Universidad de Valladolid

Máster universitario

Ingeniería Informática

Trabajo Fin de Máster

Toma de decisiones de estrategia durante las carreras de Fórmula 1 utilizando *Deep Reinforcement Learning*

Realizado por **Jorge Rebé Martín**

✠✠✠

Universidad de Valladolid

27 de junio de 2024

Tutores: Aníbal Bregón Bregón y José Belarmino Pulido Junquera

Universidad de Valladolid

Máster universitario en Ingeniería Informática

D. Aníbal Bregón Bregón y D. José Belarmino Pulido Junquera, profesores del departamento de INFORMÁTICA (ATC, CCIA Y LSI).

Exponen:

Que el alumno D. Jorge Rebé Martín, ha realizado el Trabajo final de Máster en Ingeniería Informática titulado "Toma de decisiones de estrategia durante las carreras de Fórmula 1 utilizando *Deep Reinforcement Learning*".

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección de los que suscriben, en virtud de lo cual se autoriza su presentación y defensa.

En Valladolid, 27 de junio de 2024

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. Aníbal Bregón Bregón	D. José Belarmino Pulido Junquera

Resumen

La estrategia de carrera en Fórmula 1 es uno de los elementos críticos que pueden cambiar el resultado de un piloto en una carrera. En este trabajo se ha construido un sistema basado en aprendizaje por refuerzo en el que un agente aprende a tomar las decisiones estratégicas óptimas en cada vuelta (parar o no parar, y si se para, qué neumático poner).

Se ha desarrollado un simulador de carreras de Fórmula 1 que se ha utilizado como entorno para que el agente interactúe con él y pueda aprender cuáles son las acciones a tomar en cada momento que maximicen la posición final en carrera. Se utilizan varios algoritmos (DQN, QR-DQN y A2C) y varias funciones de recompensa con los que se entrenará a varios agentes. Finalmente, se evalúan los agentes entrenados y se selecciona el mejor.

Descriptores

Aprendizaje por Refuerzo, Fórmula 1, optimización de estrategia de carrera.

Abstract

Race strategy in Formula One is one of the most critic elements that can change the final position of a driver in a race. In this work a full reinforcement learning system is built, in which the agent learns to take the optimal strategy decisions every lap (to pit or not to pit, and if pit what tyre to fit).

A Formula 1 races simulator has been developed, and it is used as the environment that the agent interacts with in order to learn what actions to take given the race state in order to maximize the final position in the race. Various algorithms (DQN, QR-DQN and A2C) and reward functions are used, with which agents will be trained. Finally, trained agents are evaluated and the best one is selected.

Keywords

Reinforcement Learning, Formula 1, race strategy optimization.

Índice general

Índice general III

Índice de figuras VI

Índice de tablas VIII

1. Introducción 1
- 1.1. Problema a Resolver . . . 2
- 1.2. Objetivos del trabajo . . . 2
- 1.3. Estructura de la memoria . . . 3

2. Planificación 4
- 2.1. Metodología . . . 4
  - 2.1.1. Scrum . . . 4
  - 2.1.2. Adaptación de Scrum al proyecto . . . 6
- 2.2. Planificación del trabajo . . . 6
- 2.3. Presupuesto . . . 7
- 2.4. Herramientas . . . 10

3. Preliminares 11
- 3.1. *Reinforcement Learning* . . . 11
  - 3.1.1. Elementos de un problema de Reinforcement Learning . . . 11
  - 3.1.2. *Markov Decision Processes* . . . 11
  - 3.1.3. Optimalidad . . . 15
  - 3.1.4. Ajustes en el Aprendizaje por Refuerzo . . . 16
  - 3.1.5. Programación Dinámica . . . 16
  - 3.1.6. Métodos de Monte Carlo . . . 20
  - 3.1.7. *Temporal-Difference learning* . . . 20
- 3.2. *Deep Reinforcement Learning* . . . 21
  - 3.2.1. Deep Q Learning . . . 22

3.2.2. *Policy Gradient Methods* 24

4. Desarrollo del Simulador **27**
4.1. Modelo de Dominio 27
4.1.1. Circuito 28
4.1.2. Combustible 28
4.1.3. Neumáticos, PitStops y Estrategias 28
4.1.4. Piloto 31
4.1.5. Carrera 31
4.2. Estado 31
4.2.1. Variables de estado 31
4.2.2. Estado Inicial (t = 0) 33
4.2.3. Estados Intermedios (t >0 & t <n_vueltas-1) 33
4.2.4. Estado Final (t = n_vueltas-1) 33
4.3. Acciones 33
4.4. Funcionamiento del simulador 34
4.4.1. Avance en el simulador (t $\geq$ 0 & t <n_vueltas-1, a $\in A$) 34
4.4.2. Pseudocódigo 36
4.4.3. Variabilidad 37
4.5. Restricciones 37
4.6. Ejemplo de Simulación 38
4.7. Adaptación del Simulador a un entorno Gymnasium 39
4.7.1. Integración con el Simulador 39

5. Diseño Experimental **40**
5.1. Observación 40
5.1.1. Normalización de la observación 40
5.2. Configuración del Simulador 41
5.2.1. Neumáticos 41
5.2.2. Estrategias 42
5.2.3. Adelantamientos, Pits y Combustible 44
5.3. Algoritmos 44
5.3.1. DQN 44
5.3.2. QR - DQN 45
5.3.3. A2C 46
5.3.4. Guardado del mejor modelo durante el entrenamiento 47
5.4. Funciones de recompensa 47
5.4.1. Elementos comunes a todas las funciones de recompensa 48
5.4.2. R1 - Posiciones ganadas por vuelta 48
5.4.3. R2 - Posiciones ganadas por vuelta 48
5.4.4. R3 - Recompensa por posición final 49

6. Experimentación **50**
6.1. Experimentos 50

6.1.1. Ejecución de los experimentos 50
6.1.2. Comparación de Algoritmos 57

7. Evaluación **61**
7.1. Evaluación y Comparación de Modelos 61
7.2. Muestra del rendimiento del mejor modelo 63
7.2.1. Simulación de ejemplo 63
7.2.2. Política del agente 65

8. Conclusiones y Líneas de trabajo futuras **67**
8.1. Conclusiones 67
8.2. Trabajo Futuro 68

Bibliografía **70**

Apéndices **72**

Apéndice A Ejemplo de Simulación de Carrera **74**

Índice de figuras

3.1. Interacción agente-entorno en un MDP . . . 12
3.2. Esquema de GPI [25]. . . . 19
4.1. Modelo de Dominio del simulador. . . . 28
5.1. Efecto del desgaste de los neumáticos. . . . 42
5.2. Estrategias de los pilotos en el Gran Premio de Mónaco del año 2021. . . . 43
6.1. Recompensa media por episodio - Experimento 1 - DQN R1 . . . 51
6.2. Recompensa media por episodio - Experimento 2 - DQN R2 . . . 51
6.3. Recompensa media por episodio - Experimento 3 - DQN R3 - Entrenamiento con Orden de Salida Estable . . . 52
6.4. Recompensa media por episodio - Experimento 4 - DQN R3 - Entrenamiento con Orden de Salida Aleatorio . . . 53
6.5. Recompensa media por episodio - Experimento 5 - QR-DQN R1 . . . 53
6.6. Recompensa media por episodio - Experimento 6 - QR-DQN R2 . . . 54
6.7. Recompensa media por episodio - Experimento 7 - QR-DQN R3 - Entrenamiento con Orden de Salida Estable . . . 54
6.8. Recompensa media por episodio - Experimento 8 - QR-DQN R3 - Entrenamiento con Orden de Salida Aleatorio . . . 55
6.9. Recompensa media por episodio - Experimento 9 - A2C R1 . . . 55
6.10. Recompensa media por episodio - Experimento 10 - A2C R2 . . . 56
6.11. Recompensa media por episodio - Experimento 11 - A2C R3 - Entrenamiento con Orden de Salida Estable . . . 56
6.12. Recompensa media por episodio - Experimento 12 - A2C R3 - Entrenamiento con Orden de Salida Aleatorio . . . 57
6.13. Comparación de la velocidad y estabilidad y capacidad de aprendizaje de los algoritmos para la función de recompensa R1. . . . 58
6.14. Comparación de la velocidad y estabilidad y capacidad del aprendizaje de los algoritmos para la función de recompensa R2. . . . 58
6.15. Comparación de la velocidad y estabilidad y capacidad del aprendizaje de los algoritmos para la función de recompensa R3, empezando los episodios con orden de salida de los pilotos estable. . . . 59

6.16. Comparación de la velocidad y estabilidad y capacidad del aprendizaje de los algoritmos para la función de recompensa R3, empezando los episodios con orden de salida de los pilotos aleatorio. 60
7.1. Cambio de posiciones durante la simulación de carrera. 64
7.2. Estrategias de los pilotos en la simulación de carrera. 65

Índice de tablas

2.1. Planificación del trabajo por sprints. 8
2.2. Presupuesto de los recursos técnicos 9
2.3. Presupuesto de los recursos humanos. 9
4.1. Descripción de la clase que representa un circuito en el que se celebra una carrera. 29
4.2. Descripción de la clase que representa el combustible usado por el coche de un piloto en una carrera. 29
4.3. Descripción de la clase que representa el compuesto de un neumático en el simulador. 30
4.4. Descripción de la clase que representa el neumático que lleva el coche de un piloto en una carrera en un momento determinado en el simulador. 30
4.5. Descripción de la clase que representa un pitstop que realiza un piloto en un momento determinado en una carrera del simulador. 30
4.6. Descripción de la clase que representa la estrategia seguida por un piloto en una carrera en el simulador. 31
4.7. Descripción de la clase que representa a un piloto que participa en una carrera en el simulador 32
4.8. Descripción de la clase que representa una carrera en el simulador. 33
5.1. Estrategias posibles en la configuración del simulador elegida para los experimentos 43
5.2. Recompensa obtenida según la posición final en la carrera. 49
7.1. Posición final media y desviación estándar en la evaluación durante 1000 carreras para los casos base. En cada carrera de evaluación el orden de salida y el neumático de salida son aleatorios y para los los pilotos que no se están controlando las estrategias son predefinidas. 62
7.2. Posición final media y desviación estándar en la evaluación durante 1000 carreras para cada uno de los modelos obtenidos en los experimentos. En cada carrera de evaluación el orden de salida y el neumático de salida son aleatorios y para los los pilotos que no se están controlando las estrategias son aleatorias. 62
7.3. Política del agente en la simulación de ejemplo en la vuelta 10. 66
7.4. Política del agente en la simulación de ejemplo en la vuelta que decide parar (vuelta 33). 66

7.5. Política del agente en la simulación de ejemplo cuando faltan 10 vueltas para el final (vuelta 68). 66

1: Introducción

La Fórmula 1 (en adelante, "F1") es una categoría deportiva de automovilismo, donde actualmente compiten 10 equipos, con 2 coches cada uno, con el objetivo de conseguir la mejor posición en cada carrera. Según Mike Law [16], ingeniero del equipo McLaren, hay seis elementos que pueden influir en la posición final de un coche: el peso, los neumáticos, la potencia, la aerodinámica, el piloto y la fiabilidad.

Los neumáticos son la única parte del coche que está en contacto con el suelo, y son por tanto son uno de los elementos más relevantes. El agarre que tienen con el asfalto, que estén en una ventana de temperatura óptima para su funcionamiento y el desgaste que sufren con el paso de las vueltas son algunos de los factores que más influyen en el rendimiento de un coche en una carrera.

Cuando los neumáticos se han desgastado lo suficiente o las condiciones de carrera son propicias para ello, los pilotos entran a boxes [1] a cambiar neumáticos, teniendo en cuenta que esto supone una pérdida de tiempo en esa vuelta, que será más o menos grande según las circunstancias de la carrera. Decidir el número de paradas, vueltas en las que se va a parar y el neumático a poner es lo que se conoce como la estrategia de una carrera.

La estrategia de paradas en boxes es una de las cosas que más puede influir en el resultado de la carrera: parar una vuelta antes que tus rivales (de tal manera que vas más rápido que ellos durante unas vueltas, y cuando ellos paren en boxes salen detrás, lo que se conoce como undercut), poner un neumático distinto, o alargar la vida útil de un neumático para parar menos veces puede significar la diferencia entre ganar una carrera o quedarse fuera del podio. Evidentemente, casi cualquier estrategia es buena cuando se tiene el coche más rápido, pero cuanto más próximos entre sí estén los rendimientos de los coches, mayor es la influencia de la estrategia en el resultado final.

[1] Los boxes, o pits, son los garajes donde entran los pilotos, a través del 'pit lane' o carril de boxes durante la carrera para cambiar neumáticos.

1.1. Problema a Resolver

Las decisiones estratégicas se van actualizando prácticamente en cada momento, teniendo en cuenta la situación de carrera: si hay un *Safety Car* [2] en pista, si nuestros rivales (ya sea los que están detrás o los que están delante) han parado o no, tiempo perdido si se para a cambiar neumáticos, posiciones perdidas tras la parada y facilidad de adelantamiento, etcétera. Hay un gran número de factores que hay que tener en cuenta de cara a decidir si se para o no y, a pesar de que los equipos cuentan con una gran cantidad de información, esas decisiones las toman personas.

Esto implica que, a veces, incluso los ingenieros de estrategia con la mejor experiencia, capacidad y conocimiento puedan cometer errores, y llegan a tomar decisiones que implican una pérdida de puntos o incluso de una victoria que sería muy valiosa para el equipo.

Por esto, se plantea la creación de un sistema basado en aprendizaje por refuerzo que de soporte los ingenieros de estrategia para facilitar la toma de decisiones en las carreras de F1. No se pretende sustituir a los ingenieros de estrategia, cuya experiencia es imprescindible para la toma correcta de decisiones, sino que se pretende darles una herramienta que les ayude a tomar esa decisión.

1.2. Objetivos del trabajo

En este Trabajo Fin de Máster se plantean los siguientes objetivos:

OBJ-1: Entrenar varios agentes utilizando aprendizaje por refuerzo para dar soporte a la toma de decisiones estratégicas en carreras de Fórmula 1.

OBJ-2: Evaluar y comparar los agentes entrenados.

Dentro de estos objetivos, podemos distinguir las siguientes tareas:

- **T1**: Crear de un simulador de carreras de Fórmula 1 para poder utilizarlo como entorno para entrenar al agente.
- **T2**: Definir varias funciones de recompensa para tratar maximizar el aprendizaje del agente.
- **T3**: Utilizar diferentes algoritmos de aprendizaje por refuerzo para entrenar el agente y poder evaluarlo.
- **T4**: Evaluar los diferentes agentes entrenados.

[2] El Safety Car es un coche que sale a pista para reagrupar a los pilotos y reducir la velocidad de carrera por motivos de seguridad.

1.3. Estructura de la memoria

El presente documento se va a estructurar por capítulos de la siguiente manera:

Capítulo 1 - Introducción : Explica el problema que se pretende resolver y se definen los objetivos del proyecto.

Capítulo 2 - Planificación : Se describen las herramientas utilizadas, la planificación temporal seguida en el proyecto y el presupuesto necesario para llevarlo a cabo.

Capítulo 3 - Preliminares : Introduce los conceptos básicos necesarios para entender el Aprendizaje Por Refuerzo y la solución desarrollada para el problema propuesto.

Capítulo 4 - Desarrollo del simulador : Explica el funcionamiento del simulador de carreras de F1 que se ha utilizado como entorno para que el agente interactúe con él y aprenda.

Capítulo 5 - Diseño Experimental : Presenta los diferentes experimentos que se van a realizar, indicando algoritmos, funciones de recompensa y condiciones experimentales.

Capítulo 6 - Experimentación : Expone los experimentos realizados, con las gráficas de aprendizaje durante la ejecución de cada uno de ellos.

Capítulo 7 - Evaluación : Muestra los modelos obtenidos durante la experimentación, y se comparan entre sí y con unas estrategias aleatorias de referencia.

Capítulo 8 - Conclusiones y Líneas de trabajo futuras : Reflexiones finales sobre el proyecto realizado y planteamiento de la continuación del trabajo para la mejora de los resultados.

Además, se incluyen los siguientes apéndices:

Apéndice A - Ejemplo de Simulación de Carrera : Se incluye un ejemplo de simulación de carrera para comprender el funcionamiento del simulador de carreras desarrollado.

2: Planificación

2.1. Metodología

La metodología escogida para el desarrollo de este proyecto ha sido Scrum, adaptado para que sea adecuado para las características del Trabajo Fin de Máster. A continuación se explican algunos conceptos básicos de Scrum, y después cómo se ha adaptado para que sea adecuado para este proyecto.

2.1.1. Scrum

Scrum [28] es una implementación de la mentalidad Agile [1], un marco de referencia. Scrum emplea un enfoque iterativo e incremental para optimizar la predictibilidad y controlar el riesgo. No se pretende hacer una descripción detallada de Scrum (que puede consultarse en [28]), pero se exponen a continuación conceptos básicos que se van a utilizar en este proyecto.

Roles

A continuación se explican cada uno de los tres roles dentro de un equipo Scrum.

- **Product Owner**: Es el responsable de maximizar el valor del producto resultante del trabajo del equipo Scrum. Sus decisiones se reflejan en el contenido y orden de los elementos del *Product Backlog*, artefacto que se describirá posteriormente.
- **Scrum Master**: Es el responsable de impulsar todas las prácticas de Scrum dentro del equipo y de la eficacia del mismo.
- **Desarrolladores**: Son las personas dentro del equipo comprometidas con la creación de cualquier aspecto de un Incremento que aporte valor cada Sprint.

Eventos

Los eventos de Scrum también tienen como objetivo crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. Estos son los eventos de Scrum:

- **Sprint**: Es un contenedor para el resto de eventos. Tienen una duración fija de un mes o menos, y un Sprint nuevo comienza inmediatamente después de la finalización del anterior.
- **Sprint Planning**: Es el evento que da inicio al Sprint, en el que se planifica el trabajo a realizar.
- **Daily Scrum**: Es una reunión diaria con una duración de aproximadamente 15 minutos para los desarrolladores del equipo Scrum. En ella, cada miembro del equipo de desarrollo informa de su progreso desde la última *Daily Scrum* y de su plan de trabajo hasta la siguiente.
- **Sprint Review**: Su objetivo es inspeccionar el resultado del Sprint y determinar las adaptaciones que hay que realizar. En esta reunión se revisa lo logrado durante el Sprint y se habla sobre qué hacer en el siguiente. Si es necesario, se modifica el *Product Backlog*.
- **Sprint Retrospective**: Es la reunión con la que termina el Sprint, en la que el equipo Scrum evalúa lo que ha ido bien durante el Sprint, problemas encontrados, y cómo se resolvieron (o no) esos problemas.

Artefactos

Los artefactos Scrum representan trabajo o valor dentro del proyecto. En Scrum se definen los siguientes artefactos:

- **Product Backlog**: Es una lista ordenada por prioridad de lo que se necesita para mejorar el producto. Los elementos del *Product Backlog* que se pueden completar en un Sprint forman parte de los que se seleccionan durante el *Sprint Planning*.
- **Sprint Backlog**: Es un plan hecho por y para los desarrolladores, una imagen que los desarrolladores planean completar para conseguir el *Sprint Goal*. Está compuesto del *Sprint Goal* (el objetivo del Sprint), los elementos del *Product Backlog* seleccionados (qué se realizará), y el plan para la entrega del incremento (el cómo).
- **Incremento**: Es un paso concreto hacia el objetivo del proyecto. Es posible que haya varios incrementos dentro del mismo Sprint, y el total de los incrementos es presentado durante la *Sprint Review*.

2.1.2. Adaptación de Scrum al proyecto

Debido a las características especiales del proyecto, como la limitación del personal, se ha tenido que realizar una adaptación de Scrum. En cuanto a roles, en este proyecto sólo habrá tres miembros involucrados en el proyecto: el estudiante, autor del TFM, y los dos tutores. El estudiante será a la vez *Product Owner* y equipo de desarrollo, mientras que los tutores cumplirán el papel del *Scrum Master*.

Los eventos *Sprint Planning*, *Sprint Review* y *Sprint Retrospective* tendrán lugar cada mes, el mismo día que finaliza el sprint y comienza el siguiente. Las *Daily Scrum* se han convertido en *Weekly Scrum*, ya que estas reuniones serán semanales.

2.2. Planificación del trabajo

El proyecto se va a desarrollar en el marco de las prácticas curriculares de la asignatura I+D+I (190 horas) y de la asignatura del Trabajo Fin de Máster (150 horas), con un total de 340 horas. Se desarrollará desde el mes de enero hasta junio, ambos incluidos. El proyecto deberá estar terminado antes del 28 de junio, fecha del cierre de actas de la primera convocatoria de la asignatura del Trabajo Fin de Máster. Se planifican 5 sprints, siendo uno de ellos un Sprint Zero [15] de 2 meses de duración, y el resto con una duración de 1 mes.

Para cumplir con los objetivos del trabajo, se plantean las siguientes épicas y las tareas en las que se descomponen, que se abordarán en los distintos sprints del proyecto:

- E0. Formación previa.
  - T0.1 Estudio sobre aprendizaje por refuerzo.
- E1. Desarrollo del simulador de carreras de F1.
  - T1.1 Análisis del simulador.
  - T1.2 Diseño del simulador.
  - T1.3 Implementación del simulador.
  - T1.4 Pruebas del simulador.
- E2. Diseño de funciones de recompensa.
  - T2.1 Estudio de funciones de recompensa.
  - T2.2 Selección de funciones de recompensa.
- E3. Uso de distintos algoritmos para entrenar agentes.
  - T3.1 Selección de los algoritmos y sus hiperparámetros.
  - T3.2 Entrenamiento de los agentes.

- E4. Evaluación de los agentes entrenados.
  - T4.1 Diseño de la evaluación.
  - T4.2 Evaluación de los agentes.
  - T4.3 Selección del mejor agente.
- E5. Redacción de la memoria.
  - T5.1 Redacción del capítulo de Planificación.
  - T5.2 Redacción del capítulo de Introducción.
  - T5.3 Redacción del capítulo de Preliminares.
  - T5.4 Redacción del capítulo de Desarrollo del simulador.
  - T5.5 Redacción del capítulo de Diseño experimental.
  - T5.6 Redacción del capítulo de Experimentación.
  - T5.7 Redacción del capítulo de Evaluación.
  - T5.8 Redacción del capítulo de Conclusiones.
  - T5.9 Bibliografía (esta historia de usuario será recurrente en cada sprint).
  - T5.10 Correcciones finales.

Diagrama de Gantt

En la Tabla 2.1 se muestra un diagrama de Gantt con la planificación por sprints del trabajo que se planea realizar.

2.3. Presupuesto

El presupuesto asociado al desarrollo del proyecto se divide en dos apartados: **1)** recursos técnicos y **2)** recursos humanos. En el primer apartado se consideran los sistemas, herramientas y programas necesarios para la realización del proyecto. En el segundo se consideran todos los gastos asociados a los trabajadores.

Recursos Técnicos

Para la realización del proyecto se van a utilizar dos máquinas distintas,

- Ordenador personal: MSI Creator 15. Precio de 1600€, con una vida útil estimada de 5 años.
- Máquina virtual del departamento con las siguientes características: 16GB de RAM, 50GB de almacenamiento, procesador con 2 núcleos 2.095 GHz. Precio estimado de 1000€, con una vida estimada de los componentes físicos de 5 años.

	Sprint				
Historia de Usuario	0	1	2	3	4
T0.1					
T1.1					
T1.2					
T5.1					
T5.2					
T5.9					
T1.3					
T1.4					
T2.1					
T5.3					
T5.4					
T5.9					
T2.2					
T3.1					
T4.1					
T5.5					
T5.9					
T3.2					
T4.2					
T4.3					
T5.6					
T5.7					
T5.8					
T5.9					
T5.10					

Tabla 2.1: Planificación del trabajo por sprints.

Será imprescindible una conexión a Internet para poder conectarse a la máquina virtual, asistir a reuniones realizadas online y consultar la bibliografía. Para ello se va a contratar una conexión a Internet de 600MB de fibra simétrica, por un precio de 40€ mensuales.

En cuanto a las herramientas y programas seleccionados, se utilizarán aplicaciones de licencia gratuita como Microsoft Teams, Overleaf y Miniconda. Los sistemas operativos no han supuesto un incremento en el presupuesto, ya que se ha utilizado Ubuntu 22.04.4 LTS, que es de libre distribución.

