompensa

El objetivo es que el extremo libre alcance una altura objetivo-designada en la menor cantidad de pasos posibles y como tal, todos los pasos que no alcancen la meta generan una recompensa de -1.

6.1.2.3 Estado inicial

Cada parámetro (theta1, theta2 y las dos velocidades angulares) se inicializa uniformemente entre -0,1 y 0,1. Esto significa que ambos enlaces apuntan hacia abajo con cierta estocasticidad inicial.

6.1.2.4 Estado final

El episodio termina si ocurre una de las siguientes situaciones:

- Terminación: El extremo libre alcanza la altura objetivo, que se construye como: -cos(theta1) – cos(theta2 + theta1) > 1,0
- Truncamiento: La duración del episodio es superior a 500 para la v1 y 200 pasa la v0.

6.1.3 Escenario 3: Mountain Car

El tercer escenario donde realizaremos pruebas de los modelos presentados anteriormente se trata del juego Mountain Car. Este es también un entorno cinemático sencillo que nos ha resultado muy útil para probar los algoritmos desarrollados en este proyecto. [14]

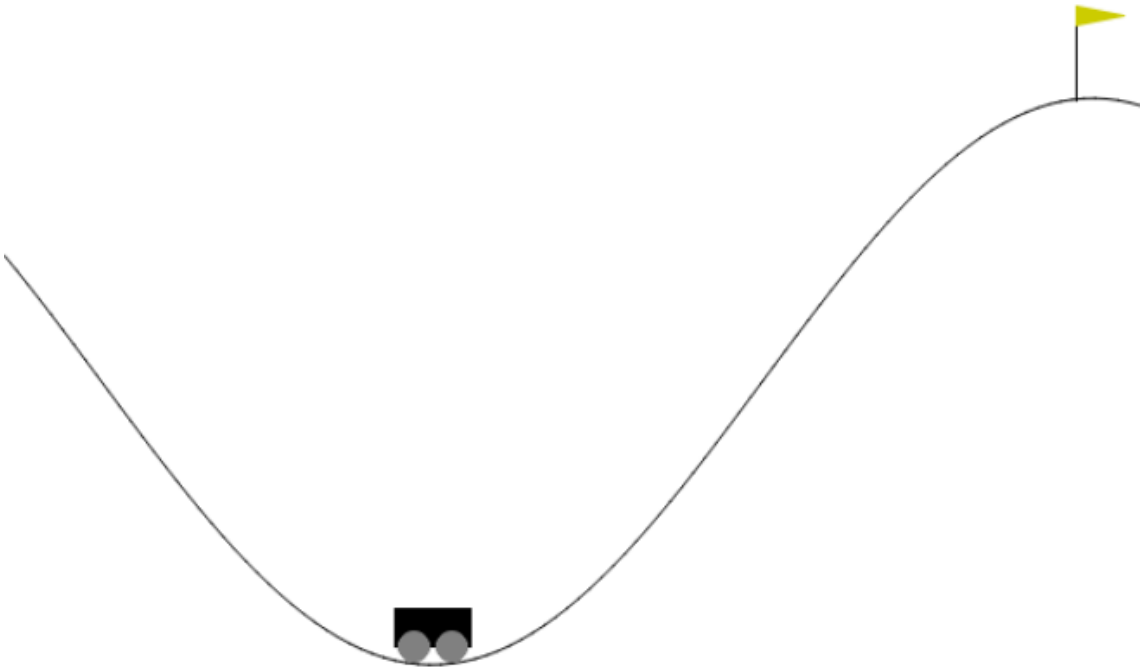

ILUSTRACIÓN 12 – MOUNTAIN CAR

En la Ilustración 12 – Mountain Car vemos una imagen representativa del juego Mountain Car.

Mountain Car es otro de los problemas más conocidos de la Inteligencia Artificial.

Una descripción del juego es la siguiente:

El Mountain Car consiste en un automóvil colocado en el fondo de un valle sinusoidal, siendo las únicas acciones posibles las aceleraciones que se pueden aplicar al automóvil en cualquier dirección. El objetivo es acelerar estratégicamente el automóvil para alcanzar el estado objetivo en la cima de la colina derecha. Hay dos versiones del Mountain Car: una con acciones discretas y otra con acciones continuas. La versión que se utilizará está basada en acciones discretas.

TABLA 4 – OBSERVACIÓN DEL ENTORNO MOUNTAIN CAR

Num	Observación	Min	Max	Unidad
0	Posición del carro a lo largo del eje x	-1.2	0.6	Posición (m)
1	Velocidad del coche	-0.07	0.07	Velocidad (v)

En la Tabla 4 – Observación del entorno Mountain Car encontramos una representación estructurada de las diferentes observaciones o estados que se pueden obtener del entorno del juego Mountain Car.

6.1.3.1 Acciones

En este apartado, se explorará en detalle las diferentes acciones disponibles para los jugadores.

Hay 3 acciones determinadas discretas, son las siguientes:

- Acelerar a la izquierda.
- No acelerar.
- Acelerar a la derecha.

6.1.3.2 Dinámica de transición

La dinámica de transición en un juego es esencial para comprender como evoluciona la jugabilidad y las situaciones a medida que los jugadores interactúan y toman decisiones.

Dada una acción, el coche de Mountain Car sigue la siguiente dinámica:

velocidad t+1 = velocidad t + (acción – 1) * fuerza – cos(3 * posición t) * gravedad

posición t+1 = posición t + velocidad t+1

Donde fuerza = 0.001 y gravedad = 0.0025.

Las colisiones en cualquier extremo son inelásticas y la velocidad se establece en 0 al chocar con la pared.

