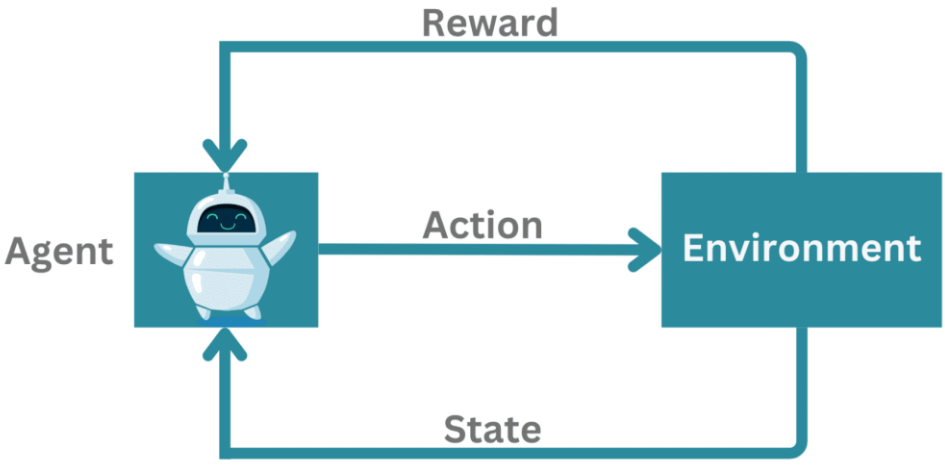
Implementación de Técnicas de Aprendizaje Automático para la Conducción Autónoma en Entornos Simulados



Alondra Hernández Sánchez

2ºDAM

ÍNDICE

1.Introducción.

En este documento se detalla el proceso de desarrollo del proyecto *Implementación de Técnicas de Aprendizaje Automático para la Conducción Autónoma en Entornos Simulados* desarrollado como trabajo final del ciclo formativo de grado superior en Desarrollo de Aplicaciones Multiplataforma.

Este documento recoge el objetivo del proyecto, las herramientas utilizadas, su planificación y la metodología de trabajo seguida en cada etapa, y los resultados obtenidos.

Se puede acceder a el proyecto en su totalidad siguiendo el siguiente enlace: https://drive.google.com/drive/folders/14GEI-f0YP8Sfty2B2_TTFna4v9nWXkPv

2.Objetivo.

El presente proyecto tiene como propósito la creación y análisis de un sistema de conducción autónoma, evaluando su comportamiento en distintos entornos simulados a través del uso de inteligencia artificial aplicada al control de agentes. Concretamente, se busca comparar la efectividad de tres enfoques de aprendizaje distintos:

- **Q-Learning**: como técnica clásica de aprendizaje por refuerzo basada en valores discretos.
- **NEAT(NeuroEvolution of Augmenting Topologies)**, como enfoque evolutivo que permite la generación automática de arquitecturas neuronales.
- **Behavioral Cloning:** una técnica supervisada basada en la imitación del comportamiento humano a partir de datos reales.

3.Metodología y Herramientas de Desarrollo.

El desarrollo del proyecto se ha dividido en varias etapas, cada una centrada en uno de los tipos de aprendizajes mencionados con anterioridad. A continuación, se describen las herramientas utilizadas:

Entorno de desarrollo y lenguaje

- **PyCharm**: Entorno de desarrollo integrado (IDE) de JetBrains utilizado durante todo el desarrollo para la escritura de scripts en Python
- **Python**: Lenguaje principal del proyecto debido a su amplia disponibilidad de librerías de IA. Se usaron versiones específicas según compatibilidad: Python 3.10 para Gymnasium y 3.7 para CARLA.

Librerías fundamentales

- **NumPy (v1.24)**: Librería utilizada en los tres enfoques para el tratamiento de vectores de observación, recompensas y estados.
- **Pygame**: Motor de juegos ligero usado para crear un simulador personalizado 2D. Para el aprendizaje por refuerzo se usó durante la implementación de radares,

sensores, colisiones y movimiento de coches. En CARLA funcionó como un visor embebido en Python que muestra las imágenes recibidas del servidor.