En la Tabla 2.2 se muestra el presupuesto de los recursos técnicos del proyecto.

Concepto	Precio por unidad	Cantidad	Total
Ordenador personal	26,66€/mes	6 meses	159,96€
Máquina virtual	16,66€/mes	6 meses	99,96€
Internet	40€/mes	6 meses	240,00€
Microsoft Teams	0€/mes		0,00€
Overleaf	0€/mes		0,00€
Miniconda	0€/mes		0,00€
Visual Studio Code	0€/mes		0,00€
Ubuntu 22.04 LTS	0€/mes		0,00€
RECURSOS TÉCNICOS			**499,92€**

Tabla 2.2: Presupuesto de los recursos técnicos

Recursos Humanos

El perfil técnico necesario para el desarrollo del proyecto es un Ingeniero de Software, cuyo salario en España es de aproximadamente 40 000€ anuales [34]. Suponiendo 40 horas semanales y prorrateando el sueldo en 12 pagas, el sueldo bruto equivale a 20,83€ por hora. Al salario en bruto de los trabajadores hay que sumar las cotizaciones a la seguridad social, que corresponde aproximadamente al 30 % del salario bruto [23].

Será necesario formar al trabajador en aprendizaje por refuerzo, para lo que se utilizará, además de los recursos gratuitos disponibles online, un curso de Coursera de la Universidad de Alberta [36]. Para completarlo, se utilizará una suscripción a Coursera Plus durante 2 meses, por un precio de 56€ por mes.

Además, se alquilará un espacio de trabajo donde poder desarrollar el proyecto. Se ha escogido un espacio de coworking, con un precio de 3€/día [4].

En la Tabla 2.3 se muestra el presupuesto de los recursos humanos para el proyecto.

Concepto	Precio por unidad	Cantidad	Total
Sueldo Ingeniero Software	20,83€/hora	340 horas	7082,20€
Seguridad Social del empleado	6,25€/hora	340 horas	2125,00€
Formación	56€/mes	2 meses	112,00€
Espacio de trabajo coworking	240€/mes	6 meses	1440,00€
RECURSOS HUMANOS			**10759,20€**

Tabla 2.3: Presupuesto de los recursos humanos.

Juntando los dos apartados anteriores, el presupuesto para el desarrollo del proyecto es de **11259,12€**.

2.4. Herramientas

Durante la realización del proyecto se va a utilizar las siguientes herramientas:

- **Python**: Lenguaje de programación muy popular utilizado para aplicaciones de inteligencia aritficial. El simulador se ha implementado en este lenguaje y la mayoría de bibliotecas de Deep Learning están implementadas en Python también.
- **Visual Studio Code**: Entorno integrado de desarrollo (IDE) de código abierto que se utilizará para desarrollar el simulador.
- **Miniconda**: Gestor de entornos virtuales y paquetes de Python. Se utiliza para mantener un entorno de Python en el que tener todos los paquetes necesarios para el proyecto y evitar incompatibilidades.
- **Stable Baselines 3**: Biblioteca con implementaciones de algoritmos de Aprendizaje por Refuerzo en PyTorch. Es compatible con Tensorboard para obtener las gráficas del entrenamiento y las implementaciones se han probado con entornos de Gymnasium para verificar su correcta implementación.

3: Preliminares

3.1. *Reinforcement Learning*

El *Reinforcement Learning* (en español, Aprendizaje por Refuerzo) [25, 29, 3] es un paradigma del *machine learning*, muy diferente a otros más conocidos como el aprendizaje supervisado o el aprendizaje no supervisado. En el Aprendizaje por Refuerzo se aprende a asociar situaciones a acciones para **maximizar una recompensa**, que tal vez sea diferida. Se descubren qué acciones dan la mayor recompensa haciendo mediante **prueba y error** una búsqueda exhaustiva del espacio de estados.

3.1.1. Elementos de un problema de Reinforcement Learning

En un sistema de aprendizaje por refuerzo llamamos **agente** a la entidad que interactúa con el **entorno**, que es todo lo que está fuera del agente. Llamamos **política** al comportamiento del agente en un momento determinado, es decir, a la acción que toma según el **estado** que percibe del entorno. Cada vez que el agente toma una acción, el entorno devuelve una señal de **recompensa**, que define cómo de bien o mal lo ha hecho el agente con la acción inmediatamente anterior, y el agente observa el nuevo estado en el que se encuentra el entorno tras la acción. Se puede decir que el agente es un agente inteligente basado en metas en el sentido definido por Russell y Norvig [27], con el objetivo de maximizar la recompensa obtenida a largo plazo.

3.1.2. *Markov Decision Processes*

Los procesos de decisión de Markov, o MDPs, son una formalización clásica de los problemas de toma de decisiones secuenciales en presencia de incertidumbre, donde las acciones no sólo influyen las recompensas inmediatas, sino también los siguientes estados y por tanto las recompensas futuras. Los MDPs son una idealización matemática del problema de aprendizaje por refuerzo que se utiliza para establecer los fundamentos teóricos del paradigma.

La interacción de un agente con su entorno en un MDP se muestra en la Figura 3.1. Esta interacción sucede en una secuencia de instantes de tiempo discretos (timesteps) $t = 0, 1, 2, 3, ...$ En cada instante t el agente recibe una representación del **estado** del entorno S_t, en base al cual realiza una **acción** A_t. En el siguiente instante $t + 1$, el agente recibe una señal de **recompensa** R_{t+1} y un nuevo estado S_{t+1}. Esto lleva a una trayectoria que comienza de la siguiente manera:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3... \tag{3.1}$$

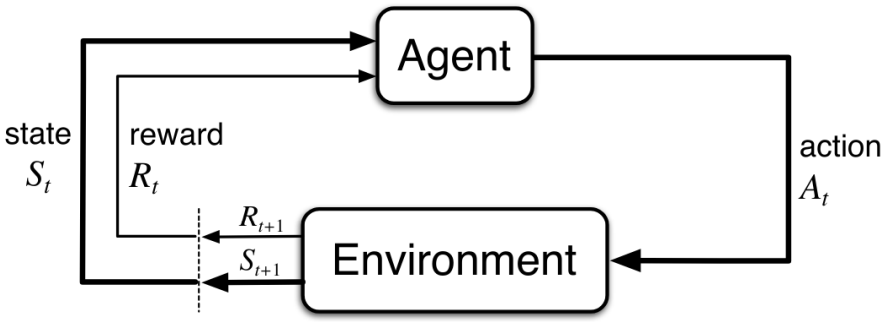


Figura 3.1: Interacción agente-entorno en un MDP [25]

En un MDP finito, los conjuntos de estados $\mathcal{S}$, acciones $\mathcal{A}$ y recompensas $\mathcal{R}$ tienen un número finito de elementos, y en ese caso, las variables aleatorias R_t y S_t tienen funciones de probabilidad discretas bien definidas y dependientes sólo del estado y la acción anteriores.

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{3.2}$$

La función $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ es una función determinista ordinaria que define las dinámicas del MDP. Además, la función p especifica una distribución de probabilidad para cada elección de s y a, lo cual implica:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{3.3}$$

Propiedad de Markov

El estado del entorno en el momento t debe contener información sobre todos los aspectos relevantes del pasado, es decir, el futuro es independiente del pasado dado el presente. Esto implica que la probabilidad de cada valor posible para S_t y R_t depende sólo del estado y acción inmediatamente anteriores, S_{t-1} y A_{t-1}, y, dados ellos, no depende para nada de los anteriores. La ecuación 3.2 cumple esto que se acaba de describir.

Objetivos y recompensas

Como se ha dicho anteriormente, el objetivo del agente es maximizar la recompensa que recibe, no sólo de forma inmediata, sino también a largo plazo. Esta idea se puede enunciar como la hipótesis de la recompensa:

> "Todo lo que entendemos por metas y propósitos puede pensarse como la maximización del valor esperado de la suma de una señal escalar recibida (llamada recompensa)" [25].

Esto implica que en un sistema de aprendizaje por refuerzo es crítico diseñar una función de recompensa que indique de verdad lo que se quiere conseguir. La función de recompensa es la forma que tenemos de decirle al agente lo que queremos conseguir, no cómo queremos que lo consiga.

Retornos y Episodios

El objetivo del agente es maximizar la suma de la recompensa recibida a largo plazo. Formalmente, si se está en el **instante** t, el objetivo es maximizar el retorno esperado, donde el **retorno**, denotado G_t, se define como la suma de la secuencia de las recompensas recibidas:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T, \tag{3.4}$$

donde T es el instante de tiempo final. Este enfoque tiene sentido en aplicaciones donde existe la noción de instante final, y donde interacción del agente con el entorno se divide en subsecuencias que se denominan **episodios**. Estas tareas se denominan tareas episódicas, y todas ellas terminan en un estado especial, llamado estado terminal. El conjunto de todos los estados no terminales lo llamamos $\mathcal{S}$, y al de todos los estados más el estado terminal, $\mathcal{S}^+$.

En el caso en el que las tareas no sean episódicas y continúen de manera indefinida, las llamamos **tareas continuas**. En este caso es necesario el concepto de **descuento**.

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{3.5}$$

donde γ es un parámetro, $0 \leq \gamma \leq 1$, denominado tasa de descuento. Además, los retornos en *timesteps* sucesivos están relacionados entre sí de forma recursiva:

$$G_t \doteq R_{t+1} + \gamma G_{t+1} \tag{3.6}$$

Se puede unificar la notación del retorno para las tareas episódicas y las tareas continuas:

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{3.7}$$

donde es posible que $T = \infty$ o que $\gamma = 1$, pero no las dos.

Políticas y funciones de valor

La mayoría de los algoritmos de aprendizaje de refuerzo estiman **funciones de valor** (o *value functions*), que son funciones de estados o de pares estado-acción, que estiman el retorno esperado al estar en un estado o al tomar una acción en un estado determinado, respectivamente.

Estas funciones se definen de acuerdo a una forma de actuar, denominada política. Formalmente, una política es un mapeo de estados a probabilidades de seleccionar cada acción posible. Si el agente está siguiendo una política π en un instante t, entonces $\pi(a|s)$ es la probabilidad de tomar la acción $A_t = a$ si $S_t = s$.

La *value function* de un estado s siguiendo una política π, denotada como $v_\pi(s)$, es el retorno esperado cuando se empieza en el estado s y se sigue la política π.

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \tag{3.8}$$

De forma similar, el valor de tomar la acción a en el estado s bajo una política π, denotado como $q_\pi(s, a)$, es el retorno esperado cuando se empieza en el estado s, se toma la acción a y después se sigue la política π después.

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \tag{3.9}$$

Una propiedad de las *value functions* que se usa en el aprendizaje por refuerzo es que satisfacen relaciones recursivas. La ecuación que expresa la relación entre el valor de un estado y los valores de estados sucesores, para v_π, se denomina ecuación de *Bellman*:

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right] \tag{3.10}$$

De forma análoga, la ecuación de *Bellman* para q_π:

$$q_\pi(s, a) \doteq \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right] \tag{3.11}$$

3.1.3. Optimalidad

Resolver una tarea de aprendizaje por refuerzo significa encontrar una política que consiga maximizar el retorno. Necesitamos poder comparar políticas y las funciones de valor dan un orden parcial entre ellas. Una política π es mejor o igual que una política π' si su retorno esperado es mayor o igual que el de π' para todos los estados. Es decir, $\pi \geq \pi'$ si y solo si $v_\pi(s) \geq v'_\pi(s)$, para todo $s \in \mathcal{S}$. Siempre hay al menos una política óptima, denotada π_* que es mejor o igual que todas las demás. Las políticas óptimas comparten la misma función estado-valor, denotada por v_*:

$$v_*(s) \doteq \max_\pi v_\pi(s), \tag{3.12}$$

para todo $s \in \mathcal{S}$.

Las políticas óptimas también comparten la misma función de acción-valor, denotada q_*:

$$q_*(s) \doteq \max_\pi q_\pi(s, a), \tag{3.13}$$

para todo $s \in \mathcal{S}$ y $a \in \mathcal{A}$

Dado que v_* es una función de valor de una política, se puede escribir la ecuación de Bellman (ver 3.10) para v_*: la *ecuación de optimalidad de Bellman*. Expresa que el valor de un estado siguiendo una política óptima debe ser igual al retorno esperado al tomar la mejor acción posible desde ese estado.

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_*(s')\right] \tag{3.14}$$

Para q_*, la ecuación de optimalidad de Bellman es:

$$q_*(s, a) = \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \max_a q_*(s', a')\right] \tag{3.15}$$

Conociendo las dinámicas del entorno y disponiendo de v_* es posible determinar una política óptima. Para cada estado s habrá una o más acciones con las que se obtiene el máximo en la ecuación de optimalidad de Bellman. Cualquier política que sea *greedy* con respecto a v_* toma decisiones basándose únicamente en las consecuencias a corto plazo. Sin embargo, si una política usa v_* para tomar decisiones a corto plazo, la política es óptima a largo plazo porque v_* ya tiene en cuenta la recompensa a largo plazo.

Tener q_* hace que elegir acciones óptimas sea más fácil, porque entonces el agente no necesita conocer los estados sucesores al tomar una acción y su valor, esto es, las dinámicas del entorno, que casi nunca son conocidas en problemas reales.

Resolver las ecuaciones de optimalidad de forma explícita da una ruta para encontrar una política óptima, pero esta solución se basa en tres suposiciones que rara vez son ciertas en la práctica: **1)** las dinámicas del entorno se conocen, **2)** los recursos computacionales son suficientes y **3)**, los estados tienen la propiedad de Markov.

La mayoría de los métodos de aprendizaje por refuerzo resuelven de forma aproximada la ecuación de optimalidad de Bellman, usando transiciones de experiencia real en lugar de conocer las transiciones esperadas.

3.1.4. Ajustes en el Aprendizaje por Refuerzo

Dilema de exploración-explotación

El dilema de la exploración-explotación es encontrar el balance entre explotar el conocimiento que se tiene sobre el entorno, tomar las acciones que se saben que van a dar un mayor retorno y explorar para poder tomar mejores decisiones en el futuro.

La explotación del conocimiento, en nuestro caso, de una función de valor consiste en que tomamos una decisión que sea *greedy* con respecto a esa función de valor.

En la práctica, para poder mantener la exploración se opta por utilizar políticas que se denominan *soft*, lo cual significa que $\pi(a \mid s) > 0$ para todo $s \in \mathcal{S}$ y toda $a \in \mathcal{A}(s)$. Dentro de las políticas soft están las políticas **ϵ-soft**, para las que $\pi(a \mid s) \geq \frac{\epsilon}{|\mathcal{A}(s)|}$ para todos los estados y para un $\epsilon > 0$.

Por último, dentro de las políticas ϵ-soft, están las políticas **ϵ-greedy**, para el cual la probabilidad de escoger la acción con el mayor retorno estimado según la función de acción-valor es $1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$, mientras que la probabilidad para escoger cualquiera de las otras acciones es $\frac{\epsilon}{|\mathcal{A}(s)|}$.

Métodos *on-policy* vs *off-policy*

Una clasificación para los algoritmos de aprendizaje por refuerzo es si son on-policy u off-policy. Los métodos on-policy intentan evaluar o mejorar la política que se está usando para tomar decisiones, mientras que los métodos off-policy evalúan o mejoran una política diferente de la que se está utilizando para generar datos.

3.1.5. Programación Dinámica

La programación dinámica (debida también a Bellman) es una técnica de programación que se puede utilizar para obtener políticas óptimas dado un modelo perfecto del entorno como un MDP (esto es, las dinámicas del entorno son conocidas).

Una de las características de los métodos basados en programación dinámica es que para estimar las funciones de valor se basan en las estimaciones del valor de estados sucesivos. En este contexto, a esta idea se le llama ***bootstrapping***.

Evaluación de Políticas

El primer problema a considerar es el cálculo de la función de estado-valor v_π para una política arbitraria π. A esto se le llama *policy evaluation* (en español, evaluación de política). En la práctica se trata de resolver la ecuación de Bellman para v_π (3.10) a través de métodos iterativos.

La primera aproximación de la función de valor v_0 se escoge de manera arbitraria, y cada aproximación sucesiva se obtiene utilizando la ecuación 3.10 como una regla de actualización:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s')\right] \tag{3.16}$$

En el Algoritmo 1 se muestra el pseudocódigo del algoritmo de evaluación de políticas.

Algoritmo 1 Evaluación de Políticas Iterativa, para estimar $V \approx v_\pi$

Entrada π, la política a evaluar
Parámetros: un pequeño umbral $\Delta > 0$ que determina la precisión de la estimación
Inicializar $V(s)$ arbitrariamente, para $s \in \mathcal{S}$, y $V(terminal)$ a 0
repeat
  for cada $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$

Mejora de Políticas

La razón para estimar la función de valor de una política es para conseguir encontrar una política que sea mejor que la que ya tenemos. Esto se consigue modificando π para que en cada estado s la acción a tomar sea la que nos lleve al estado s' cuyo valor $v(s')$ sea mayor. Es decir, que la nueva política π' sea *greedy* con respecto a v_π.

$$\pi'(s) = arg\,\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s')\right] \tag{3.17}$$

Si la nueva política *greedy* π' es igual de buena pero no mejor que la política anterior π, entonces $v_\pi = v'_\pi = v_*$, es decir, la función de valor es ya óptima, y por tanto las políticas π y π' también son óptimas. La mejora de una política usando la ecuación 3.17 debe dar como resultado una política que sea estrictamente mejor, a no ser que la política sea ya óptima.

Iteración de Políticas

Dada una política inicial arbitraria π_0, la manera de llegar a una política óptima es a través de un método denominado iteración de políticas (en inglés, *Policy Iteration*), que consiste en evaluar esa política inicial, utilizar la *value function* obtenida de esa evaluación para mejorarla y repetir el proceso indefinidamente hasta obtener una política óptima:

$$\pi_0 \xrightarrow{E} v_{\pi 0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi 1} \xrightarrow{I} \pi_2 \xrightarrow{E} ... \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

En el Algoritmo 2 se muestra el pseudocódigo de la iteración de políticas, en el que se combina la evaluación de una política con su mejora hasta llegar a una política óptima.

Algoritmo 2 Iteración de Políticas (usando evaluación de políticas iterativa) para estimar $\pi \approx \pi_*$

---

1. Inicialización:
$V(s) \in \mathbb{R}$ y $\pi(s) \in A(s)$ arbitrariamente para todo $s \in S$; $V(\text{terminal}) \doteq 0$

2. Evaluación de la Política:
repeat
  $\Delta \leftarrow 0$
  for cada $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s)) \left[r + V(s')\right]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$ (un número pequeño positivo determinando la precisión de la estimación)

3. Mejora de la política:
policy-stable $\leftarrow$ *true*
for cada $s \in S$ **do**
  old-action $\leftarrow \pi(s)$
  $\pi(s) \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a) \left[r + V(s')\right]$
  Si *old-action* $\neq \pi(s)$, entonces *policy-stable* $\leftarrow false$
end for
Si *policy-stable*, entonces parar y retornar $V \approx v_*$ y $\pi \approx \pi_*$; sino ir al paso 2

Iteración de Valores

Uno de los problemas del método de iteración de políticas es que cada iteración requiere evaluar la política, lo cual puede requerir una computación excesiva. Otra idea es hacer un solo paso de evaluación de la política (una actualización en cada estado, sin esperar a la convergencia) por cada paso de mejora de política. Se combina la mejora de la política

con la evaluación de la misma. A este algoritmo se le llama iteración de valores (*value iteration* en inglés), cuyo pseudocódigo se muestra en el Algoritmo 3.

Algoritmo 3 Iteración de Valores, para estimar $\pi \approx \pi_*$

Parámetros: un pequeño umbral $\Delta > 0$ determinando la precisión de la estimación
Inicializar $V(s)$, para todo $s \in \mathcal{S}^+$, arbitrariamente excepto $V(terminal) = 0$
repeat
  $\Delta \leftarrow 0$
  for cada $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$

Iteración de Políticas Generalizada

El término iteración de políticas generalizada (en inglés, *Generalized Policy Iteration*, GPI) se refiere a la idea de dejar a los procesos de evaluación y mejora de políticas interactuar. La mayoría de los métodos de aprendizaje por refuerzo se pueden describir como GPI: las políticas se mejoran con respecto a la función de valor y la función de valor se actualiza con respecto a la política.

La función de valor se estabiliza sólo cuando es consistente con la política actual, y la política se estabiliza sólo cuando es *greedy* con respecto a la función de valor actual. Esto es, ambos procesos se estabilizan cuando se ha encontrado una política que es *greedy* con respecto a su función de valor, lo cual implica que se cumple la ecuación de optimalidad de Bellman (ver 3.12), y por tanto la política y la función de valor son óptimas.

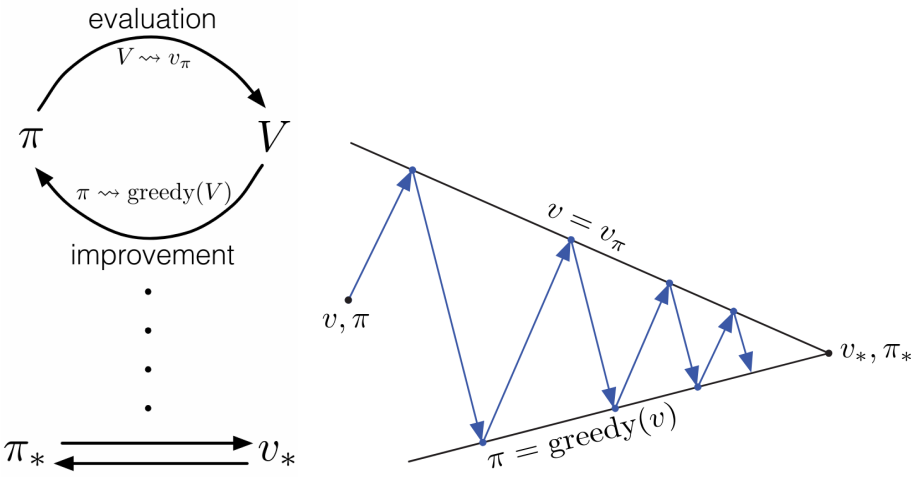


Figura 3.2: Esquema de GPI [25].

3.1.6. Métodos de Monte Carlo

Los métodos de Monte Carlo (en este contexto, el término "Monte Carlo" se refiere a los métodos que se basan en promediar retornos completos) no requieren que se conozcan las dinámicas del entorno, sino que aprenden de la experiencia (ya sea real o simulada). Si es simulada se necesita un modelo, pero sólo para generar transiciones de ejemplo, no las distribuciones de probabilidad de todas las posibles transiciones que se necesitan en los métodos de programación dinámica.

Definimos los métodos de Monte Carlo sólo para tareas episódicas, es decir, la experiencia se divide en episodios que siempre terminan sin importar qué acciones se elijan. Una vez se ha completado un episodio se actualizan las funciones de valor y las políticas. Los métodos de Monte Carlo promedian retornos para cada para estado-acción. A diferencia de los métodos basados en programación dinámica, donde se computaban funciones de valor a partir del conocimiento del MDP, aquí se aprenden las funciones de valor a partir de ejemplos de retornos con el MDP.

En el Algoritmo 4 se muestra el pseudocódigo del algoritmo de Monte Carlo para estimar la política óptima. Se genera un episodio siguiendo una política arbitraria, se estima el valor de la función de valor estado-acción ($q_\pi(s, a)$), es decir, se evalúa la política, y se mejora la política después de esa estimación. En el caso de este algoritmo, se tiene en cuenta sólo la primera ocurrencia del mismo par (estado, acción) en cada episodio.

3.1.7. *Temporal-Difference learning*

Temporal-Difference (TD) learning es una combinación de las ideas de los métodos de Monte Carlo y de la programación dinámica. Al igual que los métodos de Monte Carlo, los métodos TD aprenden de la experiencia y al igual que los métodos de programación dinámica, actualizan sus estimaciones a partir de otras estimaciones, lo que se ha llamado *bootstrapping*.

Los métodos TD actualizan la estimación de la función de valor a cada paso, en lugar de esperar al final del episodio como hacen los métodos de Monte Carlo. El método TD más simple hace la actualización

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{3.18}$$

inmediatamente en la transición a S_{t+1} mientras recibe R_{t+1}. Mientras que el objetivo para la actualización de Monte Carlo es G_t, para la actualización TD es $R_{t+1} + \gamma V(S_{t+1})$.