La posición se recorta al rango [-1.2, 0.6] y la velocidad se recorta al rango [-0.07, 0.07].

6.1.3.3 Recompensa

La recompensa es un mecanismo utilizado para proporcionar retroalimentación al agente en función de sus acciones y decisiones. Se utiliza para establecer los objetivos y los criterios de éxito del agente, así como para incentivar el aprendizaje y la mejora continua.

El objetivo es llegar lo más rápido posible a la bandera situada en lo alto de la colina derecha, por lo que el agente es penalizado con una recompensa de -1 por cada intervalo de tiempo.

La recompensa es de -200 puntos si el coche no llega la meta al finalizar la partida.

6.1.3.4 Estado inicial

El apartado de estado inicial en un juego de inteligencia artificial establece el contexto y las condiciones iniciales en las que el agente comienza a interactuar.

En el Mountain Car, la posición del automóvil se le asigna un valor aleatorio uniforme entre [-0.6, -0.4]. Y la velocidad inicial del automóvil siempre se asigna a 0.

6.1.3.5 Estado final

El apartado de estado final en un juego de inteligencia artificial marca el resultado concluyente de una partida y tiene un impacto significativo en la evaluación del desempeño del agente. Este proporciona una descripción detallada del estado final, sus criterios y consecuencias, así como la evaluación y la recompensa asociada.

Un episodio finaliza en el juego Mountain Car si se cumple una de las siguientes condiciones:

- Terminación: La posición del automóvil es mayor o igual a 0.5 (la posición de destino en la parte superior de la colina derecha).
- Truncamiento: La duración del episodio es de 200.

6.2 Resultados escenario 1: Cartpole

En este apartado, presentaremos los resultados obtenidos de los experimentos realizados con los algoritmos mencionados anteriormente en el escenario Cartpole.

6.2.1 Resultados Cartpole acciones aleatorias

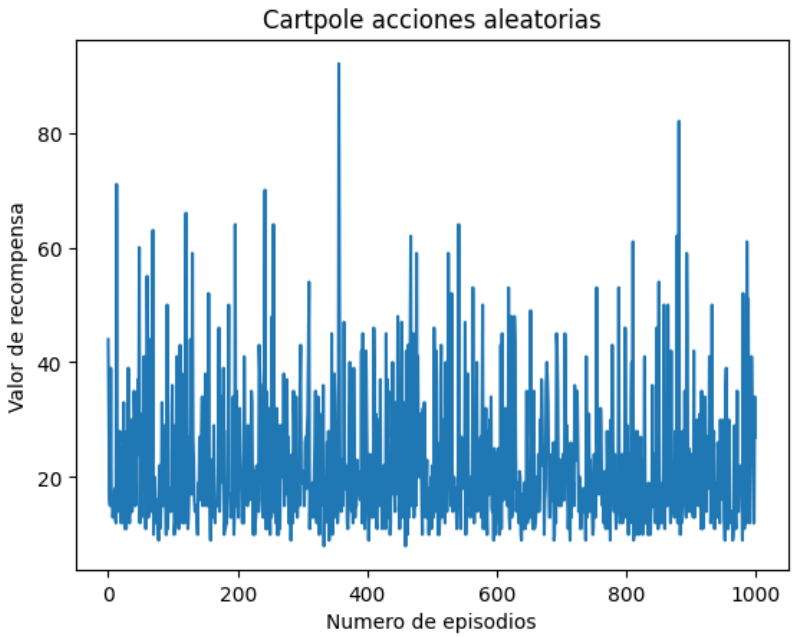

ILUSTRACIÓN 13 – GRÁFICA CARTPOLE ACCIONES ALEATORIAS

En la Ilustración 13 – Gráfica Cartpole Acciones Aleatorias se presentan los resultados obtenidos durante 1000 episodios del agente en el escenario de Cartpole, utilizando una estrategia de acciones aleatorias. Los valores registrados de recompensa reflejan un rendimiento subóptimo, con una puntuación de 02 alcanzada en un episodio.

Es importante resaltar que, al analizar la gráfica, se observa que los resultados obtenidos con las acciones aleatorias no han sido favorables para alcanzar puntuaciones altas. El agente no ha logrado desarrollar una estrategia efectiva para mantener el equilibrio del poste en el escenario de Cartpole.

Estos resultados son coherentes con la naturaleza aleatoria de la estrategia utilizada, en la cual el agente toma decisiones sin tener en cuenta la información contextual o el aprendizaje previo.

