INSTITUTO TECNOLÓGICO DE AGUASCALIENTES

APRENDIZAJE POR REFUERZO PARA LA CONDUCCIÓN AUTÓNOMA DE AGENTES MÓVILES INTELIGENTES

Tesis para obtener el grado de:

Maestría en Ciencias de la Ingeniería

PRESENTA:

Ing. Juan Miguel Hernández López

DIRECTOR:

Dr. Francisco Javier Luna Rosas

CODIRECTOR:
Dr. Julio Cesar Martínez Romo

Aguascalientes, Aguascalientes, octubre 2025

Instituto Tecnológico de Aguascalientes
División de Estudios de Posgrado e Investigación

Aguascalientes, Ags., 6/octubre/2025
Oficio No. DEPI 627/2025
Asunto: Autorización impresión tesis

C. JUAN MIGUEL HERNÁNDEZ LÓPEZ
PRESENTE

Los abajo firmantes, Miembros del Jurado para Examen de Grado de Maestría, hacemos **constar** que, habiendo revisado el Trabajo de Tesis desarrollado por Usted, bajo el título: **"Aprendizaje por Refuerzo para la Conducción Autónoma de Agentes Móviles Inteligentes"**, hemos dictaminado que éste es de aceptarse, por lo que se autoriza su impresión.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
Ingenio, Cultura y Saber, que conducen a la Excelencia®

PRESIDENTE

DR. FRANCISCO JAVIER LUNA ROSAS

SECRETARIO

DR. JULIO CESAR MARTINEZ ROMO

VOCAL

DR. MARCO ANTONIO HERNÁNDEZ VARGAS

ccp. Archivo
JFLH/yevo

2025 Año de La Mujer Indígena

Instituto Tecnológico de Aguascalientes

ESPACIO 100% LIBRE DE PLÁSTICO

MB

ABET

AUTORIZACIÓN DE USO DE DERECHOS DE AUTOR OTORGADO POR

Juan Miguel Hernández López, mayor de edad, con domicilio ubicado en Caballería 130, Municipio Libre, Ags, en mi calidad de titular y autor de la tesis denominada _ **Aprendizaje por Refuerzo para la Conducción Autónoma de Agentes Móviles Inteligentes** __ quien para todos los fines del presente documento se denominará **EL AUTOR Y/O TITULAR**, suscribo el presente documento de autorización de uso de derechos patrimoniales de autor a favor del Tecnológico Nacional de México/Instituto Tecnológico de Aguascalientes el cual se regirá por las clausulas siguientes:

PRIMERA – AUTORIZACIÓN: EL AUTOR Y/O TITULAR, mediante el presente documento autoriza la utilización de los derechos patrimoniales de autor al Tecnológico Nacional de México/Instituto Tecnológico de Aguascalientes, de la tesis denominada _ **Aprendizaje por Refuerzo para la Conducción Autónoma de Agentes Móviles Inteligentes** _, a través del Repositorio Institucional del Tecnológico Nacional de México (en lo sucesivo TecNM) y en el Repositorio Nacional, que puede ser consultado en la liga electrónica: (www.repositorionacionalcti.mx/).

SEGUNDA – OBJETO: Por medio del presente escrito, **EL AUTOR Y/O TITULAR** autoriza al Tecnológico Nacional de México/Instituto Tecnológico de Aguascalientes, a través del Repositorio Institucional del TecNM y en el Repositorio Nacional para que de conformidad con la Ley Federal del Derecho de Autor y la Ley de la Propiedad Industrial, use los derechos del documento antes referido, con fines exclusivamente académicos.

TERCERA – TERRITORIO: Los derechos aquí autorizados se dan sin limitación geográfica o territorial alguna.

CUARTA – ALCANCE: La presente autorización se da tanto para formato o soporte material, y se extiende a la utilización en medio óptico, magnético, electrónico, en red, mensajes de datos o similar conocido o por conocer, del ejemplar o número respectivo de la publicación.

QUINTA – EXCLUSIVIDAD: La autorización de uso aquí establecida no implica exclusividad en favor del Tecnológico Nacional de México/Instituto Tecnológico de Aguascalientes. Por lo tanto, **EL AUTOR Y/O TITULAR** en su carácter de autor de la obra objeto del presente documento se reserva el derecho de publicar directamente, u otorgar a cualquier tercero, autorizaciones de uso similares o en los mismos términos aquí acordados.

SEXTA – DERECHOS MORALES (Créditos y mención): La Autorización de los derechos antes mencionados no implica la cesión de los derechos morales sobre los mismos por cuanto en conformidad con lo establecido en los artículos 18, 19, 20, 21, 22 y 23 de la Ley Federal de Derechos de Autor, dada la cuenta que estos derechos son inalienables, imprescriptibles, irrenunciables e inembargables. Por lo tanto, los mencionados derechos seguirán radicados en cabeza de EL AUTOR Y/O TITULAR, y siempre deberá mencionarse su nombre cuando se utilice la obra.

SÉPTIMA – AUTORÍA: EL AUTOR Y/O TITULAR, declara y ratifica que el material objeto de la presente fue realizada por él o por ella sin violar o usurpar derechos de Propiedad Intelectual de terceros.

Aguascalientes, Aguascalientes, a ___7____ días del mes de ____octubre_____ de 2025_.

Autor de la Tesis	**Director de Tesis**
Juan Miguel Hernández López	
Nombre y firma	**Nombre y firma**

Instituto Tecnológico de Aguascalientes
División de Estudios de Posgrado e Investigación

Aguascalientes, Ags., 21/octubre/2025
Asunto: Constancia de Originalidad

CONSEJO DE POSGRADO
MAESTRÍA EN CIENCIAS DE LA INGENIERÍA
PRESENTE

El director de tesis de **Juan Miguel Hernández López**, estudiante de **Maestría en Ciencias de la Ingeniería** con número de control **G23153206**, hace constar que, después de haber realizado la revisión del informe de originalidad generado con la herramienta Turnitin, el trabajo **"Aprendizaje por Refuerzo para la Conducción Autónoma de Agentes Móviles Inteligentes"** es original conforme a los criterios establecidos. El porcentaje de similaridad es de **12%** y se adjunta resumen del reporte.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
Ingenio, Cultura y Saber, que conducen a la Excelencia

DR. FRANCISCO JAVIER LUNA ROSAS
DIRECTOR DE TESIS

Vo.Bo.

mard81

DR. MARIO ALBERTO RODRÍGUEZ DÍAZ
COORDINADOR DE LA MAESTRÍA EN CIENCIAS DE LA INGENIERÍA

ccp. Archivo

EDUCACIÓN
INSTITUTO TECNOLÓGICO DE AGUASCALIENTES
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Tesis_MCI_Juan_Miguel_Hernandez_Lopez.pdf

Juan Miguel Hernández López
Juan Miguel Hernández López
Tecnológico Nacional de Mexico

Detalles del documento

Identificador de la entrega
trn:oid:::20755:514341547

Fecha de entrega
17 oct 2025, 12:46 p.m. GMT-6

Fecha de descarga
17 oct 2025, 1:01 p.m. GMT-6

Nombre del archivo
Tesis_MCI_Juan_Miguel_Hernandez_Lopez.pdf

Tamaño del archivo
3.2 MB

85 páginas

18.268 palabras

107.837 caracteres

12% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Filtrado desde el informe

- Bibliografía
- Texto citado
- Texto mencionado
- Coincidencias menores (menos de 8 palabras)

Fuentes principales

5% Fuentes de Internet
2% Publicaciones
10% Trabajos entregados (trabajos del estudiante)

Marcas de integridad

N.° de alerta de integridad para revisión

Texto oculto
93 caracteres sospechosos en N.° de página
El texto es alterado para mezclarse con el fondo blanco del documento.

Los algoritmos de nuestro sistema analizan un documento en profundidad para buscar inconsistencias que permitirían distinguirlo de una entrega normal. Si advertimos algo extraño, lo marcamos como una alerta para que pueda revisarlo.

Una marca de alerta no es necesariamente un indicador de problemas. Sin embargo, recomendamos que preste atención y la revise.

Agradecimientos

Quiero expresar mi más profundo agradecimiento a mi madre y a mi padre, cuyo apoyo incondicional, amor y guía han sido fundamentales para alcanzar esta meta. Gracias por enseñarme el valor del esfuerzo y la perseverancia, y por estar siempre a mi lado en cada paso de este camino.

También deseo agradecer al Instituto Tecnológico de Aguascalientes por brindarme la oportunidad de formarme académicamente en un entorno de excelencia, y a todos los maestros que compartieron sus conocimientos y experiencias, guiándome y motivándome a superar cada desafío.

Resumen

El presente proyecto de investigación se centra en el análisis del aprendizaje por refuerzo como una técnica clave para mejorar el desempeño de agentes móviles inteligentes. El aprendizaje por refuerzo es un enfoque dentro de la inteligencia artificial que permite a un agente tomar decisiones óptimas mediante la interacción con su entorno y la retroalimentación obtenida a través de un sistema de recompensas.

Este análisis busca comprender los principios fundamentales del aprendizaje por refuerzo y su aplicación en la toma de decisiones de agentes móviles inteligentes. Se explorarán diferentes algoritmos de aprendizaje por refuerzo y sus capacidades para optimizar el comportamiento de estos agentes en entornos dinámicos, considerando factores como la adaptación a cambios en el entorno, la eficiencia en la navegación y la seguridad en la toma de decisiones.

A partir de este estudio, se espera identificar los principales desafíos y oportunidades en la implementación del aprendizaje por refuerzo en sistemas de conducción autónoma y otras aplicaciones de movilidad inteligente. Los resultados contribuirán al desarrollo de estrategias más eficientes para la adaptación de los agentes a escenarios cambiantes, mejorando su rendimiento y robustez en entornos del mundo real.

Abstract

This research project focuses on the analysis of reinforcement learning as a key technique to enhance the performance of intelligent mobile agents. Reinforcement learning is an approach within artificial intelligence that enables an agent to make optimal decisions through interaction with its environment and feedback received via a reward system.

This analysis aims to understand the fundamental principles of reinforcement learning and its application in the decision-making process of intelligent mobile agents. Various reinforcement learning algorithms will be explored, along with their capabilities to optimize agent behavior in dynamic environments, considering factors such as adaptation to environmental changes, navigation efficiency, and decision-making safety.

Through this study, the goal is to identify the main challenges and opportunities in the implementation of reinforcement learning for autonomous driving systems and other intelligent mobility applications. The findings will contribute to the development of more efficient strategies for agent adaptation to changing scenarios, improving their performance and robustness in real-world environments.

Índice de contenido

Agradecimientos **7**
Resumen **8**
1. Introducción **16**
1.1 Antecedentes 16
1.2 Situación problemática 17
1.3 Planteamiento del problema 17
1.4 Justificación 18
1.5 Objetivos 18
1.5.1 Objetivo general 18
1.5.2 Objetivos particulares 18
1.6 Hipótesis 19
1.6.1 Hipótesis nula 19
1.7 Metodología 19
1.7.1 Revisión de literatura y documentación técnica 20
1.7.2 Diseño del entorno de simulación y pruebas 20
1.7.3 Desarrollo y adaptación de algoritmos de aprendizaje por refuerzo 21
1.7.4 Implementación y pruebas experimentales 21
1.7.5 Análisis y evaluación de resultados 21
1.8 Conclusión del capítulo 21
2. Aprendizaje automático **22**
2.1 Características principales 22
2.2 Aprendizaje Supervisado 22
2.3 Aprendizaje no Supervisado 22
2.4 Aprendizaje por Refuerzo 23
2.4.1 Componentes principales 23
2.4.2 Proceso de decisión de Márkov (MDP) 24
2.4.3 Estados 25
2.4.4 Acciones. 27
2.4.5 Función de Transición 28
2.4.6 Función de Recompensa. 29
2.4.7 Horizonte. 31
2.4.8 Descuento 31

2.4.9 Objetivo: Política optima. 32
2.5 Conclusión del capítulo. 32
3. Algoritmos de Aprendizaje por Refuerzo 33
3.1 Algoritmo Q-learning 33
3.1.1 En búsqueda de la política optima, Inicialización de la tabla Q. 35
3.1.2 Elegir y realizar una acción (Exploración o explotación) 36
3.1.3 Obtener recompensa y actualizar la tabla Q 39
3.2 Deep Q-Network (DQN) 42
3.2.1 Q-Learning vs Deep Q-Network. 43
3.2.2 Replay Buffer. 43
3.2.3 Target Network 44
3.2.4 Q-Network. 44
3.2.5 Estrategia Explotación-Exploración 44
3.2.6 Función de Pérdida 45
3.2.7 Algoritmo DQN 45
3.2.8 Inicialización de la Q-network y Target-network 45
3.2.9 Interacción con el entorno 47
3.2.10 Almacenar la experiencia en el Replay Buffer 48
3.2.11 Muestrear minibatch aleatorios 49
3.2.12 Calcular objetivo y pérdida 50
3.2.13 Actualizar la Q-network 51
3.2.14 Copiar pesos a la Target network 51
3.2.15 Finalizar entrenamiento 52
3.2.16 Evaluación de la DQN 52
3.3 Actor-Critic: Aprendizaje basado en políticas y valores 53
3.3.1 Fundamentos de los Métodos Actor-Critic 54
3.3.2 Actor 54
3.3.3 Critic 54
3.3.4 Interacción entre actor y critic 55
3.3.5 Expresión matemática 55
3.3.6 Diagrama general 56
3.3.7 Algoritmo actor-critic 56
3.3.8 Definición de las redes actor y critic 57
3.3.9 Selección de acción 57

3.3.10 Evaluación del critic 58
3.3.11 Cálculo de la función de perdida y actualización 60
3.3.12 Bucle de entrenamiento 61
3.4 Conclusión del capítulo 62
4. Metodología 63
4.1 Características del entorno 64
4.2 Implementación técnica con Gymnasium 66
4.3 Consideraciones técnicas 68
4.4 Implementación de los algoritmos 69
4.4.1 Parámetros compartidos 69
4.4.2 Parámetros Q-Learning 70
4.4.3 Parámetros DQN 71
4.4.4 Parámetros AC 72
4.5 Conclusión del capítulo 73
5. Resultados y Conclusiones 75
5.1 Q-Learning 75
5.2 Deep Q-Network 78
5.3 Actor-Critic 82
5.4 Comparación entre algoritmos 85
5.5 Conclusiones parciales 86
5.6 Conclusión general 86
Referencias 88

Índice de figuras

Figura 1. Ciclo de la metodología del proyecto 20

Figura 2. Aprendizaje por refuerzo 23

Figura 3. El ambiente frozen lake (FL) 25

Figura 4. Los estados en el lago congelado (FL) contienen una única variable que indica el identificador de la celda en la que se encuentra el agente en cualquier paso de tiempo dado 26

Figura 5. Propiedad de Markov 26

Figura 6. Estados en el FL 27

Figura 7. Acciones en el FL 28

Figura 8. La función de transición 28

Figura 9. La función de transición en el entorno FL 29

Figura 10. La función de recompensa 30

Figura 11. Recompensa asignada a estados con transiciones que no tienen una recompensa de cero 30

Figura 12. El efecto del factor de descuento y el tiempo en el valor de las recompensas 31

Figura 13. El factor de descuento (gamma) 32

Figura 14. El entorno Norberto 34

Figura 15. El agente Norberto 35

Figura 16. Las mejores acciones para Norberto 37

Figura 17. El agente Norberto en un estado inicial aleatorio y sus acciones posibles 38

Figura 18. El agente Norberto realizando la acción Derecha 39

Figura 19. Estado – Recompensa del entorno de Norberto 40

Figura 20. Recompensa obtenida a través de los episodios. 41

Figura 21. Arquitectura DQN 43

Figura 22. Arquitectura Q-Network y Target-Network 46

Figura 23 Ciclo interacción con el entorno 47

Figura 24. Experiencia en Replay Buffer 48
Figura 25. Proceso de muestreo aleatorio de transiciones del Replay Buffer para la actualización de la Q-network 49
Figura 26. Copiar pesos a la target network desde la Q network cada K pasos 52
Figura 27. Flujo de evaluación DQN 53
Figura 28. Estructura del algoritmo actor-critic 56
Figura 29. Estructura de las redes neuronales Actor y Critic 57
Figura 30. Proceso de selección de acción 58
Figura 31. Diagrama evaluación del Critic 59
Figura 32. Diagrama de flujo del bucle de entrenamiento 61
Figura 33. Entorno ToyRace 63
Figura 34. 384 estados 64
Figura 35. Orientaciones ToyRace 64
Figura 36. Carriles ToyRace 65
Figura 37. Estados Terminales ToyRace 65
Figura 38. Acciones ToyRace 66
Figura 39. Recompensa por episodio entrenamiento QL 76
Figura 40. Recompensa máxima durante entrenamiento QL 76
Figura 41. Recompensa promedio durante evaluación QL 77
Figura 42. Tasa de éxito durante evaluación QL 77
Figura 43. Recompensa por episodio entrenamiento DQN 79
Figura 44. Recompensa máxima durante entrenamiento DQN 80
Figura 45. Recompensa promedio durante evaluación DQN 80
Figura 46. Tasa de éxito durante evaluación QL 81
Figura 47. Recompensa por episodio entrenamiento AC 82
Figura 48. Recompensa máxima durante entrenamiento AC 83
Figura 49. Recompensa promedio durante evaluación AC 83

Figura 50. Tasa de éxito durante evaluación AC 84
Figura 51. Gráficas comparativas del rendimiento de cada algoritmo 85

Índice de Tablas

Tabla 1. Representación de la inicialización de la tabla Q 35
Tabla 2. Valores Q para el estado 1 39
Tabla 3. Conocimiento adquirido para el estado 1 registrado en la Tabla Q 41
Tabla 4. Tabla Q final 42
Tabla 5. Diferencias entre Q-Learning y DQN 43
Tabla 6. Parámetros compartidos para entrenamiento 70
Tabla 7. Parámetros de entrenamiento para Q-Learning 71
Tabla 8. Parámetros de entrenamiento DQN 72
Tabla 9. Parámetros de entrenamiento AC 73
Tabla 10. Resultados por semilla QL 78
Tabla 11. Promedio de resultados de entrenamiento QL 78
Tabla 12. Resultados por semilla DQN 81
Tabla 13. Promedio de resultados de entrenamiento DQN 82
Tabla 14. Resultados por semilla AC 84
Tabla 15. Promedio de resultados de entrenamiento AC 84
Tabla 16. Comparativa entre algoritmos 85

1. Introducción

La investigación aborda los aspectos metodológicos relacionados con el aprendizaje por refuerzo, estableciendo la problemática y los objetivos que orientan el estudio. Se describen las fases del proceso de investigación y las actividades clave en el desarrollo del modelo propuesto. Además, se analiza el estado del arte, destacando los avances recientes en el aprendizaje por refuerzo y su aplicación en la toma de decisiones autónoma. Se exploran sus fundamentos teóricos, sus aplicaciones en agentes inteligentes y su potencial para mejorar la adaptabilidad y eficiencia de sistemas en entornos dinámicos.

1.1 Antecedentes

El aprendizaje por refuerzo (Reinforcement Learning, RL) es un área del aprendizaje automático que estudia cómo los agentes pueden aprender a tomar decisiones óptimas mediante la interacción directa con un entorno, recibiendo retroalimentación en forma de recompensas o penalizaciones (Sutton & Barto, 2018). Inspirado inicialmente por teorías de la psicología conductista, el RL busca replicar la capacidad de los seres vivos para aprender a través del ensayo y error (Thorndike, 1898; Skinner, 1938).

Desde sus primeras formulaciones teóricas, el aprendizaje por refuerzo ha evolucionado hasta convertirse en una de las áreas más activas e influyentes de la inteligencia artificial. Su relevancia ha crecido exponencialmente en la última década, especialmente tras la integración de redes neuronales profundas, lo que ha permitido resolver problemas complejos en dominios previamente inaccesibles para las máquinas (Mnih et al., 2015).

Actualmente, el RL es fundamental en una amplia gama de aplicaciones, incluyendo videojuegos (Mnih et al., 2015; Silver et al., 2016), robótica (Kober et al., 2013), sistemas de recomendación (Zhao et al., 2019), automatización industrial (Nguyen & Nguyen, 2020) y vehículos autónomos (Kiran et al., 2021). Estas aplicaciones demuestran el potencial del RL para enfrentar tareas que requieren aprendizaje autónomo, adaptabilidad y toma de decisiones en entornos dinámicos.

A pesar de los avances, el aprendizaje por refuerzo enfrenta retos importantes, como la eficiencia en el uso de muestras, la exploración efectiva de entornos desconocidos, la estabilidad del aprendizaje y la capacidad de generalización (Dulac-Arnold et al., 2021; Henderson et al., 2018). El desarrollo de nuevos enfoques y la integración de técnicas provenientes de otras áreas del aprendizaje automático continúan siendo temas centrales de investigación.

1.2 Situación problemática

La conducción autónoma representa la capacidad revolucionaria de los vehículos para realizar diversas misiones de conducción sin intervención humana (Raj, 2020). Impulsado por el enorme potencial de la inteligencia artificial, los vehículos autónomos o automatizados se han convertido en uno de los puntos destacados de investigación en todo el mundo (Rasouli, 2020). Fabricantes de automóviles líderes como Toyota, Tesla, Ford, Audi, Waymo, Mercedes-Benz, General Motors y otros están inmersos en el desarrollo de sus propios autos autónomos, alcanzando progresos increíbles en este emocionante campo. Paralelamente, los investigadores automotrices están concentrando sus esfuerzos en superar las barreras tecnológicas esenciales para lograr la plena automatización en los vehículos, lo cual promete un futuro de movilidad verdaderamente transformador (Gkartzonikas, 2019).

La tecnología de los vehículos autónomos se basa en una combinación de sensores avanzados, actuadores controlados por computadora y sofisticados algoritmos de inteligencia artificial, que les permiten percibir su entorno, tomar decisiones y realizar acciones de manera autónoma. Dentro de las técnicas de inteligencia artificial, el aprendizaje por refuerzo es especialmente relevante, ya que permite al vehículo adaptarse a diversas situaciones y aprender de su propia experiencia. Esta técnica se ha utilizado con éxito en la conducción autónoma, y se está aplicando en el desarrollo de comportamientos de conducción autónoma en circuitos de carreras, como en el caso del AWS Deepracer (McCalip, 2023).

A pesar de los emocionantes avances en la tecnología de vehículos autónomos, su perfección aún está lejos. La desconfianza pública hacia estos vehículos resalta de manera exagerada sus problemas, como lo evidencian los incidentes que llevaron a una reducción significativa en la flota de robotaxis de Cruise en San Francisco, poco después de su aprobación para operar como taxis (Delouya, 2023).

1.3 Planteamiento del problema

El desarrollo de agentes móviles inteligentes capaces de operar de forma autónoma en entornos dinámicos sigue siendo un reto relevante dentro de la inteligencia artificial y la robótica (Kober et al., 2013; Sutton & Barto, 2018). En tareas de conducción autónoma, los agentes deben percibir su entorno, tomar decisiones en tiempo real y adaptarse a condiciones cambiantes, lo que implica una complejidad significativa a nivel de aprendizaje y control. Aunque los algoritmos de aprendizaje por refuerzo han mostrado resultados prometedores en entornos controlados (Mnih et al., 2015; Schulman et al., 2015), aún existen desafíos importantes relacionados con su desempeño, estabilidad y capacidad de generalización en escenarios más complejos y dinámicos.