La cantidad que está entre corchetes en la actualización TD (3.18) es un error, una medida de la diferencia entre el valor estimado de S_t y la mejor estimación de ese valor $R_{t+1} + \gamma V(S_{t+1})$. A este cantidad se le denomina *TD error*.

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{3.19}$$

Algoritmo 4 On-policy First-Visit MC Control (para políticas ϵ-soft), estima $\pi \approx \pi_*$

Parámetros: pequeño $\epsilon > 0$
Inicializar:
$\pi \leftarrow$ una política ϵ-soft arbitraria
$Q(s, a) \in \mathbb{R}$ (arbitrariamente), para todo $s \in \mathcal{S}, a \in \mathbf{A}(\mathbf{s})$
$Returns(s, a) \leftarrow$ lista vacía, para todo $s \in \mathcal{S}, a \in \mathbf{A}(\mathbf{s})$
for cada episodio **do**
Generar un episodio siguiendo $\pi; S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
for cada instante del episodio, $t = T-1, T-2, \ldots, 0$ **do**
$G \leftarrow \gamma G + R_{t+1}$
if (S_t, A_t) es la primera ocurrencia en el episodio **then**
Añadir G a $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow average(Returns(S_t, A_t))$
$A^* \leftarrow argmax_a Q(S_t, a)$ (con los empates resueltos arbitrariamente)
for a $\in \mathcal{A}(S_t)$ **do**

$$\pi(a \mid S_t) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(S_t)| & if a \neq A^* \\ \epsilon/|\mathcal{A}(S_t)| & if a \neq A^* \end{cases}$$

end for
end if
end for
end for

Q-Learning

Uno de los algoritmos TD de control *off-policy* más conocidos es el llamado Q-Learning [25], en el que la actualización de la función de valor de los pares estado-acción está definido por:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \tag{3.20}$$

La función de acción-valor, Q, aproxima directamente q_*, la función de acción-valor óptima, independientemente de la política que se esté siguiendo. En el Algoritmo 5 se muestra el pseudocódigo del algoritmo Q-Learning.

3.2. *Deep Reinforcement Learning*

Cuando nos enfrentamos a problemas más complejos que tienen un número de estados infinitos, no es viable utilizar soluciones tabulares como las descritas anteriormente. No sólo por un problema de la memoria que se necesitaría para las tablas, sino por el tiempo necesario para poner en cada entrada una buena estimación del valor de la *value function*. Es por ello que se necesita generalización, porque es probable que la mayoría de los estados

Algoritmo 5 Q-Learning

Parámetros: tamaño del paso $\alpha \in (0, 1]$, pequeño $\epsilon > 0$
Inicializar $Q(s, a)$ para todo $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrariamente
excepto $Q(\text{terminal}, \cdot) = 0$
for cada episodio **do**
  Inicializar S
  while S no sea terminal **do**
    Elegir A desde S usando la política derivada de Q (e.g., ϵ-greedy)
    Tomar acción A, observar R S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  end while
end for

a los que se enfrente nuestro agente no los haya visto antes. Esta generalización es posible utilizando aproximación de funciones.

De forma práctica, esta aproximación de funciones se realiza utilizando redes neuronales artificiales. Esta combinación del *deep learning* con el *reinforcement learning* se conoce como *deep reinforcement learning*.

3.2.1. Deep Q Learning

La primera idea es utilizar el *deep reinforcement learning* para aproximar las funciones de valor de pares de estado-acción, combinando la capacidad de las redes neuronales de aproximar funciones complejas no lineales con la simplicidad del algoritmo Q-Learning.

DQN

La primera vez que se utilizó con éxito el *deep reinforcement learning* fue con varios juegos clásicos de Atari 2600 [17] con el algoritmo DQN (llamado así porque utiliza las llamadas 'Deep Q Networks'), donde se consiguió que el agente entrenado con este algoritmo llegara al nivel de probadores de juegos profesionales humanos.

Las aplicaciones de deep learning más exitosas necesitan grandes cantidades de datos etiquetados para entrenar, mientras que los algoritmos de aprendizaje por refuerzo necesitan aprender de una señal de recompensa que es dispersa, ruidosa y muchas veces llega con retraso. Si comparamos el retraso desde que se realiza una acción hasta que se obtiene la recompensa correspondiente puede ser de miles de instantes, algo desalentador en comparación al aprendizaje supervisado donde la asociación entre entradas y objetivos es directa. Otro problema es que los algoritmos de deep learning asumen que los datos son independientes entre sí, mientras que en aprendizaje por refuerzo se encuentran secuencias de datos altamente correlacionados. Además, pequeñas actualizaciones en la función de valor Q puede provocar un cambio significativo en la política (si es ϵ-greedy).

Con el objetivo de solucionar estos problemas, DQN utiliza dos ideas clave: **1)** Experience Replay y **2)** Fixed Q Targets. A continuación se explican ambas ideas y cómo ayudan a solucionar los problemas descritos.

Experience Replay Es un buffer D de tamaño N, en el que se almacena la experiencia del agente en cada instante $e_t = (S_t, A_t, R_t, S_{t+1})$, utilizada para hacer actualizaciones eficientes en la red neuronal. De esta manera, cada transición es usada potencialmente en muchas actualizaciones, al coger ejemplos aleatorios del buffer para hacer actualizaciones desaparece la correlación entre los ejemplos consecutivos.

Fixed Target Q Networks Para mejorar la estabilidad del aprendizaje, se utiliza una segunda red neuronal, que es una copia antigua de la red neuronal principal. Cada C timesteps de entrenamiento se copian los pesos de la red neuronal principal a la segunda. La segunda red se utiliza para generar los objetivos de Q-learning, $y_i = r + \gamma max'_a Q(S', A)$, lo cual añade un retraso desde que se actualiza la función Q (red neuronal principal)) y su efecto en y_i, lo que ayuda a reducir las divergencias y oscilaciones durante el aprendizaje.

En el Algoritmo 6 se muestra el pseudocódigo del algoritmo DQN, en el que se toma como parámetro el número de episodios durante el que se entrena, denotado por M.

Algoritmo 6 Deep Q-learning with Experience Replay

Parámetros: N, M, ϵ, C
Inicializar buffer D con capacidad N
Inicializar función de acción-valor Q con pesos aleatorios θ
Inicializar función de acción-valor objetivo $\hat{Q}$ con pesos $\theta^- = \theta$
for *episode* $= 1, M$ **do**
  for $t = 1, T$ **do**
    Con probabilidad ϵ elegir una acción aleatoria A_t
    en otro caso elegir $A_t = argmax_A Q(S_t, A; \theta)$
    Ejecutar la acción A_t en el entorno y observar R_t y S_{t+1}
    Almacenar la transición (S_t, A_t, R_t, S_{t+1}) en D
    Tomar un minibatch aleatorio de transiciones (S_j, A_j, R_j, S_{j+1}) de D
    Set $y_j = \begin{cases} r_j & \text{si el episodio termina en el instante } j+1 \\ r_j + \gamma max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-) & \text{en otro caso} \end{cases}$
    Tomar un paso en el descenso del gradiente en $(y_j - Q(s_j, a_j; \theta))^2$ con respecto a los parámetros θ de la red
    Cada C instantes $\hat{Q} = Q$
  end for
end for

QR-DQN

En los algoritmos descritos hasta ahora, se estima el valor de Q como la media del retorno esperado en el estado s si se toma la acción a. Cuando en un entorno hay aleatoriedad y los retornos son variables, la media del retorno puede ser insuficiente.

Para conseguir un aprendizaje más robusto y eficiente, se plantea en su lugar estimar la distribución completa de los retornos [2]. Una forma de estimar esta distribución es a través de la regresión cuantílica, con el algoritmo QR-DQN [5, 20]. La red neuronal estima N cuantiles de la distribución del retorno para cada par de estado-acción. Como se ha dicho, se aprende la distribución completa del retorno esperado en lugar del valor esperado:

$$Z(x,a) \overset{\mathrm{D}}{=} R(x,a) + \gamma Z(X',A'), \tag{3.21}$$

donde $Z(x,a) \in \mathbb{R}^{|A|\times N}$ es un conjunto de átomos que forman la función de masa de probabilidad (PMF) de la distribución del retorno. En el caso de QR-DQN, los átomos que forman $Z(x,a)$ son las medianas de N cuantiles de la distribución. Z_ϕ pasa a ser una distribución cuantílica definida como:

$$Z_\theta(x,a) = \frac{1}{N}\sum_{n=1}^{N} \delta_{\theta_i(x,a)} \tag{3.22}$$

3.2.2. *Policy Gradient Methods*

El problema de elegir las acciones según la estimación del retorno, es que aunque haya acciones que tengan un retorno estimado similar, si la política es *greedy* con respecto a la función de valor, siempre se va a tomar la misma acción cuando se esté en el mismo estado. Esto implica que no se pueden aprender políticas estocásticas, que en algunos casos son las óptimas.

En lugar de los métodos de acción-valor, se puede aprender una política parametrizada que pueda seleccionar acciones sin consultar una función de valor. Los *policy gradient methods* parametrizan la política con unos parámetros θ para tener una distribución de probabilidad sobre las acciones:

$$\pi_\theta(a_t \mid s_t) = p(a_t \mid s_t, \theta) \tag{3.23}$$

Los parámetros θ se aprenden en base al gradiente de una medida de rendimiento escalar $J(\theta)$ con respecto a los parámetros de la política. Se trata de maximizar el rendimiento, por lo que las actualizaciones de los parámetros θ buscan el ascenso del gradiente en J:

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)} \tag{3.24}$$

En el caso episódico, definimos esta medida de rendimiento de la siguiente manera:

$$J(\theta) \doteq v_{\pi_\theta}(s_0), \tag{3.25}$$

donde v_{π_θ} es la función de valor real para π_θ, la política determinada por θ. El gradiente de J se deriva en el denominado teorema del gradiente de la política (*policy gradient theorem*), que establece lo siguiente (ver capítulo 13 en [25]):

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a \mid s, \theta) \tag{3.26}$$

REINFORCE

En el Algoritmo 7 se muestra el pseudocódigo para el algoritmo on-policy *REINFORCE*, que utiliza una estimación del retorno de un episodio completo para actualizar los parámetros θ de la política.

Algoritmo 7 REINFORCE: Monte-Carlo Policy-Gradient Control (episódico) para π_*

Entrada: una parametrización de una política diferenciable $\pi(a \mid s, \theta)$
Parámetros: tamaño del paso $\alpha > 0$
Inicializar parámetros de la política $\theta \in \mathbb{R}^d$
for cada episodio **do**
  Generar un episodio $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, siguiendo $\pi(\cdot \mid \cdot, \theta)$
  for cada instante del episodio $t = 0, \ldots, T-1$ **do**
    $G \leftarrow$ retorno desde el paso t
    $\theta \leftarrow \theta + \alpha \gamma^t G \nabla_\theta \log \pi(A_t \mid S_t, \theta)$
  end for
end for

Actor Critic Methods

El problema del algoritmo *REINFORCE* es que tiene una gran varianza. Para conseguir que el agente entrene más rápido y mejor se utiliza una combinación de las dos clases de métodos explicadas hasta ahora: métodos basados en la estimación de funciones de valor y los métodos basados en el gradiente de la política, lo que se denominan *Actor Critic Methods* (métodos Actor-Crítico) [6].

En esta clase de métodos, cada uno de sus componentes (el actor y el crítico) aprende una aproximación de una función:

- Actor: una función de una política, $\pi_\theta(s)$, que controla cómo actúa el agente.
- Crítico: una función de valor, $q_w(s,a)$, utilizada para la actualización de la política midiendo cómo de buena es la acción tomada.

Con esto, se puede reescribir la ecuación del gradiente de J de la siguiente manera:

$$\nabla\theta = \alpha\nabla_\theta(log\pi_\theta(s,a))q_w(s,a) \tag{3.27}$$

Advantage Actor Critic (A2C)

Para estabilizar aún más el entrenamiento, se puede utilizar la llamada *Advantage function* en lugar de la función de acción valor. La *Advantage function*, denotada $A(s,a)$ calcula cómo de mejor es tomar la acción a en el estado s en comparación con el valor medio del estado. Los métodos que utilizan una *Advantage function* se denominan Advantage Actor Critic (A2C) [14].

$$A(s,a) = Q(s,a) - V(s) \tag{3.28}$$

Esto requeriría dos funciones de valor $Q(s,a)$ y $V(s)$, pero no es necesario si usamos el error TD (ver 3.19) como *advantage function*.

$$A(s,a) = r + \gamma V(s') - V(s) \tag{3.29}$$

Utilizando la *Advantage function*, se reescribe de nuevo el gradiente de J.

$$\nabla\theta = \alpha\nabla_\theta(log\pi_\theta(s,a))A(s,a) \tag{3.30}$$

4: Desarrollo del Simulador

Como se ha explicado, un agente entrenado con aprendizaje por refuerzo necesita interactuar con el entorno para aprender a escoger las acciones que van a maximizar la recompensa que recibe en el largo plazo. Por tanto, para conseguir un agente que sea capaz de tomar decisiones estratégicas en F1, es necesario construir un simulador de carreras de F1, el entorno, con el que pueda interactuar nuestro agente y así poder cumplir con los objetivos del proyecto.

En este capítulo se va a realizar una descripción del simulador desarrollado. Primero se van a presentar el modelo de dominio, describiendo los elementos que componen componen el simulador y los atributos que forman cada uno de ellos. Después, se hablará del estado del simulador y de las acciones que se pueden realizar dentro del mismo. Tras esto, se explicará el funcionamiento del simulador.

Finalmente, se explicará cómo se ha adaptado el simulador desarrollado al estándar de API de entornos de aprendizaje por refuerzo propuesto por Gymnasium (anteriormente conocido como Gym)[10], para poder utilizarlo con los algoritmos de aprendizaje por refuerzo.

4.1. Modelo de Dominio

Para simular una carrera vamos a necesitar una serie de conceptos (circuito, piloto, modelo de adelantamientos, neumáticos, estrategias, carrera) que darán lugar a nuestro modelo de dominio. En la descripción puede haber referencias a otros elementos del simulador que se intentarán resolver de la forma más clara posible (por ejemplo, un piloto tiene una estrategia, una estrategia tiene pitstops, y un pitstop tiene un neumático).

En la Figura 4.1 se muestra el diagrama de clases UML [19] representando el modelo de dominio del simulador, con las clases y asociaciones entre ellas. En el resto de la sección se van a describir y definir los atributos de cada una de las clases.

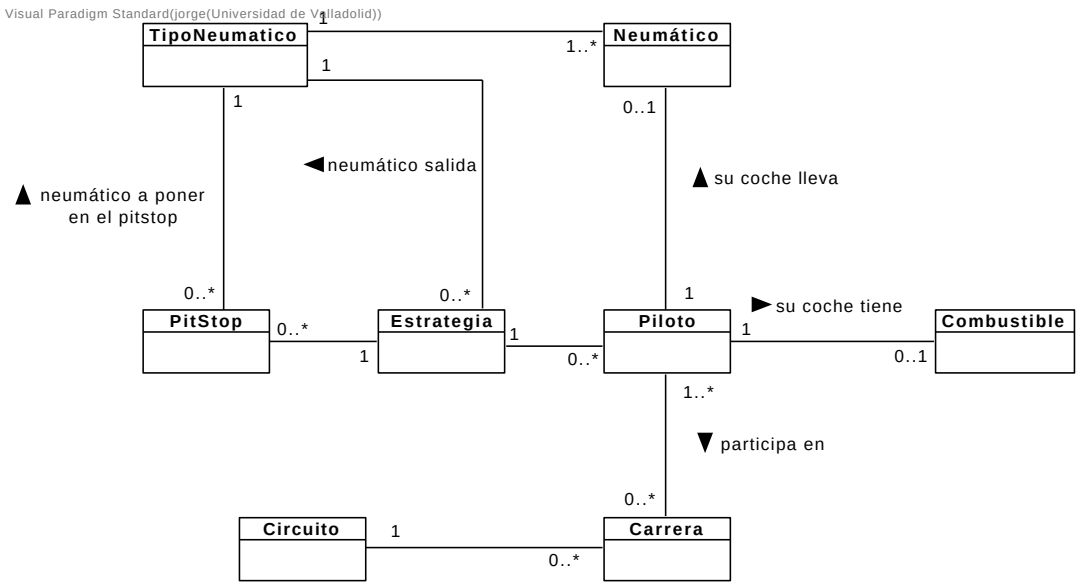


Figura 4.1: Modelo de Dominio del simulador.

4.1.1. Circuito

En la Tabla 4.1 se describe el Circuito en el que ocurre una carrera y sus atributos. Se le caracteriza por la distancia inicial en segundos entre pilotos después de la salida, el tiempo mínimo de vuelta y el porcentaje de la longitud del circuito en el que se encuentra la entrada y salida del pit lane. Además, se incluye la probabilidad de que un piloto adelante a otro si va más rápido, y la distancia entre un piloto *p1* y un piloto *p2* si *p1* intenta adelantarlo pero no lo consigue.

4.1.2. Combustible

En la Tabla 4.2 se la descripción de la clase que representa el combustible usado por un piloto en una carrera, así como la definición de los atributos que lo componen. El coche del piloto cuenta con una masa de combustible al principio de la carrera, que será la necesaria para completarla, ya que no se permite repostar combustible. En cada vuelta el coche del piloto consumirá una cantidad determinada de combustible, y el efecto de la masa de ese combustible influirá en el tiempo por vuelta.

4.1.3. Neumáticos, PitStops y Estrategias

En la Tabla 4.3 se describe la clase que representa al tipo de un neumático (también llamado compuesto), definida por un nombre (en nuestro caso, éste será blando, medio o duro) y el tiempo añadido al tiempo base de la vuelta de un circuito. Esto es, la diferencia base entre rendimiento de los neumáticos: siendo los neumáticos nuevos, el blando es más rápido que el medio y el medio es más rápido que el duro. Además, se incluye la función que determina la degradación de un neumático de ese tipo con el paso de las vueltas.

Circuito			
Definición	Circuito en el que se celebra una carrera		
Nombre	**Definición**	**Tipo**	**Restricciones**
interval_ini	Distancia inicial en segundos entre cada piloto al comenzar la carrera.	float	Rango (0, 1)
t_base	Tiempo mínimo necesario para dar una vuelta completa al circuito.	float	t_base >0
p_adelantamiento	Probabilidad de que un piloto adelante al que tiene delante si va más rápido.	float	Rango (0, 1)
d_no_ov	Distancia en segundos entre un piloto p1 y otro p2, si p1 está detrás de p2 intenta adelantarlo y no lo consigue.	float	Rango(0,1)
pit_entry	Porcentaje de la longitud del circuito en el que se encuentra la entrada al pit lane.	float	Rango (0, 1)
pit_exit	Porcentaje de la longitud del circuito en el que se encuentra la salida del pit lane.	float	Rango (0, 1)
tiempo_pitstop	Cuando se para, tiempo transcurrido desde que se entra al pit lane hasta que se sale, parada incluida. El absoluto de la diferencia entre pit_exit y pit_entry es la longitud del pit lane.	float	pitstop_time >0

Tabla 4.1: Descripción de la clase que representa un circuito en el que se celebra una carrera.

Combustible			
Definición	Combustible usado por un piloto en una carrera		
Nombre	**Definición**	**Tipo**	**Restricciones**
masa_combustible	Masa de combustible inicial, medida en kgs.	float	masa_combustible > 0
efecto_masa	Efecto de 1 kg de combustible en el tiempo por vuelta.	float	efecto_masa > 0
consumo_por_vuelta	Número de kgs de combustible consumidos por vuelta.	float	consumo_por_vuelta > 0
vueltas_consumidas	Número de vueltas de combustible consumidas.	int	Rango [0, n_vueltas]

Tabla 4.2: Descripción de la clase que representa el combustible usado por el coche de un piloto en una carrera.

TipoNeumático			
Definición	Compuesto de un neumático		
Nombre	**Definición**	**Tipo**	**Restricciones**
nombre	Nombre del tipo de neumático	string	Único
t_add	Tiempo añadido al tiempo base por vuelta de un circuito	float	t_add >= 0
degradación	Función cuadrática que marca el efecto de la degradación por vuelta	función cuadrática	Función monótona creciente

Tabla 4.3: Descripción de la clase que representa el compuesto de un neumático en el simulador.

En la Tabla 4.4 se describe la clase que representa el neumático que lleva el coche de un piloto en un momento de la carrera. Se caracteriza por su tipo y el número de vueltas que se ha usado.

Neumático			
Definición	Neumático que lleva el coche de un piloto en una carrera en un momento determinado		
Nombre	**Definición**	**Tipo**	**Restricciones**
compuesto	Tipo del neumático	TipoNeumático	
vueltas_usadas	Número de vueltas usadas del neumático	int	Rango [0, n_vueltas]

Tabla 4.4: Descripción de la clase que representa el neumático que lleva el coche de un piloto en una carrera en un momento determinado en el simulador.

Cuando un piloto hace una parada en boxes, un pitstop, éste se caracteriza por la vuelta en la que se realiza y el tipo de neumático que se va a poner en la parada. En la Tabla 4.5 se describe la clase PitStop y la definición de los atributos que la componen.

PitStop			
Definición	PitStop que realiza un piloto en una carrera		
Nombre	**Definición**	**Tipo**	**Restricciones**
vuelta	Vuelta en la que entrar a boxes	int	Rango (0, n_vueltas)
neumático	Compuesto de neumático a poner en el coche	TipoNeumático	

Tabla 4.5: Descripción de la clase que representa un pitstop que realiza un piloto en un momento determinado en una carrera del simulador.

Finalmente, la estrategia que sigue un piloto en una carrera está formada por el neumático con el que se comienza la carrera y los pitstops que se realizan. En la Tabla 4.6 se describe la clase Estrategia y se definen de los atributos que la componen.

Estrategia			
Definición	Estrategia seguida por un piloto en la carrera, formada por un neumático de salida y los pitstops a realizar		
Nombre	**Definición**	**Tipo**	**Restricciones**
neumático_salida	Compuesto de neumático con el que comenzar la carrera	TipoNeumático	Rango [0, 2]
pitstops	Lista de pitstops que forman la estrategia	Lista[PitStop]	

Tabla 4.6: Descripción de la clase que representa la estrategia seguida por un piloto en una carrera en el simulador.

4.1.4. Piloto

En la Tabla 4.7 se muestra la descripción de la clase que representa a un piloto en una carrera. Se caracteriza por un identificador único, combustible y neumáticos de su coche, estrategia a seguir, ubicación en el circuito y otra información relativa a la situación de carrera (posición, último tiempo de vuelta, número de paradas, etcétera).

4.1.5. Carrera

Finalmente, una carrera está caracterizada por los pilotos que están participando en la carrera, circuito en el que sucede, número de vueltas, si se ha empezado o no y si se ha acabado o no. En la Tabla 4.8 se describe la clase que representa una carrera en el simulador junto con sus atributos.

4.2. Estado

En el contexto de lo explicado en el capítulo anterior (ver Subsección 3.1.2), decidir la estrategia en una carrera de F1 es una tarea episódica, con el estado inicial siendo el instante discreto $t = 0$ y el estado final $t = n\_vueltas - 1$ (la decisión de parar o no se toma cada vuelta a llegar a la entrada al pit lane, pero en la última vuelta no tiene sentido parar, por lo que se excluye). En esta sección se van a explicar las variables que van a componer el estado y las características del mismo en cada instante discreto t.

4.2.1. Variables de estado

El estado del simulador estará compuesto por las siguientes variables:

Piloto			
Definición	Piloto que participa en una carrera		
Nombre	**Definición**	**Tipo**	**Restricciones**
id	Identificador único del piloto en una carrera	int	Rango [0, n_pilotos-1]
ltg	Vueltas restantes hasta el final de la carrera	int	Rango[0, n_vueltas]
porcentaje	Porcentaje de vuelta completada por el piloto en un momento determinado. Indica la ubicación en pista del piloto.	float	Rango[0, 1)
interval	Distancia en segundos al piloto de delante	float	interval >= 0
combustible	Combustible disponible en el coche del piloto para completar la carrera.	Combustible	
neumático	Neumático que tiene montado el piloto en el momento actual.	Neumático	
estrategia	Estrategia a seguir durante la carrera.	Estrategia	
posición	Posición en carrera del piloto.	int	Rango[1, n_pilotos]
plt	Tiempo de vuelta potencial. (mejor tiempo de vuelta posible)	float	plt >0
llt	Último tiempo por vuelta del piloto	float	llt >0
dos_tipos	Si el piloto ha puesto o no dos tipos de neumáticos de seco	bool	
en_pits	Si el piloto se encuentra actualmente en el pit lane	bool	
n_paradas	Número de paradas en boxes que ha hecho el piloto en la carrera	int	n_paradas >= 0

Tabla 4.7: Descripción de la clase que representa a un piloto que participa en una carrera en el simulador

- Para cada piloto, se incluyen las siguientes variables: interval, ltg, llt, posición, plt, neumático.compuesto, neumático.vueltas_usadas, dos_tipos, n_paradas.