6.2.2 Resultados Cartpole Q-Learning

Hiperparámetros

epsilon = 0.1 # Tasa de aprendizaje

gamma = 0.99 # Factor de descuento

epsilon = 1.0 # Exploración inicial

epsilon_decay = 0.999 # Decaimiento de la exploración

epsilon_min = 0.01 # Exploración mínima

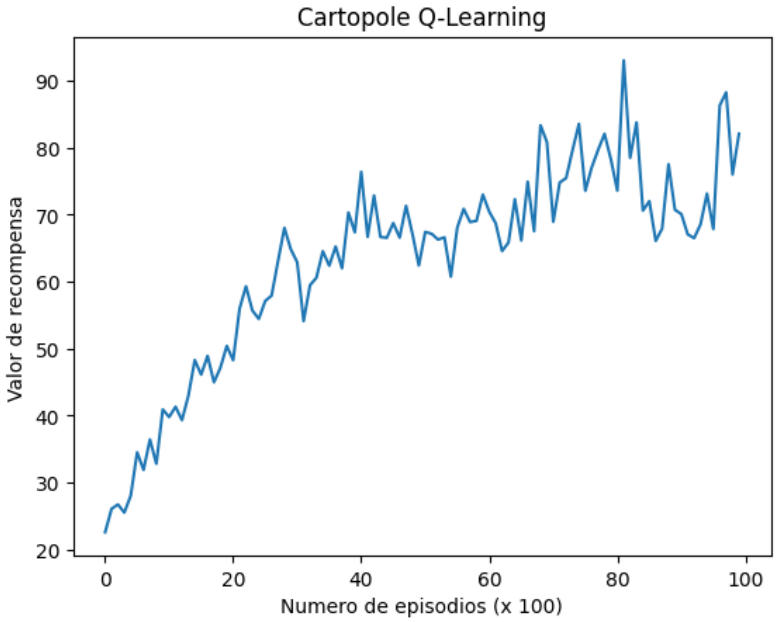

ILUSTRACIÓN 14 – GRÁFICA CARTPOLE Q-LEARNING

En la Ilustración 14 – Gráfica Cartpole Q-Learning, se muestra el rendimiento del algoritmo a lo largo de 10000 episodios. Los resultados se han promediado cada 100 recompensas. Observamos que la recompensa más alta alcanzada es de 93. Si bien esta gráfica refleja una mejora gradual a lo largo del tiempo, es importante destacar que el rendimiento aún no ha alcanzado su máximo potencial, dado que la recompensa máxima posible a obtener en este entorno es de 500.

Estos resultados son coherentes, considerando que se trata de un algoritmo de aprendizaje por refuerzo de naturaleza sencilla.

6.2.3 Resultados Cartpole DQN

Hiperparámetros

lr = 0.001 # Tasa de aprendizaje

gamma = 0.99 # Factor de descuento

epsilon = 1.0 # Valor de epsilon inicial

epsilon_decay = 0.995 # Factor de decaimiento de epsilon

replay_buffer_size = 10000 # Tamaño del buffer de experiencias

batch_size = 64 # Tamaño del lote para el entrenamiento

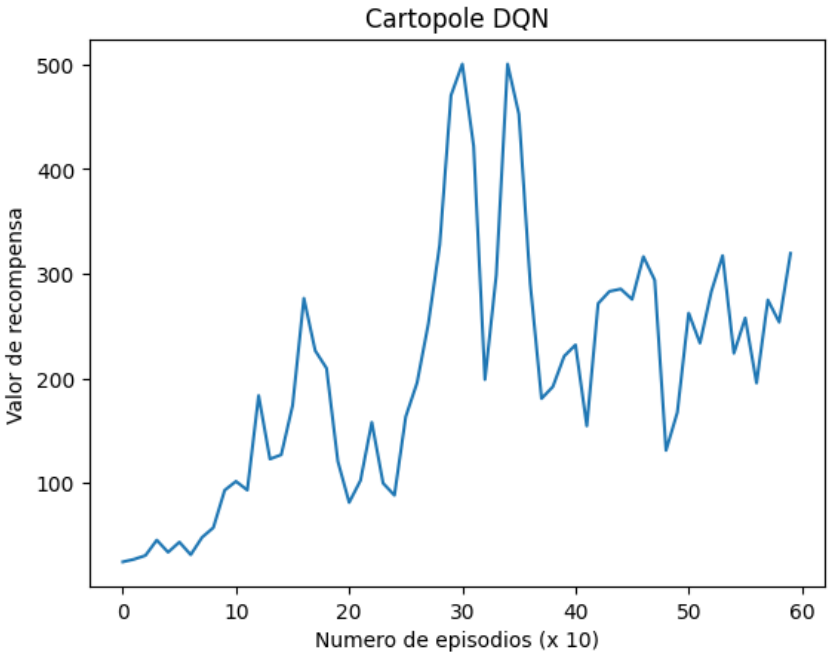

ILUSTRACIÓN 15 – GRÁFICA CARTPOLE DQN

En la Ilustración 15 – Gráfica Cartpole DQN, se presentan los resultados a lo largo de 600 episodios, con promedios calculados cada 10 episodios. Observamos que en un par de instancias dentro de estos 10 episodios, el agente logró alcanzar la recompensa máxima posible, que es 500. No obstante, en la mayoría de la gráfica, el promedio de la recompensa es estabiliza en torno a un valor cercano a 300.

6.2.4 Conclusiones resultados escenario 1: Cartpole

Al analizar los resultados del entrenamiento de un agente en el escenario Cartpole utilizando los tres algoritmos seleccionados, se pueden extraer las siguientes conclusiones:

- Acciones aleatorias: El algoritmo de acciones aleatorias no logró obtener una alta recompensa en el escenario. Su enforque basado en decisiones aleatorias no permitió al agente desarrollar estrategias efectivas para el equilibrio del poste.
- Q-Learning: El algoritmo de Q-Learning demostró una mejora significativa en comparación con las acciones aleatorias. El agente pudo aprender a tomar decisiones más informadas y alcanzar mejores resultados en términos de recompensas.
- DQN: Sin embargo, el algoritmo DQN se destacó como el más exitoso de los tres. En un menor número de episodios, logró adaptarse de manera más efectiva al escenario de Cartpole y obtener las mejores recompensas.

Estas conclusiones sugieren que la utilización de métodos más avanzados, pueden conducir a un aprendizaje más rápido y eficiente en entornos complejos de aprendizaje por refuerzo, como el escenario de Cartpole.

6.3 Resultados escenario 2: Acrobot

En este apartado, presentaremos los resultados obtenidos de los experimentos realizados con los algoritmos mencionados anteriormente en el escenario Acrobot.