Aprendizaje por Refuerzo (Q-Learning y NEAT)

- **Toy Text (Gymnasium)**: Entornos de prueba simplificados como Taxi.
- **Gymnasium**: Proporciona una API estándar para entrenamiento y evaluación de agentes por refuerzo, perteneciente a OpenAI Gym
- **NEAT-Python**: Implementación del algoritmo NEAT. Permite evolucionar tanto la topología como los pesos de redes neuronales. Fue utilizado para generar redes neuronales que controlaran al coche de Fórmula 1 en el simulador Pygame con un circuito diseñado por mi parte.

Aprendizaje supervisado

- **CARLA (v0.9.5)**: Plataforma avanzada de simulación de conducción autónoma basada en Unreal Engine que proporciona sensores (cámara, LiDAR, GPS, colisión), entornos urbanos y capacidad para registrar datos de conducción.
- **Keras + TensorFlow (v2.10)**: Utilizados en la fase de aprendizaje supervisado (Behavioral Cloning).
- **OpenCV**: Librería de visión por ordenador utilizada para procesar y transformar imágenes provenientes de los sensores de CARLA.
- **Pillow**: Librería de manipulación de imágenes y preprocesar capturas de pantalla recogidas por los sensores de CARLA durante los entrenamientos.

4.Planificación del Proyecto.

El proyecto fue realizado entre febrero y mayo de 2025, con la siguiente distribución de tiempo

Tareas	*Horas(aproximadamente)*
Buscar la idea de proyecto	6
Estudiar Python	12
Desarrollo Proyecto Q-Learning	18
Desarrollo Proyecto NEAT	24
Desarrollo Proyecto Behavioral Cloning	40
Documentación	5
Presentación	7
Total	112

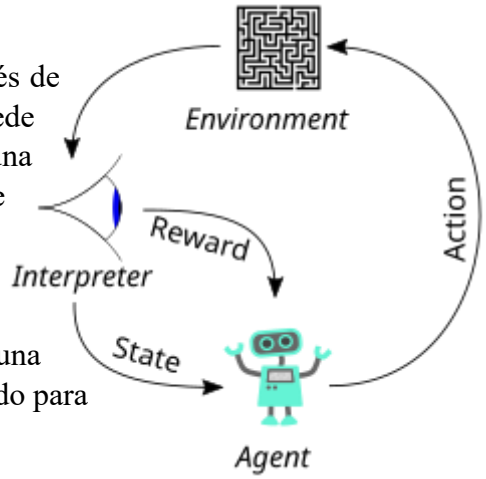
5.Desarrollo de la Aplicación.

Como se ha mencionado, el proyecto se divide en tres capítulos, cada uno asociado a un enfoque de aprendizaje distinto, con niveles crecientes de complejidad. A continuación, se presenta una introducción teórica a cada uno ellos, mientras que la implementación práctica y los detalles del código se encuentran explicados en los vídeos adjuntos.

5.1.Aprendizaje por refuerzo

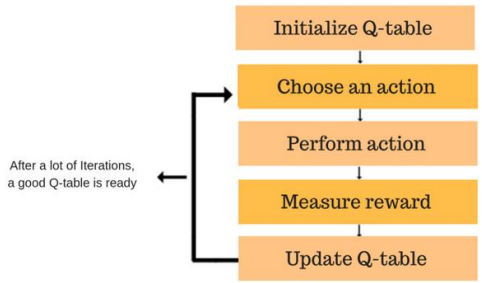
El aprendizaje por refuerzo se basa en la interacción continua entre un agente y su entorno, donde el agente debe tomar decisiones con el objetivo de maximizar una recompensa acumulada a lo largo del tiempo.

En este enfoque, el agente percibe el entorno a través de un conjunto de estados, y en cada uno de ellos puede elegir entre varias acciones posibles. Al ejecutar una acción, el agente transita de un estado a otro, y recibe una recompensa asociada a dicha acción. Esta recompensa, que puede ser positiva o negativa, actúa como una señal de retroalimentación que guía el aprendizaje. El objetivo final es encontrar una estrategia que indique qué acción tomar en cada estado para maximizar la suma de recompensas futuras.



5.1.1.Q-Learning.

Q-Learning es un algoritmo clásico de **aprendizaje por refuerzo** que permite a un agente aprender a actuar en un entorno a través de la experimentación y la retroalimentación, sin necesidad de conocer previamente el modelo del entorno. Fue desarrollado por Christopher Watkins en 1989.



El núcleo del algoritmo es la **Q-table**, una estructura que almacena el valor esperado (Q) de cada par estado-acción. Estos valores representan la utilidad esperada de realizar una acción determinada en un estado específico, siguiendo la mejor estrategia.