La literatura actual carece de estudios comparativos exhaustivos que analicen la implementación y adaptación de diferentes enfoques de aprendizaje por refuerzo en agentes móviles inteligentes, considerando tanto las características de hardware como de software de los agentes, así como la influencia de la complejidad del entorno (Kober et al., 2013; Lillicrap et al., 2016). Esta limitación dificulta la identificación de las estrategias más efectivas y limita el avance hacia soluciones robustas y eficientes que puedan trasladarse a aplicaciones reales. Por lo tanto, se requiere un análisis sistemático de diferentes métodos de aprendizaje por refuerzo aplicados a tareas de conducción autónoma, evaluando su desempeño en entornos de simulación con condiciones y obstáculos cambiantes.

1.4 Justificación

El estudio y comparación de enfoques de aprendizaje por refuerzo en el contexto de agentes móviles inteligentes resulta esencial para el avance de la conducción autónoma, una tecnología con alto potencial de impacto en la movilidad, la industria y la robótica de servicios (Kendall et al., 2019; Silver et al., 2016). La optimización del desempeño de los agentes autónomos no solo contribuye a mejorar la seguridad y la eficiencia de los sistemas de transporte, sino que también permite explorar nuevas aplicaciones en entornos complejos y dinámicos.

Al analizar y comparar diferentes métodos de aprendizaje por refuerzo, se pueden identificar las fortalezas y limitaciones de cada enfoque, además de proponer adaptaciones que respondan mejor a los desafíos de entornos reales (Mnih et al., 2015; Schulman et al., 2017). Asimismo, el desarrollo de entornos de simulación con métricas objetivas facilita una evaluación rigurosa y reproducible, lo que favorece la transferencia de conocimiento tanto a la academia como a la industria (Kober et al., 2013). Este trabajo permitirá sentar las bases para futuras investigaciones y contribuirá al desarrollo de agentes móviles inteligentes cada vez más autónomos y adaptativos.

1.5 Objetivos

1.5.1 Objetivo general

Analizar e implementar diferentes algoritmos de aprendizaje por refuerzo en agentes móviles inteligentes, con el propósito de optimizar su desempeño en tareas de conducción autónoma en entornos dinámicos.

1.5.2 Objetivos particulares

Los siguientes objetivos particulares pretenden dar cumplimiento al objetivo general

antes propuesto:

1. Investigar y comprender a fondo los fundamentos teóricos de los algoritmos de aprendizaje por refuerzo más utilizados en el ámbito de agentes móviles inteligentes, como Q-learning, Deep Q-Networks (DQN) y algoritmos basados en políticas.
2. Explorar y evaluar las capacidades de agentes móviles inteligentes, tanto en sus aspectos de hardware como de software, para determinar cómo los algoritmos de aprendizaje por refuerzo pueden aprovechar estas características para mejorar su desempeño.
3. Diseñar y configurar entornos de simulación y pruebas que permitan experimentar con diferentes algoritmos de aprendizaje por refuerzo, considerando factores como la complejidad del entorno, obstáculos y condiciones cambiantes.
4. Implementar y adaptar diferentes algoritmos de aprendizaje por refuerzo en agentes móviles inteligentes para tareas como el seguimiento de trayectorias y la adaptación a entornos dinámicos.
5. Analizar y comparar el desempeño de los algoritmos implementados mediante métricas clave como precisión, eficiencia, estabilidad y capacidad de respuesta, con el objetivo de identificar las técnicas más efectivas en el contexto de conducción autónoma.

1.6 Hipótesis

La implementación y adaptación de diferentes enfoques de aprendizaje por refuerzo en agentes móviles inteligentes permite optimizar su desempeño en tareas de conducción autónoma.

1.6.1 Hipótesis nula

La implementación y adaptación de diferentes enfoques de aprendizaje por refuerzo en agentes móviles inteligentes NO permite optimizar su desempeño en tareas de conducción autónoma.

1.7 Metodología

Para cumplir con los objetivos del proyecto, se seguirá un proceso científico-metodológico que incluye las siguientes etapas:

Figura 1. Ciclo de la metodología del proyecto

1.7.1 Revisión de literatura y documentación técnica

- Investigar y analizar la literatura relacionada con los algoritmos de aprendizaje por refuerzo, como Q-learning, Deep Q-Networks (DQN), y algoritmos basados en políticas, enfocados en su aplicación a agentes móviles inteligentes.

- Estudiar documentación técnica sobre agentes móviles, explorando casos de uso y ejemplos de implementación previos que utilicen aprendizaje por refuerzo en tareas de conducción autónoma.

1.7.2 Diseño del entorno de simulación y pruebas

- Diseñar y configurar entornos de simulación que permitan evaluar el desempeño de diferentes algoritmos de aprendizaje por refuerzo en agentes móviles.

- Considerar factores como la complejidad de los circuitos, la presencia de obstáculos, las condiciones dinámicas del entorno (como iluminación o clima) y la seguridad para el agente durante las pruebas.

1.7.3 Desarrollo y adaptación de algoritmos de aprendizaje por refuerzo

- Implementar y adaptar diferentes algoritmos de aprendizaje por refuerzo, incluyendo enfoques basados en valor (Q-learning, DQN) y políticas, para resolver problemas de conducción autónoma en agentes móviles inteligentes.
- Diseñar estrategias de entrenamiento que permitan optimizar el desempeño del agente en tareas específicas, como el seguimiento de trayectorias y la adaptación a cambios en el entorno.

1.7.4 Implementación y pruebas experimentales

- Realizar pruebas experimentales en los entornos diseñados para evaluar el comportamiento y desempeño de los algoritmos desarrollados.
- Medir métricas clave como la precisión en el seguimiento de trayectorias, la capacidad de respuesta ante cambios en el entorno, la eficiencia y la estabilidad del sistema.

1.7.5 Análisis y evaluación de resultados

- Analizar los resultados obtenidos en las pruebas para identificar los comportamientos más efectivos de los algoritmos bajo diversas condiciones.
- Comparar el desempeño de los algoritmos en términos de eficacia y eficiencia, destacando sus fortalezas, limitaciones y posibles áreas de mejora.
- Generar conclusiones que permitan seleccionar los algoritmos más adecuados para la conducción autónoma de agentes móviles en entornos dinámicos.

1.8 Conclusión del capítulo

Este capítulo presentó el contexto y fundamentos del aprendizaje por refuerzo aplicado a agentes móviles inteligentes, destacando su potencial en la conducción autónoma. Se expuso la problemática actual, los objetivos de la investigación y la metodología a seguir. Con ello, se establece una base sólida para analizar e implementar algoritmos que mejoren el desempeño de agentes autónomos en entornos dinámicos.

2. Aprendizaje automático

En este capítulo se presentan conceptos fundamentales de la Inteligencia Artificial (IA) y los tipos de aprendizaje que la componen. Se describen algoritmos de Aprendizaje Automático que han demostrado buenos resultados en tareas de percepción, planificación de trayectorias y control de vehículos autónomos. De acuerdo con estudios recientes, los modelos matemáticos desarrollados con estos algoritmos permiten que los vehículos interpreten su entorno mediante sensores, cámaras y radares, extraigan patrones de datos multimodales y tomen decisiones en tiempo real. Aun así, enfrentan desafíos como la obtención de datos etiquetados de gran escala, las limitaciones de hardware para procesamiento rápido, el manejo de situaciones imprevistas, y cuestiones éticas sobre responsabilidad y seguridad en la conducción autónoma (Grigorescu et al., 2019; Petryshyn et al., 2024).

2.1 Características principales

La Inteligencia Artificial es una disciplina de la informática que busca emular funciones cognitivas humanas mediante algoritmos, hardware y software avanzados. Dentro de la IA, el Aprendizaje Automático (Machine Learning, ML) es un subconjunto que permite que los sistemas aprendan de datos sin necesidad de programar explícitamente todas las reglas. En el ámbito de los vehículos autónomos, el ML se aplica en visión computacional, fusión de sensores, predicción del tráfico y control de maniobras. Los principales enfoques son el aprendizaje supervisado y el no supervisado (Grigorescu et al., 2019; Valverde, Moutinho & Zacchi, 2025).

2.2 Aprendizaje Supervisado

En el aprendizaje supervisado, los modelos se entrenan usando ejemplos con etiquetas previamente asignadas, para que puedan reconocer objetos, señales, peatones u obstáculos en el entorno. El objetivo es minimizar el error de clasificación o predicción, ajustando una función de coste basada en esas etiquetas (por ejemplo detectar señales, carriles, peatones). Este enfoque permite alta precisión cuando los datos son abundantes y de buena calidad (Valverde, Moutinho & Zacchi, 2025).

2.3 Aprendizaje no Supervisado

El aprendizaje no supervisado se utiliza cuando los datos no tienen etiquetas, permitiendo descubrir estructuras latentes o patrones inherentes. En el contexto de los

vehículos autónomos, este enfoque se aplica en la segmentación de escenas, la detección de anomalías en el tráfico y la agrupación de comportamientos similares de otros conductores. Estos algoritmos forman clusters en función de similitudes, ayudando a mejorar la percepción en entornos no estructurados (Grigorescu et al., 2019; Ma et al., 2020).

2.4 Aprendizaje por Refuerzo

En el aprendizaje por refuerzo (RL) hay un agente (la máquina que aprende) que interactúa con un medio ambiente (el mundo en el que se mueve), toma decisiones y recibe retroalimentación en forma de recompensas (la respuesta a sus acciones) (ver figura 2). El objetivo del agente es descubrir la mejor manera de actuar para obtener la mayor recompensa a lo largo del tiempo. Después de realizar una acción, el agente recibe información sobre si esa acción fue buena (recompensa positiva), mala (recompensa negativa), o neutral. A diferencia del aprendizaje donde te dicen qué hacer (supervisado), aquí el agente aprende por sí mismo, sin necesidad de ejemplos específicos.

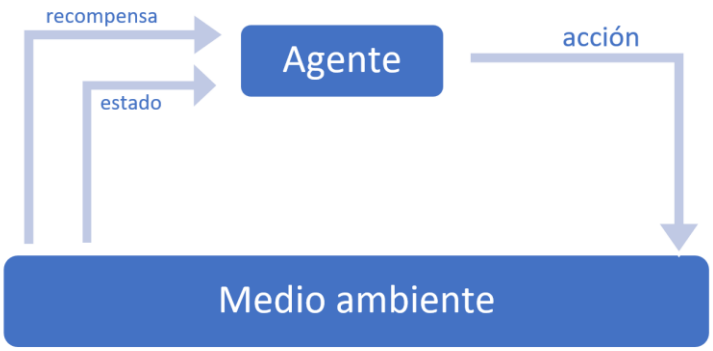

Figura 2. Aprendizaje por refuerzo

2.4.1 Componentes principales

Agente: Es la entidad que toma decisiones, este decide qué acciones llevar a cabo en el medio ambiente que se encuentra para así intentar llegar a sus metas. Las acciones pueden ser movimientos físicos, decisiones lógicas o cualquier otra cosa que pueda afectar su estado. No podemos decir que tenemos un agente si este no se encuentra en un contexto. Su capacidad más destacada es la de aprender mediante la retroalimento y así mejorar su desempeño, ya que en esto se basa el RL.

Medio ambiente: Es el entorno en el que el agente decide las acciones a realizar,

este se puede visualizar como un escenario donde ocurre todo el proceso de aprendizaje. Puede ser el mundo físico, una simulación digital, o cualquier contexto. El entorno afecta directamente al agente, en el se presentan desafíos, oportunidades, obstáculos o cualquier otro elemento que influya en cómo el agente debe actuar para alcanzar sus metas.

Acción: Son los movimientos que el agente elige realizar en un momento determinado. Puede ser un paso, una elección, un movimiento físico, o cualquier otra decisión tomada por el agente. Cada uno de estos movimientos tiene un efecto en el agente con respecto al medio ambiente. Puede alterar el estado del medio ambiente o modificar cualquiera de las condiciones del agente como su posición.

Estado: Es una representación de la situación o configuración particular en la que se encuentra el medio ambiente en un momento especifico. Este contiene los datos que necesita el agente para la toma de decisiones como puede ser su ubicación, presencia de objetos, condiciones del medio ambiente o cualquier otra información relevante. El estado cambia a medida que el agente realiza una acción.

Recompensa: Es una señal que el agente recibe como retroalimentación después de realizar una acción, la cual puede ser positiva (premio), negativa (castigo) o neutra. El agente con estas señales ajusta su comportamiento para incrementar la acumulación de recompensas positivas que contribuye al aprendizaje y así mejorar su desempeño.

2.4.2 Proceso de decisión de Márkov (MDP)

El Proceso de Decisión de Markov, o MDP por sus siglas en inglés (Markov Decision Process), es marco matemático que se utiliza en inteligencia artificial y teoría de control. Un MDP nos ayuda a modelar situaciones en las que debemos tomar decisiones a lo largo del tiempo, en un medio ambiente que puede cambiar de manera incierta, pero donde nuestras decisiones influyen en los resultados futuros. Las decisiones futuras solo dependen del estado actual, el futuro es independiente del pasado. Los agentes de aprendizaje por refuerzo no precisan conocer el MDP específico de un problema para desarrollar comportamientos sólidos, pero es fundamental que tengan un entendimiento general de los MDP ya que comúnmente se diseñan los agentes con la suposición de que hay un MDP operando detrás de escena.

Los componentes de un MDP se refieren a los estados que el agente puede experimentar, las acciones que puede realizar en esos estados, las recompensas asociadas con esas acciones y estados, y las probabilidades de transición que determinan cómo cambian los estados cuando se toman acciones. Abordaremos en este subtema un problema denominado frozen lake o lago congelado (FL), en el cual

construiremos un Proceso de Decisión de Markov (MDP).

El FL es una cuadrícula de 4 × 4 (tiene 16 celdas, identificadas como 0-15). El agente aparece en la celda de INICIO al comienzo de cada nuevo episodio. Al llegar a la celda de META, recibe una recompensa de +1; de lo contrario, la recompensa es 0. Debido a que la superficie está resbaladiza, el agente se mueve solo un tercio del tiempo según la intención. Los otros dos tercios se dividen equitativamente en direcciones ortogonales. Por ejemplo, si el agente elige moverse hacia abajo, hay un 33.3% de probabilidad de que se mueva hacia abajo, un 33.3% de probabilidad de que se mueva hacia la izquierda y un 33.3% de probabilidad de que se mueva hacia la derecha. Hay una cerca alrededor del lago, por lo que, si el agente intenta salir del mundo de la cuadrícula, volverá a la celda desde la cual intentó moverse. Hay cuatro agujeros en el lago. Si el agente cae en uno de estos agujeros, el juego termina.

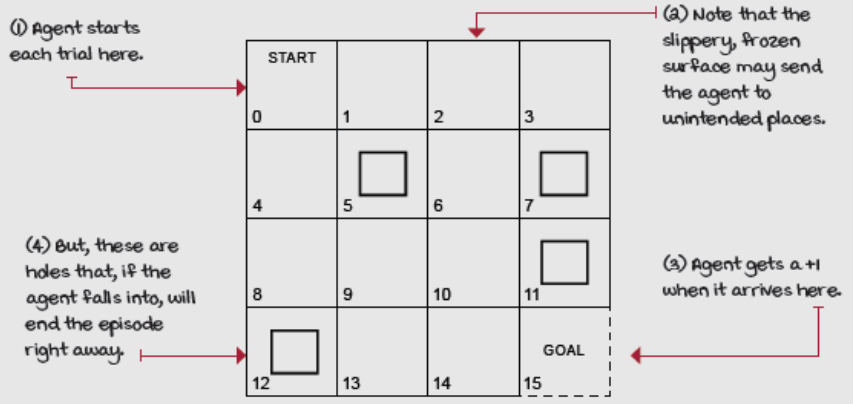

Figura 3. El ambiente frozen lake (FL)

2.4.3 Estados

Un estado (S) es una configuración única del problema, y el espacio de estados es el conjunto de todas las posibles configuraciones. Mientras que el espacio de estados puede ser infinito, el conjunto de variables que componen un estado individual siempre es finito y de tamaño constante.

En el lago congelado (FL), asignamos los identificadores de 1 a 15, de izquierda a derecha y de arriba hacia abajo.

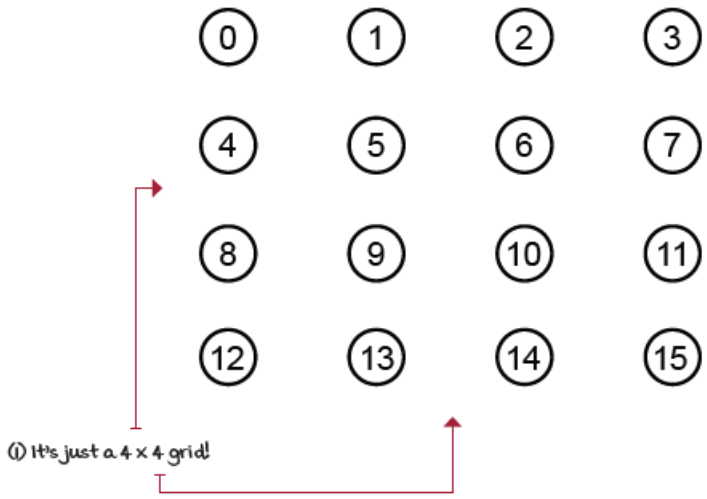

Figura 4. Los estados en el lago congelado (FL) contienen una única variable que indica el identificador de la celda en la que se encuentra el agente en cualquier paso de tiempo dado

En los MDP, la probabilidad de pasar al siguiente estado después de tomar una acción desde el estado actual no depende de la historia de interacciones. Esta propiedad, conocida como la propiedad de Markov, establece que la probabilidad de transición entre dos estados dados y una acción específica es la misma en diferentes ocasiones, independientemente de los estados o acciones previas que se hayan experimentado.

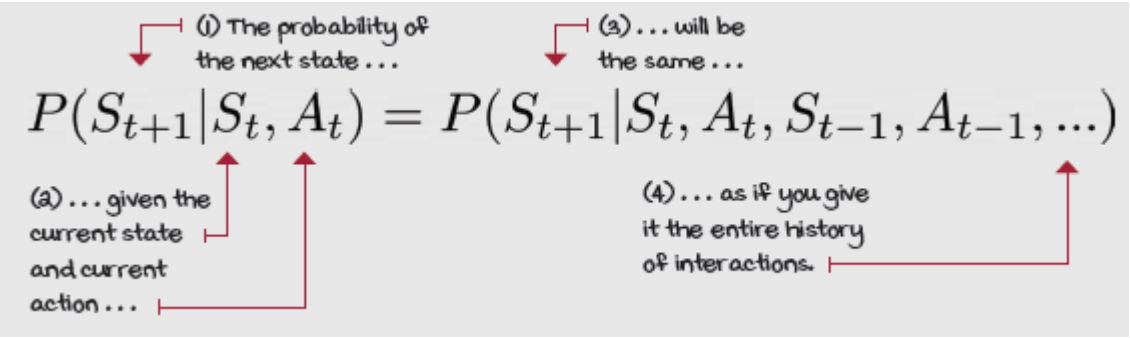

Figura 5. Propiedad de Markov

En el FL, hay un único estado de inicio (estado 0) y cinco estados terminales (o cinco

estados que conducen a un mismo estado terminal, dependiendo de la preferencia). Para evitar confusiones, se utiliza la convención de múltiples estados terminales (5, 7, 11, 12 y 15) en las ilustraciones y el código; sin embargo, cada estado terminal es considerado como un estado terminal individual.

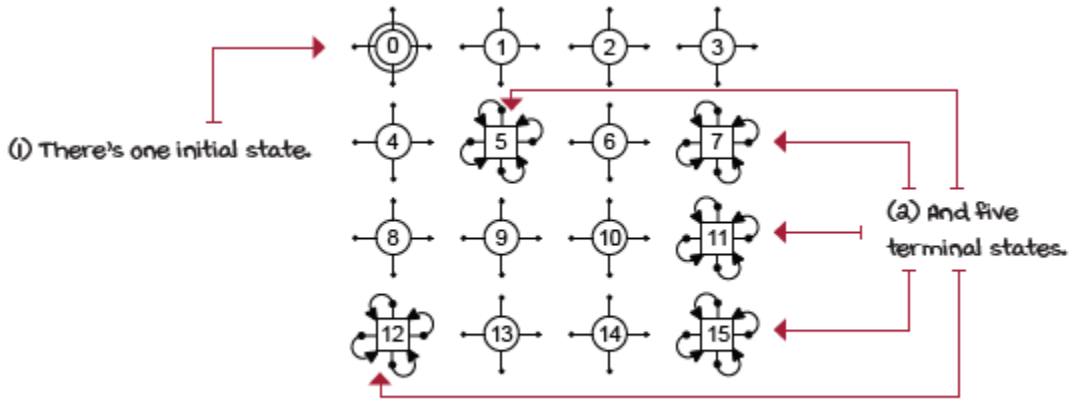

Figura 6. Estados en el FL

2.4.4 Acciones.

Los MDPs ofrecen un conjunto de acciones A que varía según el estado. Esto significa que puede haber acciones que no sean válidas en ciertos estados; en realidad, A es una función que recibe un estado como argumento, es decir, A(s).

En FL, las acciones son elementos individuales que indican la dirección en la que el agente intentará moverse, específicamente, hay cuatro opciones de acciones en todos los estados: hacia arriba, hacia abajo, hacia la derecha o hacia la izquierda. Cada acción se representa mediante una variable única, y el espacio de acciones tiene un tamaño de cuatro.

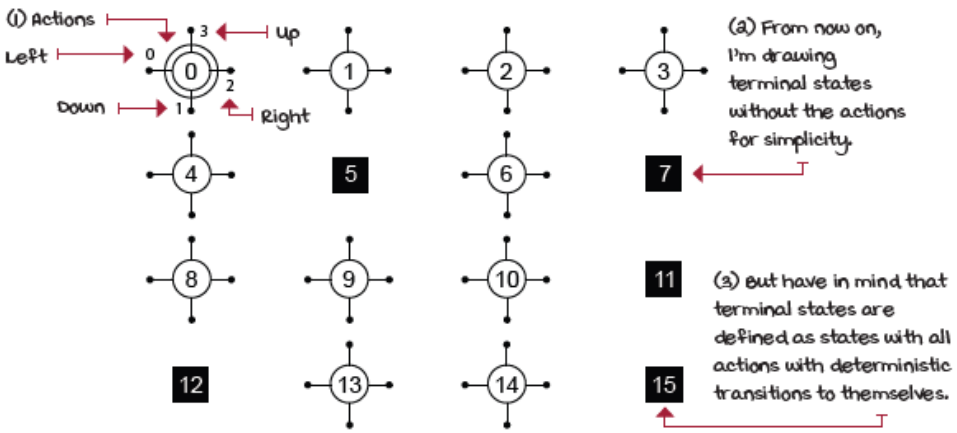

Figura 7. Acciones en el FL

2.4.5 Función de Transición.

La manera en que el entorno responde a las acciones se conoce como las probabilidades de transición de estado, o simplemente la función de transición, denotada como T(s, a, s'). La función de transición T asigna una probabilidad a una tupla de transición s, a, s'; es decir, al proporcionar un estado s, una acción a y un estado siguiente s', devuelve la probabilidad correspondiente de transición de s a s' al tomar la acción a. También se puede expresar como T(s, a) y devolver un diccionario que relaciona los siguientes estados con sus respectivas probabilidades de transición.