6.3.1 Resultados Acrobot acciones aleatorias

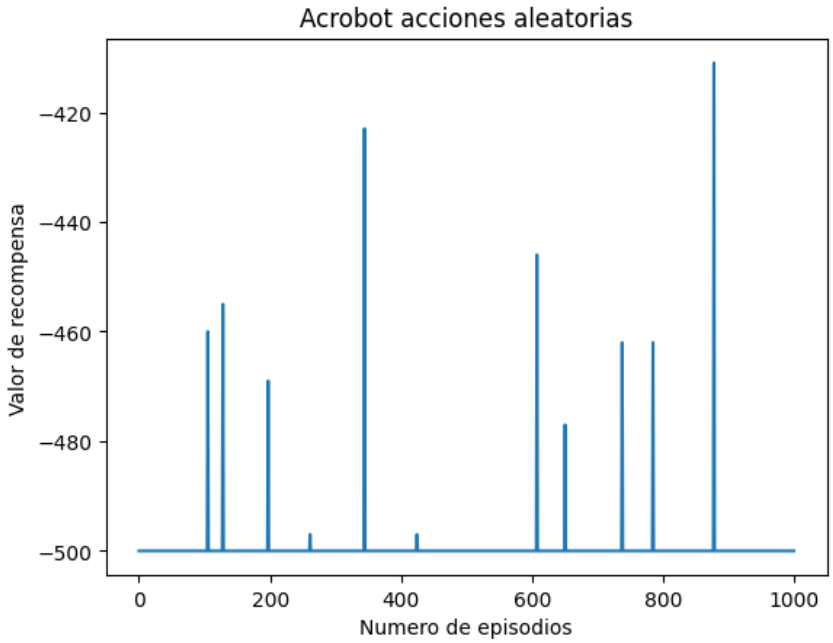

ILUSTRACIÓN 16 - GRÁFICA ACROBOT ACCIONES ALEATORIAS

En la Ilustración 16 - Gráfica Acrobot Acciones Aleatorias, se muestra el rendimiento de un agente en el escenario Acrobot a lo largo de 1000 episodios, en los cuales el agente tomó decisiones completamente al azar. En la mayoría de estos episodios, la recompensa obtenida fue de alrededor de -500, lo que indica que el agente no logró alcanzar el objetivo deseado. Sin embargo, en algunos pocos episodios el agente tuvo éxito en cumplir su objetivo, y en el mejor caso obtuvo una recompensa de -410. Esto demuestra que, al depender únicamente de acciones aleatorias, el agente tiene dificultades para lograr un rendimiento consistente y efectivo.

6.3.2 Resultados Acrobot Q-Learning

Hiperparámetros

epsilon = 0.1 # Tasa de aprendizaje

gamma = 0.99 # Factor de descuento

epsilon = 1.0 # Exploración inicial

epsilon_decay = 0.999 # Decaimiento de la exploración

epsilon_min = 0.01 # Exploración mínima

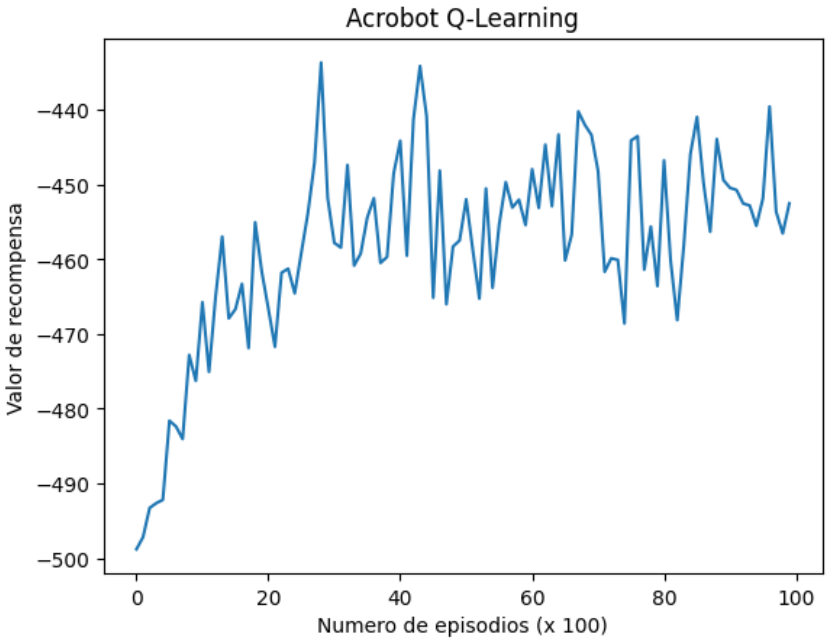

ILUSTRACIÓN 17 - GRÁFICA ACROBOT Q-LEARNING

En la Ilustración 17 - Gráfica Acrobot Q-Learning, representa el resultado de un proceso de entrenamiento que constó de 10000 episodios, y se ha generado una gráfica con un promedio cada 100 episodios. Aunque se observa una mejora gradual en la pendiente de la gráfica, la recompensa máxima alcanzada fue de -452. Esto indica que, si bien el agente ha experimentado un progreso, no ha logrado un aprendizaje óptimo y el algoritmo Q-Learning no se ha mostrado altamente efectivo en este escenario.