Lo anterior implica que, si contamos con 20 pilotos en la carrera, habrá un total de 160 variables en referencia a los pilotos que componen el estado del simulador.

Carrera			
Definición	Carrera del simulador		
Nombre	**Definición**	**Tipo**	**Restricciones**
pilotos	Lista de pilotos participando en la carrera	Lista[Piloto]	Lista no vacía
circuito	Circuito en el que se corre la carrera	Circuito	
n_vueltas	Número de vueltas de la carrera	int	n_vueltas >0
empezada	Si la carrera ha empezado o no	bool	
terminada	Si la carrera ha terminado o no (a todos los pilotos de la carrera les falta 0 vueltas para terminar)	bool	

Tabla 4.8: Descripción de la clase que representa una carrera en el simulador.

4.2.2. Estado Inicial (t = 0)

Cuando se arranque la simulación, el piloto que estamos controlando saldrá en una posición, que puede ser aleatoria o puede ser la última (dependiendo de la configuración del simulador). Durante esa primera vuelta no habrá adelantamientos y el estado que se tendrá en $t = 0$ es del momento en el que el piloto que controlamos llega a la entrada del pit lane para tomar la decisión de si parar o no.

4.2.3. Estados Intermedios (t > 0 & t < n_vueltas-1)

Los estados intermedios corresponden al momento en el que el piloto que se está controlando se encuentra la entrada al pit lane (momento último para decidir si se para o no), desde la segunda vuelta (t = 1) hasta la penúltima vuelta incluida, t = n_vueltas-2, ya que en la última vuelta no se puede hacer una parada.

4.2.4. Estado Final (t = n_vueltas-1)

El simulador se encuentra en el estado final (terminal) ($t = n\_vueltas - 1 = T$) cuando la carrera ha finalizado, el número de vueltas para el final de todos los pilotos es igual a cero.

4.3. Acciones

Hay 4 posibles acciones que se pueden realizar en cada paso de la simulación (momento en el que el piloto que se está controlando llega a la entrada del pit lane):

- 0: No parar.
- 1: Parar y poner un nuevo juego de neumáticos blandos.
- 2: Parar y poner un nuevo juego de neumáticos medios.
- 3: Parar y poner un nuevo juego de neumáticos duros.

Por tanto, se define A como el conjunto de acciones posibles,

$$A = \{0, 1, 2, 3\}$$

4.4. Funcionamiento del simulador

En esta sección se va a explicar el funcionamiento del simulador, cómo se va a avanzar desde el instante t al instante $t + 1$ teniendo en cuenta las acciones del agente que estará interactuando con él.

4.4.1. Avance en el simulador (t $\geq$ 0 & t $<$ n_vueltas-1, a $\in A$)

Tiempo por vuelta: Neumáticos y Combustible

El tiempo por vuelta de cada uno piloto se calcula según dos variables: el rendimiento del neumático actual que tiene cada uno en su coche y el efecto de la masa del combustible restante en su coche.

Efecto de los Neumáticos Sea *ci* el circuito en el que el piloto está corriendo, n un neumático que lleva el coche de piloto, se define $n.t\_min$ como el efecto en el tiempo por vuelta de un neumático nuevo:

$$n.t\_min = ci.t\_base + n.compuesto.t\_add \tag{4.1}$$

Teniendo en cuenta la degradación, se define el efecto en el tiempo por vuelta del neumático en un momento determinado:

$$n.efecto\_neumaticos = n.t\_min + n.compuesto.degradacion(n.vueltas\_usadas) \tag{4.2}$$

Efecto del Combustible Sea f el combustible del coche de un piloto en una carrera, se define el combustible consumido en un momento determinado:

$$f.combustible\_consumido = (f.consumo\_por\_vuelta \cdot f.vueltas\_consumidas) \tag{4.3}$$

El combustible restante:

$$f.combustible\_restante = f.masa\_combustible - f.combustible\_consumido \tag{4.4}$$

Por tanto, se define el efecto de la masa del combustible:

$$f.efecto\_combustible = f.combustible\_restante \cdot f.efecto\_masa \tag{4.5}$$

Tiempo de vuelta potencial Por tanto, sea p un piloto, se define su tiempo por vuelta si se encuentra en aire limpio [3], es decir, el tiempo de vuelta potencial (plt) como:

$$p.plt = n.efecto\_neumaticos + f.efecto\_combustible \tag{4.6}$$

Paso del tiempo

La forma de llegar avanzar en el simulador de un estado S_t al estado S_{t+1} dada la acción A_t es ir simulando pequeños pasos del tiempo, según la situación de cada piloto.

En concreto, se simula el tiempo mínimo necesario para que ocurra un evento relacionado con cualquier piloto. Ese evento podrá ser: **1)** llegar a la entrada del pit lane, **2)** es llegar a la salida del pit lane si el piloto ha entrado a boxes y **3)** será llegar a meta si se encuentra en la última vuelta (en la que no puede parar). Si un piloto p ha terminado la carrera ($p.ltg = 0$) no va a tener más eventos, por lo que para él, su tiempo mínimo necesario para que ocurra un evento será ∞.

Por tanto, sea p un piloto, se define el tiempo que tiene que transcurrir hasta que ocurra el siguiente evento como:

$$tiempo\_hasta\_evento(p) = \begin{cases} \infty, & \text{si } p.ltg = 0 \\ p.tiempo\_hasta\_meta, & \text{si } p.ltg = n\_vueltas \\ p.tiempo\_hasta\_pit\_exit, & \text{si } p.en\_pits \\ p.tiempo\_hasta\_pit\_entry, & \text{en otro caso} \end{cases} \tag{4.7}$$

Por tanto, el tiempo mínimo a simular será:

$$tiempo\_a\_simular = \min_{p \in pilotos} p.tiempo\_hasta\_evento(p) \tag{4.8}$$

[3] Se considera que un piloto está en aire limpio si no tiene ningún otro piloto delante que vaya más lento y, por tanto, su tiempo por vuelta real es igual al tiempo por vuelta potencial. Si está en aire sucio, su tiempo por vuelta real es más lento que su tiempo por vuelta potencial.

El tiempo mínimo para que suceda cualquiera de esos eventos entre todos los pilotos transcurrirá para todos igual y se calculará la nueva posición de cada piloto en pista (porcentaje de vuelta completado) de acuerdo a su tiempo de vuelta potencial y el tiempo transcurrido. Por último, se recalculan los intervalos y las posiciones de todos los pilotos tras haber pasado ese tiempo.

Porcentaje a avanzar Una vez se conoce para un piloto su *plt* y el *tiempo_a_simular*, suponiendo que está en aire limpio, se puede calcular el porcentaje que va a avanzar un piloto p en ese tiempo.

Se define *porcentaje_a_avanzar* como:

$$p.porcentaje\_a\_avanzar = \frac{tiempo\_a\_simular}{p.plt} \tag{4.9}$$

Adelantamientos

Una de las partes más relevantes dentro del simulador es el modelo de adelantamientos. En cada circuito es más o menos difícil adelantar, según el ancho de la pista, el número de curvas lentas y la longitud de las curvas.

Para simplificarlo, en este simulador, si en el tiempo mínimo a simular (ver ecuación 4.8) se estima que un piloto va a avanzar más que el que tiene delante y, además, le es suficiente para adelantarlo, entonces adelantará ese piloto con una cierta probabilidad *p_adelantamiento* particular a cada circuito.

Sea x un número pseudoaleatorio en el rango $[0, 1)$, c un circuito donde se está celebrando una carrera, p_1 un piloto, p_2 otro piloto detrás de p_1, *tiempo_hasta_siguiente* el tiempo necesario para que p_2 alcance a p_1, siendo $tiempo\_hasta\_siguiente < tiempo\_a\_simular$, y siendo :

$$p_2.porcentaje\_a\_avanzar = \begin{cases} \frac{tiempo\_a\_simular}{p_2.plt}, & \text{si } x < c.p\_adelantamiento \\ \frac{tiempo\_hasta\_siguiente - d\_no\_ov}{p_2.plt}, & \text{en otro caso} \end{cases} \tag{4.10}$$

Es decir, si p_2 consigue adelantar a p_1, avanza todo lo que pueda y, sino, se queda detrás de p_1 a cierta distancia preestablecida. Si $tiempo\_hasta\_siguiente \geq tiempo\_a\_simular$, entonces *porcentaje_a_avanzar* se define de acuerdo a 4.9.

4.4.2. Pseudocódigo

En el Algoritmo 8 se muestra el pseudocódigo que describe el funcionamiento del simulador, en el que se simula el paso del tiempo desde el estado t hasta el estado $t+1$, con $0 \leq t \leq n\_vueltas - 1$. Se utilizan funciones auxiliares que se comentan aquí brevemente:

- **debe_parar()**: Se comprueba la estrategia del piloto que recibe la llamada a esta función y, si tiene algún pitstop cuya vuelta de parada coincida con la vuelta en la que está el piloto, entonces retorna $true$, sino $false$.
- **pit(a)**: El piloto que recibe la llamada a esta función entrará en boxes. Si es el piloto que está controlando el agente, pondrá el neumático indicado por el argumento opcional a (referido a la acción del agente en el instante t). Si el piloto no es el controlado por el agente, el argumento se ignora y se realiza la parada para poner el neumático del pitstop correspondiente a la vuelta en la que se encuentra.
- **get_tiempo_minimo_a_simular()**: calcula el tiempo mínimo a simular de acuerdo a la ecuación 4.8.
- **avanzar(tiempo_a_simular)**: El piloto que recibe la llamada a esta función progresa por el circuito de acuerdo a lo descrito en la Subsección 4.4.1.
- **recalcula_posiciones_e_intervalos(pilotos)**: Ordena a los pilotos de forma ascendente por el número de vueltas restantes hasta el final (ltg) y como segunda clave de ordenación de forma descendente por el porcentaje de vuelta completado (porcentaje). Una vez reordenados los pilotos, se les actualiza su posición y la distancia en segundos con el piloto que tienen inmediatamente delante (interval).

4.4.3. Variabilidad

En la realidad, no hay dos momentos iguales en una carrera de Fórmula 1. El rendimiento de los neumáticos varía en cada stint de la misma carrera, por lo que existirá una variación mínima aleatoria en el tiempo por vuelta y degradación por vuelta entre cada juego de neumáticos del mismo compuesto.

Cada uno de los pilotos que no se está controlando utilizará una estrategia predefinida estimada de carreras reales, pero no será siempre la misma, sino que habrá variabilidad en el número de vuelta en la que se realiza cada pitstop. Además, cada estrategia predefinida tendrá una probabilidad de ser asignada a un piloto. El neumático de salida del piloto que se está controlando será el correspondiente al de una de las estrategias predefinidas, con la misma probabilidad que para el resto de pilotos.

4.5. Restricciones

Según el artículo 30.5 m) del reglamento deportivo de la F1 [8], si la carrera es en seco, es necesario que un piloto utilice al menos dos tipos de neumáticos diferentes durante una carrera (p. ej., blandos y medios), y uno de ellos ha de ser el tipo de neumático obligatorio para la carrera. Por tanto, si un piloto no utiliza dos neumáticos distintos en una carrera, pasará automáticamente a ser el último clasificado y se le dará una recompensa negativa adicional.

Algoritmo 8 Pseudocódigo de la simulación del S_t al estado S_{t+1}, con $0 \leq t < n\_vueltas - 1$. A_t es la acción a tomar. Retorna el nuevo estado tras tomar la acción a.

```
simular_paso_del_tiempo(A_t):
    if a ≠ 0 then
        piloto_siendo_controlado.pit(A_t)
    end if
    nuevo_estado ← false
    while not carrera.terminada do
        tiempo_a_simular ← get_tiempo_minimo_a_simular()
        for all p en carrera.pilotos [con pilotos en pits al final] do
            if p.ltg = carrera.n_vueltas then
                continue
            end if
            p.avanzar(tiempo_a_simular)
            cerca_de_pits ← piloto.porcentaje = circuito.pit_entry
            if p = piloto_siendo_controlado and cerca_de_pits then
                nuevo_estado ← true
                continue
            end if
            if cerca_de_pits and p.debe_parar() then
                p.pit()
            end if
        end for
        recalcula_posiciones_e_intervalos(carrera.pilotos)
        if nuevo_estado then
            return estado_simulacion
        end if
    end while
    return estado_simulacion
```

Dado que en la vida real los pilotos no tienen neumáticos ilimitados, y que en el simulador cada vez que se para los neumáticos son nuevos, se dará una recompensa negativa cada vez que paren si han parado 5 veces o más.

4.6. Ejemplo de Simulación

En el Apéndice A se muestra un ejemplo de simulación de una carrera, mostrando para cada instante discreto t el estado S_t y la acción A_t tomada según ese estado.

4.7. Adaptación del Simulador a un entorno Gymnasium

Gymnasium (anteriormente conocido como Gym) [10] es una librería de aprendizaje por refuerzo que ofrece una API para que los entornos y los algoritmos de aprendizaje se comuniquen de manera estándar. Ofrece una gran cantidad de entornos en los que poder probar los algoritmos de aprendizaje por refuerzo, desde algunos básicos que pueden solucionarse con algoritmos tabulares, a los clásicos juegos de Atari que el algoritmo DQN resolvió con éxito [17]. Gymnasium permite también la creación de un entorno propio, que es lo que se ha hecho con el simulador desarrollado.

4.7.1. Integración con el Simulador

Para integrar el simulador con Gymnasium se crea un entorno, que es una clase Python que debe implementar una interfaz con los siguientes métodos [11]:

- **reset()**: Resetea el entorno al estado inicial (ver Subsección 4.2.2). Retorna lo siguiente:
  - **observation**: Observación del estado inicial del entorno.
  - **info**: Diccionario que contiene información auxiliar.
- **step(a)**: Ejecuta la acción a en el entorno (siguiendo lo descrito en la Sección 4.4), y retorna lo siguiente:
  - **observation**: Observación del nuevo estado del entorno.
  - **reward**: Recompensa obtenida al tomar la acción a.
  - **terminated**: Señal que indica si el agente ha llegado al estado terminal (en nuestro caso, al final de la carrera).
  - **truncated**: Señal que indica si se termina el episodio de manera prematura antes de llegar al estado terminal (no se usará en el caso de este entorno).
  - **info**: Diccionario que contiene información auxiliar.

5: Diseño Experimental

En este capítulo se va a explicar el diseño de los experimentos que se van a realizar. Primero, se explica cómo se va a construir la observación que va a tener el agente del estado del simulador. Después, se explicará la configuración del simulador en los experimentos que se van a realizar, incluyendo neumáticos, estrategias predefinidas, configuración de pits, adelantamientos, etcétera. Tras esto, se explicarán los algoritmos que se van a utilizar y sus hiperparámetros, así como también las funciones de recompensa diseñadas.

5.1. Observación

La observación que va a tener el agente del estado del simulador es la descrita en la Sección 4.2, una observación completa del mismo.

5.1.1. Normalización de la observación

Dado que vamos a utilizar *deep reinforcement learning*, es conveniente que las observaciones estén normalizadas en el rango [0,1], que es el rango con el que mejor trabajan las redes neuronales. A continuación se explica cómo se ha realizado la normalización de cada uno de los atributos del piloto que forman el estado (descritos en 4.2.1).

- **interval**: La distancia en segundos con el piloto de delante se normaliza dividiéndolo por el tiempo por vuelta potencial del piloto actual (que es el tiempo necesario para alcanzarle si el piloto de delante se quedara parado).
- **ltg**: El número de vueltas para el final se normaliza dividiéndolo por el número total de vueltas de la carrera.
- **llt**: El último tiempo de vuelta se normaliza dividiéndolo por el peor último tiempo de vuelta de entre todos los pilotos.
- **posición**: La posición de un piloto se normaliza dividiendo esa posición-1 por el número de pilotos en carrera - 1.

- **plt**: El tiempo por vuelta potencial se normaliza de forma similar a llt: se divide por el peor tiempo de vuelta potencial del resto de los pilotos.
- **neumático.compuesto**: Hay un total de tres tipos de neumáticos (blando, medio, duro). Al ser una variable categórica, le vamos a aplicar una codificación *One Hot*[4], de tal manera que ahora se tendrán para cada piloto 3 entradas a la red representando el tipo de neumático que lleva en ese momento.
- **neumático.vueltas_usadas**: El número de vueltas usadas de un compuesto se normaliza dividiéndolo por el número de vueltas totales de la carrera.
- **dos_tipos**: Es una variable booleana, por lo que está ya en el rango [0,1].
- **n_paradas**: El número de paradas de un piloto en una carrera se normaliza de la siguiente manera: si es mayor o igual al número máximo de paradas permitido, su valor es 1. En caso contrario se normaliza dividiéndolo por el número máximo de paradas permitido.

Por tanto, habrá un total de 11 variables por piloto. Es decir, en el caso en el que hay 20 pilotos en pista, se tendrán 220 variables que forman la observación del estado de la carrera.

5.2. Configuración del Simulador

Los experimentos se realizarán utilizando una configuración correspondiente al Gran Premio de Mónaco, para el que se ha estimado el desgaste de cada compuesto de neumáticos y las estrategias posibles utilizando datos históricos de carreras reales. En concreto, se han utilizado los datos del Gran Premio de Mónaco del año 2021, en el que no hubo incidentes y la carrera fue en seco, por lo que es más sencillo estimar los datos de dicha carrera.

Para obtener datos de carreras reales de F1 se ha utilizado el paquete de Python FastF1 [7], que provee de datos reales de Grandes Premios de Fórmula 1 para analizar resultados, datos de telemetría y cronometraje.

5.2.1. Neumáticos

Para estimar la degradación de los neumáticos, se cogieron tiempos por vuelta de stints con cada uno de los tipos de neumáticos y se aproximó una función cuadrática para los tiempos por vuelta en función del número de vuelta. En la Figura 5.1 se muestra el efecto en el tiempo por vuelta de los neumáticos para cada tipo. Se puede ver que cuando son nuevos, los más rápidos son los blandos, luego los medios y los más lentos los duros. Con el paso de las vueltas el desgaste consigue que el neumático duro sea el más rápido de los

[4]La codificación One Hot convierte una variable categórica en tantas variables numéricas como categorías haya, con el objetivo de que la red neuronal no asuma un orden natural entre las categorías.

tres. Los neumáticos **blandos** se representan con el **color rojo**, los **medios** con el **color amarillo** y los **duros** con el **color blanco**.

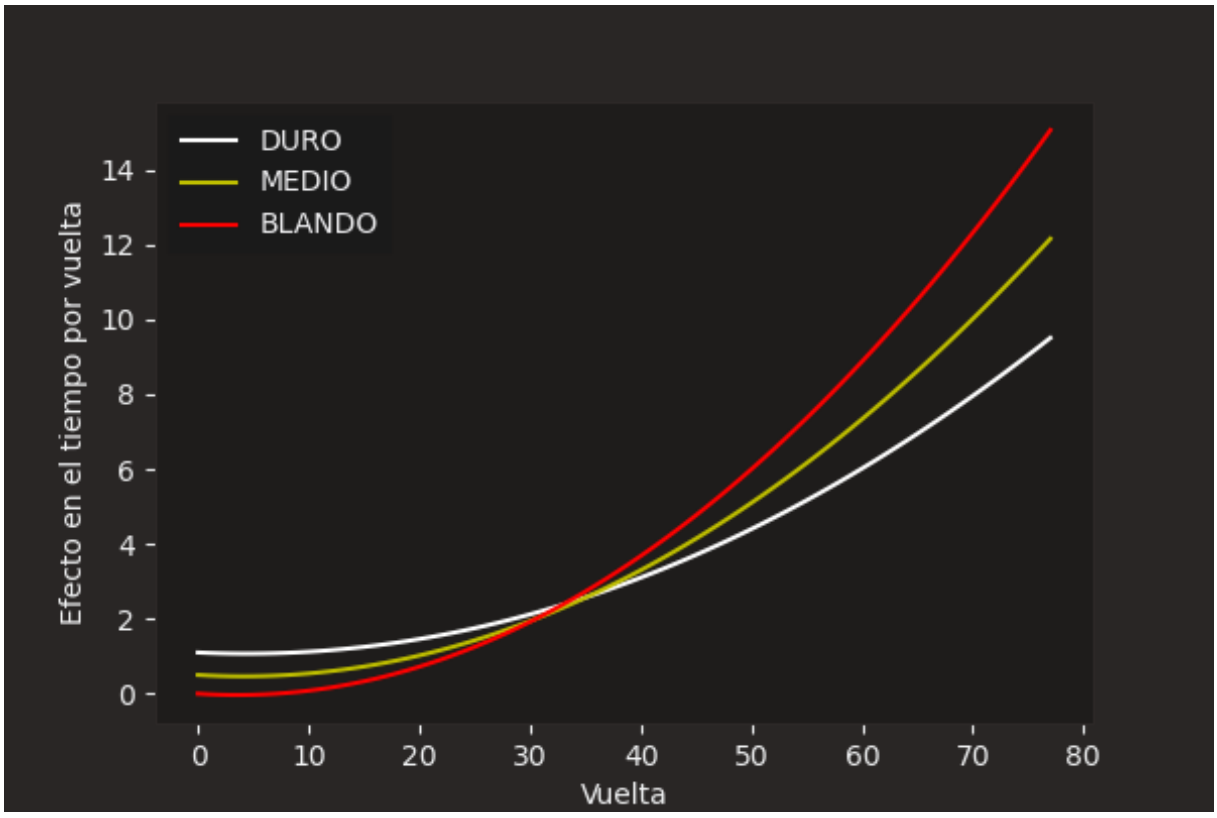


Figura 5.1: Efecto del desgaste de los neumáticos.

5.2.2. Estrategias

En la Figura 5.2 se muestran las estrategias que siguió cada uno de los pilotos en la carrera del Gran Premio de Mónaco 2021. Podemos ver que hay cinco estrategias diferentes, todas ellas a una parada: empezar con neumáticos blandos y parar a poner neumáticos duros, o parar a poner neumáticos medios; empezar con neumáticos medios y parar a poner neumáticos blandos o parar a por neumáticos duros; o empezar con neumáticos duros y parar a por blandos.

Para cada estrategia (siempre que se haya completado con éxito) se ha cogido el número medio de vueltas dadas con cada neumático para obtener la vuelta de parada en cada estrategia. La probabilidad de que un piloto siga una de las estrategias descritas viene dada por la frecuencia relativa de cada una de ellas. Por último, la vuelta en la que se realiza la parada tendrá cierta variación. En la Tabla 5.1 se muestran los datos sobre las estrategias posibles.

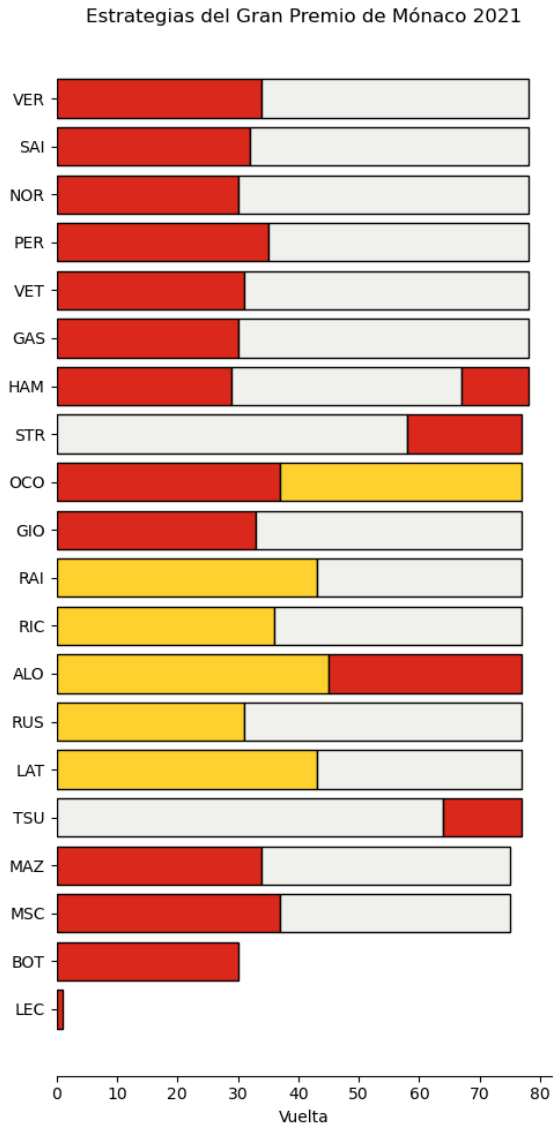


Figura 5.2: Estrategias de los pilotos en el Gran Premio de Mónaco del año 2021.