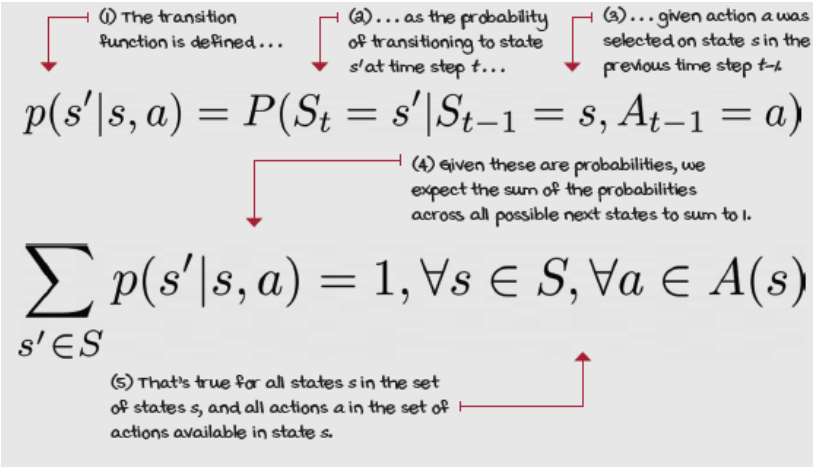

Figura 8. La función de transición

En el entorno FL, existe una probabilidad del 33.3% de realizar la transición a la celda objetivo deseada, mientras que hay una probabilidad del 66.6% de realizar la transición en direcciones ortogonales. Además, existe la posibilidad de retroceder al estado anterior si este se encuentra junto a una pared.

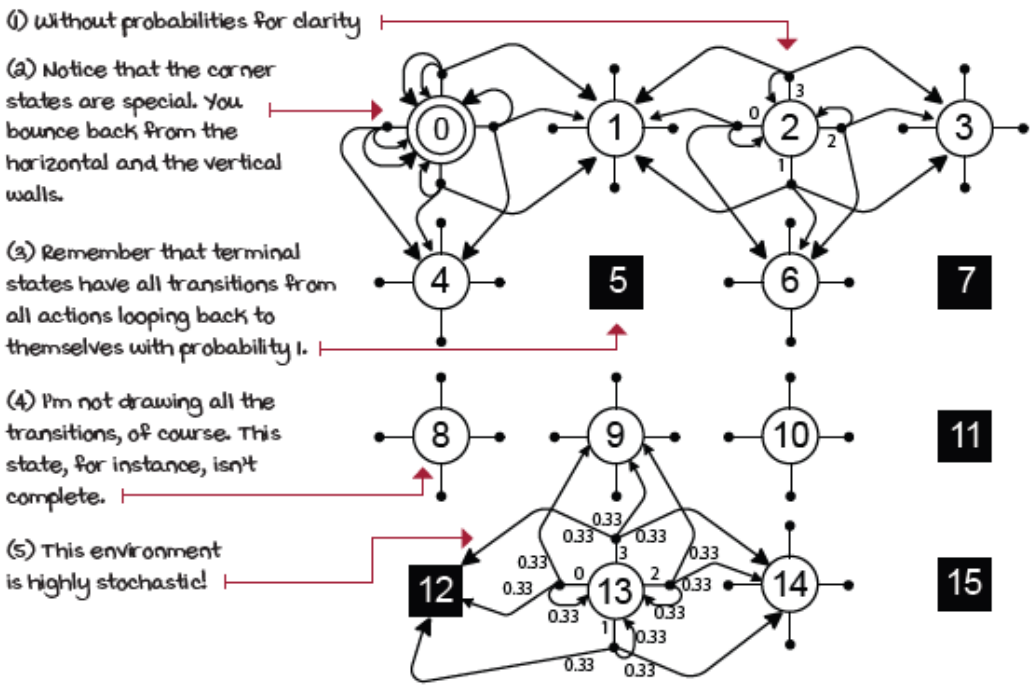

Figura 9. La función de transición en el entorno FL

2.4.6 Función de Recompensa.

La función de recompensa R asigna un valor numérico a una secuencia de transición s, a, s', representándola con un escalar. Esta función proporciona una indicación numérica de la calidad de las transiciones. Cuando la señal es positiva, podemos interpretar la recompensa como un beneficio o premio. La mayoría de los problemas presentan al menos una señal positiva, como ganar una partida de ajedrez o alcanzar el destino deseado.

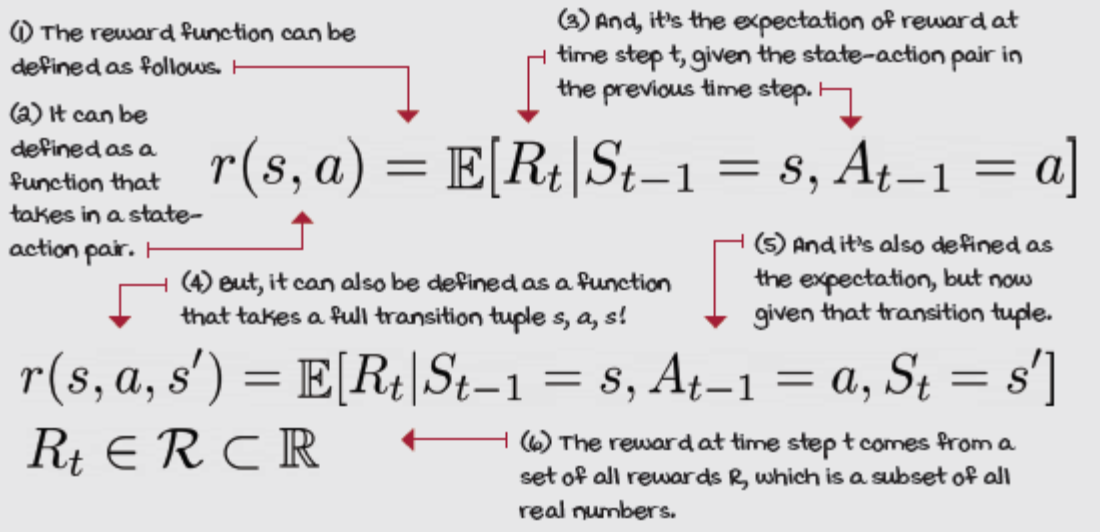

Figura 10. La función de recompensa

En el entorno FL, se otorga una recompensa de +1 al alcanzar el estado 15, mientras que en todos los demás casos la recompensa es de 0. Solo hay tres formas de alcanzar el estado 15. Si se elige la acción Derecha en el estado 14, el agente tiene una probabilidad del 33.3% de llegar allí (33.3% al estado 10 y 33.3% de regreso al 14). No obstante, seleccionar la acción Arriba y la acción Abajo desde el estado 14 también conducirán al agente al estado 15 de manera no intencionada, con una probabilidad del 33.3% para cada acción.

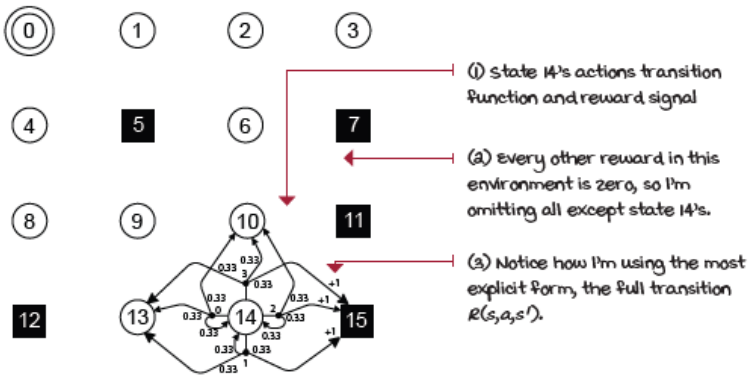

Figura 11. Recompensa asignada a estados con transiciones que no tienen una recompensa de cero

2.4.7 Horizonte.

Podemos también representar la noción del tiempo en los MDPs. Un paso temporal, también conocido como época, ciclo, iteración o incluso interacción, es un reloj global que sincroniza a todas las partes y divide el tiempo en unidades discretas. Tener un reloj da lugar a dos tipos de tareas posibles. Una tarea episódica es aquella en la que existe un número finito de pasos temporales, ya sea porque el reloj se detiene o porque el agente alcanza un estado terminal. Por otro lado, están las tareas continuas, las cuales no tienen un final definido; no hay estados terminales, lo que implica que hay un número infinito de pasos temporales. En este tipo de tarea, es necesario detener al agente manualmente.

El entorno FL se clasifica como una tarea episódica debido a la presencia de estados terminales que representan objetivos claros y estados de fracaso. En FL, el horizonte de planificación es indefinido, lo que significa que el agente planifica para un número infinito de pasos, pero las interacciones pueden detenerse en cualquier momento.

2.4.8 Descuento.

Debido a la posibilidad de tener secuencias infinitas de pasos temporales en tareas de horizonte infinito, es necesario aplicar un descuento al valor de las recompensas a lo largo del tiempo. Esto implica comunicar al agente que recibir recompensas de +1 es más beneficioso cuanto antes. Usualmente utilizamos un valor real positivo menor a uno para aplicar un descuento exponencial al valor de las recompensas futuras. Mientras más lejana sea la recompensa en el futuro, menos valor tendrá en el presente. Este número se conoce como factor de descuento, o gamma.

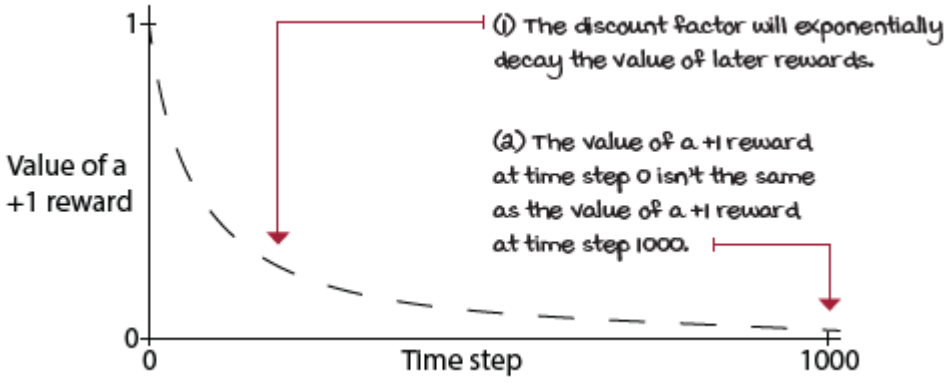

Figura 12. El efecto del factor de descuento y el tiempo en el valor de las recompensas

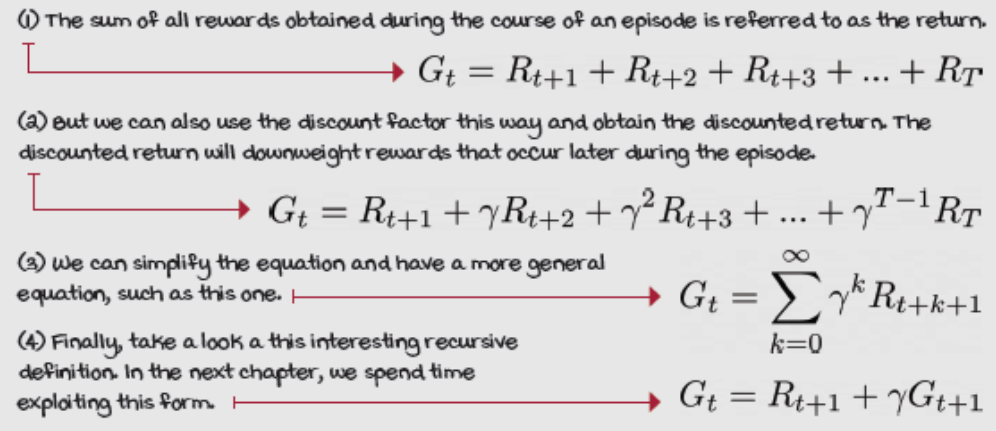

Figura 13. El factor de descuento (gamma)

2.4.9 Objetivo: Política optima.

Inicialmente, parece que el objetivo del agente es determinar una serie de acciones que maximicen el rendimiento: la suma de recompensas (descontadas o no descontadas, según el valor de gamma) durante un episodio o durante toda la vida útil del agente, dependiendo de la tarea.

La dificultad con los planes radica en que no consideran la aleatoriedad en los entornos, y cuando un entorno es estocástico; las acciones realizadas no siempre producirán los resultados deseados. Lo que el agente debe crear se denomina política. Las políticas son planes generales; abarcan todos los estados posibles. Es necesario planificar para cada estado potencial.

2.5 Conclusión del capítulo.

El Aprendizaje Automático es esencial en el desarrollo de vehículos autónomos, ya que combina enfoques que cubren distintas necesidades. El aprendizaje supervisado permite una percepción precisa del entorno mediante datos etiquetados, mientras que el no supervisado ayuda a detectar patrones ocultos y situaciones inesperadas. Por su parte, el aprendizaje por refuerzo aporta flexibilidad en la toma de decisiones dinámicas y en entornos inciertos. En conjunto, estos métodos ofrecen una base sólida para la conducción autónoma, aunque persisten retos relacionados con la calidad de los datos, el procesamiento en tiempo real y las implicaciones éticas de la seguridad vial.

3. Algoritmos de Aprendizaje por Refuerzo

El Aprendizaje Profundo (DL por sus siglas en inglés) es un área dentro del campo de la inteligencia artificial, en el que destaca el uso de Redes Neuronales Artificiales para la creación de modelos de aprendizaje automático. Que gracias a la capacidad de las redes neuronales profundas ha sido posible resolver problemas con tareas complejas con grandes volúmenes de datos que antes se consideraban imposibles de resolver.

La integración del Aprendizaje por Refuerzo con el Aprendizaje Profundo ha generado una disciplina conocida como Aprendizaje por Refuerzo Profundo (Deep Reinforcement Learning, DRL). Esto ha permitido superar muchas limitaciones de los enfoques clásicos, al aprovechar el poder de las redes neuronales profundas dentro del marco de RL. Esto ha abierto la puerta a aplicaciones de gran complejidad, como el control de robots autónomos, juegos competitivos y sistemas de recomendación personalizados.

En este capítulo se presentan las características principales del DRL y los algoritmos Deep Q-Network (DQN) y Actor-Critic (AC), en un entorno de simulación de un circuito de carreras, donde un agente representado por un carro debe aprender a desplazarse eficientemente por la pista. El agente cuenta con diferentes acciones disponibles y debe ajustar su orientación y posición en función de la retroalimentación recibida. El sistema de recompensas está diseñado para incentivar que el vehículo permanezca lo más cerca posible del centro de la pista, penalizando las desviaciones y recompensando la trayectoria óptima.

3.1 Algoritmo Q-learning

Q-Learning es un algoritmo de aprendizaje por refuerzo basado en valores que se utiliza para encontrar la política óptima (tabla Q) de selección de acciones mediante la Función Q. Esto le permite al agente preguntar, ¿Cuál es la mejor acción?

Tabla Q: Una tabla Q es una estructura que almacena las combinaciones de estados y acciones. Usamos el algoritmo Q-learning para actualizar sus valores.

Función Q (*Q(s,a)*): La función Q usa la ecuación de Bellman para calcular qué tan buena es una acción en un estado determinado. Esta ecuación ayuda al agente a aprender de forma más eficiente, facilitando el cálculo de los valores que indican la mejor decisión en cada situación.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + (\gamma\, maxQ(s',a')) - Q(s,a)] \quad \text{(ec. 1)}$$

- α es la tasa de aprendizaje.
- γ es el factor de descuento.
- r es la recompensa inmediata.
- $maxQ(s', a')$ es el valor máximo de la función Q dado el nuevo estado.

Abordaremos en este subtema un problema denominado el entorno de Norberto. Este ejemplo ilustra cómo un agente puede aprender una política óptima con Q-learning mediante la interacción con un entorno, utilizando recompensas para guiar su comportamiento.

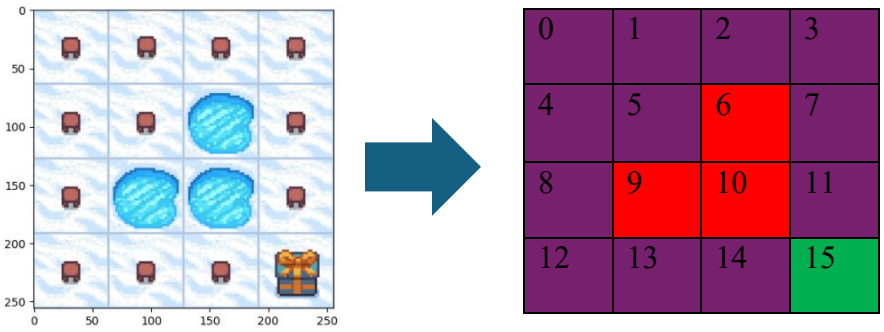

Figura 14. El entorno Norberto

El entorno de Norberto (figura 14) es una cuadrícula de 4 × 4 (tiene 16 celdas, identificadas como 0-15). El agente (figura 15) aparece en cualquier celda AZUL de forma aleatoria al comienzo de cada nuevo episodio. El objetivo es alcanzar la celda VERDE, al llegar recibe una recompensa de +10; de lo contrario, la recompensa es -1 (celdas AZUL). Hay tres agujeros en el entorno (celdas ROJO). Si el agente cae en uno de estos agujeros, el juego termina y recibe una penalización de -10. Una de las limitaciones que tiene Norberto es que solo puede realizar 10 movimientos para llegar

a la celda VERDE. Las acciones posibles son desplazar el agente hacia arriba, izquierda, abajo o derecha, una celda. Hay una cerca alrededor del entorno, por lo que, los movimientos para salir del mundo de la cuadrícula no son permitidas y permanecerá en la celda desde la cual intentó moverse.

Figura 15. El agente Norberto

3.1.1 En búsqueda de la política optima, Inicialización de la tabla Q.

Primero se construye una tabla Q (Tabla 1). En esta hay un número n de columnas, donde n es igual al número de acciones (a). Norberto puede moverse hacia arriba, abajo, izquierda o derecha. Si intenta salir de la cuadrícula, permanece en la celda actual. También hay un número m de filas, donde m es igual al número de estados (s). Asignamos los identificadores de 0 a 15, de izquierda a derecha y de arriba hacia abajo. Al principio inicializaremos todos los valores Q(s,a) en 0, como punto de partida, esto indica que el modelo aún no tiene conocimiento.

Tabla 1. Representación de la inicialización de la tabla Q

Estado \ Acción	Izquierda	Abajo	Derecha	Arriba
Estado 0	0	0	0	0
Estado 1	0	0	0	0
Estado 2	0	0	0	0
Estado 3	0	0	0	0
Estado 4	0	0	0	0
Estado 5	0	0	0	0
Estado 6	0	0	0	0

Estado 7	0	0	0	0
Estado 8	0	0	0	0
Estado 9	0	0	0	0
Estado 10	0	0	0	0
Estado 11	0	0	0	0
Estado 12	0	0	0	0
Estado 13	0	0	0	0
Estado 14	0	0	0	0
Estado 15	0	0	0	0

Una matriz en cero puede ser indicativo de que el modelo no ha aprendido nada aun, y esto se puede observar también a través de los episodios como van cambiando los valores, aunque puede ser que algunos permanezcan en cero, avanzado el entrenamiento.

3.1.2 Elegir y realizar una acción (Exploración o explotación)

En el entorno de Norberto, cualquier celda AZUL puede ser el estado inicial y cuatro estados terminales tanto las celdas ROJO y la celda VERDE.

Una de las características distintivas del aprendizaje por refuerzo (RL) es que el agente tiene control total sobre las decisiones que toma en su interacción con el entorno. Por ello, resulta fundamental definir un mecanismo para que el agente tome dichas decisiones, comenzando con un enfoque simple que pueda ser desarrollado y perfeccionado posteriormente.

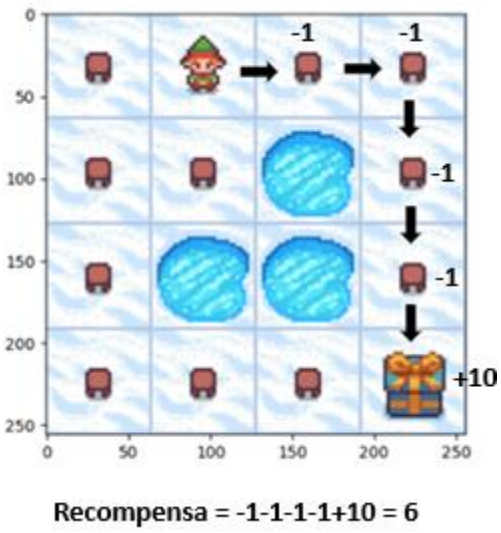

Figura 16. Las mejores acciones para Norberto

El acumulado de recompensas de Norberto, se espera que sea el mayor posible, pero con el menor número de acciones. Para identificar la mejor acción, es imprescindible analizar las recompensas obtenidas de todas las posibles opciones en múltiples ocasiones. Al mismo tiempo, es crucial que el agente utilice las acciones con mejores resultados conocidos para maximizar las recompensas inmediatas. Este equilibrio define el compromiso entre la exploración de nuevas opciones y la explotación del conocimiento adquirido (figura 16).

- **Exploración aleatoria**: Es una estrategia sencilla en la que el agente selecciona acciones completamente al azar (figura 17).

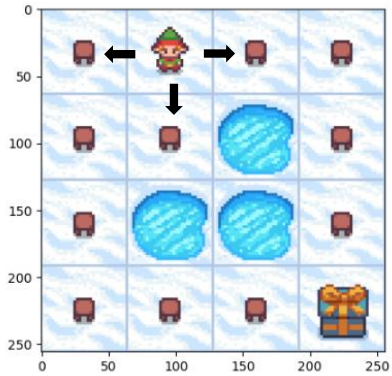

Figura 17. El agente Norberto en un estado inicial aleatorio y sus acciones posibles

Al utilizar este tipo de estrategia corremos un alto riesgo de que el modelo tarde demasiado tiempo en descubrir la política optima o incluso que no la descubra, ya que, aunque el agente esté aprendiendo que acciones le da mayor recompensa, al ser un proceso totalmente aleatorio no estamos explotando el conocimiento ya adquirido.

- **Épsilon - Greedy**; Permite al agente explorar nuevas acciones con una cierta probabilidad (épsilon) y explotar acciones ya aprendidas con la probabilidad restante (1 – épsilon).

$$\pi(a \mid s) = \begin{cases} arg\ \max Q(s,a), & con\ probabilidad\ 1-epsilon\ (explotación) \\ una\ acción\ aleatoria, & con\ probabiblidad\ epsilon\ (exploración) \end{cases} \text{(ec. 2)}$$

- $\pi(a|s)$: Es la probabilidad de seleccionar la acción *a* en el estado *s*.
- épsilon: Es la probabilidad de exploración (un valor entre 0 y 1).
- $\arg\max Q(s,a)$: Es la acción que maximiza la función Q en el estado s.

Explotación: Con probabilidad 1 – épsilon, el agente elige la acción optima según la función Q(s,a), es decir, la acción que maximiza la recompensa esperada.

Exploración: Con probabilidad épsilon, el agente elige una acción entre todas las posibles.

Un ejemplo sería un épsilon = 0.2, esto se traduce a un 20% de probabilidad de que el agente realice una acción aleatoria lo que se conoce como exploración, por lo tanto,

se tiene 80% (1 – 0.2 = 0.8) de probabilidad de que explote el conocimiento ya adquirido.

En la explotación el agente utilizara los valores aprendidos en la exploración mediante la función Q.

Tabla 2. Valores Q para el estado 1

Estado \ Acción	Izquierda	Abajo	Derecha	Arriba
Estado 1	2.0	2.5	3.0	0

Aquí entra el termino $\arg\max Q(s,a)$, el agente se encuentra en el estado s = 1 y se observa en la tabla 2 que la acción que maximiza la recompensa esperada es “Derecha” (figura 18).