6.3.3 Resultados Acrobot DQN

Hiperparámetros

lr = 0.001 # Tasa de aprendizaje

gamma = 0.99 # Factor de descuento

epsilon = 1.0 # Valor de epsilon inicial

epsilon_decay = 0.995 # Factor de decaimiento de epsilon

replay_buffer_size = 10000 # Tamaño del buffer de experiencias

batch_size = 64 # Tamaño del lote para el entrenamiento

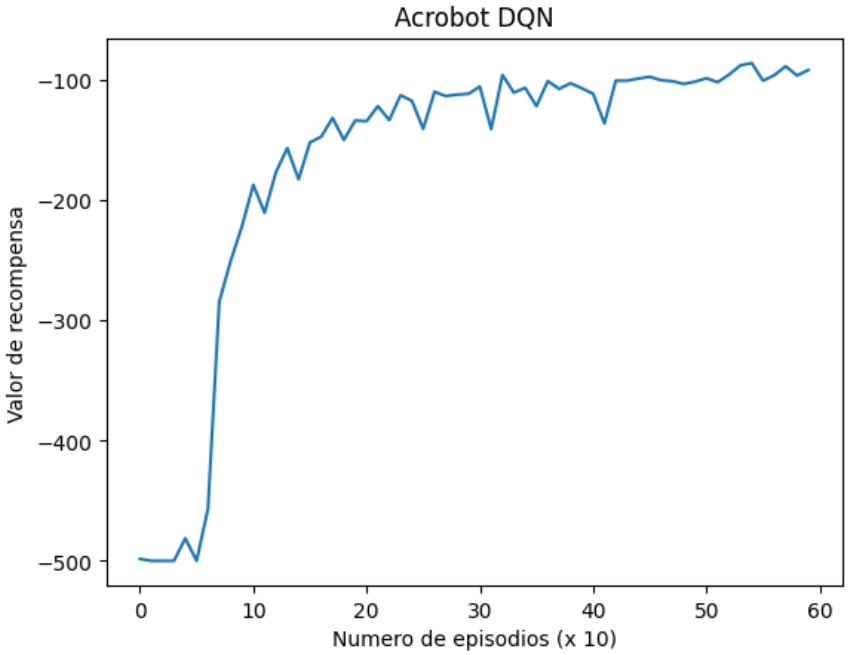

ILUSTRACIÓN 18 - GRÁFICA ACROBOT DQN

En la Ilustración 18 - Gráfica Acrobot DQN, se presentan los resultados de 600 episodios con un promedio calculado cada 10 episodios. La gráfica refleja un aprendizaje notablemente efectivo, ya que el agente alcanza una recompensa de -86.5. Estos resultados sugieren que el algoritmo DQN ha demostrado ser altamente efectivo en este escenario.

6.3.4 Conclusiones resultados escenario 2: Acrobot

Tras analizar los resultados del entrenamiento de un agente en el escenario Acrobot utilizando los tres algoritmos seleccionados, podemos derivar las siguientes conclusiones:

- Acciones aleatorias: El algoritmo de acciones aleatorias no logró obtener una alta recompensa en el escenario. En la mayoría de los episodios, el agente obtuvo la peor recompensa posible, que es -500.
- Q-Learning: El algoritmo de Q-Learning demostró una mejora significativa en comparación con las acciones aleatorias, ya que en muy pocos episodios obtuvo la peor recompensa, pero su aprendizaje fue bastante limitado, alcanzando una mejor recompensa de -452, que sigue siendo bastante baja.
- DQN: Por otro lado, el algoritmo DQN se destacó como el más exitoso de los tres. En un número mucho menor de episodios, logró que el agente aprendiera de manera muy efectiva a moverse en el escenario y, como resultado, obtuvo recompensas considerablemente más altas.

Estas conclusiones sugieren que la implementación de métodos más avanzados puede conducir a un aprendizaje más rápido y eficiente en entornos de aprendizaje por refuerzo complejos, como el escenario Acrobot.

6.4 Resultados escenario 3: Mountain Car

En este apartado, presentaremos los resultados obtenidos de los experimentos realizados con los algoritmos mencionados anteriormente en el escenario Mountain Car.

6.4.1 Resultados Mountain Car acciones aleatorias

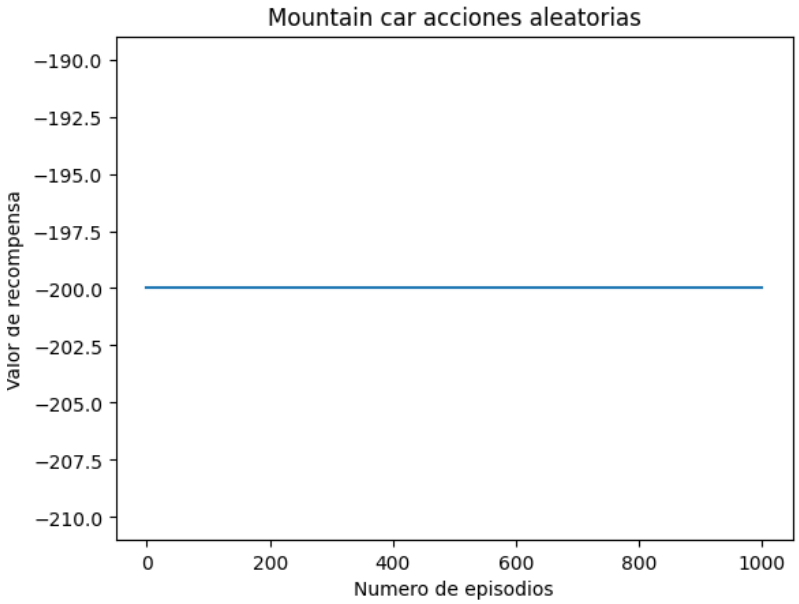

ILUSTRACIÓN 19 - GRÁFICA MOUNTAIN CAR ACCIONES ALEATORIAS

En la Ilustración 19 - Gráfica Mountain Car Acciones Aleatorias, se presentan los resultados obtenidos tras realizar 1000 episodios en el escenario de Mountain Car utilizando una estrategia de acciones aleatorias. Se observa que la recompensa se ha mantenido constante en -200 en todos los episodios, lo que indica que el agente no ha logrado alcanzar el objetivo en ningún momento.