A medida que el agente interactúa con el entorno, la tabla se actualiza mediante la siguiente ecuación:

$$\mathbf{Q(s, a) \leftarrow Q(s, a) + \alpha\,[r + \gamma * \max Q(s', a') - Q(s, a)]}$$

Donde:

- α es la tasa de aprendizaje,
- γ es el factor de descuento,
- r es la recompensa recibida,
- s es el estado actual,
- a es la acción tomada,

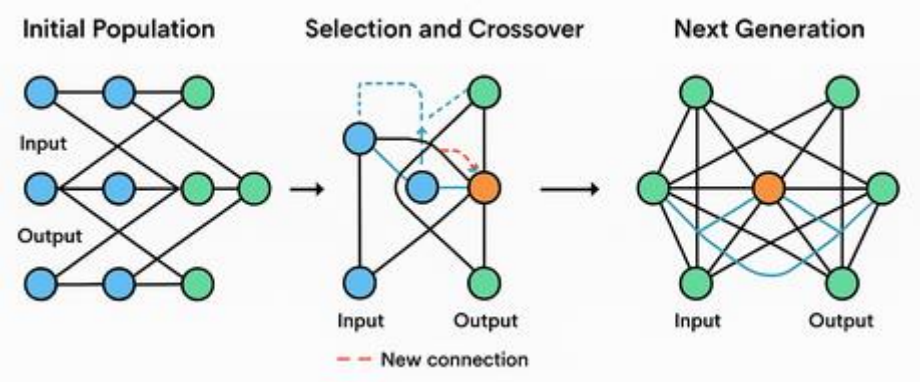
- s’ es el nuevo estado alcanzado.

Para este proyecto, Q-Learning se aplicó en el entorno **Taxi-v3** de **Gymnasium**, una simulación en la que un taxi debe recoger y dejar pasajeros en un hotel, siendo cada ubicación una cuadrícula.

5.1.2. NeuroEvolution of Augmenting Topologies(NEAT)

NEAT es un algoritmo de neuroevolución diseñado por Kenneth Stanley, que permite entrenar agentes mediante la evolución progresiva de redes neuronales artificiales.

El proceso se basa en un enfoque evolutivo: se crea una población inicial de agentes que interactúan con el entorno y son evaluados en función de su desempeño. Aquellos que presentan mejor comportamiento son seleccionados para reproducirse, cruzando sus topologías y aplicando mutaciones controladas. Con el tiempo, los agentes se adaptan al problema planteado, descubriendo estrategias de comportamiento efectivas.



En este proyecto, NEAT se aplicó a un simulador personalizado ambientado en el circuito urbano de Fórmula 1 de Marina Bay (Singapur), donde el agente, representado por un coche, debía recorrer el circuito sin colisionar, utilizando como entradas los datos de sensores tipo radar, que controlaban el giro y avance del vehículo. El entrenamiento se realizó a lo largo de múltiples generaciones, utilizando como métrica de evaluación la distancia total recorrida sin accidente.

5.2.Aprendizaje supervisado

El aprendizaje supervisado se basa en el uso de un conjunto de datos que contienen pares de entrada y salida. El objetivo de este enfoque es que un modelo aprenda una función que relacione ambas variables, de modo que, ante nuevas entradas, pueda predecir con precisión su salida correspondiente.

Este tipo de aprendizaje se denomina "supervisado" porque el modelo es guiado por ejemplos durante el entrenamiento, actuando como si recibiera la supervisión de un maestro. A medida que procesa los datos, el algoritmo ajusta sus parámetros internos para minimizar el error de predicción entre la salida que genera y la salida esperada.

5.2.1.Behavioral Cloning.

En este proyecto, se empleó CARLA como entorno de simulación avanzada para la implementación de Behavioral Cloning. Para ello, se desarrolló un sistema de captura de

datos en el que un conductor humano controlaba manualmente un vehículo dentro del simulador. Durante cada sesión de entrenamiento, se registraron las imágenes captadas por la cámara frontal del vehículo.

Este conjunto de datos sirvió como base para entrenar un modelo de red neuronal convolucional, desarrollado con Keras y TensorFlow, cuyo objetivo era aprender a predecir la acción de giro adecuada únicamente a partir de los datos recolectados. El resultado fue un agente capaz de reproducir el comportamiento del conductor humano, tomando decisiones de forma autónoma y basada exclusivamente en la información visual, sin requerir exploración activa del entorno ni un sistema de recompensa, a diferencia de los proyectos anteriores.