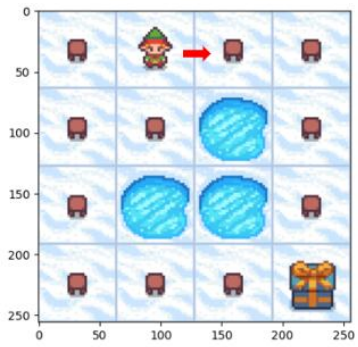

Figura 18. El agente Norberto realizando la acción Derecha

3.1.3 Obtener recompensa y actualizar la tabla Q

Ahora que se ha tomado una acción, el agente se encontrara en un estado diferente al anterior por lo que se debe de obtener un resultado, esta es la recompensa que se obtiene con los parámetros definidos en el entorno de Norberto, casilla azul (-1), roja

(-10) y verde (+10) como se muestra en la figura 19.

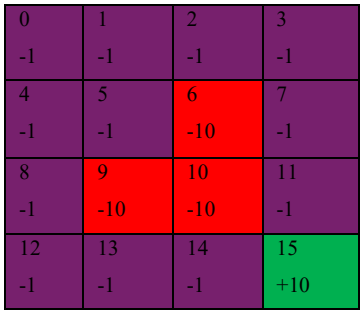

Figura 19. Estado – Recompensa del entorno de Norberto

Siguiendo el ejemplo de Norberto en una posición inicial Estado 1, realizando la acción "Derecha", se observa que se obtiene la nueva posición Estado 2. Este resultado nos arroja una recompensa de -1. Para que nuestro modelo aprenda de estos resultados aquí es cuando entra la Función Q antes mencionada.

$$Q'(s,a) \leftarrow Q(s,a) + \alpha[r + (\gamma\, maxQ(s',a')) - Q(s,a)] \quad \text{(ec. 3)}$$

Donde:

$Q(s,a) = 0$ $\quad ya\ que\ nuestro\ modelo\ aun\ no\ tiene\ conociemiento$

$\alpha = 0.1$ $\quad la\ tasa\ de\ aprendizaje$

$r = -1$ $\quad la\ recompensa\ del\ Estado\ 2$

$maxQ(s',a') = 0$ $\quad valor\ optimo\ esperado\ de\ las\ posibles\ acciones\ del\ Estado\ 2$

$\gamma = 0.5$ $\quad la\ tasa\ de\ descuento$

Sustituyendo:

$$Q'(s,a) \leftarrow 0 + 0.1[(-1) + (0.5\ (0)) - 0]$$

$$Q'(s,a) = -0.1$$

Este nuevo valor que se ha generado con la función Q es la relación que existirá entre

la posición anterior que es el Estado 1 y la acción que hizo que llegara al Estado 2, la acción “Derecha”. Por lo que se tiene que actualizar la tabla Q, como conocimiento adquirido de esa experiencia (tabla 3). La siguiente ves que se consulte la tabla Q tiene que contener este conocimiento para que pueda seguir adquiriendo más aprendizaje.

Tabla 3. Conocimiento adquirido para el estado 1 registrado en la Tabla Q

Estado \ Acción	Izquierda	Abajo	Derecha	Arriba
Estado 1	0	-0.1	0	0

El proceso de aprendizaje debe de continuar hasta cierto número de episodios, la convergencia del Q-learning ocurre cuando los valores de la tabla Q dejan de cambiar significativamente. Esto dice que el agente ha encontrado una política que maximiza las recompensas. Una forma en la que se puede observar el avance del aprendizaje es graficando las recompensas totales obtenidas a través de cada episodio. En la figura 20 se puede ver cómo va incrementando la recompensa total, pero después de cierta cantidad de episodios el modelo empieza a estabilizarse.

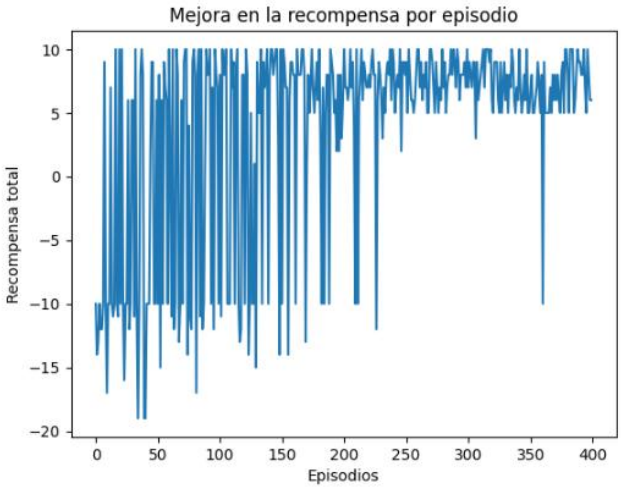

Figura 20. Recompensa obtenida a través de los episodios.

Después de entrenar con Q-learning, la tabla Q final almacena los valores aprendidos para cada combinación de estado y acción (tabla 4), reflejando la calidad de cada decisión según la experiencia acumulada. En cada estado, la mejor acción es aquella con el valor más alto en la tabla, lo que permite al agente tomar decisiones óptimas

sin necesidad de seguir explorando. En esencia, la Q-table final actúa como la memoria del agente, guiando su comportamiento de manera eficiente dentro del entorno.

Tabla 4. Tabla Q final

	Izquierda	Abajo	Derecha	Arriba
Estado 0	0.000000	1.178505	-0.995362	0.000000
Estado 1	-0.846418	-1.008353	-0.624336	0.000000
Estado 2	-0.957609	-9.528987	2.939342	0.000000
Estado 3	-0.612580	6.044698	0.000000	0.000000
Estado 4	0.000000	3.114176	-1.764602	-0.986697
Estado 5	1.619526	-1.900000	-1.000000	-0.979724
Estado 6	0.000000	0.000000	0.000000	0.000000
Estado 7	-2.710000	7.987995	0.000000	0.000000
Estado 8	0.000000	4.579400	-8.332282	-0.814698
Estado 9	0.000000	0.000000	0.000000	0.000000
Estado 10	0.000000	0.000000	0.000000	0.000000
Estado 11	-1.900000	9.999843	0.000000	0.000000
Estado 12	0.000000	0.000000	6.199973	-0.116885
Estado 13	0.204838	0.000000	8.000000	-1.900000
Estado 14	0.449000	0.000000	10.000000	0.000000
Estado 15	0.000000	0.000000	0.000000	0.000000

3.2 Deep Q-Network (DQN)

El algoritmo Deep Q-Network (DQN) es un método de aprendizaje por refuerzo profundo que utiliza redes neuronales para aproximar la función de valor Q, permitiendo a los agentes aprender directamente de datos de alta dimensión como imágenes. En DQN, el agente interactúa con el entorno y almacena sus experiencias en un "replay buffer", desde donde se seleccionan muestras aleatorias para entrenar la red neuronal. Esto ayuda a romper la correlación entre experiencias consecutivas y mejora la estabilidad del aprendizaje. Además, DQN introduce el uso de una red objetivo separada, la cual se actualiza de manera periódica, para evitar oscilaciones e inestabilidad durante el proceso de aprendizaje (figura 21). Gracias a estas innovaciones, DQN logró resultados a nivel humano en varios juegos de Atari (Mnih et al., 2015).

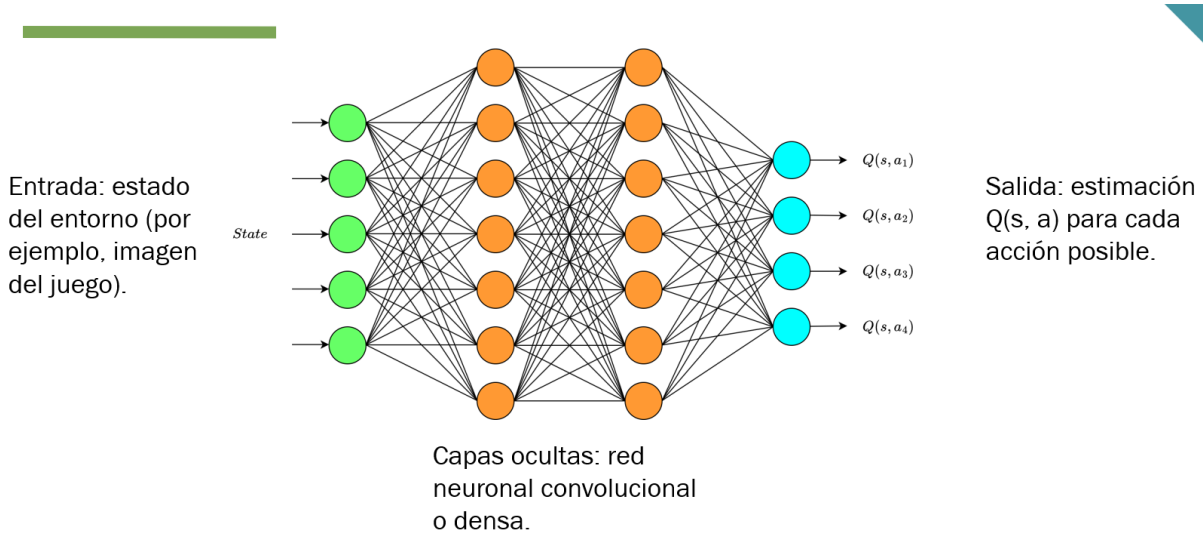

Figura 21. Arquitectura DQN

3.2.1 Q-Learning vs Deep Q-Network.

El algoritmo Deep Q-Learning (DQN) es una extensión del Q-Learning tradicional, pero con una diferencia clave: en lugar de usar una tabla para guardar los valores Q, utiliza una red neuronal (NN) para aproximarlos.

Tabla 5. Diferencias entre Q-Learning y DQN

Q-LEARNING	DEEP Q-LEARNING
• Tablas Q se vuelven inviables en ambientes grandes no continuos.	• Combina Q-learning con redes neuronales profundas.
• No generaliza entre estados similares.	• Usa una red neuronal para aproximar la función Q(s,a)
• Difícil de escalar a tareas complejas (como videojuegos o robótica)	• Permite manejar entornos de mayor dimensiones o tareas mas complejas.

3.2.2 Replay Buffer.

El replay buffer o memoria de repetición es una técnica fundamental en los algoritmos modernos de aprendizaje por refuerzo profundo, especialmente en arquitecturas como DQN. Esta memoria almacena de manera continua las experiencias del agente, registradas como cuádruplas que contienen el estado, la acción tomada, la recompensa recibida y el estado siguiente (state, action, reward, next state).

Posteriormente, durante la fase de entrenamiento, el agente selecciona muestras aleatorias de este buffer para actualizar la red neuronal. Esta técnica, denominada “experience replay”, contribuye a la estabilidad y eficiencia del aprendizaje, ya que permite romper la correlación entre muestras secuenciales y mejora la convergencia del algoritmo al reutilizar experiencias previas (Mnih et al., 2015). Además, el uso del replay buffer ayuda a suavizar las actualizaciones y reduce el riesgo de sobreajuste a transiciones recientes o patrones particulares de exploración.

3.2.3 Target Network

La target network o red objetivo es un componente auxiliar introducido para abordar el problema de inestabilidad durante el entrenamiento de redes neuronales en aprendizaje por refuerzo. En DQN, se emplean dos redes: la red Q principal, que se entrena constantemente, y la target network, que se utiliza exclusivamente para calcular los valores objetivo durante la actualización de la función de valor Q. La target network se mantiene fija durante varios pasos de entrenamiento y solo se actualiza periódicamente copiando los pesos de la red principal. Esto introduce un retraso controlado que ayuda a estabilizar las actualizaciones y reduce la propagación de errores en las estimaciones de los valores Q, lo cual es crucial para evitar que el entrenamiento diverja (Mnih et al., 2015).

3.2.4 Q-Network

La Q-network es una red neuronal profunda diseñada para aproximar la función de valor Q, es decir, una función que estima el valor esperado de la recompensa acumulada al tomar una determinada acción en un estado específico y seguir la política aprendida en adelante. El propósito de esta aproximación es permitir que el agente tome decisiones óptimas en entornos complejos y de alta dimensionalidad, donde las funciones de valor tabulares resultan inviables. Durante el entrenamiento, la Q-network ajusta sus parámetros minimizando el error entre los valores Q predichos y los valores objetivo calculados con la ayuda de la target network. Esta aproximación permite que los agentes aprendan políticas efectivas incluso en tareas con espacios de estados y acciones muy grandes (Sutton & Barto, 2018).

3.2.5 Estrategia Explotación-Exploración

El aprendizaje por refuerzo enfrenta el desafío fundamental de balancear la exploración y la explotación. La exploración implica seleccionar acciones nuevas o menos conocidas con el objetivo de descubrir opciones potencialmente más beneficiosas, mientras que la explotación consiste en elegir las acciones que, según

el conocimiento actual, maximizan la recompensa esperada. Una de las estrategias más utilizadas es la política ε-greedy, donde el agente selecciona una acción aleatoria con probabilidad ε (exploración) y, en caso contrario, elige la acción con el mayor valor Q estimado (explotación). Ajustar correctamente este parámetro es fundamental, ya que una exploración insuficiente puede llevar al agente a políticas subóptimas, mientras que una exploración excesiva puede ralentizar la convergencia del aprendizaje (Sutton & Barto, 2018). Existen variantes más sofisticadas, como el uso de estrategias de "decaying epsilon" o métodos basados en incertidumbre, que buscan adaptar dinámicamente el equilibrio entre exploración y explotación a lo largo del proceso de aprendizaje.

3.2.6 Función de Pérdida

Para entrenar la red, se utiliza una función de pérdida, comúnmente el error cuadrático medio (MSE) entre el valor objetivo (Target network) y el valor Q estimado (Q network).

$$L = \left(r + \gamma \cdot \max_{a'} Q_{\text{target}}(s', a') - Q(s, a)\right)^2 \quad \text{(ec. 4)}$$

3.2.7 Algoritmo DQN

A continuación, se describe el funcionamiento general del algoritmo el cual abordaremos con el entorno de Toyrace mencionado en el Capítulo 3 de este trabajo. Este ejemplo ilustra cómo un agente puede aprender una política óptima mediante el algoritmo DQN, el cual utiliza redes neuronales profundas para estimar la función de valor Q. El agente aprende interactuando con el entorno, utilizando las recompensas recibidas para ajustar progresivamente su comportamiento hacia decisiones que maximicen el rendimiento esperado.

3.2.8 Inicialización de la Q-network y Target-network

Al inicio del entrenamiento, se definen dos redes neuronales con la misma arquitectura: la **Q-network** principal, que es la que se entrenará directamente, y la **target network**, que se utilizará para calcular los valores objetivo al actualizar la Q-network (figura 22). Inicialmente, ambas redes comparten los mismos pesos.

```
Q Network: Sequential(
  (0): Linear(in_features=384, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=128, bias=True)
  (3): ReLU()
  (4): Linear(in_features=128, out_features=3, bias=True)
)
Target Network: Sequential(
  (0): Linear(in_features=384, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=128, bias=True)
  (3): ReLU()
  (4): Linear(in_features=128, out_features=3, bias=True)
)
```

Figura 22. Arquitectura Q-Network y Target-Network

Arquitectura de las Redes:

- **Capa de Entrada Lineal (Fully Connected):**
  - Entrada: 384 estados del entorno Toyrace
  - Salida: 128
  - Función de activación : ReLU
- **Capa oculta Lineal (Fully Connected):**
  - Entrada: 128
  - Salida: 128
  - Función de activación : ReLU
- **Capa Salida (Fully Connected):**
  - Entrada: 128
  - Salida: 3 acciones posibles

La capa de salida no utiliza ninguna función de activación, es decir, la salida es directa. En cuanto a las dimensiones de la red, el valor 384 corresponde al tamaño del vector de entrada, que representa el estado del entorno; el valor 128 indica la cantidad de neuronas ocultas en las dos primeras capas lineales; y el valor 3 corresponde al número de acciones posibles en el entorno, de modo que cada salida de la red representa el valor Q estimado para una de estas acciones.

3.2.9 Interacción con el entorno

El agente observa el estado actual del entorno y, con base en ese estado, selecciona una acción a través de su política (por ejemplo, usando ε-greedy, donde la mayoría de las veces elige la acción con mayor valor Q estimado, pero ocasionalmente explora otras acciones). Luego, el agente ejecuta la acción seleccionada en el entorno, lo que provoca una transición: el entorno responde devolviendo un nuevo estado resultante de esa acción y una recompensa inmediata, además de indicar si el episodio ha concluido, como se muestra en el diagrama de flujo de la figura 23.

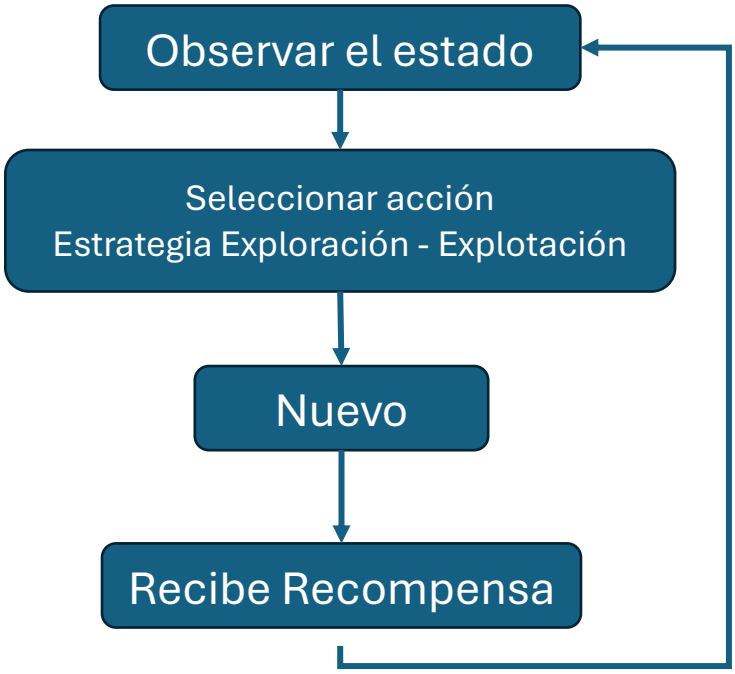

Figura 23 Ciclo interacción con el entorno

- **Observar el estado inicial:**

Al hacer Estado = env.reset(), el entorno devuelve el estado inicial. Este estado es transformado en un vector one-hot (vector_state) para alimentar la red neuronal.

- **Seleccionar acción:**

El agente elige una acción usando la función select_action(), que incorpora una estrategia ε-greedy. Aquí, la acción puede depender del estado, de la dirección y del número de visitas, promoviendo tanto exploración como explotación.

- **Ejecutar acción y observar transición:**

La acción seleccionada se aplica con env.step(action), que retorna:

  - new_state: el nuevo estado al que llega el agente.
  - reward: la recompensa recibida por esa acción.
  - done: si el episodio terminó.
  - new_direction: la nueva orientación del agente.

3.2.10 Almacenar la experiencia en el Replay Buffer

Cada transición observada (estado actual, acción tomada, recompensa recibida, siguiente estado y si el episodio terminó) se guarda en una memoria llamada Replay Buffer. Este buffer tiene una capacidad finita y, cuando se llena, las experiencias más antiguas se eliminan para dar espacio a las nuevas. Toda esta transición en Toyrace (vector_state, action, reward, new_state_np, done) se guarda en el Replay Buffer, permitiendo que el agente aprenda posteriormente de esta experiencia como se observa en la figura 24.

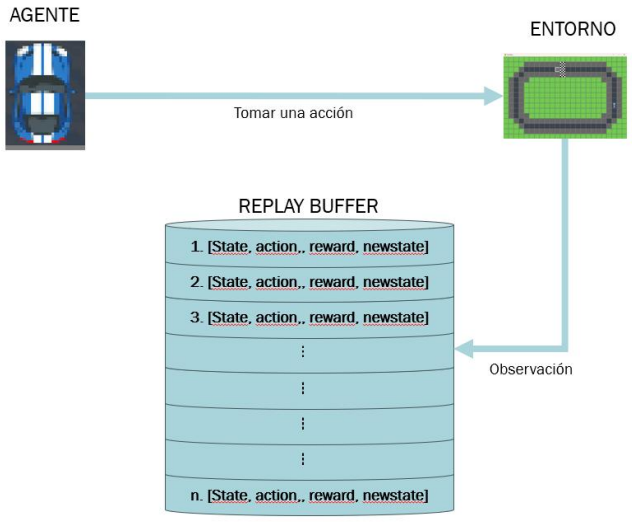

Figura 24. Experiencia en Replay Buffer

El tamaño de esta memoria está limitado a una cierta cantidad máxima de experiencias; por ejemplo, se puede fijar que el buffer guarde hasta diez mil

experiencias distintas. Cuando se alcanza este límite, las experiencias más antiguas se van eliminando para hacer espacio a las nuevas. De este modo, el agente siempre cuenta con un conjunto de datos reciente y relevante, evitando saturar la memoria y manteniendo la información fresca para el aprendizaje.

3.2.11 Muestrear minibatch aleatorios

A lo largo del entrenamiento del agente, es necesario extraer un conjunto de experiencias desde el Replay Buffer para proceder con el aprendizaje por lotes. Este momento se alcanza cuando el Replay Buffer contiene un número igual o superior al tamaño establecido para el minibatch. En el caso del entorno ToyRace, implementado en el código proporcionado, este valor corresponde específicamente a 64 transiciones, determinado por el parámetro batch_size.

En otras palabras, cada vez que el Replay Buffer del agente en el entorno ToyRace ha registrado al menos 64 transiciones, se activa el proceso de muestreo. En ese instante, se selecciona de forma aleatoria un conjunto de 64 experiencias almacenadas, conformando así el minibatch que será utilizado en la siguiente etapa del aprendizaje. Este procedimiento puede repetirse al finalizar cada episodio o tras un número definido de interacciones, de acuerdo con la configuración del entrenamiento.

Cada una de las transiciones seleccionadas conserva la información completa sobre el estado, la acción, la recompensa, el siguiente estado y la señal de finalización de episodio. Así, en el entorno ToyRace, la extracción del minibatch ocurre exactamente cuando el número de experiencias almacenadas en el Replay Buffer alcanza o supera el valor de 64, permitiendo que este conjunto de datos se utilice inmediatamente como entrada para la actualización de la red y la mejora del comportamiento del agente como se muestra en la Figura 25.

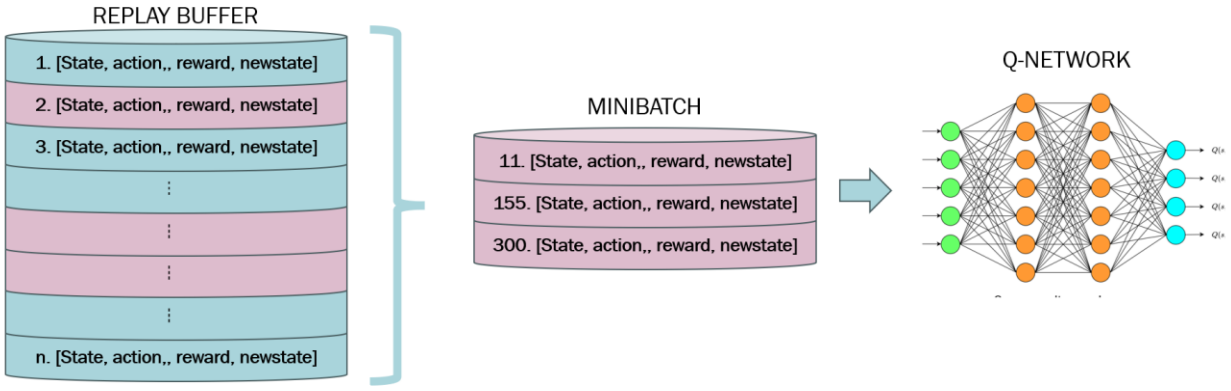

Figura 25. Proceso de muestreo aleatorio de transiciones del Replay Buffer para la actualización de la Q-network

3.2.12 Calcular objetivo y pérdida

Una vez que se ha extraído el minibatch de transiciones desde el Replay Buffer en el entorno ToyRace, el siguiente paso del proceso de aprendizaje consiste en calcular, para cada transición, el valor objetivo y la pérdida. Este procedimiento es fundamental para orientar la actualización de la red del agente.