Estrategia	Neumatico Salida	PitStop			Probabilidad
		Neumático	Vuelta	Variación Vuelta	
blando-duro	Blando	Duro	32	2	0,6
blando-medio	Blando	Medio	37	2	0,05
medio-blando	Medio	Blando	45	2	0,05
medio-duro	Medio	Duro	38	2	0,2
duro-blando	Duro	Blando	61	2	0,1

Tabla 5.1: Estrategias posibles en la configuración del simulador elegida para los experimentos

5.2.3. Adelantamientos, Pits y Combustible

Si un piloto va más rápido que el que tiene delante y tiene opción a adelantarle, habrá un 15 % de probabilidad de que el adelantamiento tenga éxito. Se asume que la entrada a los pits está en el 98 % del circuito, y la salida en el 2 %. Si se para, el tiempo desde que se entra hasta que se sale de los pits será de 24 segundos, incluyendo la parada. Se estima un efecto en el tiempo por vuelta por unidad de masa de combustible (en este caso, por cada kg) de 0,07 segundos, con un consumo por vuelta de 1,282 kgs. Cada carrera tendrá un total de 78 vueltas.

5.3. Algoritmos

Para los experimentos se van a utilizar los algoritmos DQN, QR-DQN y A2C, explicados en la Sección 3.2. La implementación de los algoritmos DQN y A2C se obtiene de la versión estable de Stable Baselines 3 [24] y la de QR-DQN de su versión experimental Contrib [24]. A continuación se describen los hiperparámetros elegidos para cada algoritmo para la ejecución de los experimentos.

5.3.1. DQN

A continuación se listan los hiperparámetros de la implementación de DQN de Stable Baselines 3 [31] cuyo valor es distinto al que viene por defecto, así como una pequeña descripción de los mismos y de los distintos valores probados para cada uno. Los que no aparecen mantienen su valor por defecto.

- **gamma** (γ): factor de descuento de la recompensa, en rango [0, 1]. Como las funciones de recompensa estarán basadas en la posición de nuestro piloto, y cuanto más cerca estemos del instante final T más importancia tiene esa posición, $gamma = 1$.
- **exploration**: probabilidad de tomar una acción aleatoria en cada instante de tiempo. Comprende los siguientes hiperparámetros:
  - **exploration_fraction**: Fracción del tiempo de entrenamiento durante el que se reducirá la tasa de exploración, $exploration\_fraction = 0{,}2$. Incrementado de 0,1 a 0,2 para aumentar el tiempo de exploración. Se probó también 0,4, pero ese valor era demasiado grande.
  - **exploration_initial_eps**: Valor inicial de la tasa de exploración, $exploration\_initial\_eps = 0{,}25$. Reducido de 1 a 0,25, ya que con un valor tan alto el agente no era capaz de aprender nada.
  - **exploration_final_eps**: Valor final de la tasa de exploración, que se alcanzará cuando haya pasado la fracción del entrenamiento definida por exploration_fracion, $exploration\_final\_eps = 0{,}020$. Reducido de 0,05 a 0,02 para evitar demasiada exploración, de esta manera hay una acción aleatoria cada episodio aproximadamente.

- **learning_rate**: tasa de aprendizaje del modelo, $learning\_rate = 0{,}00005$. Reducida de 0,0001 a 0,00005 debido a que el entrenamiento con el valor original era muy poco estable.
- **train_freq**: frecuencia de actualización del modelo, $train\_freq = 16$ timesteps. Aumentado ligeramente de 4 a 16.
- **buffer_size**: Tamaño del buffer de repetición, $buffer\_size = 100000$. Reducido de 1000000 a 100000 para tratar de conseguir que se utilicen lo mismo las experiencias recientes que las antiguas.
- **batch_size**: Tamaño del minibatch para la actualización del modelo, $batch\_size = 256$. Aumentado de 32 a 256 para mejorar la estabilidad del entrenamiento, al coger un número de timesteps equivalente aproximadamente a 3 episodios para hacer la actualización de la red.
- **learning_start**: Número de transiciones a almacenar antes de comenzar el entrenamiento, $learning\_start = 5000$. Incrementado de 100 a 5000 para tener un número suficiente de transiciones para empezar a aprender.
- **net_arch**: Es la arquitectura de la red neuronal que aproxima la función de valor de pares estado-acción. Se han dejado los valores por defecto, pero conviene comentarlos: Una capa de entrada con 220 neuronas, una capa oculta con 64 neuronas y una capa de salida con 4 neuronas (una por cada acción).
- **n_timesteps**: Número de timesteps durante los que se va a entrenar $n\_timesteps = 1000000$.

5.3.2. QR - DQN

A continuación se listan los hiperparámetros de la implementación de QR-DQN de Stable Baselines 3 [33] cuyo valor es distinto al que viene por defecto, así como una pequeña descripción de los mismos y de los distintos valores probados para cada uno. Los que no aparecen mantienen su valor por defecto.

- **n_quantiles**: número de cuantiles que se van a aproximar, $n\_quantiles = 21$. Reducido de 200 a 21, porque el tiempo de entrenamiento era excesivamente grande con ese valor. Se probó también con algún valor más bajo como 50, pero ocurría lo mismo. Finalmente se dejó el valor 21.
- **gamma** (γ): factor de descuento de la recompensa, en rango [0, 1]. Como las funciones de recompensa estarán basadas en la posición de nuestro piloto, y cuanto más cerca estemos del instante final T más importancia tiene esa posición, $gamma = 1$.
- **exploration**: probabilidad de tomar una acción aleatoria en cada instante de tiempo. Comprende los siguientes hiperparámetros:

  - **exploration_fraction**: Fracción del tiempo de entrenamiento durante el que se reducirá la tasa de exploración, $exploration\_fraction = 0{,}2$. Incrementado de 0,1 a 0,2 para aumentar el tiempo de exploración. Se probó también 0,4, pero ese valor era demasiado grande.
  - **exploration_initial_eps**: Valor inicial de la tasa de exploración, $exploration\_initial\_eps = 0{,}25$. Reducido de 1 a 0,25, ya que con un valor tan alto el agente no era capaz de aprender nada.
  - **exploration_final_eps**: Valor final de la tasa de exploración, que se alcanzará cuando haya pasado la fracción del entrenamiento definida por exploration_fracion, $exploration\_final\_eps = 0{,}020$. Reducido de 0,05 a 0,02 para evitar demasiada exploración, de esta manera hay una acción aleatoria cada episodio aproximadamente.

- **learning_rate**: tasa de aprendizaje del modelo, $learning\_rate = 0{,}00005$. Reducida de 0,0001 a 0,00005 debido a que el entrenamiento con el valor original era muy poco estable.
- **train_freq**: frecuencia de actualización del modelo, $train\_freq = 16$ timesteps. Aumentado ligeramente de 4 a 16.
- **buffer_size**: Tamaño del buffer de repetición, $buffer\_size = 100000$. Reducido de 1000000 a 100000 para tratar de conseguir que se utilicen lo mismo las experiencias recientes que las antiguas.
- **batch_size**: Tamaño del minibatch para la actualización del modelo, $batch\_size = 256$. Aumentado de 32 a 256 para mejorar la estabilidad del entrenamiento, al coger un número de timesteps equivalente aproximadamente a 3 episodios para hacer la actualización de la red.
- **learning_start**: Número de transiciones a almacenar antes de comenzar el entrenamiento, $learning\_start = 5000$. Incrementado de 100 a 5000 para tener un número suficiente de transiciones para empezar a aprender.
- **net_arch**: Es la arquitectura de la red neuronal que aproxima la distribución del retorno para cada par estado-acción. Se han dejado los valores por defecto, pero conviene comentarlos: Una capa de entrada con 220 neuronas, una capa oculta con 64 neuronas y una capa de salida con $4 \times n\_quantiles = 84$ neuronas (4 acciones con 21 cuantiles a estimar por cada acción).
- **n_timesteps**: Número de timesteps durante los que se va a entrenar $n\_timesteps = 1000000$.

5.3.3. A2C

A continuación se listan los hiperparámetros de la implementación de A2C de Stable Baselines 3 [30] cuyo valor es distinto al que viene por defecto, así como una pequeña

descripción de los mismos. Los que no aparecen mantienen su valor por defecto. En el caso de este algoritmo se utilizan varios entornos ejecutándose en paralelo n_env, lo cual hace que se tenga una mejor estabilidad en el entrenamiento.

- **gamma (γ)**: factor de descuento de la recompensa, en rango [0, 1]. Como las funciones de recompensa estarán basadas en la posición de nuestro piloto, y cuanto más cerca estemos del instante final T más importancia tiene esa posición, $gamma = 1$.
- **n_steps**: número de pasos a ejecutar en cada entorno por actualización de la política, $n\_steps = 16$. En este caso habrá dos entornos ejecutándose en paralelo, $n\_envs = 2$, por lo que $batch\_size = n\_steps \cdot n\_env = 32$.
- **learning_rate**: tasa de aprendizaje del modelo, $learning\_rate = 0{,}00035$. Reducida de 0,0007 a 0,00035 porque con el valor por defecto el entrenamiento no era estable.
- **net_arch**: Son las arquitecturas de las redes neuronales que van a aproximar la función de la política y la función de valor. La arquitectura de la red que aproxima la función de la política tiene una capa de entrada con 220 neuronas, una capa oculta con 64 neuronas y una capa de salida con 4 neuronas (cada una indicando la probabilidad de tomar cada acción). La arquitectura de la red neuronal que aproxima la función de valor tiene una capa de entrada con 220 neuronas, una capa oculta con 64 neuronas y una capa de salida con 1 neurona (el valor del estado).
- **n_timesteps**: Número de timesteps durante los que se va a entrenar, $n\_timesteps = 1000000$.

5.3.4. Guardado del mejor modelo durante el entrenamiento

Es posible que durante la ejecución de un experimento el rendimiento de un modelo llegue a un máximo y después comience a bajar, por lo que se va a utilizar un *Evaluation Callback* [32]. Utilizando este Callback se va a evaluar el modelo cada 25 000 instantes de entrenamiento con 20 carreras distintas y se va a medir la recompensa media obtenida durante esa evaluación. Si es mayor que la de la anterior evaluación, se guardará ese modelo como el mejor obtenido hasta el momento.

Cada vez que se realiza la evaluación, las condiciones de inicio de las carreras son las mismas (orden de la parrilla de salida, estrategias y neumáticos de salida) para asegurar que los resultados no se vean influidos por la posición de salida y se puedan comparar.

5.4. Funciones de recompensa

Como se ha explicado ya, la función de recompensa es un elemento crítico en un sistema basado en aprendizaje por refuerzo. Es la única señal que indica al agente cómo de bien o mal lo está haciendo, por lo que es crítico diseñar una buena función de recompensa y estudiar varias para ver cuál es con la que mejores resultados se consiguen. En esta sección

se van a explicar las funciones de recompensa que se van a utilizar en los experimentos que se van a realizar.

5.4.1. Elementos comunes a todas las funciones de recompensa

Para cumplir con las restricciones del simulador (ver Sección 4.5), se va a dar una recompensa negativa al agente cada vez que tome una acción con la que se incumpla cualquiera de ellas:

- Si el agente al finalizar la carrera no ha utilizado los dos tipos diferentes de compuestos de seco, entonces tendrá una recompensa adicional de -1 (además de ser el último clasificado) al final de la carrera (cuando se certifica que ha incumplido la norma).
- Si el agente supera el número máximo de paradas establecido, significa que está intentando parar cuando no le quedan neumáticos disponibles, por lo que se le dará una recompensa de -1 cada vez que pare una vez haya superado el límite.

5.4.2. R1 - Posiciones ganadas por vuelta

La primera función de recompensa que se va a estudiar es bastante sencilla, en el instante discreto t de la carrera se va a dar al agente una recompensa que va a ser igual al número de posiciones ganadas (o perdidas) por el agente desde el estado $t-1$ hasta el instante t, es decir, posiciones ganadas (o perdidas) por vuelta.

$$R_1 = posicion_{t-1} - posicion_t \tag{5.1}$$

Los experimentos realizados con esta función de recompensa serán carreras en las que el piloto que se está controlando saldrá último siempre.

5.4.3. R2 - Posiciones ganadas por vuelta

La segunda función de recompensa que se va a utilizar es muy similar a la anterior, pero con la diferencia de que se va a dar más importancia al cambio de posiciones según queden menos vueltas.

$$R_2 = (posicion_{t-1} - posicion_t) \cdot \frac{(t+1)}{\text{n\_vueltas - 1}} \tag{5.2}$$

Los experimentos realizados con esta función de recompensa serán carreras en las que el piloto que se está controlando saldrá último siempre.

5.4.4. R3 - Recompensa por posición final

La última función de recompensa que se va a estudiar consiste en dar una recompensa según la posición final de piloto. De esta manera se podrá entrenar al agente en condiciones más realistas (con una parrilla de salida donde el orden de los pilotos es aleatorio).

Desde la primera posición a la última se ha asociado una recompensa a cada posición, que se dará al final de la carrera al agente según su posición. Se define la función $r3\_aux(p)$ como la función que recibe una posición y retorna la recompensa asociada, tal y como se muestra en la Tabla 5.2.

Posición	Recompensa	Posición	Recompensa	Posición	Recompensa	Posición	Recompensa
1	15	2	13.5	3	12,25	4	11
5	9,75	6	8,75	7	8,0	8	7,25
9	6,5	10	5,75	11	5,0	12	4,25
13	3,75	14	3,25	15	1,75	16	1,25
17	0,75	18	0,5	19	0,25	20	0

Tabla 5.2: Recompensa obtenida según la posición final en la carrera.

$$R3 = \begin{cases} r3\_aux(posicion), & \text{si } t = T \\ 0 & \text{en otro caso} \end{cases} \tag{5.3}$$

Se realizarán experimentos utilizando R3 con el agente saliendo último y con el agente saliendo en posición aleatoria, es decir, se ejecutarán el doble de experimentos para esta función de recompensa.

6: Experimentación

En este capítulo se explica cómo se ha llevado a cabo el proceso de experimentación y se presentan las gráficas de entrenamiento para cada experimento realizado, así como una comparación del rendimiento de los algoritmos según la función de recompensa.

6.1. Experimentos

6.1.1. Ejecución de los experimentos

A continuación se muestran los 12 experimentos realizados (3 algoritmos $\times$ 4 (funciones de recompensa R1, R2 y R3 con el piloto que se controla saliendo el último, y R3 saliendo en orden aleatorio)), y para cada uno de ellos se van a mostrar las curvas de entrenamiento (recompensa media por episodio a lo largo de los timesteps de entrenamiento). Las curvas de entrenamiento sombreadas son los valores reales, y las opacas están ligeramente suavizadas para que se puedan leer mejor. Las gráficas se han obtenido de Tensorboard [35].

Con el objetivo de conseguir resultados que no estén sesgados por la semilla inicial [13], se van a repetir cada uno de los experimentos planteados en el capítulo anterior tres veces, cada uno con una semilla de aleatoriedad diferente (en este caso, las semillas utilizadas han sido 1, 2 y 3).

Esto implica que se han realizado un total de $12 \times 3 = 36$ entrenamientos, teniendo cada uno de ellos una duración de entre 3 y 4 horas.

Experimento 1 - DQN R1

En la Figura 6.1 se muestra la recompensa media durante el entrenamiento para el primer experimento, para las tres repeticiones del mismo. Durante los primeros 200 000 timesteps en los que se está explorando bastante, la recompensa es negativa (de acuerdo con la recompensa negativa que se da al agente cada vez que para si supera el límite establecido), pero a partir de ahí la recompensa media por episodio va subiendo poco a poco a lo largo de todo el entrenamiento, llegando a una media de 5 posiciones ganadas

por episodio (como durante el entrenamiento sale último, esto significa que de media acaba 15º).

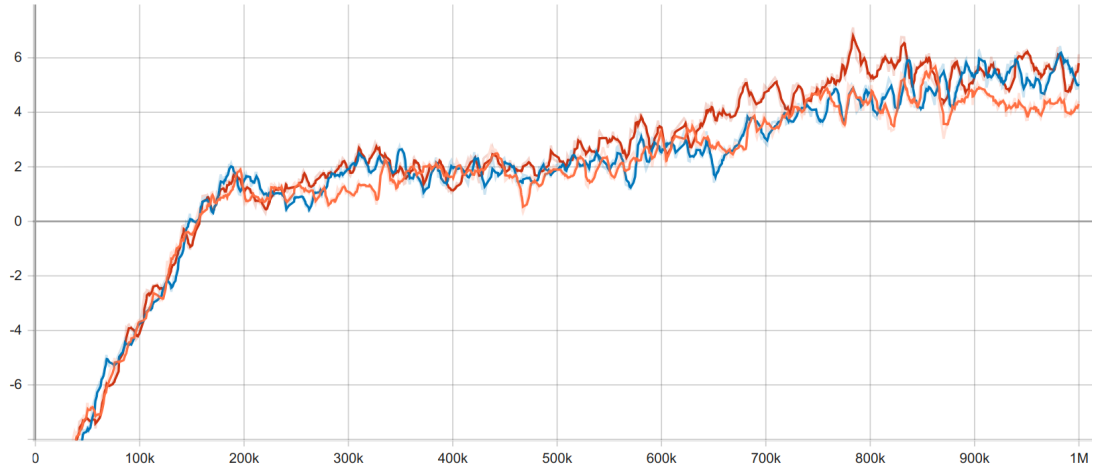


Figura 6.1: Recompensa media por episodio - Experimento 1 - DQN R1

Experimento 2 - DQN R2

En la Figura 6.2 se muestran la recompensa media durante el entrenamiento para el experimento 2, para las tres repeticiones del mismo. Teniendo en cuenta la definición de la función de recompensa R2 (ver Subsección 5.4.3), lo primero que podría estar aprendiendo el agente (a partir el instante 200 000, cuando se ha reducido mucho la exploración) es a no parar y ganar posiciones por ello, pero finalmente las perdería más tarde en la carrera y esto implica una recompensa acumulada negativa al final de la carrera. Por tanto, la subida en la recompensa media desde el instante 500 000 hasta el final del entrenamiento se podría explicar la subida en la recompensa media con que el agente aprende que es más importante mantener una posición alta al final de carrera que al principio.

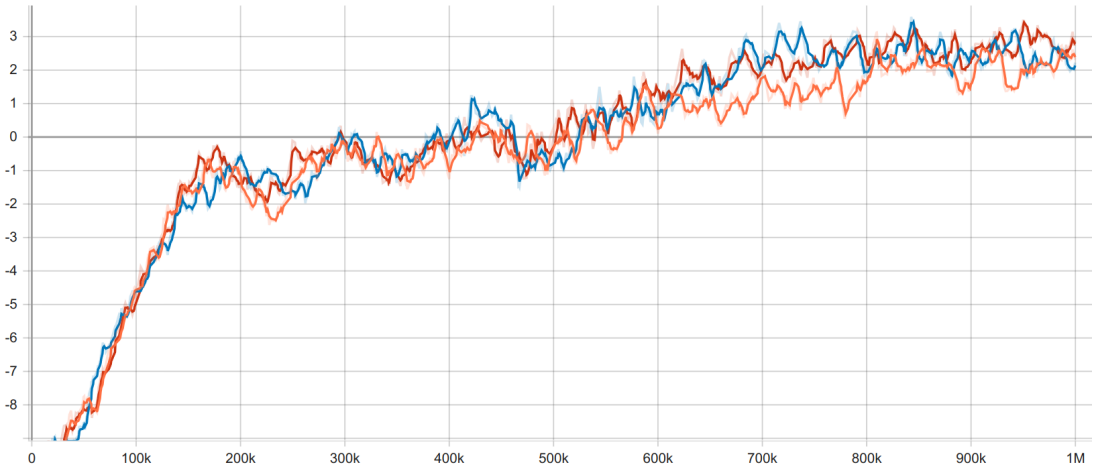


Figura 6.2: Recompensa media por episodio - Experimento 2 - DQN R2

Experimento 3 - DQN R3 Orden Salida Estable

En la Figura 6.3 se muestran la recompensa media durante el entrenamiento para el experimento 3, para las tres repeticiones del mismo. No supera el valor 2 de recompensa media, lo cual significa que no se supera la 15ª posición de media (ver Tabla 5.2), mientras que en el experimento 1 se llega casi a la 14ª posición. Según la gráfica, la recompensa media por episodio llega a un máximo local con valor 2 pasado el instante 500 00, y luego fluctúa hasta el final del entrenamiento. Esto puede deberse a que la recompensa se da al final del episodio y el buffer es demasiado pequeño, por lo que el agente deja de usar experiencias antiguas para entrenar y las 'olvida', lo que se conoce como *catastrophic forgetting* [26]. Una forma de mejorar el entrenamiento podría ser incrementar tanto $buffer\_size$ como $batch\_size$, de forma que se entrenara con experiencias antiguas más veces.

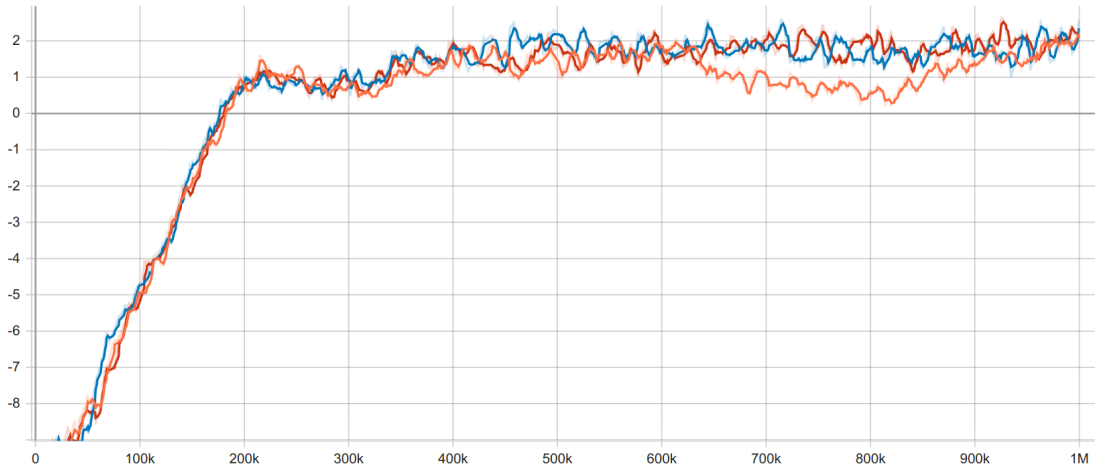


Figura 6.3: Recompensa media por episodio - Experimento 3 - DQN R3 - Entrenamiento con Orden de Salida Estable

Experimento 4 - DQN R3 Orden Salida Aleatorio

En la Figura 6.4 se muestran la recompensa media durante el entrenamiento para el experimento 4, para las tres repeticiones del mismo. Igual que en el experimento anterior, la recompensa media por episodio fluctúa mucho una vez pasado el instante 500 000. Los posibles motivos y soluciones son los mismos que en el experimento anterior.