6.Conclusión.

La mayor dificultad encontrada durante el desarrollo del tercer proyecto estuvo relacionada con los requisitos de hardware del simulador CARLA, los cuales excedían las capacidades de la GPU disponible en mi equipo. Esto obligó a utilizar una versión anterior y más ligera del simulador. Como consecuencia, la simulación se limitó a la interacción básica con el entorno (aceleración y giro del vehículo), sin posibilidad de controlar elementos más avanzados como líneas de circulación o señales de tráfico.

Esta limitación también implicó un cambio en la planificación inicial, que contemplaba la implementación de un modelo de aprendizaje por refuerzo basado en Deep Q-Learning (DQN) dentro de CARLA. Sin embargo, debido al alto consumo de recursos del script correspondiente, se optó por sustituir esta técnica por Behavioral Cloning, una alternativa más ligera y viable dadas las circunstancias.

Problemas de rendimiento, aunque en menor medida, también se presentaron durante el segundo proyecto. Específicamente, se observó parpadeo en la pantalla al reiniciar una nueva generación de agentes, un efecto visual asociado a las limitaciones gráficas del sistema.

A pesar de estas dificultades técnicas,la inversión de tiempo en búsqueda de información teórica y práctica la realización del proyecto ha resultado altamente satisfactoria. De haber contado con una GPU más potente, me hubiese gustado explorar el uso de sensores LiDAR implementados en versiones recientes de CARLA, así como el entrenamiento de redes neuronales más complejas. Con suerte, serán desarrollados en un futuro como proyectos de interés personal.

7. Referencias.

1. Python

- *Python Full Course for free*. YouTube. https://www.youtube.com/watch?v=XKHEtdqhLK8

- NumPy (v1.24.4). https://pypi.org/project/numpy/1.24.4/#files

2. Q-Learning

- DataCamp. (s.f.). *Introducción a Q-Learning: Tutorial para principiantes*. https://www.datacamp.com/es/tutorial/introduction-q-learning-beginner-tutorial
- Wikipedia. (s.f.). *Q-learning*. https://es.wikipedia.org/wiki/Q-learning
- Inesdi. (2023). *Q-learning: qué es y cómo funciona este algoritmo de IA*. https://www.inesdi.com/blog/Q-learning/
- freeCodeCamp. (2022). *Introducción a Q-learning: Aprendizaje por refuerzo*. https://www.freecodecamp.org/espanol/news/introduccion-a-q-learning-aprendizaje-por-refuerzo/
- *Q-Learning Tutorial 1: Train Gymnasium FrozenLake-v1 with Python Reinforcement Learning: https://www.youtube.com/watch?v=ZhoIgo3qqLU*
- Gym – Repositorio oficial. https://github.com/openai/gym
- Gymnasium:
    - Sitio oficial: https://gymnasium.farama.org/
    - Repositorio GitHub: https://github.com/Farama-Foundation/Gymnasium
    - Entorno Taxi-v3: https://gymnasium.farama.org/environments/toy_text/taxi/

3. NEAT (NeuroEvolution)

- *Simple AI Tutorial with NEAT-Python*. YouTube. https://www.youtube.com/watch?v=2o-jMhXmmxA
- NEAT-Python – Documentación oficial. https://neat-python.readthedocs.io/en/latest/

4. CARLA y Behavioral Cloning

- CARLA Simulator – Sitio oficial. https://carla.org/
- Repositorio oficial de CARLA. https://github.com/carla-simulator/carla/releases
- TensorFlow. https://www.tensorflow.org/
- *Deep Q-Learning Tutorial*. PythonProgramming.net. https://pythonprogramming.net/deep-q-learning-dqn-reinforcement-learning-python-tutorial/

- IBM. (s.f.). *What is Supervised Learning?* https://www.ibm.com/cloud/learn/supervised-learning
- NVIDIA Developer Blog. (s.f.). *Behavioral Cloning with Deep Learning*. https://developer.nvidia.com/blog/deep-learning-self-driving-cars/
- Towards Data Science. (2020). *Behavioral Cloning: Teaching AI to Drive by Imitation*. https://towardsdatascience.com/behavioral-cloning-teaching-ai-to-drive-by-imitation-465fdf4ffcd8