En primer lugar, para cada transición contenida en el minibatch, se utiliza la información de la recompensa obtenida y del siguiente estado alcanzado para calcular el valor objetivo. Específicamente, el valor objetivo se define como la suma de la recompensa inmediata y el valor máximo estimado que el agente podría obtener en el siguiente estado, ponderado por el factor de descuento (gamma). Este valor máximo se obtiene evaluando todas las acciones posibles a partir del siguiente estado usando la target network.

$$y = \begin{cases} r & si\ el\ episodio\ termina\ en\ s'(done = 1) \\ r + \gamma . max_{a'} Q_{target}(s', a') & en\ caso\ contrario \end{cases} \quad \text{(ec. 5)}$$

donde:

- *r* es la recompensa obtenida al realizar la acción *a* en el estado *s.*
- *γ* es el factor de descuento.
- $Q_{target}(s', a')$ es el valor estimado por la Target network para el siguiente estado *s'* y cada posible acción *a'.*
- $max_{a'} Q_{target}(s', a')$ selecciona el mayor valor estimado entre todas las acciones posibles en *s'.*

A continuación, se compara el valor objetivo calculado para cada transición con el valor que la Q network del agente predice actualmente para el par de estado y acción correspondientes. La diferencia entre estos dos valores constituye el error o pérdida, que representa qué tanto debe corregirse la predicción de la red para acercarse al valor esperado real de la acción tomada.

$$L = (y - Q(s, a))^2 \quad \text{(ec. 6)}$$

donde:

- L es el error o la perdida,
- y es el valor objetivo
- $Q(s, a)$ es el valor estimado por la Q network

En el código del entorno ToyRace, este cálculo se realiza de manera conjunta para todas las transiciones del minibatch, generando así un conjunto de valores objetivo y de pérdidas. Este conjunto de datos es el que se utiliza inmediatamente después para guiar el ajuste de los parámetros internos de la red, permitiendo que el agente mejore su estimación de las acciones más convenientes en cada estado del entorno.

3.2.13 Actualizar la Q-network

Calculados el valor objetivo y la pérdida para el minibatch de transiciones, se procede a la actualización de la Q network, que es la red responsable de estimar los valores de las acciones en cada estado.

El objetivo de esta etapa es ajustar los parámetros internos de la Q network de modo que sus predicciones se aproximen lo más posible a los valores objetivo calculados en el paso anterior. Para lograrlo, se emplea un algoritmo de optimización, que generalmente es una variante del método de gradiente descendente.

Concretamente, el proceso consiste en calcular los gradientes de la función de pérdida con respecto a los parámetros de la red. Estos gradientes indican la dirección y magnitud en la que se deben modificar los parámetros para reducir el error de predicción. Posteriormente, se realiza una actualización de dichos parámetros en sentido opuesto al gradiente, es decir, en la dirección que minimiza la pérdida.

Este procedimiento se aplica a todo el minibatch de transiciones, de modo que la red Q aprende simultáneamente a mejorar sus estimaciones en diferentes situaciones vividas por el agente. Así, en el entorno ToyRace, la actualización de la Q network permite que el agente refine su política de selección de acciones a partir de las experiencias almacenadas y el feedback recibido, logrando un aprendizaje progresivo y efectivo en tareas de toma de decisiones secuenciales.

3.2.14 Copiar pesos a la Target network

En este paso, se realiza una copia exacta de los parámetros de la Q network hacia la target network. Es decir, todos los valores y configuraciones aprendidos hasta ese momento por la Q network se transfieren de forma íntegra a la red objetivo. Este procedimiento se ejecuta cada vez que se alcanza el número de pasos o episodios especificado por el parámetro target_update en el entorno ToyRace como se observa en la Figura 26. Una vez copiados, la target network utilizará estos nuevos parámetros en los siguientes ciclos de entrenamiento hasta la próxima actualización.

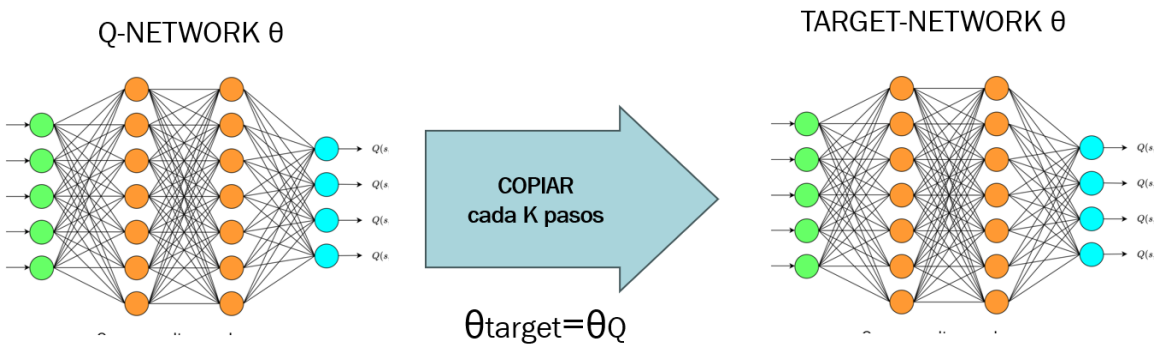

Figura 26. Copiar pesos a la target network desde la Q network cada K pasos

Durante el entrenamiento en el entorno ToyRace, se lleva un registro del número de pasos o iteraciones realizados. En cada ciclo de entrenamiento, se compara este contador con el valor del parámetro target_update. Por ejemplo, si target_update está definido en 50, cada vez que el contador alcanza o supera el valor de 50, se procede a copiar todos los parámetros actuales de la Q network a la target network. Después de realizar esta copia, el contador se reinicia y continúa acumulando hasta que vuelva a alcanzar el valor establecido, repitiendo el proceso de actualización periódica de la target network.

3.2.15 Finalizar entrenamiento

El proceso de entrenamiento se repite siguiendo todos los pasos descritos, incluyendo la recolección de experiencias, el muestreo de minibatches, el cálculo de objetivos y pérdidas, la actualización de la Q-network y la sincronización periódica con la target network, hasta que se cumple el número total de episodios definido al inicio. En el caso del entorno ToyRace, este límite está determinado por el parámetro n_episodes. Una vez alcanzado este número de episodios, el entrenamiento finaliza y el agente detiene el aprendizaje, conservando los parámetros finales de su Q network como resultado de todo el proceso.

3.2.16 Evaluación de la DQN

Tras finalizar el proceso de entrenamiento, es fundamental evaluar el desempeño del agente para determinar en qué medida ha aprendido a interactuar de manera efectiva con el entorno ToyRace. Para ello, se realiza una fase de evaluación en la que el agente utiliza la política aprendida, es decir, selecciona las acciones que considera óptimas según los valores almacenados en su Q network, ilustrado en la figura 27.

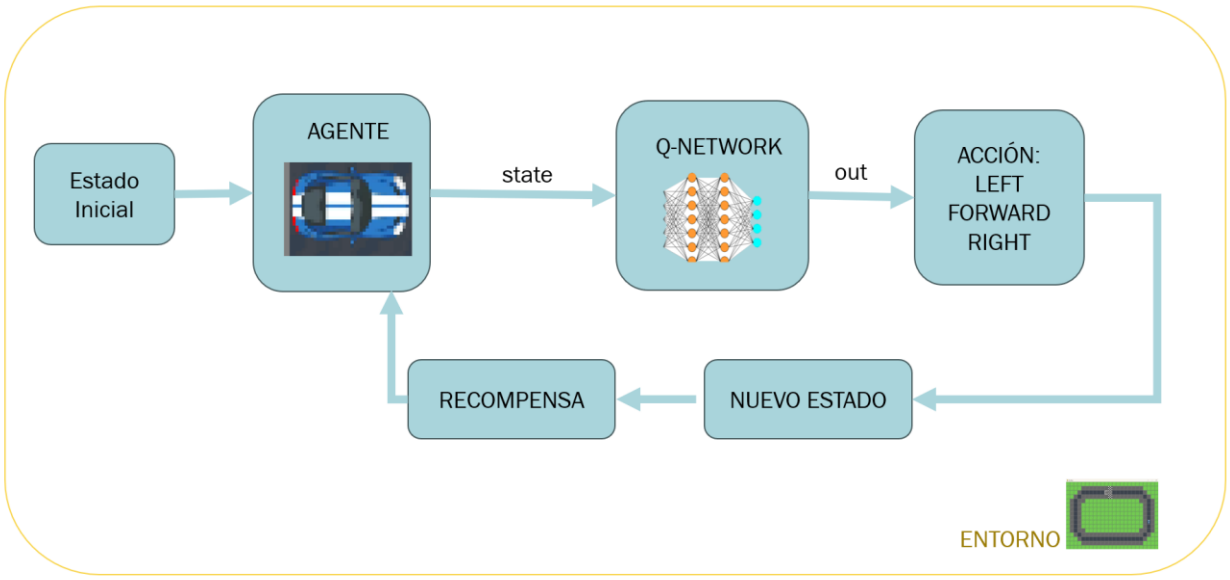

Figura 27. Flujo de evaluación DQN

Durante la evaluación, el agente se enfrenta nuevamente al entorno, pero a diferencia de la etapa de entrenamiento, ya no se emplea exploración aleatoria. De este modo, el agente elige siempre la acción con el mayor valor Q estimado en cada estado.

El desempeño se mide ejecutando un conjunto de episodios de prueba y registrando métricas relevantes, como la recompensa total acumulada en cada episodio, la cantidad de pasos necesarios para alcanzar el objetivo o la frecuencia con la que logra completar la tarea correctamente. Estos resultados permiten cuantificar objetivamente la eficacia de la política aprendida y comparar su rendimiento con otras configuraciones o enfoques alternativos.

Finalmente, los datos obtenidos durante la evaluación se pueden utilizar para generar gráficos, tablas o análisis estadísticos que ilustren el nivel de desempeño alcanzado por el agente en el entorno ToyRace, proporcionando una visión clara y cuantitativa de la capacidad de generalización y eficiencia del modelo DQN entrenado.

3.3 Actor-Critic: Aprendizaje basado en políticas y valores

En los métodos de aprendizaje por políticas y valores, el objetivo es que un agente aprenda a tomar buenas decisiones combinando dos ideas principales. Por un lado, aprender una política, que es básicamente una guía que le dice al agente qué acción tomar en cada situación. Por otro lado, calcular valores, que le ayudan a saber qué tan buenas o malas son esas decisiones a largo plazo.

El método Actor-Critic junta estas dos formas de aprendizaje en un solo sistema. El

actor es la parte que decide qué acción tomar (siguiendo la política), y el critic es la parte que evalúa esas acciones y le dice al actor si lo está haciendo bien o mal (calculando los valores). Gracias a esta combinación, el agente puede mejorar su forma de actuar de manera más rápida y estable que usando solo políticas o solo valores.

En este capítulo veremos cómo funciona este método y por qué es útil juntar estas dos ideas para lograr mejores resultados al entrenar agentes inteligentes.

3.3.1 Fundamentos de los Métodos Actor-Critic

El método Actor-Critic es un enfoque de aprendizaje donde se utilizan dos componentes principales, generalmente implementados con redes neuronales: un actor que aprende la política, es decir, decide qué acción tomar en cada estado, y un critic que estima el valor de los estados o acciones, evaluando la calidad de las decisiones tomadas por el actor (Konda & Tsitsiklis, 2000; Silver et al., 2014).

A diferencia de métodos como DQN, que solo pueden manejar espacios de acción discretos, Actor-Critic es eficiente en problemas con espacios de acción continuos, donde las posibles acciones no se pueden enumerar fácilmente (Silver et al., 2014). Además, al combinar política y valor, el aprendizaje tiende a ser más estable y reduce la varianza en la actualización de la política, permitiendo una exploración más eficiente (Konda & Tsitsiklis, 2000).

Los métodos de gradiente de política (Policy Gradient) optimizan directamente la política, mientras que los métodos basados en valores (Value-Based) buscan encontrar la mejor acción estimando valores para cada posible acción (Sutton & Barto, 2018). Actor-Critic actúa como un “puente” entre ambos: el actor actualiza la política, y el critic mejora la precisión de esa actualización al estimar valores, combinando así las ventajas de ambos enfoques (Konda & Tsitsiklis, 2000; Sutton & Barto, 2018).

3.3.2 Actor

El actor aprende una política $\pi(a|s; \theta)$, la cual define la probabilidad de tomar una acción a dado un estado s, y está parametrizada por θ. Su objetivo es seleccionar las mejores acciones para maximizar la recompensa esperada (Silver et al., 2014).

3.3.3 Critic

El critic estima una función de valor, que puede ser $V(s; w)$ para el valor del estado, o $Q(s, a; w)$ para el valor de una acción en un estado, donde w son los parámetros del critic. Esto permite evaluar cuán buena fue la acción elegida por el actor (Konda & Tsitsiklis, 2000).

3.3.4 Interacción entre actor y critic

El critic observa la acción seleccionada por el actor y proporciona una señal de retroalimentación, indicando si la acción fue mejor o peor de lo esperado. El actor ajusta su política usando esta información para mejorar en el futuro. Ambos actualizan sus parámetros con funciones de pérdida propias: el actor para maximizar la recompensa esperada y el critic para mejorar la precisión en la estimación del valor (Sutton & Barto, 2018).

3.3.5 Expresión matemática

Formulas básicas:

- Gradiente de la política (para el actor). El actor actualiza sus parámetros siguiendo el gradiente de la política, expresado como:

$$\nabla_\theta J(\theta) = E_{s,a}[\nabla_\theta \, log \, \pi(a \mid s;\theta) \cdot \delta] \text{ (ec. 7)}$$

  - π(a|s;θ): Es la política, o sea, la probabilidad de elegir la acción a cuando el agente está en el estado s, según los parámetros θ (que son los "pesos" de la red del actor).
  - $\nabla_\theta \, log \, \pi(a \mid s;\theta$: Es una forma matemática de preguntar "¿qué tanto afecta cambiar los parámetros θ a la decisión de tomar la acción a?".
  - δ**:** Es la retroalimentación que le da el critic al actor; indica si la acción tomada fue mejor o peor de lo esperado.
  - $E_{s,a}$: Significa que esto se calcula en promedio, considerando varios estados y acciones.

- Actualización del critic (TD error). El critic ajusta sus parámetros minimizando el error temporal:

$$\delta = r + \gamma V(s';w) - V(s;w) \text{ (ec. 8)}$$

  - r: Recompensa recibida después de tomar la acción.
  - V(s;w): Valor esperado del estado actual antes de tomar la acción (calculado por el critic con sus propios parámetros w).
  - V(s′;w): Valor esperado del nuevo estado, después de la acción.
  - γ: Factor de descuento; indica cuánto le importan al agente las recompensas futuras.
  - δ: Diferencia entre lo que se esperaba y lo que realmente se obtuvo (esto

es el “TD error”, y es la señal que ayuda tanto al critic como al actor a mejorar).

3.3.6 Diagrama general

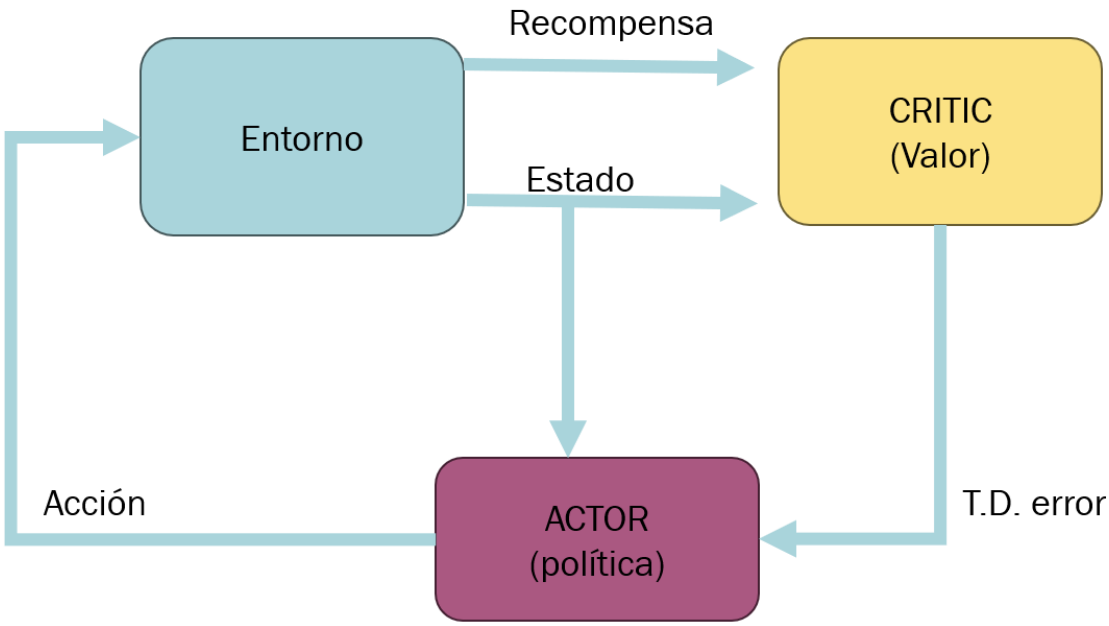

Figura 28. Estructura del algoritmo actor-critic

- El actor recibe el estado y selecciona una acción.
- El entorno responde con una nueva observación y recompensa.
- El critic evalúa la acción del actor y le brinda retroalimentación.

Ambos componentes ajustan sus parámetros en base a esta información (Silver et al., 2014).

3.3.7 Algoritmo actor-critic

En base al entorno ToyRAce que se desarrolló anteriormente se describirá la implementación del algoritmo Actor-Critic. Como ya se sabe el agente debe moverse por un grid siguiendo diferentes direcciones, eligiendo entre tres acciones posibles (izquierda, avanzar, derecha). En este algoritmo se define dos redes neuronales (actor

y critic), gestiona la interacción con el entorno y actualiza ambos modelos en cada paso, buscando maximizar la recompensa total por episodio.

3.3.8 Definición de las redes actor y critic

El primer paso es construir las redes neuronales que conforman el actor y el critic (Figura 29). El actor toma como entrada el estado actual del entorno (por ejemplo, la posición y orientación del agente) y genera una distribución de probabilidades sobre las posibles acciones. Esto se logra utilizando una red completamente conectada con una capa de salida softmax, que asegura que las probabilidades sumen uno. El critic, por su parte, está formado por una red similar, pero en vez de producir múltiples probabilidades, su salida es un solo valor: la estimación del valor del estado actual.

```
ACTOR
Sequential(
  (0): Linear(in_features=7, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=3, bias=True)
  (3): Softmax(dim=-1)
)
CRITIC
Sequential(
  (0): Linear(in_features=7, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=1, bias=True)
)
```

Figura 29. Estructura de las redes neuronales Actor y Critic

Esta separación entre actor y critic permite que cada red se especialice: el actor se enfoca en la toma de decisiones, mientras que el critic se dedica a evaluar cuán buenas son esas decisiones en función de la recompensa esperada.

3.3.9 Selección de acción

Para elegir una acción, el estado actual se introduce en la red del actor, que devuelve una distribución de probabilidades sobre las acciones posibles. A partir de esta distribución, se selecciona una acción utilizando muestreo probabilístico, lo que introduce aleatoriedad y fomenta la exploración. Además, se incorpora un mecanismo que considera el número de veces que se ha visitado un estado, lo que ayuda a

equilibrar la exploración de nuevas rutas con la explotación de rutas conocidas.

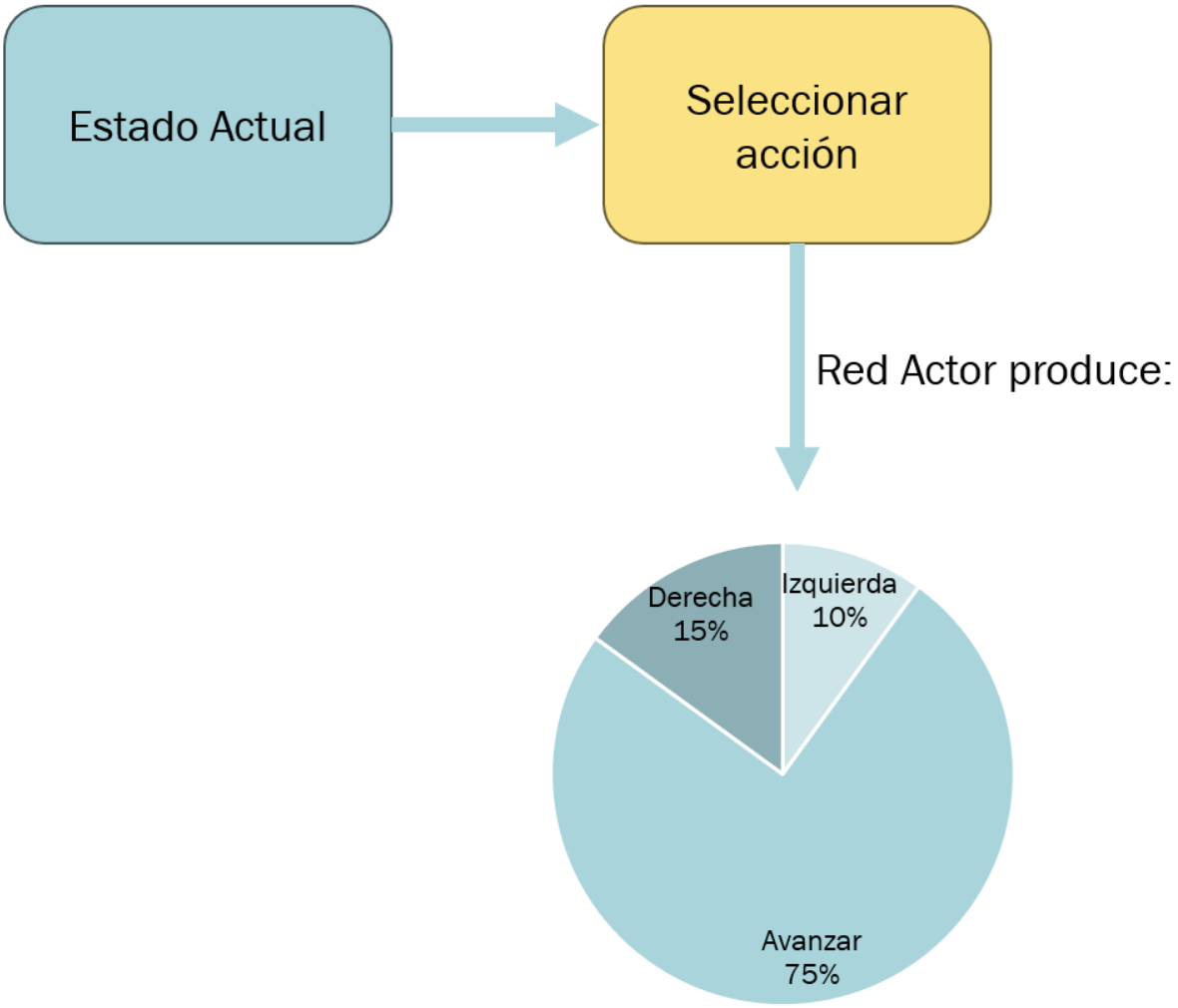

Figura 30. Proceso de selección de acción

Podemos definir este proceso (Figura 30) como una ruleta, cada vez que el agente necesita actuar, consulta la ruleta de probabilidades que crea el actor, gira la ruleta y elige la acción que resulta seleccionada. Así, el proceso no es completamente aleatorio ni totalmente determinista, sino que balancea ambas cosas según lo que el agente ha aprendido.