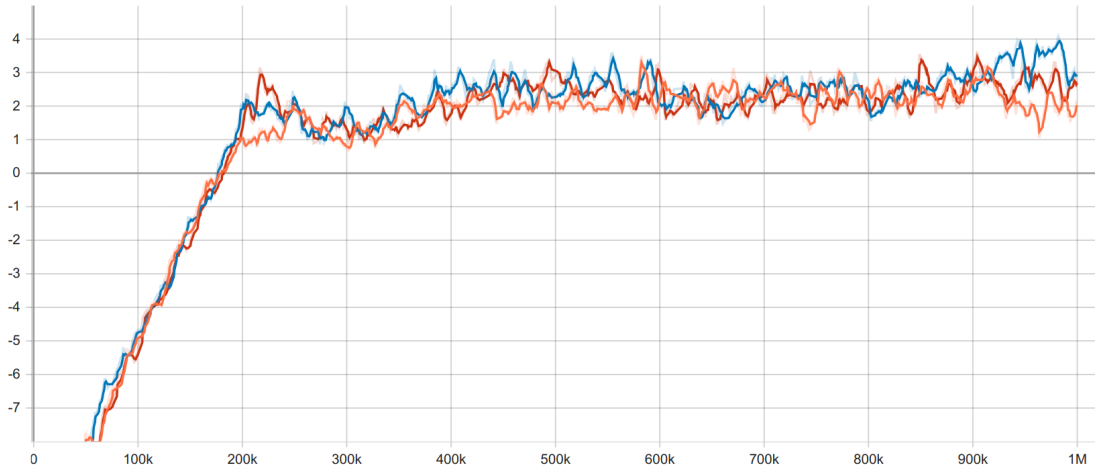


Figura 6.4: Recompensa media por episodio - Experimento 4 - DQN R3 - Entrenamiento con Orden de Salida Aleatorio

Experimento 5 - QR-DQN R1

En la Figura 6.5 se muestran la recompensa media durante el entrenamiento para el experimento 5, para las tres repeticiones del mismo. En comparación con DQN, el aprendizaje fluctúa más, y además el agente consigue ganar de media una posición menos con respecto a DQN, lo cual puede indicar que el aprendizaje ha sido peor.

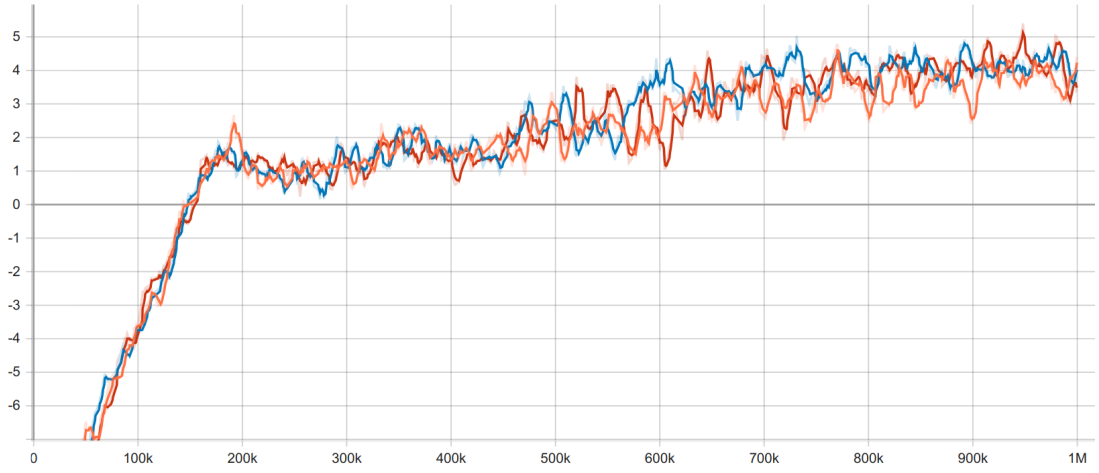


Figura 6.5: Recompensa media por episodio - Experimento 5 - QR-DQN R1

Experimento 6 - QR-DQN R2

En la Figura 6.6 se muestran la recompensa media durante el entrenamiento para el experimento 6, para las tres repeticiones del mismo. En este caso, comparado con DQN la estabilidad parece similar y el aprendizaje es más lento también.

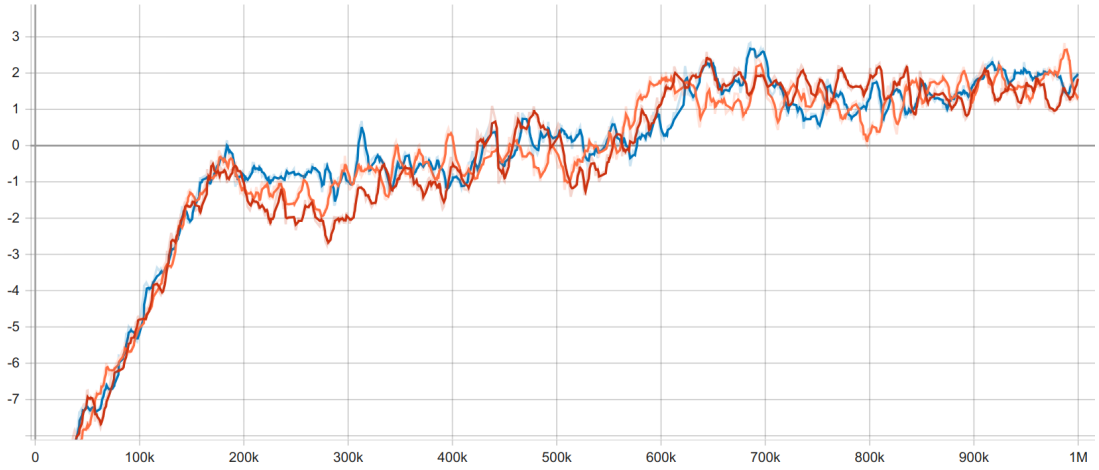


Figura 6.6: Recompensa media por episodio - Experimento 6 - QR-DQN R2

Experimento 7 - QR-DQN R3 Orden Salida Estable

En la Figura 6.7 se muestra la recompensa media durante el entrenamiento para el experimento 7, para las tres repeticiones del mismo. A partir del instante 400 000 la recompensa media comienza a caer hasta llegar al 0, y a partir del instante 700 000 sube de nuevo, quedándose cerca del máximo anterior. Es probable los motivos de esto sean similares a lo que se propone para DQN en el experimento 3, y se podrían probar las mismas soluciones.

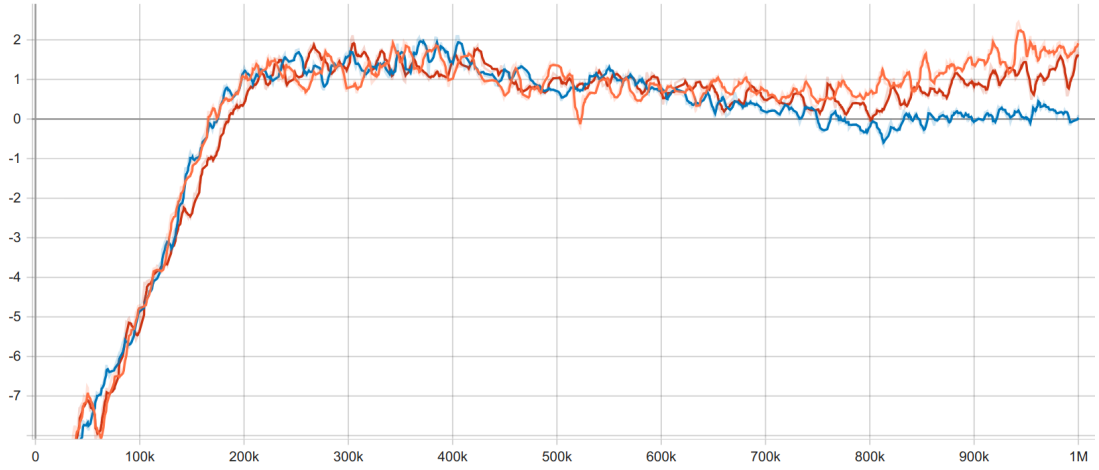


Figura 6.7: Recompensa media por episodio - Experimento 7 - QR-DQN R3 - Entrenamiento con Orden de Salida Estable

Experimento 8 - QR-DQN R3 Orden Salida Aleatorio

En la Figura 6.8 se muestran la recompensa media durante el entrenamiento para el experimento 8, para las tres repeticiones del mismo. Al igual que en el experimento

anterior, la recompensa cae, pero no recupera los valores de antes de la bajada. Los posibles motivos y soluciones de esto son los mismos que en el experimento anterior.

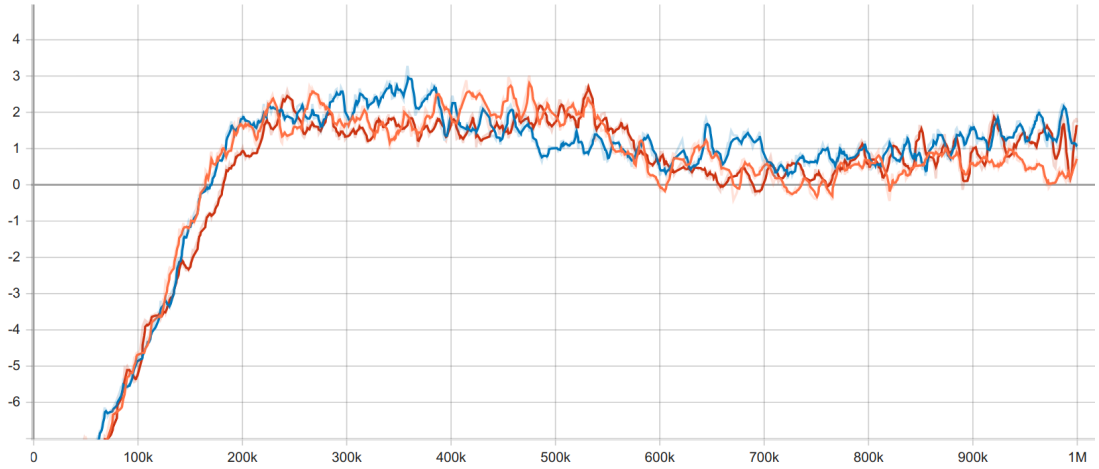


Figura 6.8: Recompensa media por episodio - Experimento 8 - QR-DQN R3 - Entrenamiento con Orden de Salida Aleatorio

Experimento 9 - A2C R1

En la Figura 6.9 se muestran la recompensa media durante el entrenamiento para el experimento 9, para las tres repeticiones del mismo. El agente aprende de forma rápida y monótona, y parece que la recompensa media por episodio seguiría aumentando con más instantes de entrenamiento.

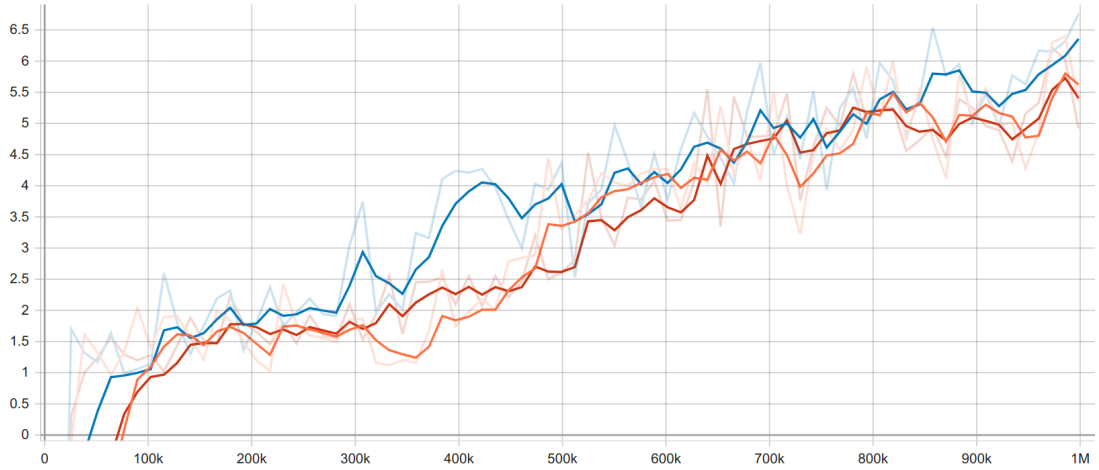


Figura 6.9: Recompensa media por episodio - Experimento 9 - A2C R1

Experimento 10 - A2C R2

En la Figura 6.10 se muestran la recompensa media durante el entrenamiento para el experimento 10, para las tres repeticiones del mismo. Igual que para R1, el aprendizaje es monótono y rápido.

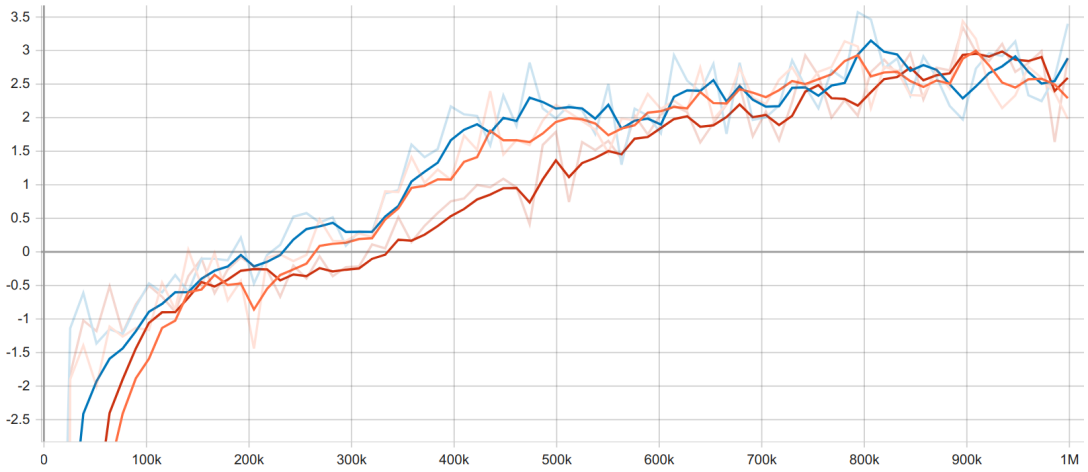


Figura 6.10: Recompensa media por episodio - Experimento 10 - A2C R2

Experimento 11 - A2C R3 Orden Salida Estable

En la Figura 6.11 se muestran la recompensa media durante el entrenamiento para el experimento 11, para las tres repeticiones del mismo. A diferencia de DQN y QR-DQN, el entrenamiento no se frena y parece que la recompensa media por episodio seguiría subiendo con más instantes de entrenamiento.

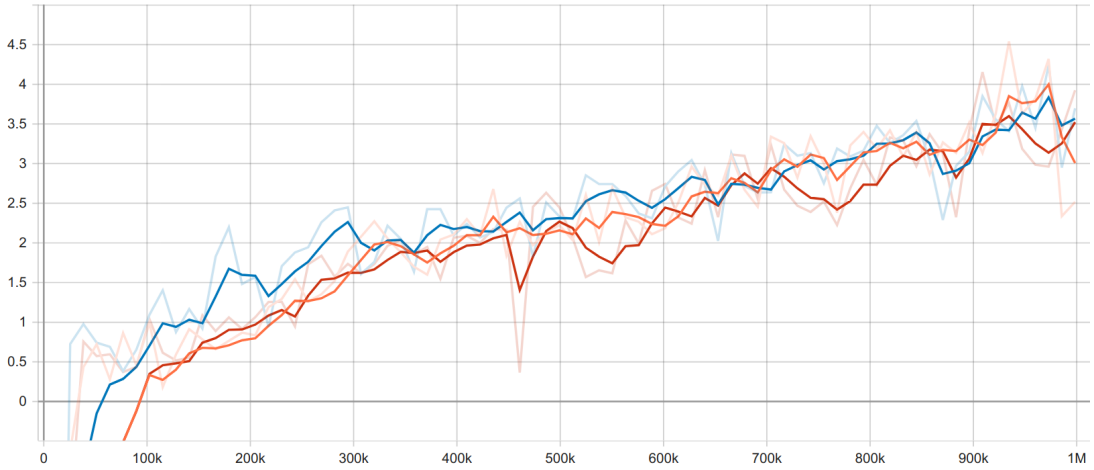


Figura 6.11: Recompensa media por episodio - Experimento 11 - A2C R3 - Entrenamiento con Orden de Salida Estable

Experimento 12 - A2C R3 Orden Salida Aleatorio

En la Figura 6.12 se muestran la recompensa media durante el entrenamiento para el experimento 12, para las tres repeticiones del mismo. En este caso, a pesar del orden de salida aleatorio y de la recompensa diferida, A2C es capaz de aprender de forma rápida, hasta una recompensa media por episodio de 8, lo cual equivale aproximadamente a que termina cada carrera de media en la 7ª posición (ver Tabla 5.2).

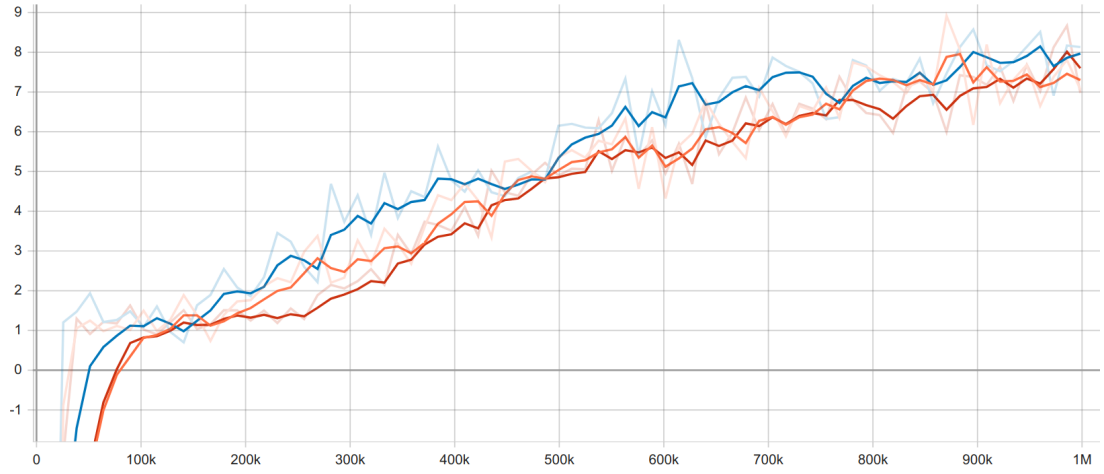


Figura 6.12: Recompensa media por episodio - Experimento 12 - A2C R3 - Entrenamiento con Orden de Salida Aleatorio

6.1.2. Comparación de Algoritmos

A continuación se van a mostrar las gráficas de entrenamiento por función de recompensa, mostrando la media de los tres entrenamientos de cada algoritmo. De esta manera se podrá comparar la velocidad, estabilidad y capacidad de aprendizaje de los algoritmos para cada función de recompensa.

R1

En la Figura 6.13 se muestra la recompensa media por episodio para cada algoritmo con la función de recompensa R1. Se puede ver que DQN y QR-DQN son menos estables debido a que las políticas que usan para generar los datos son ϵ-greedy. A2C es más estable y consigue la mayor recompensa media por episodio al final del entrenamiento, ganando 1 posición de media por carrera más que DQN y 2 más que QR-DQN.

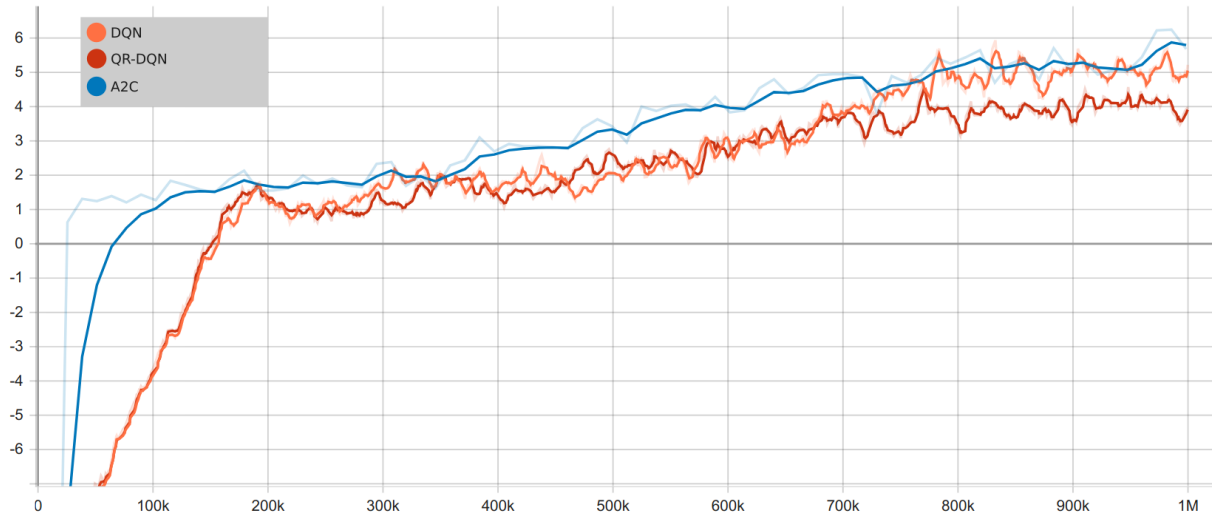


Figura 6.13: Comparación de la velocidad y estabilidad y capacidad de aprendizaje de los algoritmos para la función de recompensa R1.

R2

En la Figura 6.14 se muestra la recompensa media por episodio para cada algoritmo con la función de recompensa R2. Igual que para R1, DQN y QR-DQN son menos estables y aprenden más lento que A2C, pero DQN consigue alcanzar la recompensa media por episodio de A2C al final del entrenamiento. El rendimiento de QR-DQN sigue siendo el peor de los tres algoritmos.

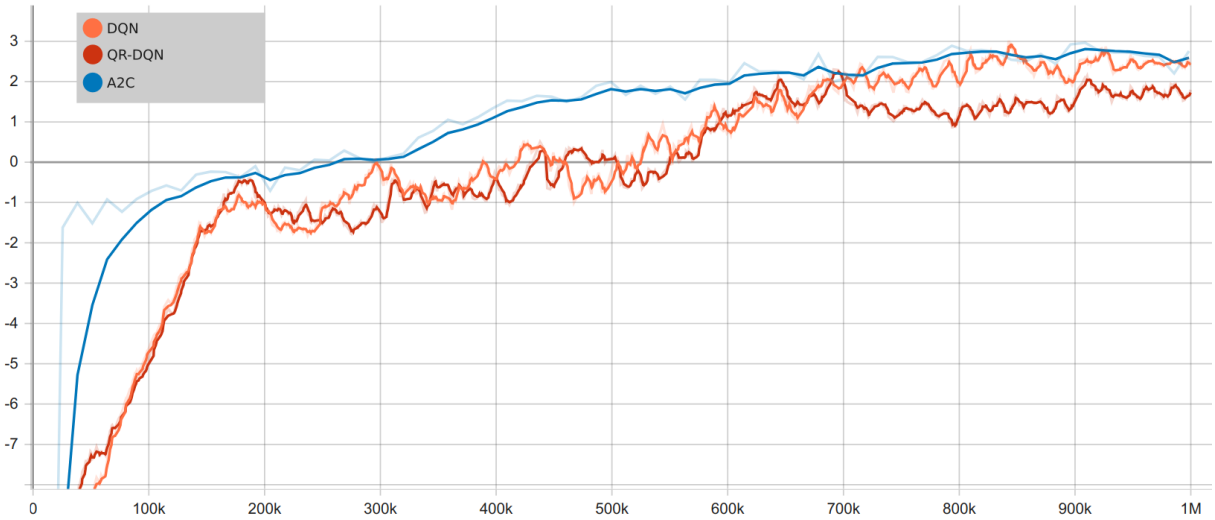


Figura 6.14: Comparación de la velocidad y estabilidad y capacidad del aprendizaje de los algoritmos para la función de recompensa R2.

R3 - Orden Salida Estable

En la Figura 6.15 se muestra la recompensa media por episodio para cada algoritmo con la función de recompensa R3, para la que se entrena con la parrilla en orden estable. A2C supera claramente a DQN y QR-DQN tanto en velocidad, estabilidad y capacidad de aprendizaje. Con esta función de recompensa crecen las diferencias entre los tres algoritmos.