Estos resultados revelan un rendimiento deficiente por parte del agente al utilizar acciones aleatorias en el entorno de Mountain Car.

6.4.2 Resultados Mountain Car Q-Learning

Hiperparámetros

epsilon = 0.1 # Tasa de aprendizaje

gamma = 0.99 # Factor de descuento

epsilon = 1.0 # Exploración inicial

epsilon_decay = 0.999 # Decaimiento de la exploración

epsilon_min = 0.01 # Exploración mínima

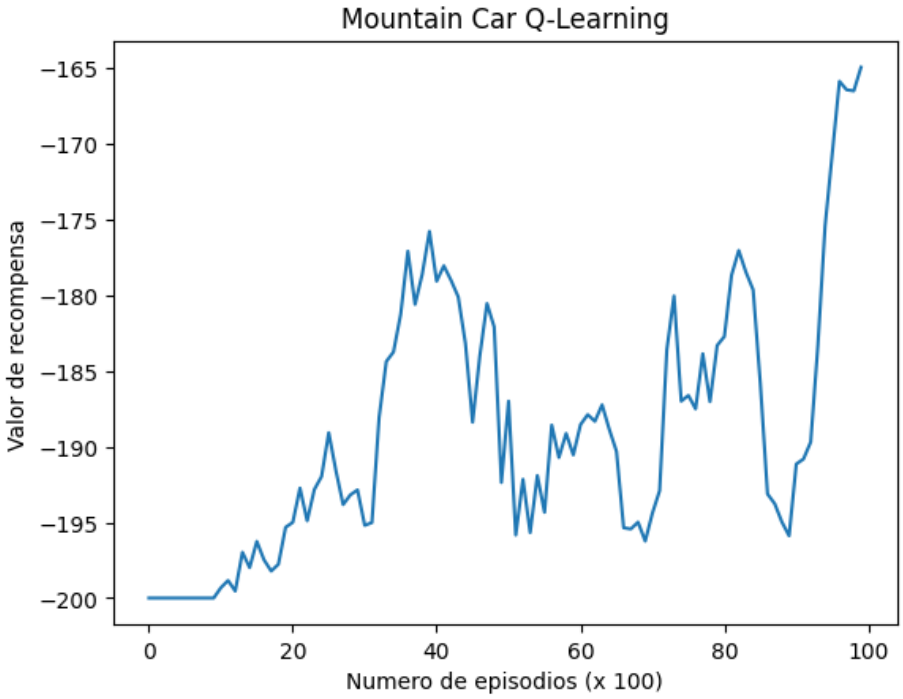

ILUSTRACIÓN 20 - GRÁFICA MOUNTAIN CAR Q-LEARNING

En la Ilustración 20 - Gráfica Mountain Car Q-Learning, se presentan los resultados del entrenamiento de un agente a lo largo de 10000 episodios, con un promedio de resultados calculado cada 100 episodios. La gráfica muestra un aprendizaje significativo en la primera mitad de los episodios, pero luego parece haberse estancado. Sin embargo, en los episodios más recientes, el agente logró mejorar su rendimiento, alcanzando su mejor recompensa registrada, que fue de -165.

6.4.3 Resultados Mountain Car DQN

Hiperparámetros

lr = 0.0001 # Tasa de aprendizaje

gamma = 0.95 # Factor de descuento

epsilon = 0.5 # Valor de epsilon inicial

epsilon_decay = 0.995 # Factor de decaimiento de epsilon

replay_buffer_size = 30000 # Tamaño del buffer de experiencias

batch_size = 64 # Tamaño del lote para el entrenamiento

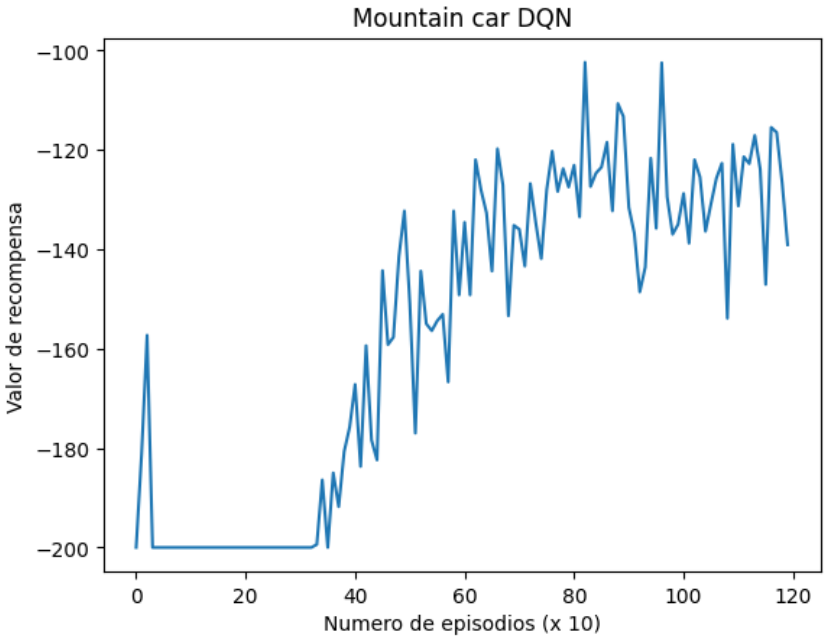

ILUSTRACIÓN 21 - GRÁFICA MOUNTAIN CAR DQN

En la Ilustración 21 - Gráfica Mountain Car DQN, se presenta el desempeño de un agente que fue entrenado a lo largo de 1200 episodios, y se calculó un promedio cada 10 episodios para su representación en la gráfica. Al principio de la gráfica, se observa un pico, el cual se debe a que el agente no lograba aprender de manera efectiva utilizando el mismo algoritmo que se aplicó en los otros 2 escenarios. Para abordar esta situación, se implementaron 30 episodios de exploración guiada, en los cuales el agente fue programado para tomar acciones hacia la izquierda durante un número específico de iteraciones y luego tomar acciones hacia la derecha en las iteraciones restantes. Esto se hizo con el propósito de que el agente aprendiera como ascender la pendiente y alcanzar la meta.