3.3.10 Evaluación del critic

Cada vez que el agente toma una acción y el entorno responde con un nuevo estado

y recompensa, la red critic evalúa el valor tanto del estado actual como del siguiente estado. Esta evaluación es esencial para calcular el llamado "error temporal" o TD error, que mide la diferencia entre el valor predicho por el critic y el valor realmente observado tras recibir la recompensa y avanzar al siguiente estado. Así, el critic proporciona una señal de retroalimentación no solo para mejorar su propia predicción, sino también para ayudar al actor a ajustar su política de selección de acciones.

Supongamos que el agente está en el Estado A y elige la acción “Derecha”. El critic toma la observación del Estado A y lo procesa a través de su red neuronal, la cual entrega un valor numérico. Ese número representa la expectativa de recompensa futura si el agente parte desde ese estado y sigue actuando como hasta ahora. Después, el critic utiliza estos valores para calcular el TD error (error temporal), que mide la diferencia entre lo que esperaba y lo que realmente sucedió. A continuación, en la figura 31 se muestra el flujo de la evaluación que realiza el critic.

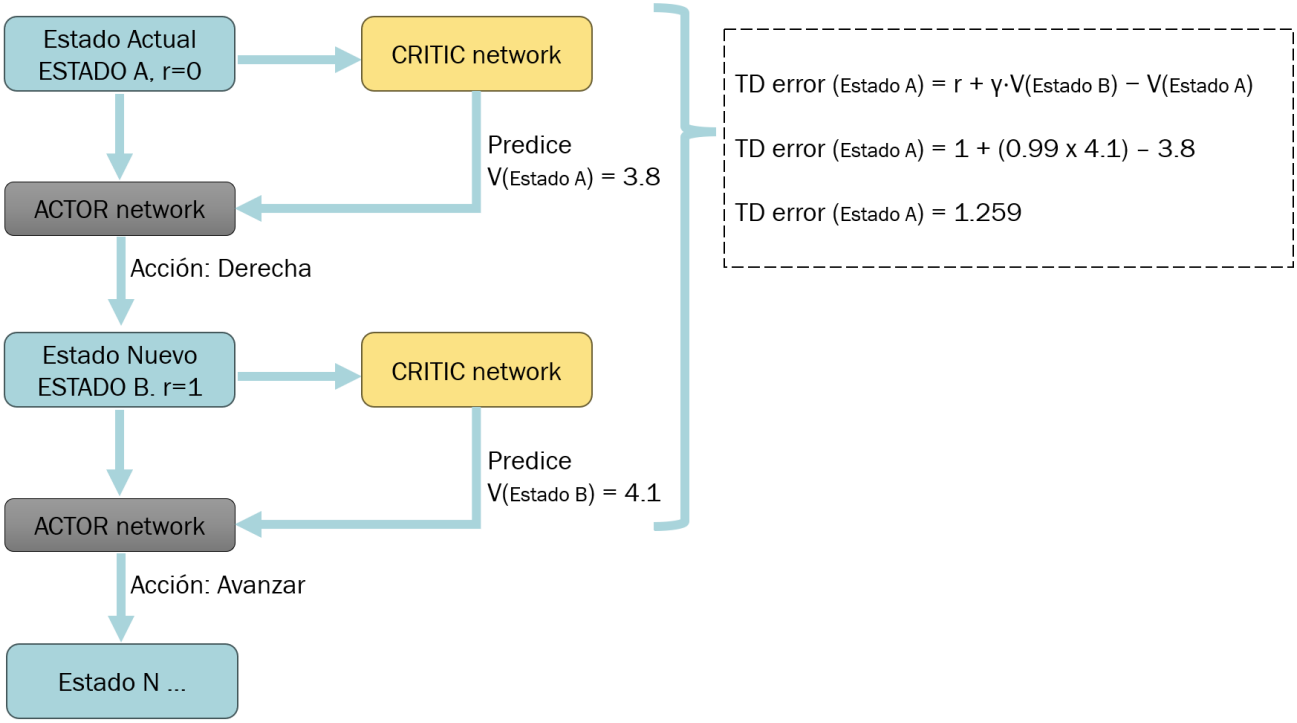

Figura 31. Diagrama evaluación del Critic

- Si el TD error es positivo (como en la figura 31, 1.259), indica que las cosas salieron mejor de lo esperado: el actor debería reforzar esta acción y el critic debe ajustar su predicción al alza.
- Si el TD error es negativo, fue peor de lo esperado: el actor debería tomar otra decisión la próxima vez, y el critic debe bajar su predicción.

3.3.11 Cálculo de la función de perdida y actualización

- Función de pérdida del critic:

  El objetivo del critic es aprender a estimar correctamente el valor esperado de cada estado. Por eso, la función de pérdida del critic se define como el error cuadrático medio (MSE) entre el valor actual predicho y el valor objetivo (calculado usando la recompensa y el valor del siguiente estado):

  $$Lcritic = \delta^2 = TD\ error^2 \text{ (ec. 9)}$$

  Esto significa que, si el critic estaba muy equivocado (TD error grande), la penalización será mayor y la red aprenderá más rápido. Si su predicción era acertada, la corrección será menor.

- Función de pérdida del actor:

  El actor, por su parte, utiliza la información de cuán buena fue la acción seleccionada según el critic. La función de pérdida para el actor en tu código pondera el logaritmo de la probabilidad de la acción elegida por el TD error:

  $$Lactor = -log\pi(a \mid s) \cdot \delta \text{ (ec. 10)}$$

  Esta expresión fomenta que el actor aumente la probabilidad de aquellas acciones que llevaron a resultados mejores de lo esperado (δ>0) y disminuya la probabilidad de acciones que resultaron peores de lo previsto (δ<0).

- Actualización de los modelos:

  Se calculan ambas pérdidas y se actualizan los parámetros de las redes utilizando algoritmos de optimización (como Adam o SGD).
  La actualización ocurre así:

  1. Se calcula el TD error con las predicciones actuales del critic.
  2. Se obtiene la pérdida del critic y se ajustan sus pesos para que sus futuras predicciones sean más precisas.
  3. Se calcula la pérdida del actor y se ajustan sus pesos para que su política favorezca acciones que el critic considera ventajosas.
  4. Este proceso se repite en cada paso y en cada episodio, permitiendo que ambos modelos aprendan de la experiencia acumulada.

3.3.12 Bucle de entrenamiento

El bucle de entrenamiento es el corazón del algoritmo Actor-Critic y representa el proceso mediante el cual el agente aprende a partir de la experiencia directa con el entorno (figura 32). Este ciclo se repite durante un número determinado de episodios (n_episodes), y dentro de cada episodio se suceden múltiples pasos hasta que el agente llega a una condición de finalización (como alcanzar la meta o toparse con un obstáculo).

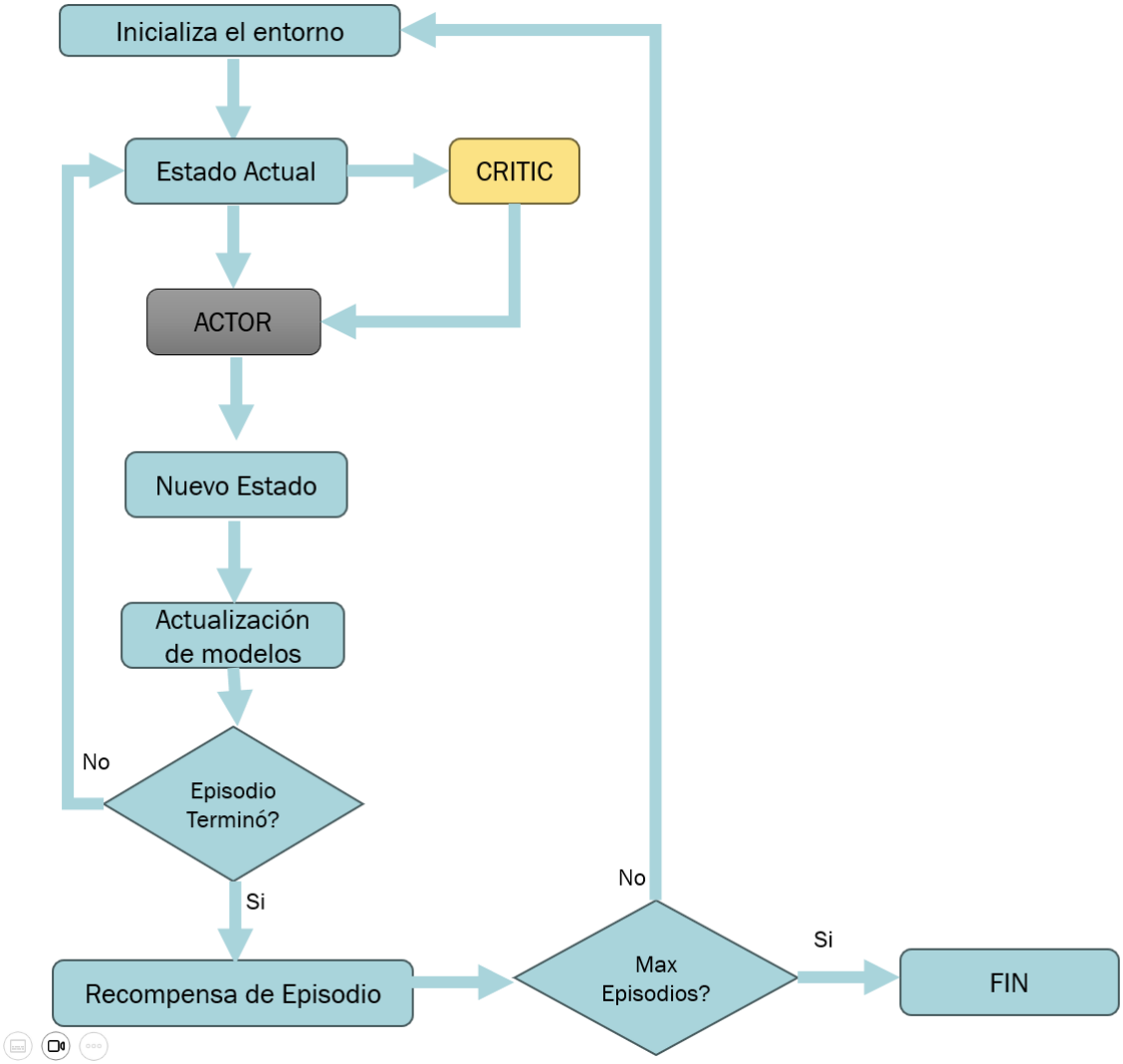

Figura 32. Diagrama de flujo del bucle de entrenamiento

El ciclo comienza reiniciando el entorno a un estado inicial. Se configura la posición y orientación del agente, y se transforma este estado en un formato que pueda ser entendido por las redes neuronales.

En cada paso del episodio, el actor recibe el estado actual del agente y, basándose en su política, selecciona una acción. Esta acción se ejecuta en el entorno, el cual responde con un nuevo estado, una recompensa inmediata y una indicación de si el episodio ha terminado. Cada transición, compuesta por el estado, la acción, la recompensa y el siguiente estado, constituye una experiencia nueva para el agente.

Después de obtener esta experiencia, el agente procede a actualizar sus modelos. Una vez actualizados ambos modelos, el agente avanza al siguiente estado y repite el proceso: selecciona una nueva acción, interactúa con el entorno, recibe retroalimentación y actualiza sus parámetros. Este ciclo de selección de acción, obtención de experiencia y actualización de los modelos se repite hasta que el episodio termina.

Al concluir cada episodio, se registra la recompensa total obtenida, lo que permite monitorear el progreso del aprendizaje a lo largo del tiempo. Todo el proceso se reinicia para un nuevo episodio, partiendo de un estado inicial posiblemente distinto, y así sucesivamente durante cientos de episodios.

3.4 Conclusión del capítulo

El aprendizaje por refuerzo, desde sus bases en Q-learning hasta su integración con redes neuronales profundas, constituye un enfoque poderoso para la toma de decisiones secuenciales. Conceptos como los MDP, las recompensas y las estrategias de exploración-explotación, junto con elementos avanzados como replay buffer y target network, permiten a los agentes aprender políticas efectivas incluso en entornos complejos. Los ejemplos en Frozen Lake, Norberto y ToyRace mostraron cómo los agentes mejoran progresivamente su desempeño, confirmando el potencial del aprendizaje por refuerzo profundo en aplicaciones como la robótica, los juegos y la optimización de sistemas.

4. Metodología

ToyRace es un entorno personalizado desarrollado con la biblioteca Gymnasium de OpenAI. Está inspirado en el entorno Frozen Lake, pero adaptado a un circuito de carreras en ovalo de tres carriles, con una posición de inicio y salida.

El objetivo principal es que el agente, obtenga la mayor cantidad de recompensa posible mientras recorre el circuito (figura 33). Para lograrlo, el agente debe aprender a tomar las decisiones más eficientes en cada paso, optimizando su trayecto hasta llegar a la meta. Así, ToyRace no solo pone a prueba la capacidad del agente para completar la vuelta, sino también su habilidad para maximizar las recompensas durante todo el recorrido.

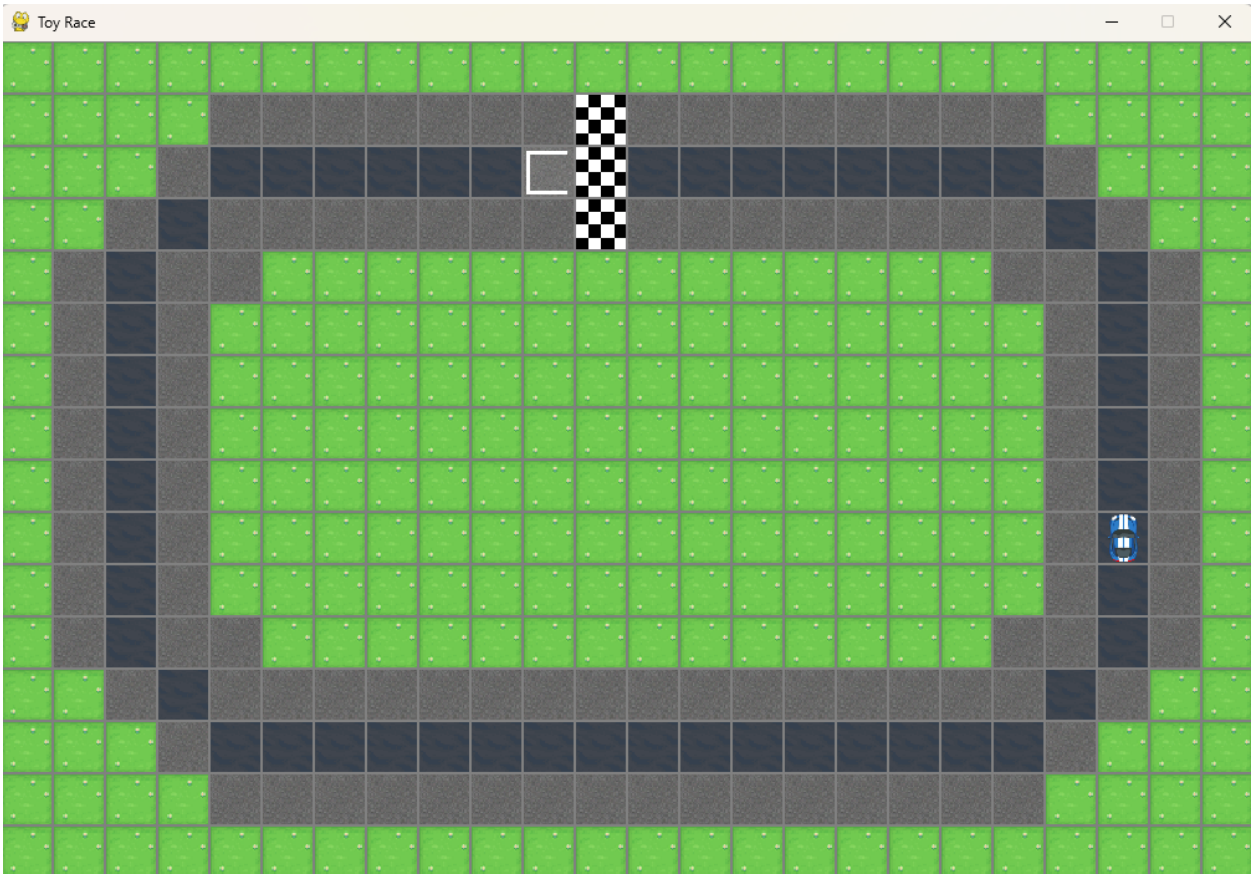

Figura 33. Entorno ToyRace

4.1 Características del entorno

En ToyRace, cada estado se define principalmente por tres atributos: la posición del carro en el grid, la orientación del carro y el tipo de carril en el que se encuentra.

Posición:
El circuito está representado como una cuadrícula de 16 filas por 24 columnas (figura 34), con 384 posibles posiciones. Cada celda puede pertenecer al carril central, a uno de los dos carriles laterales, o estar fuera de la pista.

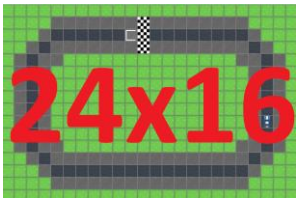

Figura 34. 384 estados

Orientación:
El carro puede tener una de ocho orientaciones posibles (figura 35):

- 0: Norte (N)
- 1: Noreste (NE)
- 2: Este (E)
- 3: Sureste (SE)
- 4: Sur (S)
- 5: Suroeste (SO)
- 6: Oeste (O)
- 7: Noroeste (NO)

Figura 35. Orientaciones ToyRace

Carril:
Cada posición de la pista pertenece a uno de los tres carriles: central, lateral izquierdo o lateral derecho (figura 36).

Figura 36. Carriles ToyRace

Estados terminales:

Un estado es terminal si:

- El carro sale de la pista.
- El carro llega a la meta tras completar la vuelta.

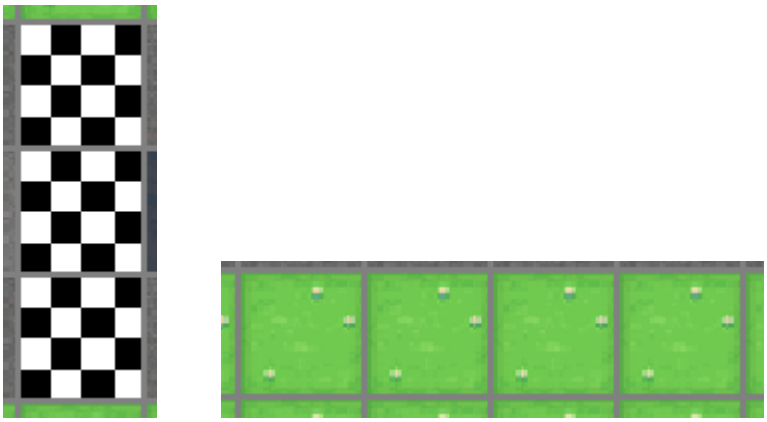
Figura 37. Estados Terminales ToyRace

Recompensas:

- Estática según el carril:
  - Carril central: 1
  - Carriles laterales: 0.3

- Meta: 10
- Salirse de la pista: -1

Acciones y transiciones:

- El agente puede ejecutar tres acciones (figura 38):
  - 0: LEFT (gira 45° a la izquierda)
  - 1: FORWARD (mantiene la orientación y avanza)
  - 2: RIGHT (gira 45° a la derecha)
- La orientación del carro cambia instantáneamente según la acción elegida, y el movimiento se actualiza en el grid en función de la nueva orientación.

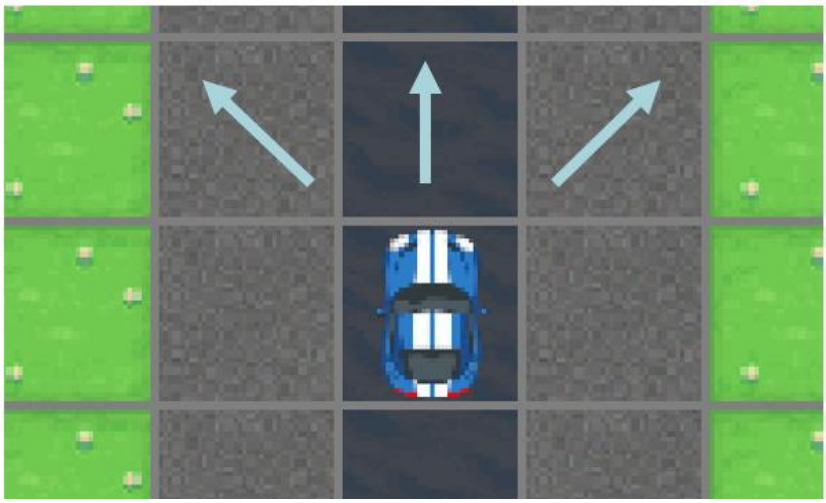

Figura 38. Acciones ToyRace

4.2 Implementación técnica con Gymnasium

Gymnasium es una biblioteca de Python diseñada para crear, gestionar y simular entornos de aprendizaje por refuerzo. Proporciona una interfaz sencilla y estandarizada que permite a los usuarios desarrollar y probar algoritmos de RL en diferentes entornos, desde los clásicos hasta personalizados. Gymnasium facilita la interacción entre el agente y el entorno mediante una estructura basada en episodios, estados, acciones y recompensas, haciendo más fácil el diseño, la simulación y la comparación de experimentos.

Ventajas de usar esta biblioteca

- **Estandarización:** Ofrece una interfaz uniforme y bien definida, facilitando la integración de entornos personalizados con algoritmos existentes.
- **Compatibilidad:** Es compatible con una gran variedad de frameworks y herramientas de RL, lo que permite experimentar y comparar resultados fácilmente.
- **Flexibilidad:** Permite la creación y modificación de entornos propios de manera sencilla, como fue el caso con ToyRace.
- **Facilidad de uso:** La estructura de la API hace que la interacción entre el agente y el entorno sea intuitiva, simplificando el ciclo de desarrollo y prueba.
- **Comunidad y recursos:** Gymnasium cuenta con una amplia comunidad y abundantes ejemplos, tutoriales y documentación, lo que facilita la resolución de dudas y la incorporación de buenas prácticas.

Clase principal que extiende gym.Env

El entorno ToyRace se desarrolla creando una clase principal que hereda de gym.Env. Esta herencia garantiza que el entorno es compatible con la API de Gymnasium, facilitando su integración con algoritmos de aprendizaje por refuerzo.

Métodos implementados

- **reset()**
  Inicializa el entorno, colocando el carro en la posición de inicio y reiniciando la orientación y demás variables de estado. Retorna el estado inicial y, opcionalmente, información adicional.

```
def reset(self, seed=None, options=None):
    # Inicializa la posición y orientación del carro
    self.s = posición_inicial  # por ejemplo, la casilla 'S' del mapa
    self.orientation = 6       # orientación inicial (por ejemplo, oeste)
    return int(self.s), {"prob": 1}
```

- **step(action)**
  Recibe una acción (girar a la izquierda, avanzar o girar a la derecha), actualiza la

orientación y posición del carro según la dinámica de la pista y devuelve:

- El nuevo estado
- La recompensa obtenida
- Un indicador de si el episodio ha terminado (estado terminal)
- La orientación actual
- Un diccionario de información extra

```
def paso(self, a):
    # Actualiza la orientación según la acción (LEFT, FORWARD, RIGHT)
    if a == 0:
        self.orientation = ... # Gira a la izquierda
    elif a == 2:
        self.orientation = ... # Gira a la derecha
    # Calcula la nueva posición usando la orientación actual
    new_row, new_col = ...
    # Determina la recompensa y si es estado terminal
    letter = self.desc[new_row, new_col]
    terminal = letter in b"GH"
    if letter == b"F":      r = 10
    elif letter == b"C":    r = 100
    elif letter == b"H":    r = -100
    elif letter == b"G":    r = 10000
    else:                   r = 0
    return (int(s), r, terminal, self.orientation, {})
```

- **render()**
  Permite visualizar el entorno, mostrando la pista y la posición/orientación actual del carro, útil para análisis y debugging.
- **close()**
  Libera los recursos y cierra las ventanas o procesos asociados con la visualización.