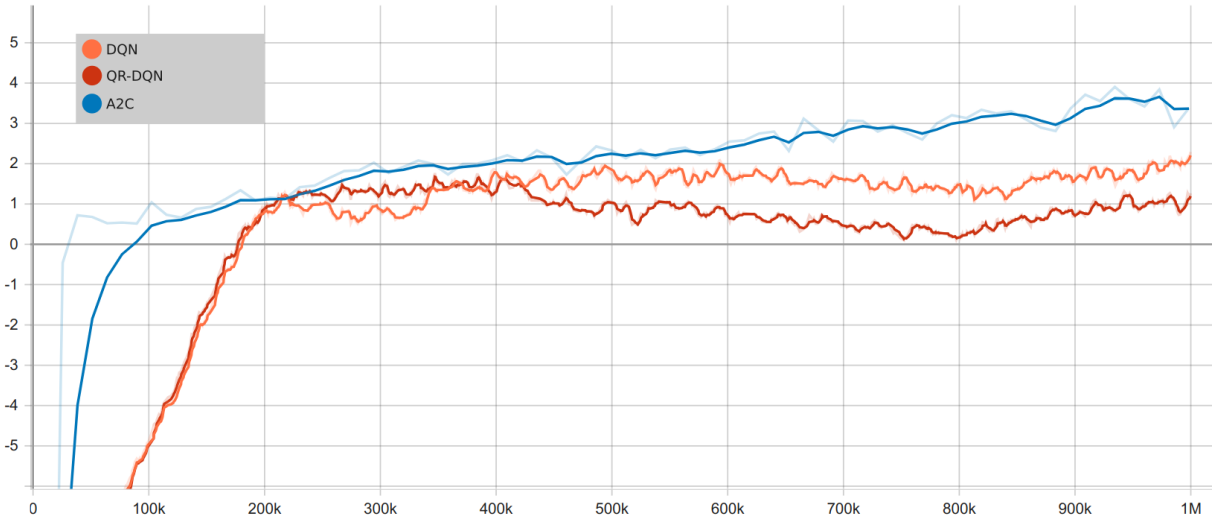


Figura 6.15: Comparación de la velocidad y estabilidad y capacidad del aprendizaje de los algoritmos para la función de recompensa R3, empezando los episodios con orden de salida de los pilotos estable.

R3 - Orden Salida Aleatorio

En la Figura 6.16 se muestra la recompensa media por episodio para cada algoritmo con la función de recompensa R3, para la que se entrena con la parrilla con orden de salida aleatorio. Este entrenamiento es más estocástico que los anteriores, y podemos ver cómo claramente DQN y QR-DQN no son capaces de aprender prácticamente nada. De hecho, QR-DQN llega a 'olvidarse' de lo que ha aprendido desde el instante de entrenamiento 500 000. Por el contrario, A2C aprende mucho más, más rápido y de forma más estable. Parece que si se hubiera entrenado durante más timesteps se podría haber mejorado más la recompensa media por episodio.

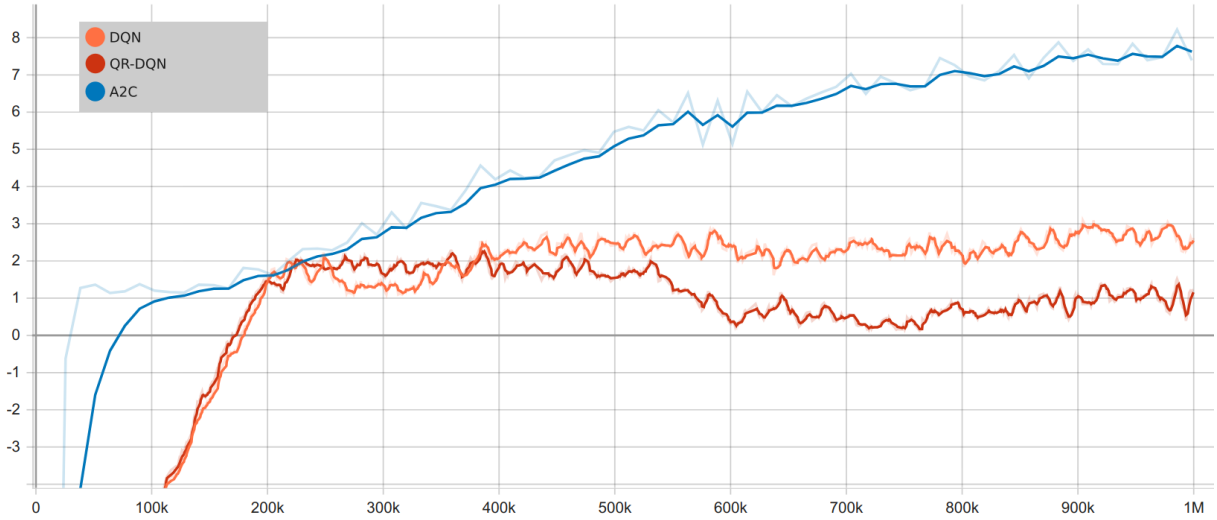


Figura 6.16: Comparación de la velocidad y estabilidad y capacidad del aprendizaje de los algoritmos para la función de recompensa R3, empezando los episodios con orden de salida de los pilotos aleatorio.

7: Evaluación

Para ver el desempeño real de los agentes entrenados en el Capítulo 6, en este capítulo se va a realizar una evaluación de ellos. Primero, se va a ver cuál es el desempeño si el piloto al que se controla siguiera una estrategia aleatoria o la misma estrategia predefinida que el resto de los pilotos. Después, se cogerá el mejor modelo de cada experimento realizado y se evaluará bajo las mismas condiciones, pudiendo comparar su rendimiento con el de los casos base y el del resto de los modelos entrenados.

Finalmente, se va a presentar un ejemplo de la toma de decisiones de un agente en una carrera, para comprender mejor el funcionamiento del mismo.

7.1. Evaluación y Comparación de Modelos

La **evaluación** consistirá en, para cada experimento, competir en **1000 carreras distintas**. Las carreras son distintas entre sí, con **ordenes de la parrilla de salida y estrategias aleatorias**. Evidentemente, cada caso a evaluar se enfrentará a las mismas 1000 carreras, con las mismas condiciones de salida (mismo orden de parrilla, neumático de salida y mismas estrategias del resto de pilotos que no se están controlando).

La **métrica de evaluación** será la **posición final media en carrera y su desviación estándar**, a lo largo de las 1000 carreras. Es decir, cuanto más bajo sea el valor de la métrica, mejor será el modelo evaluado.

Evaluación de los casos base

En la Tabla 7.1 se muestran los resultados de la evaluación para tres casos base: **1)** primero, el caso en el que se realizan entre 1 y 3 paradas en cualquier momento de la carrera. Después, **2)** el caso en el que se realizan entre 1 y 3 paradas que se realizan entre el primer y el segundo tercio de la carrera, es decir, hay conocimiento sobre el dominio y se sabe que parar muy al principio o muy al final no es la mejor opción. Finalmente, **3)** el caso en el que se siguiera alguna de las estrategias predefinidas, que son las que usan el resto de pilotos, y están basadas en las estrategias de años anteriores.

Cabe destacar que para los tres casos base se ha obligado a que el piloto controlado cumpliera con las restricciones descritas en la Sección 4.5, para asegurar que la evaluación fuera justa.

Entre 1 y 3 paradas parando en cualquier vuelta	Entre 1 y 3 paradas parando entre el primer tercio y el segundo tercio de la carrera	Estrategias predefinidas (igual al resto de pilotos)
16.219 ± 4.6416	13.559 ± 4.61459	10.488 ± 5.7120

Tabla 7.1: Posición final media y desviación estándar en la evaluación durante 1000 carreras para los casos base. En cada carrera de evaluación el orden de salida y el neumático de salida son aleatorios y para los los pilotos que no se están controlando las estrategias son predefinidas.

Evaluación de los agentes entrenados

En la Tabla 7.2 se muestran los resultados de la evaluación para los agentes entrenados en cada uno de los experimentos realizados, para cada algoritmo (ver Sección 5.3) y para cada función de recompensa (ver Sección 5.4) utilizados. Debido a las características de R1 y R2, se entrenó con la parrilla de salida en orden estable siempre, es decir, con el piloto que se controla saliendo siempre último. Para R3, se entrenó en un caso con el orden de salida aleatorio en cada carrera, y en otro con orden estable.

	Función de recompensa			
	R1	**R2**	**R3**	
	Orden de salida de los pilotos durante el entrenamiento			
Algoritmo	Estable, piloto controlado por el agente sale siempre último			Aleatorio
DQN	11,267 ± 5,8142	11,904 ± 4,4028	11,645 ± 5,6116	11,266 ± 4,8718
QR-DQN	10,459 ± 6,3136	10,836 ± 5,9980	11,939 ± 5,3223	13,091 ± 6,0941
A2C	**9,802 ± 6,8928**	**10,069 ± 7,1327**	**7,684 ± 5,5076**	**7,208 ± 5,2351**

Tabla 7.2: Posición final media y desviación estándar en la evaluación durante 1000 carreras para cada uno de los modelos obtenidos en los experimentos. En cada carrera de evaluación el orden de salida y el neumático de salida son aleatorios y para los los pilotos que no se están controlando las estrategias son aleatorias.

A la vista de los resultados obtenidos, podemos ver que los modelos entrenados con el algoritmo A2C superan con creces a los entrenados con los algoritmos DQN y QR-DQN, con independencia de la función de recompensa utilizada.

Viendo las funciones de recompensa, R2 tiene una peor posición final media que R1 para los tres algoritmos. R3 mejora a R2 en el caso de A2C, pero DQN y QR-DQN, como se ha visto en el capítulo anterior, no consiguen aprender con esa función de recompensa.

En cuanto a los algoritmos, viendo primero DQN, sólo es capaz de superar el rendimiento de los dos casos de toma de decisiones de forma aleatoria, pero no alcanza el rendimiento del agente que utiliza estrategias basadas en carreras de otros años.

La posición final media de QR-DQN es mejor que la de DQN en las funciones de recompensa R1 y R2 (0.8 y 1.1 posiciones mejor, respectivamente), pero con mucha más desviación típica. Sin embargo, al llegar a R3, en particular con el caso en el que se entrena con parrilla de salida en orden aleatorio, su rendimiento cae. Solo consigue superar ligeramente el rendimiento del agente que utiliza estrategias basadas en carreras de otros años con R1, por tan solo 0.03 posiciones.

Por último, se puede ver que **A2C mejora de forma clara para todas las funciones de recompensa al agente que utiliza estrategias basadas en carreras de otros años**. En particular, su rendimiento es muy bueno con la función de recompensa R3, y el mejor de todos los modelos se obtiene cuando se le entrena en las mismas condiciones de evaluación (esto es, con orden de salida aleatorio).

Al ser un entorno estocástico, tiene sentido que **A2C tenga un mejor rendimiento que DQN y QR-DQN**, ya que los *policy gradient methods* son los propicios para este tipo de entornos, y los resultados lo confirman.

7.2. Muestra del rendimiento del mejor modelo

7.2.1. Simulación de ejemplo

A continuación se va a mostrar el rendimiento y las decisiones tomadas por el mejor modelo entrenado (el correspondiente al algoritmo A2C con la función de recompensa R3, entrenado con el orden de salida aleatorio) para un ejemplo de una carrera.

En la simulación de ejemplo, nuestro **agente sale en la 15ª posición** con neumáticos medios y es capaz de **remontar hasta la 7ª posición**. Para ello para en la vuelta 33, momento en el que está en la 12ª posición. Gracias a la diferencia de elección de neumático y al momento de la parada consigue finalizar en 7ª posición, aunque a falta de menos de 10 vueltas para el final estaba en la 5ª. En la Figura 7.1 se muestra el cambio de posiciones en cada vuelta durante la simulación. Cada piloto se representa por un identificador del 0 al 18, y el piloto que controla el agente está denotado por la etiqueta 'Agente', con el color naranja.

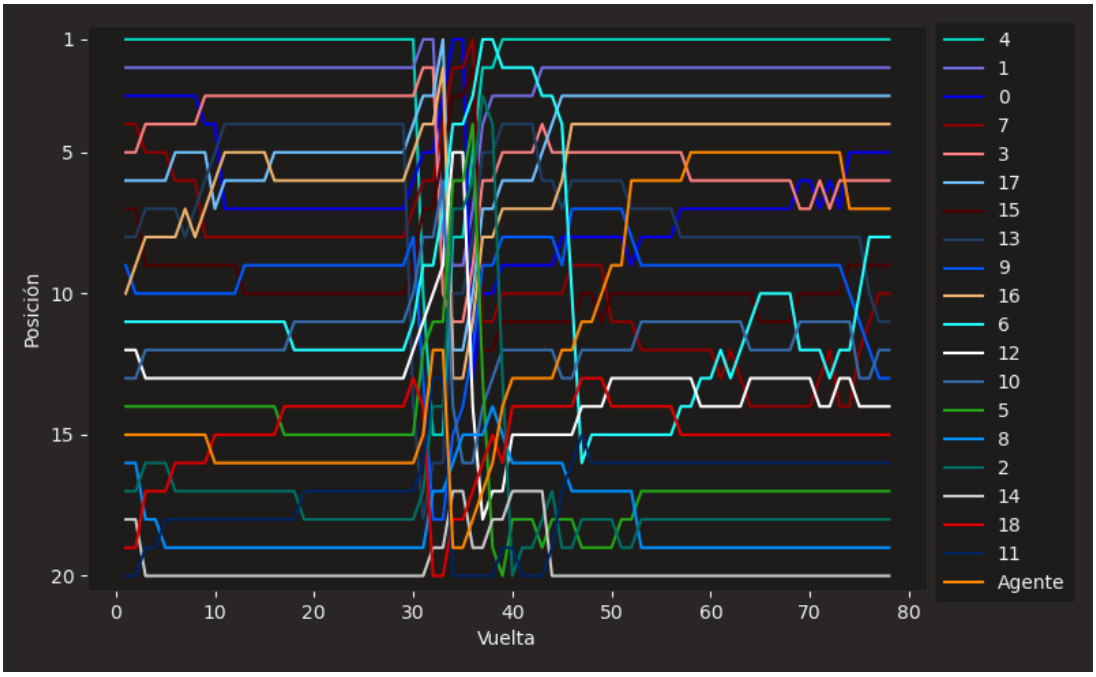


Figura 7.1: Cambio de posiciones durante la simulación de carrera.

Las estrategias seguidas por todos los pilotos se pueden ver en la Figura 7.2. El agente decide que el piloto que está controlando pare bastante pronto, en la vuelta 33, y pone el neumático blando. Es una parada pronto porque lleva el neumático medio, y deja más de la mitad de la carrera para un neumático que degrada más. Sin embargo, adelantando la parada consigue ganar la posición al piloto 6. Mientras que el piloto 6 sigue en pista con un neumático medio viejo, el piloto que controla el agente va más rápido con un blando nuevo, y cuando el piloto 6 hace su parada, sale detrás del piloto controlado por el agente.

Debido a la diferencia en la elección del tipo de neumático, el piloto controlado por el agente también consigue adelantar en pista a los pilotos 0 y 3, cuyo neumático para el segundo stint es el duro. Sin embargo, al final de la carrera la degradación del neumático blando es notoria y en las últimas vueltas el agente pierde la posición con ellos dos, finalizando la carrera en la 7ª posición.

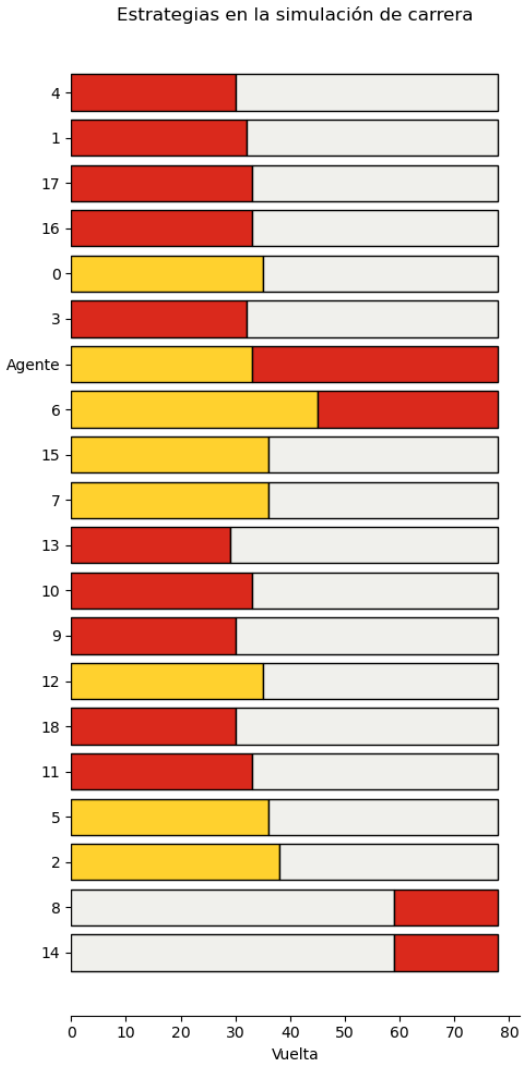


Figura 7.2: Estrategias de los pilotos en la simulación de carrera.

7.2.2. Política del agente

Finalmente, se va a mostrar cual es la política del agente en determinadas situaciones (esto es, probabilidad de tomar cada una de las acciones (ver Sección 4.3) según el momento de la carrera).

Al comenzar la carrera

La primera situación es cuando han pasado las 10 primeras vueltas. En la Tabla 7.3 se muestra la probabilidad de tomar cada acción cuando han pasado las primeras 10 vueltas. Claramente, la acción preferida del agente es no parar, lo cual tiene sentido dado que es demasiado pronto y los neumáticos no durarían el resto de la carrera con un rendimiento aceptable.

Acción	0	1	2	3
Probabilidad	0.9998	0.0002	0	0

Tabla 7.3: Política del agente en la simulación de ejemplo en la vuelta 10.

Antes de la parada

Justo cuando se va a tomar la decisión de parar, la política del agente es la que se muestra en la Tabla 7.4. La acción con más probabilidad es la 1, es decir, poner el neumático blando. La siguiente es continuar otra vuelta sin parar, cubriendo ahí el 95 % de la probabilidad. Con una probabilidad mucho más baja está poner otro neumático medio (acción 2), lo cual obligaría al piloto a realizar otra parada para cumplir con la normativa. Poner el neumático duro es la acción menos preferida, seguramente debido a que al ser un neumático más lento dificultaría la ganancia de posiciones del piloto.

Acción	0	1	2	3
Probabilidad	0.2186	0.7369	0.0328	0.0117

Tabla 7.4: Política del agente en la simulación de ejemplo en la vuelta que decide parar (vuelta 33).

A falta de 10 vueltas para el final

Por último, en la Tabla 7.5 se muestra la política del agente cuando quedan 10 vueltas para el final. De nuevo, la acción con la probabilidad mayoritaria es no parar, al igual que al principio de la carrera. Parar al final de la carrera provocaría que el piloto fuera mucho más rápido que el resto de pilotos, pero no tendría tiempo de recuperar las posiciones perdidas en la parada. En el resto de las acciones, las que más probabilidad tienen son, por orden, 2, 1 y 3 (es decir, por orden preferiría poner neumático medio, blando y duro).

Acción	0	1	2	3
Probabilidad	0.9848	0.0013	0.0137	0.0002

Tabla 7.5: Política del agente en la simulación de ejemplo cuando faltan 10 vueltas para el final (vuelta 68).

8: Conclusiones y Líneas de trabajo futuras

En este capítulo se presentan las conclusiones obtenidas una vez completado el proyecto, así como las líneas de trabajo futuro.

8.1. Conclusiones

Primero, se va a analizar si se han cumplido o no los objetivos planteados en la Sección 1.2:

- (OBJ-1) En el Capítulo 3 se explica el desarrollo del simulador y su utilización como entorno para que los agentes interactúen con él durante el aprendizaje basado por refuerzo. Tras esto, se diseñaron los experimentos a realizar, detallando los algoritmos y las funciones de recompensa que se iban a utilizar en el Capítulo 4. Tras esto, se presentan los resultados de la ejecución de los experimentos en el Capítulo 5.
- (OBJ-2) En el Capítulo 6 se han evaluado y comparado los agentes entrenados según algoritmo y función de recompensa, seleccionando el que mejores resultados ha obtenido.

De acuerdo a esta información, se puede afirmar que se han cumplido los dos objetivos planteados al comienzo del proyecto, OBJ-1 y OBJ-2, de forma satisfactoria.

Personalmente, este proyecto me ha permitido aumentar mucho mi conocimiento sobre el Aprendizaje por Refuerzo, desarrollando un sistema completo basado en este paradigma del machine learning. Se ha desarrollado un simulador de carreras de Fórmula 1 para usarlo como entorno para entrenar a los agentes, que es extensible para poder añadirle complejidad en el futuro, de forma que sea más realista y ser usado para que los agentes puedan aprender comportamientos más complejos.

8.2. Trabajo Futuro

A continuación se plantean las líneas de trabajo futuro que no se han podido abordar en este proyecto, pero que son el camino a seguir para conseguir más y mejores resultados.

Mejoras en el Simulador

Se plantean varias mejoras a realizar en el simulador para hacerlo más realista y también para acelerar el aprendizaje de los agentes:

- Añadir nuevas situaciones de carrera: añadir la salida aleatoria de un Safety Car o un Virtual Safety Car, situaciones en las que las decisiones a tomar son más críticas y hay menos tiempo para que los ingenieros de estrategia puedan reaccionar, por lo que se podrían beneficiar más del sistema desarrollado.
- Mejorar el modelo de adelantamientos: en lugar de una probabilidad fija de adelantamiento en cualquier zona del circuito, que la probabilidad de adelantamiento sea en base a la zona del circuito en la que se encuentran los pilotos y la diferencia en el tiempo de vuelta potencial entre ellos.
- Generalizar a varios circuitos: Conviene entrenar al agente en circuitos con diferentes características, diferente longitud, diferente tiempo de parada en boxes, diferente rendimiento de neumáticos, etcétera para que pueda llegar a utilizarse en una situación real.
- Optimización del simulador: Para que los entrenamientos sean más rápidos y que el cuello de botella no sea el tiempo necesario para simular las carreras, convendría utilizar un compilador de código Python como Numba[18], que permite acelerar la ejecución del código al compilar partes del mismo en código máquina.

Mejoras en el aprendizaje de los agentes

- Como se ha visto en los resultados, el mejor método ha sido un *policy gradient method*, en concreto A2C. El siguiente paso es probar más algoritmos de este tipo, como PPO[21].
- Buscar los valores de los hiperparámetros de los algoritmos y la arquitectura más adecuada de la red neuronal que haga que el agente pueda aprender lo mejor posible para el problema de la toma de decisiones estratégicas en F1.
- En lugar de entrenar al agente contra pilotos que utilizan estrategias predefinidas, lo cual limita al agente a competir contra el conocimiento humano, se le puede entrenar contra pilotos cuya estrategia es controlada por versiones antiguas de si mismo. Esto es lo que consiguió DeepMind con AlphaGo Zero [9] [22].

Evaluación en situaciones de carrera reales

El paso del prototipo construido en este proyecto a un sistema que pueda utilizarse en la vida real implica utilizar carreras reales y comparar las decisiones que han tomado los ingenieros reales con las que habría tomado el mejor agente entrenado en este trabajo.

Mejoras Alternativas

En este trabajo se ha supuesto que el único objetivo es maximizar el resultado del piloto que está controlando el agente, pero la Fórmula 1 es un deporte de equipo, en el que cada equipo cuenta con dos coches.

Los objetivos de un equipo de Fórmula 1 pueden ser múltiples: maximizar el resultado de los dos pilotos, maximizar el resultado para el equipo, maximizar la diferencia de posiciones entre un piloto y su rival, etcétera. Para ello, se puede entrenar un agente que tenga múltiples objetivos, lo que se conoce como Aprendizaje por Refuerzo Multiobjetivo (Multi-Objective Reinforcement Learning, MORL) [12].

Bibliografía

[1] Beck, Beedle, Bennekum, Cockburn y 13 autores más. Manifesto for agile software development. https://agilemanifesto.org/. Último acceso el 05/02/2023.

[2] Bellemare, M. G., Dabney, W., and Munos, R. A Distributional Perspective on Reinforcement Learning. https://doi.org/10.48550/arXiv.1707.06887, 2017.

[3] Brunskill, E. Stanford CS234: Reinforcement Learning | Winter 2019. url: https://www.youtube.com/playlist?list=PLoROMvodv4rOSOPzutgyCTapiGlY2Nd8u, 2019.