Como resultado de esta implementación, se redujo el valor de “épsilon” a 0.5. Esta adaptación condujo a un aprendizaje mucho más efectivo por parte del agente, que se refleja en la gráfica con una recompensa máxima alcanzada de -102.4.

6.4.4 Conclusiones resultados escenario 2: Mountain Car

Con respecto al entrenamiento de un agente en el escenario de Mountain Car utilizando tres algoritmos diferentes, se pueden extraer las siguientes conclusiones:

- Acciones aleatorias: El agente que tomó acciones aleatorias nunca logró alcanzar la meta durante el entrenamiento. Este enfoque demostró ser altamente ineficiente en este escenario, ya que la probabilidad de que el agente tome las acciones correctas de manera aleatoria es extremadamente baja. Como resultado, la recompensa obtenida fue muy baja y consistente en episodios sin éxito.
- Q-Learning: El algoritmo de Q-Learning representó una mejora significativa en comparación con las acciones aleatorias. A lo largo del entrenamiento, el agente aprendió gradualmente a navegar hacia la meta. Sin embargo, a pesar de esta mejora el rendimiento del agente todavía quedó por debajo de las expectativas, ya que la recompensa máxima obtenida no alcanzó un valor muy alto -165. Esto

sugiere que el algoritmo Q-Learning tuvo dificultades para encontrar una estrategia óptima para maximizar la recompensa.

- DQN con exploración guiada: El algoritmo DQN, combinado con la exploración guiada durante los primeros episodios, destacó como la estrategia más efectiva entre los tres. A través de la exploración guiada, el agente adquirió conocimientos valiosos sobre como abordar el escenario. Luego, después de la fase inicial de exploración guiada, el agente pudo continuar mejorando su rendimiento de manera constante. Esto se reflejó en la recompensa máxima obtenida, siente esta -102.4.

En conclusión, las acciones aleatorias resultaron ser ineficientes para resolver el problema de Mountain Car, mientras que Q-Learning permitió un aprendizaje gradual pero insuficiente. Por otro lado, el enfoque de con exploración guiada demostró ser altamente efectivo, ya que permitió al agente aprender una estrategia exitosa y obtener recompensas más altas.

Estos resultados enfatizan la importancia de estrategias de exploración efectivas y algoritmos de aprendizaje más avanzados para abordar con éxito entornos complejos de aprendizaje por refuerzo como el escenario de Mountain Car.

7. Conclusiones y trabajo futuro

Las conclusiones obtenidas en este estudio demuestran que se han alcanzado con éxito los siguientes objetivos:

- Se ha logrado una comprensión profunda de los fundamentos teóricos del aprendizaje por refuerzo, incluyendo la teoría detrás de los algoritmos comunes utilizados en este campo como los algoritmos de Q-Learning o las redes neuronales profundas.
- Se han seleccionado cuidadosamente los entornos de juego apropiados proporcionados por la biblioteca Gymnasium para llevar a cabo el desarrollo y entrenamiento del agente.
- Se han implementado y entrenado agentes utilizando una variedad de algoritmos, incluyendo Acciones Aleatorias, Q-Learning y DQN, lo que permitió una comparación significativa del desempeño de cada uno.
- Se evaluó y comparó exhaustivamente el rendimiento de los agentes en términos de las recompensas obtenidas y la mejora a lo largo del tiempo en escenarios desafiantes de aprendizaje por refuerzo.
- Todos los resultados, incluyendo los algoritmos implementados, los entornos de juego seleccionados y las conclusiones derivadas del estudio, se ha documentado de manera completa y clara.

En conclusión, este estudio ha cumplido sus objetivos al proporcionar una exploración integral del aprendizaje por refuerzo y una evaluación comparativa de diferentes algoritmos en múltiples escenarios de juego, así como la implementación de todos los algoritmos mencionados. Las conclusiones obtenidas brindan información valiosa sobre la efectividad de estos algoritmos en entornos de aprendizaje por refuerzo y su aplicabilidad en situaciones del mundo real.

En cuanto a los trabajos futuros, se plantea la exploración de nuevos desafíos y enfoques en el ámbito del Trabajo de Final de Grado. Entre las posibilidades a considerar se encuentran:

1. Juegos más complejos: Una dirección prometedora es ampliar el estudio a juegos más complejos y desafiantes, que requieran una mayor capacidad de aprendizaje y adaptación por parte del agente. Esto permitirá evaluar el desempeño de los algoritmos de Reinforcement Learning en entornos más realistas y sofisticados.
2. Algoritmos más avanzados: Es importante investigar y experimentar con algoritmos más avanzados de Reinforcement Learning. Esto incluye métodos como el aprendizaje profundo (Deep Reinforcement Learning), algoritmos basados en políticas (Policy-based methods), y algoritmos de aprendizaje por refuerzo basados en modelos (Model-based Reinforcement Learning). Estos enfoques pueden proporcionar nuevas perspectivas y mejoras significativas en el rendimiento del agente.
3. Juegos multijugador: Otra área interesante a explorar es el entrenamiento de agentes en juegos de multijugador, donde interactúan con otros agentes o jugadores humanos. Este escenario introduce desafíos adicionales, como la coordinación, la competencia y la colaboración entre agentes, lo que requiere estrategias más sofisticadas y adaptativas.