4.3 Consideraciones técnicas

Lenguaje de programación y dependencias

- Lenguaje de programación: Python
- Versión de Python recomendada: 3.10 o superior
- Versión de Gymnasium: 0.29.1

- Otras dependencias relevantes:
    - **NumPy:** para manipulación de arrays y generación de mapas aleatorios
    - **pygame:** (opcional) para visualización gráfica del entorno
    - **Gymnasium/toy_text:** para la estructura de entornos tipo grid y utilidades auxiliares

4.4 Implementación de los algoritmos

La implementación de los algoritmos consistió en el diseño y adaptación de tres enfoques clásicos del aprendizaje por refuerzo: Q-Learning (QL), Deep Q-Network (DQN) y Actor-Critic (AC). Cada uno de ellos se programó de manera independiente, manteniendo parámetros de referencia compartidos para asegurar una comparación parcial.

Con el objetivo de garantizar la reproducibilidad de los resultados y una evaluación consistente, se emplearon semillas aleatorias (seeds) que fijaron la inicialización de cada entrenamiento. Cada algoritmo fue ejecutado con cinco semillas diferentes (0–4), lo que permitió obtener medidas más robustas y reducir la influencia del azar en los resultados finales.

También se establecieron un conjunto de parámetros definidos de forma común para todos los algoritmos, de manera que las comparaciones se centraran exclusivamente en las diferencias de los métodos de aprendizaje y no en factores externos. Estos parámetros incluyeron el número de episodios de entrenamiento, el esquema de exploración ε-greedy, el factor de descuento y el intervalo de evaluación.

En primer lugar, se presentan los parámetros compartidos que definen la base de la comparación, y posteriormente se detallan las configuraciones específicas de cada algoritmo.

4.4.1 Parámetros compartidos

Con el fin de asegurar condiciones de entrenamiento consistentes, se definió un conjunto de parámetros compartidos para los tres algoritmos. Estos parámetros establecieron la base de la comparación y permitieron garantizar que las diferencias en los resultados fueran atribuibles únicamente a la naturaleza de cada método de aprendizaje.

A continuación, se muestra la Tabla 6, donde se resumen los parámetros comunes utilizados en la implementación:

Tabla 6. Parámetros compartidos para entrenamiento

Parámetro	Descripción	Valor		
		QL	DQN	AC
Entorno	ToyRace	Si	Si	Si
n_directions	Número de orientaciones posibles del vehículo	8	8	8
n_actions	Número de acciones disponibles (LEFT, FORWARD, RIGHT)	3	3	3
n_states	Número total de estados (filas × columnas)	384	384	384
γ	Factor de descuento	0.99	0.99	0.99
Epsilon inicial	Valor inicial de exploración	0.9	0.9	0.9
Epsilon mínimo	Valor mínimo de exploración	0.01	0.01	0.01
Decaimiento de epsilon	Reducción progresiva de exploración	0.999	0.999	0.999
n_episodes	Número total de episodios de entrenamiento	2000	2000	2000
Semillas	Valores usados para inicializar el entrenamiento de forma reproducible.		[0, 1, 2, 3, 4]	
Evaluación periódica	Frecuencia con la que se registraron métricas de desempeño	10	10	10

4.4.2 Parámetros Q-Learning

El algoritmo Q-Learning se implementó de manera tabular, asignando a cada par estado-acción un valor que representa la expectativa de recompensa futura. La tabla Q fue inicializada en cero y actualizada en cada interacción siguiendo la ecuación de Bellman.

Para la selección de acciones se utilizó una política ε-greedy, que permitió balancear exploración y explotación. Al inicio del entrenamiento el agente exploraba con alta probabilidad, y esta se redujo gradualmente hasta alcanzar un mínimo establecido.

El entrenamiento del agente se realizó durante múltiples episodios, en los cuales se actualizaron los valores Q en función de las recompensas recibidas. Adicionalmente, se estableció un umbral de desempeño que permitió determinar el momento de convergencia del aprendizaje.

Tabla 7. Parámetros de entrenamiento para Q-Learning

Parámetro	**Descripción**	**Valor QL**
α (learning rate)	Controla cuánto se actualizan los valores Q en cada paso de aprendizaje.	0.1

4.4.3 Parámetros DQN

El algoritmo Deep Q-Network se implementó sustituyendo la tabla Q por una red neuronal profunda capaz de aproximar los valores de acción. Para mejorar la estabilidad del entrenamiento, se utilizó el esquema clásico de DQN que incluye una red objetivo (target network) y una memoria de repetición (Replay Buffer).

La red principal y la red objetivo compartieron la misma arquitectura inicial, y la sincronización entre ambas se realizó periódicamente. Durante el entrenamiento, las experiencias recolectadas por el agente se almacenaron en un buffer de tamaño fijo y se muestrearon en mini-lotes para romper la correlación temporal de las transiciones.

El proceso de exploración se basó en una política ε-greedy, iniciando con un nivel de exploración máximo y decreciendo de manera progresiva hasta alcanzar un valor mínimo. El entrenamiento fue llevado a cabo durante múltiples episodios, actualizando los parámetros de la red de manera frecuente.

Tabla 8. Parámetros de entrenamiento DQN

Parámetro	Descripción	Valor DQN
Arquitectura de la red	Dos capas ocultas densas con 128 neuronas y activación ReLU.	Input → [128, ReLU] → [128, ReLU] → Output
Optimizador	Algoritmo de optimización utilizado para actualizar los pesos de la red principal.	Adam (lr = 0.001)
Replay Buffer	Almacena experiencias del agente para muestreo aleatorio.	10,000 transiciones
Batch size	Tamaño de los mini-lotes extraídos del buffer en cada actualización.	64
Warmup	Transiciones iniciales acumuladas antes de comenzar el entrenamiento de la red.	1000 pasos
Target update	Intervalo en el que se sincroniza la red objetivo con la red principal.	200 pasos

4.4.4 Parámetros AC

El algoritmo Actor-Critic se implementó utilizando dos redes neuronales: una encargada de la política (actor) y otra de la función de valor (crítico). El actor produce una distribución de probabilidades sobre las acciones posibles, mientras que el crítico estima el valor de cada estado, lo que permite retroalimentar la actualización de la política.

La política de selección de acciones se basó en una salida softmax, que fue combinada con un esquema ε-greedy para mantener cierto grado de exploración. Durante el entrenamiento, las actualizaciones se realizaron en cada paso utilizando la diferencia temporal (TD error) como señal de aprendizaje.

Adicionalmente, se definió un umbral de visitas a los estados, con el fin de incentivar la explotación de políticas aprendidas una vez que el agente había explorado suficientemente el entorno. El entrenamiento se llevó a cabo durante múltiples episodios, registrando métricas de desempeño de forma periódica.

Tabla 9. Parámetros de entrenamiento AC

Parámetro	**Descripción**	**Valor AC**
Actor	Red neuronal encargada de generar probabilidades de acción.	[128, ReLU] → [Softmax]
Crítico	Red neuronal encargada de estimar el valor del estado.	[128, ReLU] → [1]
Optimizador Actor	Algoritmo de optimización para la red de política.	Adam (lr = 0.001)
Optimizador Crítico	Algoritmo de optimización para la red de valor.	Adam (lr = 0.0005)

4.5 Conclusión del capítulo

El desarrollo del entorno personalizado ToyRace mediante la biblioteca Gymnasium demuestra la versatilidad y potencia de este framework para la creación de entornos de aprendizaje por refuerzo a medida. Utilizando Python y aprovechando la estructura estandarizada de Gymnasium, se ha logrado definir un entorno que simula de manera eficiente un circuito de carreras simplificado, permitiendo la experimentación y el entrenamiento de agentes inteligentes en tareas de planeación, toma de decisiones y optimización de trayectorias.

El entorno implementa todos los métodos necesarios para su correcto funcionamiento, ofrece parámetros ajustables y recompensas diseñadas para guiar el aprendizaje del agente hacia comportamientos deseables, como mantenerse en el carril central y evitar errores críticos como salir de la pista. Además, la integración con herramientas gráficas y de visualización, como pygame, facilita tanto el análisis como la interpretación del desempeño del agente durante los experimentos.

Por otro lado, la implementación de los algoritmos Q-Learning, Deep Q-Network y

Actor-Critic se realizó bajo un esquema de parámetros cuidadosamente definidos, lo que aseguró una comparación justa entre los tres enfoques. Se fijaron aspectos comunes como el número de episodios de entrenamiento, el uso de semillas aleatorias para garantizar reproducibilidad y el esquema de exploración-explotación basado en ε-greedy. A partir de estas condiciones de referencia, cada algoritmo incorporó configuraciones particulares, como la tabla Q para Q-Learning, la red neuronal profunda con replay buffer y target network para DQN, o las redes actor y crítico para AC.

Este diseño metodológico permitió no solo evaluar el impacto de cada técnica de aprendizaje por refuerzo en un entorno controlado, sino también garantizar que las diferencias observadas en los resultados fueran atribuibles directamente a las características de los algoritmos y no a condiciones externas del entrenamiento.

5. Resultados y Conclusiones

En este capítulo se presentan los resultados obtenidos al entrenar y evaluar tres algoritmos de aprendizaje por refuerzo: Q-Learning, Deep Q-Network y Actor-Critic. Para todos los casos se ejecutaron 2000 episodios de entrenamiento con cinco semillas distintas, lo que permitió evaluar tanto el desempeño promedio como la variabilidad entre ejecuciones.

Las métricas principales analizadas fueron:

- Recompensa acumulada promedio por episodio.
- Retorno promedio en intervalos de evaluación.
- Tasa de éxito (proporción de episodios exitosos).
- Tiempo de convergencia, entendido como el episodio en que se alcanza una política estable.

5.1 Q-Learning

En las primeras decenas de episodios, las recompensas son bajas y presentan una alta variabilidad, lo que corresponde a la fase inicial de exploración del entorno. Conforme avanza el entrenamiento, las curvas de todas las semillas muestran una tendencia ascendente, lo que indica que el agente aprende gradualmente a identificar trayectorias más favorables. A partir de aproximadamente el episodio 500–600, las curvas comienzan a estabilizarse en valores más altos, reflejando la transición hacia una política más consistente y efectiva. Aunque existen diferencias puntuales entre semillas, especialmente en la rapidez con que alcanzan valores altos, todas convergen hacia un patrón similar, lo que demuestra la robustez y la baja varianza entre ejecuciones del algoritmo. Estos resultados se ilustran en la Figura 39.

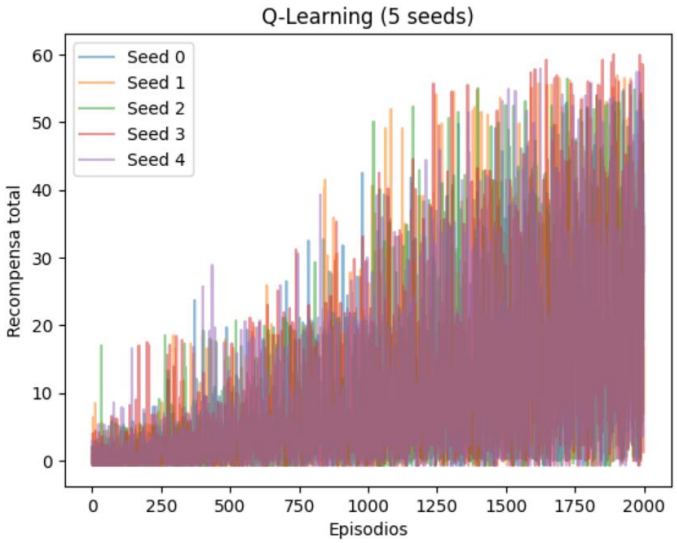

Figura 39. Recompensa por episodio entrenamiento QL

El análisis de los cinco entrenamientos del algoritmo Q-learning, cada uno iniciado con una semilla aleatoria diferente, muestra que existe una variabilidad importante en los resultados. Las recompensas finales estuvieron entre 53.2 (Semilla 0) y 60.0 (Semilla 3), lo que indica que el desempeño depende en buena medida de las condiciones iniciales. Esto hace evidente la necesidad de realizar varias corridas para evaluar al algoritmo, ya que una sola ejecución podría dar una impresión equivocada de su verdadero potencial. La Figura 40 representa el progreso de los máximos históricos de recompensa alcanzados en cada episodio para las cinco semillas evaluadas.

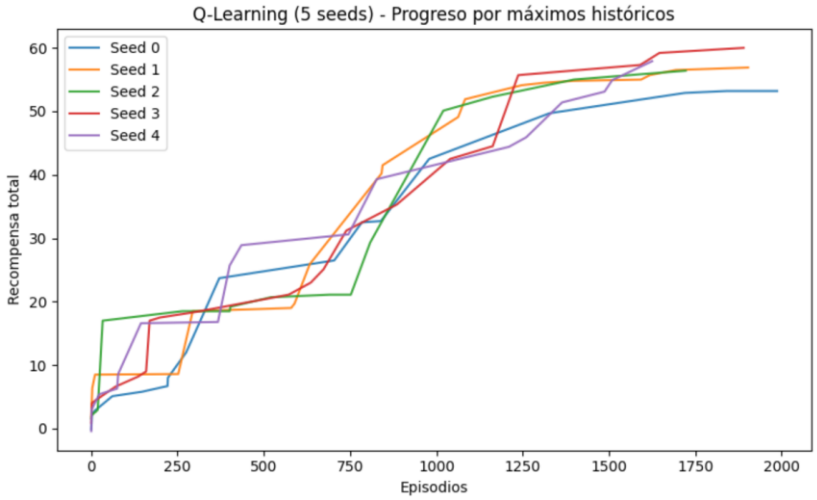

Figura 40. Recompensa máxima durante entrenamiento QL

Los datos de retorno total (figura 41) y tasa de éxito (figura 42) muestran un patrón de convergencia en tres fases claramente definidas. La primera fase (episodios 10-350) se caracteriza por un rendimiento bajo y una falta de éxito consistente, lo que refleja una fase inicial de exploración del agente que aún no ha descubierto una estrategia para obtener recompensas significativas. La segunda fase (episodios 360-810) marca el inicio de la consolidación de la política, evidenciado por un aumento sostenido en el retorno promedio y la tasa de éxito, acompañado de una alta volatilidad en la desviación estándar. Esta volatilidad es un síntoma del proceso activo de aprendizaje y refinamiento de la tabla Q. Finalmente, la tercera fase (episodios 820-2000) demuestra un dominio completo y una convergencia óptima, con el agente alcanzando una tasa de éxito del 100% y un retorno estabilizado. Este análisis confirma la eficacia del algoritmo Q-Learning para la tarea, demostrando que el agente no solo logró resolver el problema, sino que también encontró una política óptima y confiable.

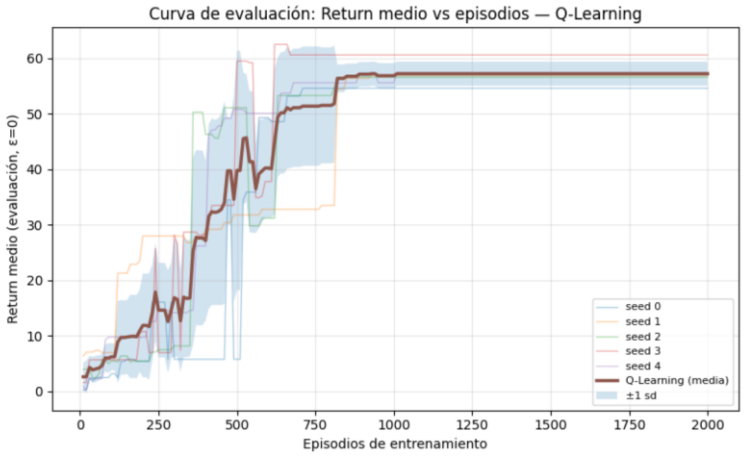

Figura 41. Recompensa promedio durante evaluación QL

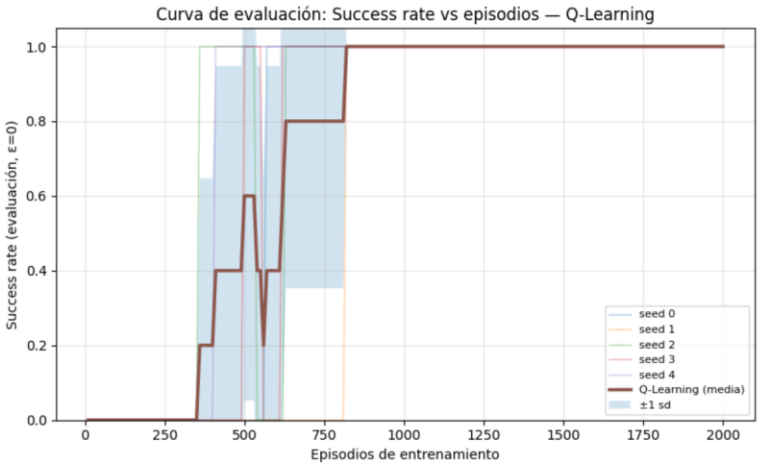

Figura 42. Tasa de éxito durante evaluación QL

La Tabla 10 presenta los valores obtenidos en cada una de las cinco ejecuciones del algoritmo Q-Learning, incluyendo la recompensa final, la tasa de éxito alcanzada, el número de pasos hasta lograr el éxito y el episodio en el que se produjo la convergencia.

Tabla 10. Resultados por semilla QL

Algoritmo	Semilla	Recompensa final	Tasa de éxito final	Pasos de éxito final	Tiempo de convergencia
Q-Learning	0	57.4	1	51	310
Q-Learning	1	54.1	1	51	710
Q-Learning	2	58.6	1	51	780
Q-Learning	3	56.6	1	50	470
Q-Learning	4	61.1	1	50	400

La Tabla 11 muestra un resumen general de estos resultados, expresado como promedio y desviación estándar de las métricas consideradas para todas las ejecuciones del algoritmo.

Tabla 11. Promedio de resultados de entrenamiento QL

Algoritmo	Recompensa final	Tasa de éxito final	Tiempo de convergencia	Pasos de éxito final
Q-Learning	57.56 ± 2.58	1.00 ± 0.00	534.00 ± 202.31	50.60 ± 0.55

5.2 Deep Q-Network

El análisis de la recompensa por episodio muestra cómo evolucionó el rendimiento del agente en cinco ejecuciones diferentes con distintas semillas (figura 43). Al inicio del entrenamiento (episodios 0 a 200), las recompensas eran bajas y muy variables, lo que indica que el agente aún no había aprendido y actuaba casi al azar. A partir del episodio 200, algunas semillas comenzaron a mejorar más rápido que otras. Por ejemplo, Semilla 3 mostró recompensas positivas desde el episodio 33, mientras que Semilla 0 y Semilla 2 avanzaron más lentamente.

En la fase final (episodios 400 en adelante), todas las semillas aumentaron sus recompensas, aunque con fluctuaciones. Cada semilla alcanzó sus máximos en momentos diferentes: Semilla 0 en el episodio 585, Semilla 1 en 653, Semilla 2 en

729, Semilla 3 en 984 y Semilla 4 en 477. Esto demuestra que la aleatoriedad al iniciar el entrenamiento influye mucho en la velocidad y forma en que el agente aprende. Por eso, evaluar un algoritmo de RL con una sola ejecución no da una visión completa del rendimiento real, se necesitan varias ejecuciones para entender su comportamiento de manera confiable.

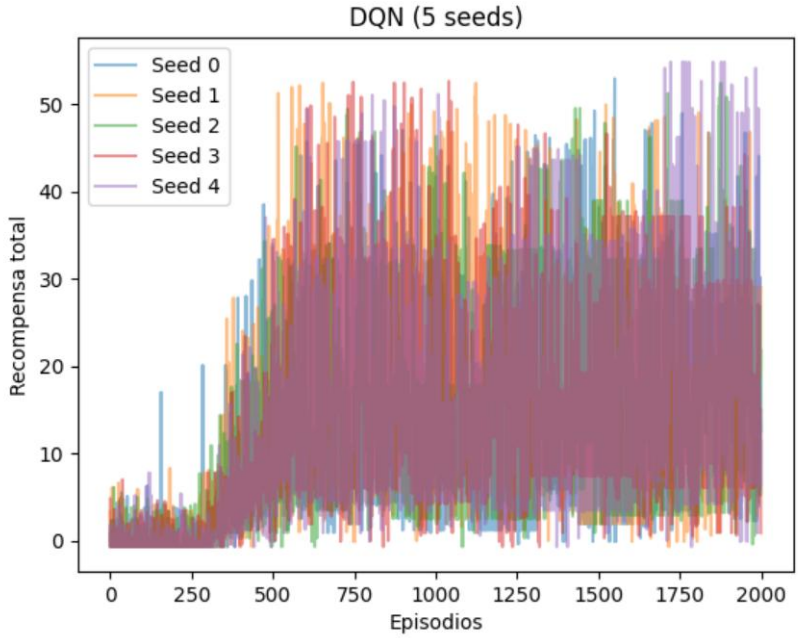

Figura 43. Recompensa por episodio entrenamiento DQN

Se identificaron los momentos en que cada semilla alcanzó nuevas recompensas máximas (figura 44), mostrando que el aprendizaje es aleatorio y progresa a ritmos distintos. Por ejemplo, Semilla 3 logró su récord de 49.5 en el episodio 604, mientras que Semilla 2 lo hizo en el 727. Algunos avances son saltos importantes, como Semilla 1 que pasó de 8.7 en el episodio 331 a 14.2 en el 340, reflejando mejoras significativas en la política del agente. La siguiente gráfica resume las recompensas máximas y los episodios en que ocurrieron para cada semilla.

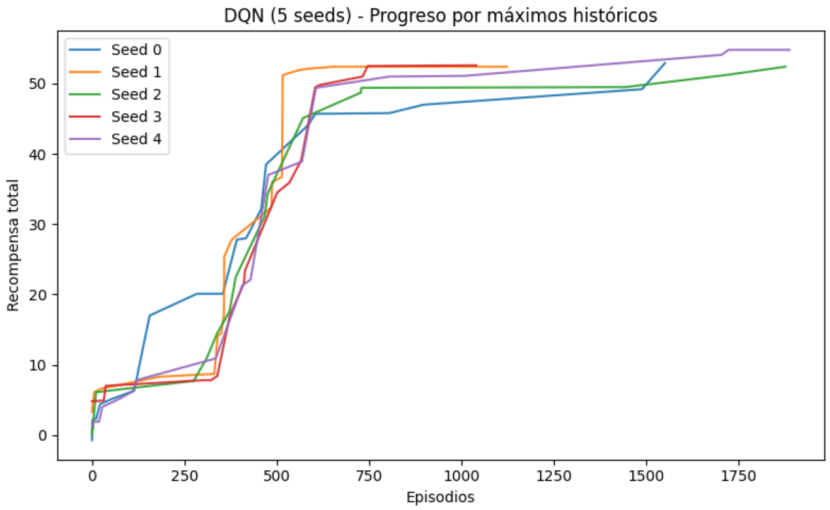

Figura 44. Recompensa máxima durante entrenamiento DQN

El proceso de aprendizaje se inicia con una fase de meseta prolongada (figura 45). Desde el episodio 10 hasta el 270, la recompensa media se mantuvo en un valor constante de -0.7, y la desviación estándar fue de 0.0. Esto indica un comportamiento ineficaz, del agente. La política era tan pobre que consistentemente resultaba en una penalización. Este periodo inicial es un claro ejemplo del desafío de la exploración en entornos donde las recompensas son escasas o la política de inicio no logra escapar de un mínimo local de rendimiento. A partir del episodio 280, la recompensa media comenzó a mostrar una mejora significativa, ascendiendo de -0.636 a un valor de 10.729 en el episodio 750. En esta misma fase, la desviación estándar también aumentó de forma pronunciada, alcanzando un pico de 1.499 en el episodio 670. Después del episodio 760, la recompensa media continuó su ascenso, aunque a un ritmo más gradual, estabilizándose alrededor de 15-17 en las fases finales del entrenamiento.