[4] Cámara Oficial de Comercio, Industria y Servicios de Palencia. PÍO XII Coworking Center. https://cocipa.es/zonas-compartidas/. Último acceso el 22/06/2024.

[5] Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. Distributional reinforcement learning with quantile regression. https://doi.org/10.48550/arXiv.1710.10044, 2017.

[6] de Kleut., A. V. Actor-Critic Methods, Advantage Actor-Critic (A2C) and Generalized Advantage Estimation (GAE). https://avandekleut.github.io/a2c/, 2021.

[7] FastF1. FastF1 3.3.8. https://docs.fastf1.dev/. Último acceso el 16/06/2024.

[8] Federation Internationale de l'Automobile. FIA 2024 Formula 1 Sporting Regulations - Issue 6 V2. https://www.fia.com/sites/default/files/fia_2024_formula_1_sporting_regulations_-_issue_6_-_2024-04-30_v2.pdf. Último acceso el 15/05/2024.

[9] Google DeepMind. AlphaGo Zero: Starting from scratch. https://deepmind.google/discover/blog/alphago-zero-starting-from-scratch/. Último acceso el 22/06/2024.

[10] Gymnasium. An API standard for reinforcement learning with a diverse collection of reference environments. https://gymnasium.farama.org/. Último acceso el 27/04/2024.

[11] Gymnasium. Env - Gymnasium Documentation. https://gymnasium.farama.org/api/env/. Último acceso el 10/06/2024.

[12] Hayes, C. F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L. M., Dazeley, R., Heintz, F., Howley, E., Irissappane, A. A., Mannion, P., Nowé, A., Ramos, G., Restelli, M., Vamplew, P., and Roijers, D. M. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems 36*, 1 (Apr. 2022).

[13] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. https://doi.org/10.48550/arXiv.1709.06560, 2019.

[14] HuggingFace. Advantage actor critic (a2c). https://huggingface.co/blog/deep-rl-a2c. Último acceso el 22/06/2024.

[15] Martin, R. *Clean Agile*. Robert C. Martin Series. Pearson Education, 2019.

[16] Mike Law. An Introduction to FSAE Vehicle Dynamics - Mike Law at the University of Surrey - 06/12/2022. https://www.youtube.com/watch?v=1bihTZEiyjU&ab_channel=MikeLaw. Último acceso el 25/04/2024.

[17] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari with Deep Reinforcement Learning. https://doi.org/10.48550/arXiv.1312.5602, 2013.

[18] Numba. User Manual. https://numba.readthedocs.io/en/stable/user/index.html. Último acceso el 23/06/2024.

[19] Object Management Group (OMG). Unified Modeling Language, v2.5.1. https://www.omg.org/spec/UML/2.5.1/PDF. Último acceso el 14/06/2024.

[20] of the VITALab (Videos & Images Theory, W., and of Sherbrooke University., A. L. Distributional Reinforcement Learning with Quantile Regression. https://vitalab.github.io/article/2021/03/26/QR_DQN.html, 2021.

[21] OpenAI. Proximal Policy Optimization — Spinning Up documentation. https://spinningup.openai.com/en/latest/algorithms/ppo.html. Último acceso el 22/06/2024.

[22] Plaat, A. *Deep Reinforcement Learning*, 1 ed. Computer Science, Computer Science (R0). Springer Singapore, Singapore, 2022. The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Singapore Pte Ltd. 2022.

[23] Quipu Blog. ¿Cuánto cuesta contratar a un trabajador? https://getquipu.com/blog/cuanto-cuesta-contratar-un-trabajador/. Último acceso el 22/06/2024.

[24] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research 22*, 268 (2021), 1–8.

[25] Richard S. Sutton, Andrew G. Barto. *Reinforcement Learning: An Introduction*, 2 ed. MIT Press, Cambridge, MA, 2018, 2020. http://incompleteideas.net/book/RLbook2020.pdf.

[26] Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., and Wayne, G. Experience Replay for Continual Learning. https://arxiv.org/abs/1811.11682, 2019.

[27] Russell, S. J., and Norvig, P. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

[28] Schwaber, Sutherland. The scrum guide - the definitive guide to scrum: The rules of the game. https://scrumguides.org/scrum-guide. Último acceso el 05/02/2023.

[29] Silver, D. Lectures on Reinforcement Learning. url: https://www.davidsilver.uk/teaching/, 2015.

[30] Stable Baselines3. A2C. https://stable-baselines3.readthedocs.io/en/master/modules/a2c.html. Último acceso el 07/06/2024.

[31] Stable Baselines3. DQN. https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html. Último acceso el 07/06/2024.

[32] Stable Baselines3. EvalCallback. https://stable-baselines3.readthedocs.io/en/master/guide/callbacks.html#evalcallback. Último acceso el 12/06/2024.

[33] Stable Baselines3. QR-DQN. https://sb3-contrib.readthedocs.io/en/master/modules/qrdqn.html. Último acceso el 07/06/2024.

[34] talent.com. Salario medio para ingeniero de software en españa, 2024. https://es.talent.com/salary?job=ingeniero+de+software. Último acceso el 22/06/2024.

[35] TensorBoard | TensorFlow. A suite of visualization tools to understand, debug, and optimize TensorFlow programs for ML experimentation. https://www.tensorflow.org/tensorboard. Último acceso el 12/06/2024.

[36] University of Alberta. Reinforcement Learning. https://www.ualberta.ca/admissions-programs/online-courses/reinforcement-learning. Último acceso el 22/06/2024.

Apéndices

Apéndice A

Ejemplo de Simulación de Carrera

Simulación de carrera

Simulación de carrera para ver el funcionamiento del simulador.

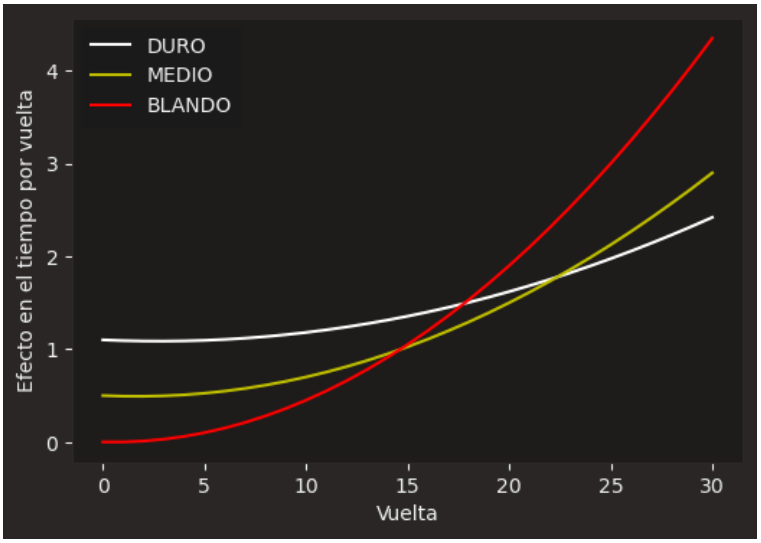
Habrá 5 pilotos en este ejemplo, carrera a 30 vueltas. La entrada al pit lane está en el porcentaje 0.98 de la vuelta, la salida en el porcentaje 0.02. El invertido en la parada en boxes será de 24 segundos.

La probabilidad de adelantamiento sera 0.15.

El piloto que vamos a controlar será el piloto con id = 4

Neumáticos

Se muestra como va a ser el efecto de los neumáticos para esta simulación de ejemplo



Estrategias

A continuación se muestran las estrategias de cada piloto que participa en la carrera:

```
Estrategia del piloto 3
        Neumático de salida:  Blando
        PitStops:
                Vuelta:  11 ___ Neumático: Duro


Estrategia del piloto 4
        Neumático de salida:  Medio


Estrategia del piloto 0
        Neumático de salida:  Blando
        PitStops:
                Vuelta:  11 ___ Neumático: Duro


Estrategia del piloto 1
        Neumático de salida:  Medio
        PitStops:
                Vuelta:  22 ___ Neumático: Blando


Estrategia del piloto 2
        Neumático de salida:  Medio
        PitStops:
                Vuelta:  22 ___ Neumático: Blando
```

Estado Inicial t = 0

Estado Inicial, momento en el que el piloto que estamos controlando (id = 4) llega a la entrada del pit lane, cuando tiene que decidir si entrar o no. Aparecen algunas variables más que no forman parte de la observación del agente del estado, pero ayudan a comprender la simulación (**percentage**, ubicación del piloto en el circuito, **gap**, distancia en segundos al líder).

El compuesto 0 es el neumático blando, el compuesto 1 es el neumático medio y el compuesto 2 es el neumático duro.

```
display(race_simulation.to_df())
```

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.993	0.000	30	1	1	70.5769	0	0	False	0
1	4	0.890	0.980	0.890	30	1	2	71.0769	1	0	False	0
2	0	0.296	0.976	1.186	30	1	3	70.5769	0	0	False	0
3	1	0.501	0.969	1.686	30	1	4	71.0769	1	0	False	0
4	2	0.400	0.963	2.086	30	1	5	71.0769	1	0	False	0

Estados Intermedios 1 <= t <= 29

```
race_simulation.simulate_timestep(0)
display(race_simulation.to_df())
```

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	1.000	0.000	29	70.57690	1	70.55767	0	1	False	0
1	4	1.383	0.980	1.383	29	71.47682	2	71.05067	1	1	False	0
2	0	0.274	0.976	1.657	29	71.76400	3	70.55767	0	1	False	0
3	1	0.522	0.969	2.179	29	72.27653	4	71.05067	1	1	False	0
4	2	0.400	0.963	2.579	29	72.67639	5	71.05067	1	1	False	0

Cálculo del tiempo de vuelta potencial

En t = 2, el tiempo de vuelta potencial para el piloto con id 3 será:

```
driver_4 = race_simulation.get_driver_by_id(4)
print("Información del combustible:", driver_4.fuel.__dict__)
f = driver_4.fuel
efecto_combustible = (f.fuel_mass-((f.consumed_laps+1)*f.consumption_per_lap))*f.mass_effect
print("Efecto del combustible:", efecto_combustible)
print("Información del neumático:", driver_4.tyre.__dict__)
n = driver_4.tyre
efecto_neumatico = round(n.base_lap_time+n.squared_deg*(n.used_laps+1)**2+n.linear_deg*(n.used_la
print("Efecto del neumático:", efecto_neumatico)
plt = efecto_combustible+efecto_neumatico
print("Tiempo de vuelta potencial para el piloto con id=4 en la siguiente vuelta:", plt)
```

```
Información del combustible: {'fuel_mass': 38.46, 'mass_effect': 0.015, 'consumption_per_lap': 1.
282, 'consumed_laps': 1}
Efecto del combustible: 0.53844
Información del neumático: {'compound': 1, 'base_lap_time': 70.5, 'squared_deg': 0.003, 'linear_d
eg': -0.01, 'used_laps': 1}
Efecto del neumático: 70.492
Tiempo de vuelta potencial para el piloto con id=4 en la siguiente vuelta: 71.03044
```

```
race_simulation.simulate_timestep(0)
display(race_simulation.to_df())
```

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.006	0.000	27	70.54844	1	70.54921	0	3	False	0
1	4	1.865	0.980	1.865	28	71.05061	2	71.03044	1	2	False	0
2	0	0.280	0.976	2.146	28	71.02900	3	70.54844	0	2	False	0
3	1	0.516	0.969	2.661	28	71.05038	4	71.03044	1	2	False	0
4	2	0.400	0.963	3.061	28	71.05027	5	71.03044	1	2	False	0

Como se puede ver, el tiempo de vuelta potencial del piloto con id=4 coincide con el calculado antes. Se sigue simulando hasta que a nuestro piloto (piloto con id=4) le queden 10 vueltas para el final

```
while driver_4.laps_to_go > 10:
    race_simulation.simulate_timestep(0)
    display(race_simulation.to_df())
```

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.013	0.000	26	70.54921	1	70.55998	0	4	False	0
1	4	2.332	0.980	2.332	27	71.03040	2	71.01621	1	3	False	0
2	0	0.287	0.976	2.620	27	71.03724	3	70.54921	0	3	False	0
3	1	0.509	0.969	3.128	27	71.03024	4	71.01621	1	3	False	0
4	2	0.400	0.963	3.528	27	71.03016	5	71.01621	1	3	False	0

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.019	0.000	25	70.55998	1	70.58075	0	5	False	0
1	4	2.781	0.980	2.781	26	71.01618	2	71.00798	1	4	False	0
2	0	0.294	0.976	3.075	26	71.02332	3	70.55998	0	4	False	0
3	1	0.502	0.969	3.577	26	71.01609	4	71.00798	1	4	False	0
4	2	0.400	0.963	3.977	26	71.01605	5	71.00798	1	4	False	0

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.025	0.000	24	70.58075	1	70.61152	0	6	False	0
1	4	3.206	0.980	3.206	25	71.00797	2	71.00575	1	5	False	0
2	0	0.302	0.976	3.508	25	71.01551	3	70.58075	0	5	False	0
3	1	0.494	0.969	4.003	25	71.00795	4	71.00575	1	5	False	0
4	2	0.400	0.963	4.402	25	71.00794	5	71.00575	1	5	False	0

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.031	0.000	23	70.61152	1	70.65229	0	7	False	0
1	0	3.507	0.981	3.507	24	71.01061	2	70.61152	0	6	False	0
2	4	0.098	0.980	3.605	24	71.00576	3	71.00952	1	6	False	0
3	1	0.799	0.969	4.404	24	71.00580	4	71.00952	1	6	False	0
4	2	0.400	0.963	4.804	24	71.00582	5	71.00952	1	6	False	0

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.036	0.000	22	70.65229	1	70.70306	0	8	False	0
1	0	3.507	0.987	3.507	23	70.61136	2	70.65229	0	7	False	0
2	4	0.466	0.980	3.972	23	71.00954	3	71.01929	1	7	False	0
3	1	0.799	0.969	4.772	23	71.00960	4	71.01929	1	7	False	0
4	2	0.400	0.963	5.171	23	71.00964	5	71.01929	1	7	False	0

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	0	0.000	0.991	0.000	22	70.65229	1	70.70306	0	8	False	0
1	3	0.291	0.987	0.291	22	70.65229	2	71.52306	2	0	True	1
2	4	4.305	0.980	4.596	22	71.01931	3	71.03506	1	8	False	0
3	1	0.799	0.969	5.396	22	71.01937	4	71.03506	1	8	False	0
4	2	0.400	0.963	5.795	22	71.01941	5	71.03506	1	8	False	0

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	0	0.000	0.995	0.000	21	70.70299	1	70.76383	0	9	False	0
1	4	1.092	0.980	1.092	21	71.03508	2	71.05683	1	9	False	0
2	1	0.799	0.969	1.892	21	71.03513	3	71.05683	1	9	False	0
3	2	0.400	0.963	2.291	21	71.03515	4	71.05683	1	9	False	0
4	3	15.975	0.738	18.267	21	81.28900	5	71.50383	2	0	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	0	0.000	0.982	0.000	20	70.76383	1	71.4846	2	0	True	1
1	4	1.343	0.980	1.343	20	71.05685	2	71.0846	1	10	False	0
2	1	0.799	0.969	2.142	20	71.05691	3	71.0846	1	10	False	0
3	2	0.400	0.963	2.542	20	71.05693	4	71.0846	1	10	False	0
4	3	16.379	0.733	18.921	20	82.07322	5	71.4764	2	1	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	4	0.000	0.980	0.000	19	71.08451	1	71.11837	1	11	False	0
1	1	0.799	0.969	0.799	19	71.08468	2	71.11837	1	11	False	0
2	2	0.400	0.963	1.199	19	71.08471	3	71.11837	1	11	False	0
3	3	16.726	0.728	17.925	19	71.47606	4	71.45257	2	2	True	1
4	0	2.133	0.698	20.058	19	81.41791	5	71.46537	2	0	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	4	0.000	0.980	0.000	18	71.11837	1	71.15814	1	12	False	0
1	1	0.799	0.969	0.799	18	71.11845	2	71.15814	1	12	False	0
2	2	0.400	0.963	1.199	18	71.11848	3	71.15814	1	12	False	0
3	3	17.016	0.724	18.215	18	71.45244	4	71.43234	2	3	True	1
4	0	2.140	0.694	20.355	18	82.03553	5	71.43794	2	1	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	4	0.000	0.980	0.000	17	71.15814	1	71.20391	1	13	False	0
1	1	0.799	0.969	0.799	17	71.15822	2	71.20391	1	13	False	0
2	2	0.400	0.963	1.199	17	71.15825	3	71.20391	1	13	False	0
3	3	17.244	0.721	18.443	17	71.43223	4	71.41571	2	4	True	1
4	0	2.140	0.691	20.583	17	71.43748	5	71.41411	2	2	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	4	0.000	0.980	0.000	16	71.20391	1	71.25568	1	14	False	0
1	1	0.799	0.969	0.799	16	71.20399	2	71.25568	1	14	False	0
2	2	0.400	0.963	1.199	16	71.20402	3	71.25568	1	14	False	0
3	3	17.409	0.719	18.608	16	71.41562	4	71.40268	2	5	True	1
4	0	2.133	0.689	20.741	16	71.41372	5	71.39388	2	3	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	4	0.000	0.980	0.000	15	71.25568	1	71.31345	1	15	False	0
1	1	0.799	0.969	0.799	15	71.25576	2	71.31345	1	15	False	0
2	2	0.400	0.963	1.199	15	71.25579	3	71.31345	1	15	False	0
3	3	17.507	0.718	18.706	15	71.40262	4	71.39325	2	6	True	1
4	0	2.119	0.688	20.825	15	71.39356	5	71.37725	2	4	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	4	0.000	0.980	0.000	14	71.31345	1	71.37722	1	16	False	0
1	1	0.799	0.969	0.799	14	71.31353	2	71.37722	1	16	False	0
2	2	0.400	0.963	1.199	14	71.31356	3	71.37722	1	16	False	0
3	3	17.537	0.718	18.736	14	71.39321	4	71.38742	2	7	True	1
4	0	2.098	0.688	20.834	14	71.37700	5	71.36422	2	5	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	4	0.000	0.980	0.000	13	71.37722	1	71.44699	1	17	False	0
1	1	0.799	0.969	0.799	13	71.37730	2	71.44699	1	17	False	0
2	2	0.400	0.963	1.199	13	71.37733	3	71.44699	1	17	False	0
3	3	17.496	0.718	18.694	13	71.38741	4	71.38519	2	8	True	1
4	0	2.070	0.689	20.764	13	71.36404	5	71.35479	2	6	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	4	0.000	0.980	0.000	12	71.44699	1	71.52276	1	18	False	0
1	1	0.799	0.969	0.799	12	71.44707	2	71.52276	1	18	False	0
2	2	0.400	0.963	1.199	12	71.44710	3	71.52276	1	18	False	0
3	3	17.380	0.720	18.579	12	71.38520	4	71.38656	2	9	True	1
4	0	2.034	0.692	20.613	12	71.35468	5	71.34896	2	7	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	4	0.000	0.980	0.000	11	71.52276	1	71.60453	1	19	False	0
1	1	0.799	0.969	0.799	11	71.52284	2	71.60453	1	19	False	0
2	2	0.400	0.963	1.199	11	71.52287	3	71.60453	1	19	False	0
3	3	17.188	0.723	18.387	11	71.38658	4	71.39153	2	10	True	1
4	0	1.992	0.695	20.378	11	71.34892	5	71.34673	2	8	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	4	0.000	0.980	0.000	10	71.60453	1	71.6923	1	20	False	0
1	1	0.799	0.969	0.799	10	71.60461	2	71.6923	1	20	False	0
2	2	0.400	0.963	1.199	10	71.60464	3	71.6923	1	20	False	0
3	3	16.916	0.727	18.115	10	71.39155	4	71.4001	2	11	True	1
4	0	1.941	0.700	20.057	10	71.34676	5	71.3481	2	9	True	1

Parada

Paramos en boxes para un nuevo neumático blando para las 10 vueltas de carrera restantes

```
race_simulation.simulate_timestep(1)
```

Simulamos hasta el momento t = 29, esto es, hasta que a nuestro piloto le quede 1 vuelta para el final, momento en el que no hay que tomar más decisiones.

```
while driver_4.laps_to_go > 1:
    display(race_simulation.to_df())
    race_simulation.simulate_timestep(0)
```

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	1	0.000	0.011	0.000	8	82.35035	1	70.15384	0	0	True	1
1	2	0.400	0.011	0.400	8	82.34863	2	70.15384	0	0	True	1
2	3	2.314	0.007	2.713	8	71.41221	3	71.42804	2	13	True	1
3	0	1.868	0.981	4.581	9	71.34814	4	71.35307	2	10	True	1
4	4	0.045	0.980	4.626	9	82.25845	5	70.17307	0	0	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.009	0.000	7	71.42804	1	71.44741	2	14	True	1
1	0	1.801	0.984	1.801	8	71.35313	2	71.36164	2	11	True	1
2	4	0.270	0.980	2.071	8	80.86795	3	70.15384	0	1	True	1
3	1	0.979	0.966	3.050	8	82.35035	4	70.15384	0	0	True	1
4	2	0.400	0.960	3.449	8	82.34863	5	70.15384	0	0	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.008	0.000	6	71.44741	1	71.47038	2	15	True	1
1	0	1.728	0.984	1.728	7	71.36153	2	71.37381	2	12	True	1
2	4	0.261	0.980	1.989	7	71.48782	3	70.14461	0	2	True	1
3	2	0.148	0.978	2.137	7	80.75039	4	70.13461	0	1	True	1
4	1	0.134	0.976	2.271	7	80.75842	5	70.13461	0	1	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.007	0.000	5	71.47038	1	71.49695	2	16	True	1
1	0	1.647	0.984	1.647	6	71.37399	2	71.38958	2	13	True	1
2	4	0.272	0.980	1.919	6	71.38556	3	70.14538	0	3	True	1
3	2	0.291	0.976	2.210	6	70.16098	4	70.12538	0	2	True	1
4	1	0.291	0.972	2.501	6	70.69249	5	70.12538	0	2	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.006	0.000	4	71.49695	1	71.52712	2	17	True	1
1	0	1.559	0.984	1.559	5	71.38954	2	71.40895	2	14	True	1
2	4	0.283	0.980	1.842	5	71.39461	3	70.15615	0	4	True	1
3	2	0.300	0.976	2.142	5	71.53778	4	70.12615	0	3	True	1
4	1	0.301	0.971	2.443	5	71.69490	5	70.12615	0	3	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.005	0.000	3	71.52712	1	71.56089	2	18	True	1
1	0	1.464	0.984	1.464	4	71.40897	2	71.43192	2	15	True	1
2	4	0.294	0.980	1.758	4	71.41949	3	70.17692	0	5	True	1
3	2	0.309	0.976	2.068	4	71.42896	4	70.13692	0	4	True	1
4	1	0.310	0.971	2.378	4	71.43872	5	70.13692	0	4	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.003	0.000	2	71.56089	1	71.59826	2	19	True	1
1	0	1.362	0.984	1.362	3	71.43194	2	71.45849	2	16	True	1
2	4	0.306	0.980	1.668	3	71.44281	3	70.20769	0	6	True	1
3	2	0.319	0.975	1.986	3	71.45230	4	70.15769	0	5	True	1
4	1	0.320	0.971	2.306	3	71.46205	5	70.15769	0	5	True	1

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0.002	0.000	1	71.59826	1	71.63923	2	20	True	1
1	0	1.252	0.984	1.252	2	71.45850	2	71.48866	2	17	True	1
2	4	0.318	0.980	1.570	2	71.46972	3	70.24846	0	7	True	1
3	1	0.193	0.977	1.763	2	71.47926	4	70.18846	0	6	True	1
4	2	0.135	0.975	1.898	2	71.47923	5	70.18846	0	6	True	1

Estado Final t = 30

El estado final de la simulación de la carrera es el siguiente

```
display(race_simulation.to_df())
```

	id	interval	por	gap	ltg	llt	pos	plt	compuesto	v_usados	dos_tipos	n_paradas
0	3	0.000	0	0.000	0	71.63923	1	71.6838	2	21	True	1
1	0	1.134	0	1.134	0	71.52206	2	71.5598	2	19	True	1
2	4	0.309	0	1.443	0	71.51868	3	70.3600	0	9	True	1
3	1	0.337	0	1.780	0	71.65623	4	70.2800	0	8	True	1
4	2	0.339	0	2.119	0	71.85270	5	70.2800	0	8	True	1