En conclusión, los trabajos futuros del Trabajo de Final de Grado se centran en expandir el ámbito de estudio hacia juegos más complejos, algoritmos más avanzados y juegos

de multijugador. Estas investigaciones permitirán un mayor entendimiento de las capacidades y limitaciones de los algoritmos de Reinforcement Learning, así como su aplicación en entornos más desafiantes y realistas.

7.1 Relación con asignaturas cursadas

En este capítulo, se explorarán los fundamentos y la aplicación de las asignaturas cursadas durante la carrera que están directamente relacionadas con el Trabajo de Final de Grado. Estas asignaturas proporcionaron los conocimientos y habilidades necesarios para comprender y abordar los desafíos planteados en el desarrollo del presente trabajo.

A continuación, se detallan las asignaturas y su relevancia en el contexto del TFG:

1. Algorítmica y matemáticas para juegos: Esta asignatura sentó las bases teóricas y prácticas para comprender los conceptos algorítmicos fundamentales aplicados en el ámbito de los juegos. Proporcionó conocimientos sobre algoritmos y estructuras de datos que son esenciales en el diseño de algoritmos de aprendizaje automático y toma de decisiones en entornos de juego.
2. Programación 1 y Programación 2: Estas asignaturas introdujeron los conceptos básicos de programación.
3. Inteligencia artificial: Esta asignatura proporcionó los conocimientos fundamentales sobre la inteligencia artificial. Los conceptos aprendidos en esta asignatura fueron aplicados en el desarrollo e implementación de algoritmos de Reinforcement Learning en el TFG.
4. Asignaturas de proyectos: Estas son las asignaturas en las que se trabajó con la metodología SCRUM y me brindaron una base sólida en la gestión de proyectos de desarrollo de software. El conocimiento adquirido en estas asignaturas permitió una planificación efectiva, colaboración en equipo y seguimiento adecuado del progreso del TFG.
    a. Proyecto Diseño y Programación Web.
    b. Proyecto Internet de las cosas y Aplicaciones móviles.
    c. Proyecto Aplicaciones Multimedia Interactivas. Videojuegos.
    d. Proyecto Aplicaciones de Biometría y Medio Ambiente.
    e. Proyecto de Robótica.
    f. Proyecto Entornos Interactivos Avanzados.

8. Bibliografía

[1] Bagnato, J. I. (s.f.). Aprendizaje por refuerzo. Aprendizaje Machine Learning. https://www.aprendemachinelearning.com/

[2] Silva, M. (2019, 30 de mayo). Aprendizaje por refuerzo: Procesos de decisión de Markov (Parte 1). Aprendizaje por Refuerzo: Introducción al Mundo del Machine Learning. https://medium.com/aprendizaje-por-refuerzo-introducci%C3%B3n-al-mundo-del/aprendizaje-por-refuerzo-procesos-de-decisi%C3%B3n-de-markov-parte-1-8a0aed1e6c59

[3] Watkins, C. J. C. H. y Dayan, P. Q-learning. Machine Learning, vol. 8(3), páginas 279–292, 1992. ISSN 1573-0565.

[4] van Hasselt, H., Guez, A., & Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1). https://doi.org/10.1609/aaai.v30i1.10295

[5] Gymnasium. (2023). Página principal de Gymnasium. Gymnasium. https://gymnasium.farama.org/

[6] NumPy (Versión 1.25.0). (2023). GitHub. https://github.com/numpy/numpy

[7] Matplotlib (Versión 3.7.1). (2023). GitHub. https://github.com/matplotlib/matplotlib

[8] Biblioteca OS en Python (Versión 3.11.3). (2023). Python Software Foundation. https://docs.python.org/3/library/os.html

[9] Torch. (2023). PyTorch (Versión 2.0.1). [Software]. https://pytorch.org/

[10] ImageIO. (Versión 2.31.1) (2023). Biblioteca Python para leer y escribir datos de imágenes. Github. https://github.com/imageio/imageio

[11] OpenCV-python (Versión 4.8.0.74) (2023) opencv-python en PyPI. https://pypi.org/project/opencv-python/

[12] Gymnasium. (2023). Cartpole - Gymnasium. https://gymnasium.farama.org/environments/classic_control/cart_pole/

[13] Gymnasium. (2023). Acrobot - Gymnasium. https://gymnasium.farama.org/environments/classic_control/acrobot/

[14] Gymnasium. (2023). Mountain Car - Gymnasium. https://gymnasium.farama.org/environments/classic_control/mountain_car/

[15] Kak, A., & Bouman, C. (2023). Reinforcement Learning with Discrete and Continuous State Spaces. Purdue University.

[16] Zhu, J., Wu, F., & Zhao, J. (2021). An Overview of the Action Space for Deep Reinforcement Learning. En 2021 4th International Conference on Algorithms, Computing and Artificial Intelligence (ACAI'21), diciembre 22–24, 2021, Sanya, China. ACM, New York, NY, USA. https://doi.org/10.1145/3508546.3508598

[17] Tokio School. (2023). Tipos de Deep Learning. https://www.tokioschool.com/noticias/tipos-deep-learning/

ANEXO

Enlace a mi repositorio de GitHub con los documentos empleados durante los entrenamientos: https://github.com/danieltortosazango/aprendizaje-por-refuerzo.git