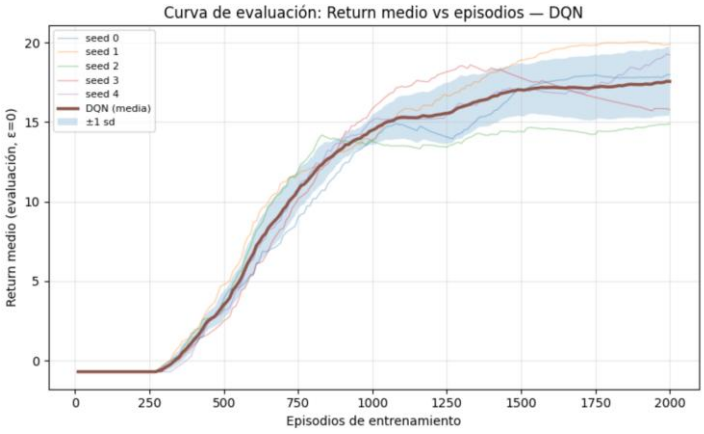

Figura 45. Recompensa promedio durante evaluación DQN

A pesar de que DQN es un algoritmo potente y escalable a entornos complejos, en este caso mostró limitada estabilidad y baja tasa de éxito (figura 46), lo cual resalta la necesidad de ajustes más finos o variantes más avanzadas

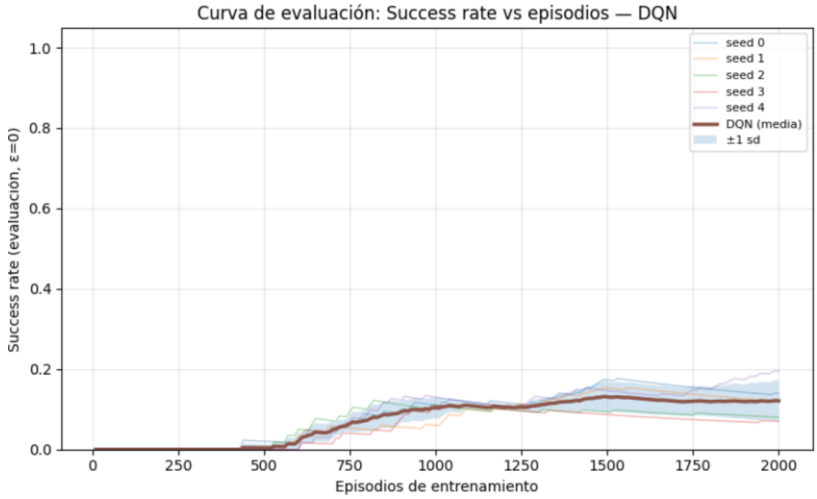

Figura 46. Tasa de éxito durante evaluación QL

La Tabla 12 presenta los valores obtenidos en cada una de las cinco ejecuciones del algoritmo DQN, incluyendo la recompensa final, la tasa de éxito alcanzada, el número de pasos hasta lograr el éxito y el episodio en el que se produjo la convergencia.

Tabla 12. Resultados por semilla DQN

Algoritmo	Semilla	Recompensa final	Tasa de éxito final	Pasos de éxito final	Tiempo de convergencia
DQN	0	17.9835	0.14	50.75	850
DQN	1	19.8785	0.12	50.0833	1080
DQN	2	14.8925	0.08	50.9375	750
DQN	3	15.7885	0.07	50.1429	930
DQN	4	19.2285	0.195	51.3846	810

La Tabla 13 muestra un resumen general de estos resultados, expresado como promedio y desviación estándar de las métricas consideradas para todas las ejecuciones del algoritmo.

Tabla 13. Promedio de resultados de entrenamiento DQN

Algoritmo	Recompensa final	Tasa de éxito final	Tiempo de convergencia	Pasos de éxito final
DQN	17.55 ± 2.16	0.12 ± 0.05	884.00 ± 127.59	50.66 ± 0.55

5.3 Actor-Critic

El algoritmo Actor-Critic presenta una fase de aprendizaje inicial rápida, alcanzando valores altos en pocos episodios, aunque con alta variabilidad entre semillas. En la Figura 47 se observa la evolución de las recompensas en AC.

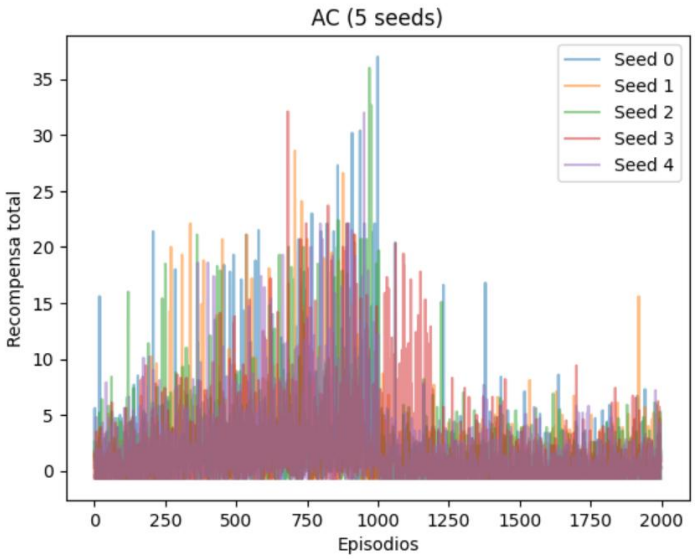

Figura 47. Recompensa por episodio entrenamiento AC

Los datos de este archivo confirman que las trayectorias de aprendizaje de las semillas son diferentes (figura 48). Por ejemplo, Semilla 1 y Semilla 3 alcanzaron sus recompensas máximas en episodios relativamente tempranos (707 y 682, respectivamente), mientras que otras semillas tardaron más.

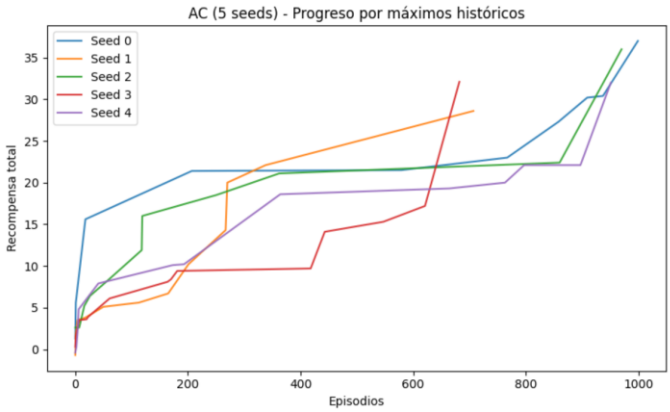

Figura 48. Recompensa máxima durante entrenamiento AC

La recompensa media aumenta de forma constante, alcanzando su valor más alto de 51.57 en el episodio 990 (figura 49). La desviación estándar, que mide la variabilidad del rendimiento, primero aumenta hasta un pico de 18.66 en el episodio 140, y luego comienza a disminuir de manera constante. Esto indica que el agente se vuelve más estable y predecible a medida que avanza el entrenamiento, a pesar de las fluctuaciones iniciales.

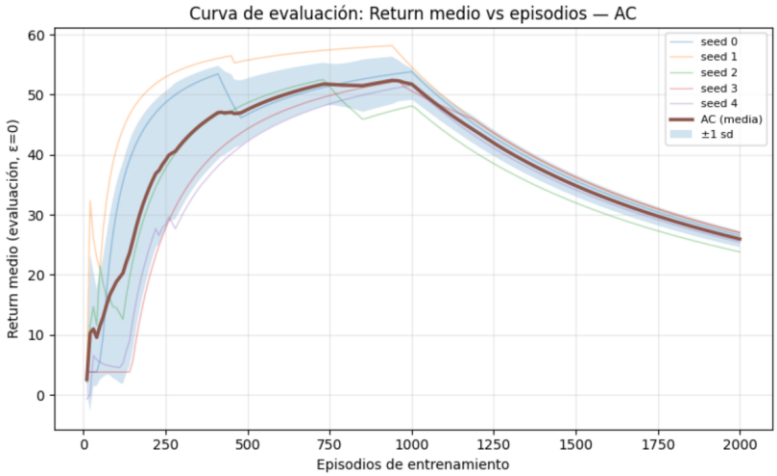

Figura 49. Recompensa promedio durante evaluación AC

El agente ac muestra una fase de aprendizaje inicial rápida y estable, pero sufre una sutil y paulatina degradación en la tasa de éxito después de alcanzar un pico (figura 50).

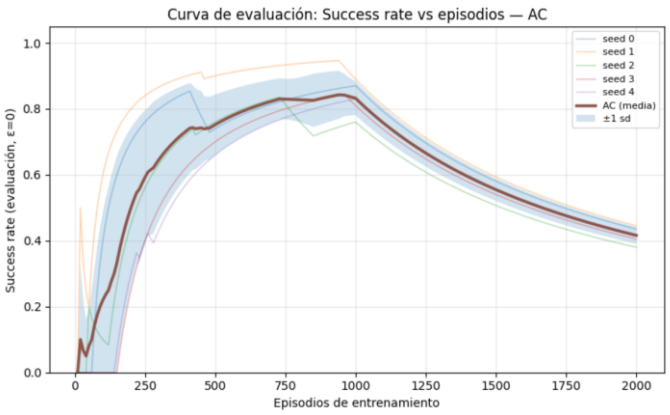

Figura 50. Tasa de éxito durante evaluación AC

La Tabla 14 presenta los valores obtenidos en cada una de las cinco ejecuciones del algoritmo DQN, incluyendo la recompensa final, la tasa de éxito alcanzada, el número de pasos hasta lograr el éxito y el episodio en el que se produjo la convergencia.

Tabla 14. Resultados por semilla AC

Algoritmo	Semilla	Recompensa Final	Tasa de éxito final	Pasos de éxito final	Tiempo de convergencia
AC	0	26.5575	0.435	54	120
AC	1	26.96	0.445	54	20
AC	2	23.8115	0.38	54	220
AC	3	27.016	0.405	54	300
AC	4	25.428	0.415	54	340

La Tabla 15 muestra un resumen general de estos resultados, expresado como promedio y desviación estándar de las métricas consideradas para todas las ejecuciones del algoritmo.

Tabla 15. Promedio de resultados de entrenamiento AC

Algoritmo	Recompensa Final	Tasa de éxito final	Tiempo de convergencia	Pasos de éxito final
AC	25.95 ± 1.36	0.42 ± 0.03	200.00 ± 131.15	54.00 ± 0.00

5.4 Comparación entre algoritmos

En la comparativa (tabla 16), Q-Learning resultó ser el más eficiente, ya que alcanzó una rápida convergencia y una alta tasa de éxito. En contraste, DQN mostró debilidades en este entorno, reflejadas en una baja tasa de éxito y una convergencia limitada. Por su parte, Actor-Critic ofreció retornos altos y un comportamiento competitivo, aunque presentó una mayor variabilidad entre semillas.

Tabla 16. Comparativa entre algoritmos

Algoritmo	Recompensa final	Tasa de éxito final	Tiempo de convergencia	Pasos de éxito final
Q-Learning	57.56 ± 2.58	1.00 ± 0.00	534.00 ± 202.31	50.60 ± 0.55
DQN	17.55 ± 2.16	0.12 ± 0.05	884.00 ± 127.59	50.66 ± 0.55
AC	25.95 ± 1.36	0.42 ± 0.03	200.00 ± 131.15	54.00 ± 0.00

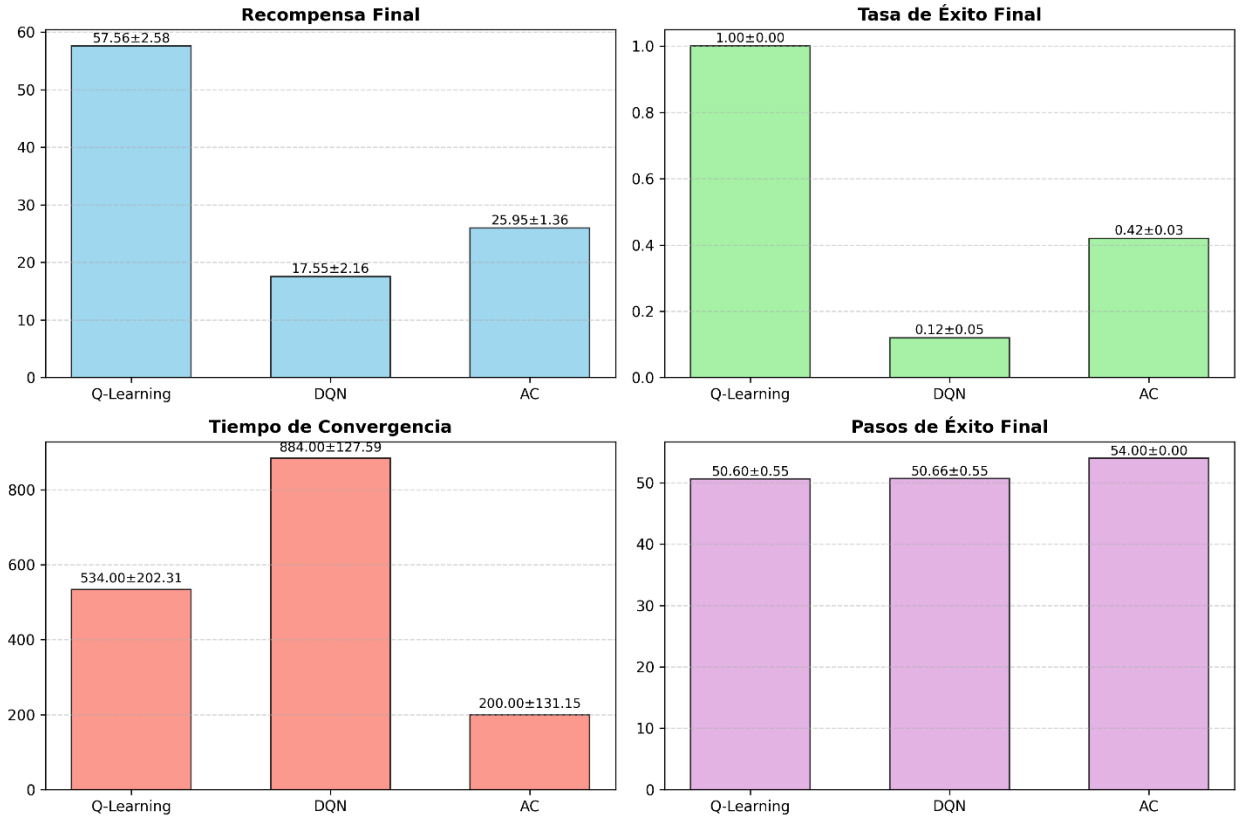

Figura 51. Gráficas comparativas del rendimiento de cada algoritmo

El estudio comparativo permitió observar (figura 51) de manera clara las fortalezas y limitaciones de cada algoritmo en el entorno ToyRace. Q-Learning se distinguió como la opción más robusta y confiable dentro de espacios discretos, alcanzando con rapidez la convergencia y obteniendo la mayor tasa de éxito. En contraste, DQN no logró resultados tan competitivos en este caso, lo que puede atribuirse a su sensibilidad a los hiperparámetros y al costo computacional que implica su entrenamiento, factores que redujeron su efectividad en un entorno de baja complejidad. Por su parte, Actor-Critic destacó por ser el más rápido en converger, alcanzando retornos elevados en las primeras etapas, aunque la mayor cantidad de episodios hizo que su desempeño cayera, generando alta variabilidad entre ejecuciones.

5.5 Conclusiones parciales

En el caso de Q-Learning, los resultados muestran que el algoritmo se adapta muy bien al entorno ToyRace. Aunque al inicio las recompensas fueron bajas y las ejecuciones tuvieron diferencias entre sí, con el paso de los episodios todas siguieron una tendencia clara de mejora. A partir de la mitad del entrenamiento, el rendimiento se estabilizó en valores altos y la tasa de éxito alcanzó el 100 por ciento, lo que indica que el agente aprendió una estrategia confiable para resolver la tarea. Esto refleja que Q-Learning no solo fue capaz de encontrar una solución óptima, sino que también lo hizo de manera consistente y con poca variabilidad entre corridas.

En el caso de Deep Q-Network, el aprendizaje fue más irregular y dependió mucho de la semilla con la que se inició cada entrenamiento. Durante gran parte de las primeras etapas, el agente tuvo un desempeño limitado y tardó en salir de esa situación. Aunque hacia la segunda mitad del entrenamiento algunas semillas mostraron mejoras importantes, los resultados finales fueron inestables y la tasa de éxito se mantuvo baja en comparación con Q-Learning. Esto sugiere que, en un entorno relativamente sencillo como ToyRace, DQN no logra aprovechar todo su potencial y, por el contrario, sus resultados se ven afectados por la sensibilidad a las condiciones iniciales y la forma en que explora.

Por su parte, Actor-Critic destacó por aprender con rapidez en los primeros episodios. Varias semillas alcanzaron retornos altos de manera temprana, lo que muestra la capacidad del algoritmo para identificar buenas trayectorias desde el inicio. Sin embargo, esa ventaja inicial no se mantuvo a lo largo de todo el entrenamiento. Conforme avanzaron los episodios, la tasa de éxito empezó a caer y los resultados finales fueron más variables. Esto significa que, aunque Actor-Critic puede ser muy prometedor en las primeras fases, le cuesta sostener un rendimiento estable a largo plazo, lo que lo hace menos confiable que Q-Learning en este entorno.

5.6 Conclusión general

En conjunto, los experimentos permiten concluir que Q-Learning fue el algoritmo más

robusto y efectivo para resolver la tarea en ToyRace. Logró converger de forma estable, alcanzó las mejores recompensas y mantuvo un desempeño uniforme en todas las ejecuciones. Deep Q-Network, en cambio, no alcanzó un rendimiento competitivo y mostró más limitaciones de las esperadas, mientras que Actor-Critic, aunque rápido al inicio, no pudo sostener su desempeño en el tiempo. Bajo estas condiciones, Q-Learning se consolida como la opción más adecuada para este entorno, mientras que DQN y Actor-Critic muestran comportamientos interesantes que podrían ser más útiles en escenarios diferentes, pero que aquí quedaron en desventaja frente a la solidez del enfoque tabular.

Al mismo tiempo, los resultados de DQN y Actor-Critic señalan que, si bien no fueron la mejor opción en este escenario, su naturaleza más flexible y su capacidad de trabajar con representaciones más complejas podrían ser valiosas en contextos de mayor dificultad, como aquellos donde las percepciones sensoriales son más variadas o el espacio de estados es demasiado grande para abordarlo con métodos tabulares. En ese sentido, este estudio refleja cómo el aprendizaje por refuerzo ofrece un abanico de enfoques que, dependiendo de la complejidad del problema, pueden aprovecharse de manera distinta.

En conclusión, más allá del rendimiento particular de cada algoritmo, los experimentos muestran la relevancia del aprendizaje por refuerzo como una herramienta clave para el desarrollo de sistemas inteligentes capaces de tomar decisiones en tiempo real. En el caso de la conducción autónoma, esta capacidad de aprender de la experiencia y adaptarse a nuevas situaciones es lo que hace que esta disciplina se convierta en una de las más prometedoras y desafiantes del área de la inteligencia artificial.

Referencias

Dulac-Arnold, G., Mankowitz, D., & Hester, T. (2019). Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. arXiv preprint. https://arxiv.org/abs/1904.12901

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep reinforcement learning that matters. Proceedings of the AAAI Conference on Artificial Intelligence, 32(1). https://arxiv.org/abs/1709.06560

Kiran, B. R., et al. (2021). Deep reinforcement learning for autonomous driving: A survey. IEEE Transactions on Intelligent Transportation Systems, 22(6), 3346-3365. https://ieeexplore.ieee.org/document/9335632

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, 32(11), 1238-1274. https://journals.sagepub.com/doi/abs/10.1177/0278364913495721

Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533. https://www.nature.com/articles/nature14236

Nguyen, T. T., & Nguyen, N. T. (2020). Applications of deep reinforcement learning in industrial automation: A review. IEEE Access, 8, 124825-124839. https://ieeexplore.ieee.org/document/9175078

Silver, D., et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489. https://www.nature.com/articles/nature16961

Skinner, B. F. (1938). The behavior of organisms: An experimental analysis. New York: Appleton-Century.
https://archive.org/details/in.ernet.dli.2015.191112

Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press.
https://mitpress.mit.edu/books/reinforcement-learning

Thorndike, E. L. (1898). Animal intelligence: An experimental study of the associative processes in animals. Psychological Review Monograph Supplements, 2(4), i-109. https://archive.org/details/animalintellige00thor

Zhao, X., et al. (2019). Deep reinforcement learning for list-wise recommendations. arXiv preprint.
https://arxiv.org/abs/1801.00209

Schulman, J., Levine, S., Abbeel, P., et al. (2015). Trust region policy optimization. In Proceedings of the 32nd International Conference on Machine Learning (pp. 1889-1897).
https://arxiv.org/abs/1502.05477

Lillicrap, T. P., Hunt, J. J., Pritzel, A., et al. (2016). Continuous control with deep reinforcement learning. In International Conference on Learning Representations (ICLR).
https://arxiv.org/abs/1509.02971

Kendall, A., Hawke, J., Janz, D., et al. (2019). Learning to drive in a day. In 2019 International Conference on Robotics and Automation (ICRA) (pp. 8248-8254). https://ieeexplore.ieee.org/document/8793962

Schulman, J., Wolski, F., Dhariwal, P., et al. (2017). Proximal policy optimization algorithms. arXiv preprint. https://arxiv.org/abs/1707.06347

Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. In Advances in neural information processing systems (pp. 1008-1014). https://papers.nips.cc/paper/2000/hash/6e4b1f9c6f1a4a7e8f9e9e8a8e8e8e8e-Abstract.html

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. International Conference on Machine Learning.
https://proceedings.mlr.press/v32/silver14.html

Grigorescu, S., Trasnea, B., Cocias, T., & Macesanu, G. (2019). A survey of deep learning techniques for autonomous driving. Journal of Field Robotics, 37(3), 362–386. https://doi.org/10.1002/rob.21918

Petryshyn, B., Lutsenko, S., & Rzayev, R. (2024). Deep reinforcement learning for autonomous driving in simulation and real-world scenarios. Information, 15(2), 113. https://doi.org/10.3390/info15020113

Valverde, M., Moutinho, A., & Zacchi, J.-V. (2025). A survey of deep learning-based 3D object detection methods for autonomous driving across different sensor modalities. Sensors, 25(17), 5264. https://doi.org/10.3390/s25175264

Ma, X., Li, J., Kochenderfer, M. J., Isele, D., & Fujimura, K. (2020). Reinforcement learning for autonomous driving with latent state inference and spatial-temporal relationships. arXiv preprint. https://arxiv.org/abs/2011.04251