

Entrenamiento de un Vehículo (Agente) Autónomo Usando Deep Q-Learning y ML-Agents en Unity

Juan Barragán
Luis Coronell

Universidad Nacional de Colombia, Sede de La Paz

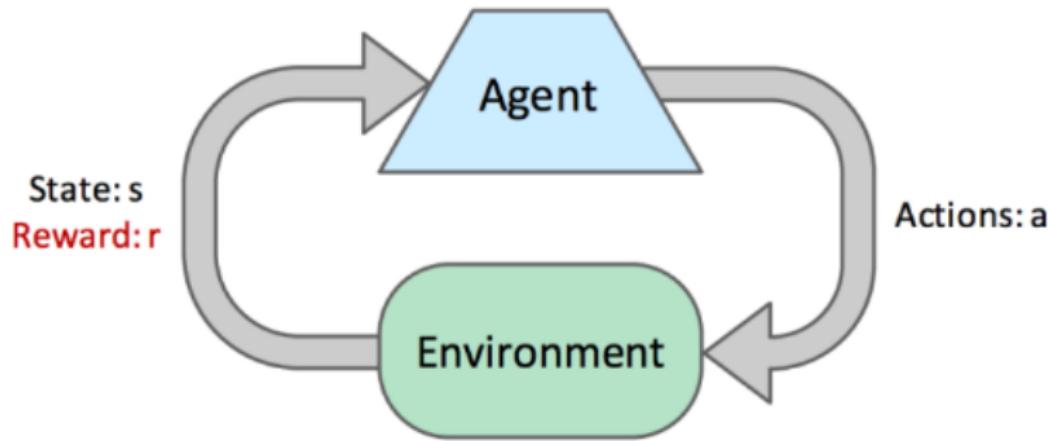
25 de julio de 2025

Contenido

- 1 Introducción
- 2 Objetivos
- 3 Arquitectura del Sistema
- 4 Diseño del Entorno
- 5 Observaciones y Acciones
- 6 Aprendizaje Reforzado
- 7 ML-Agents y Configuración
- 8 Visualización y Evaluación

Introducción

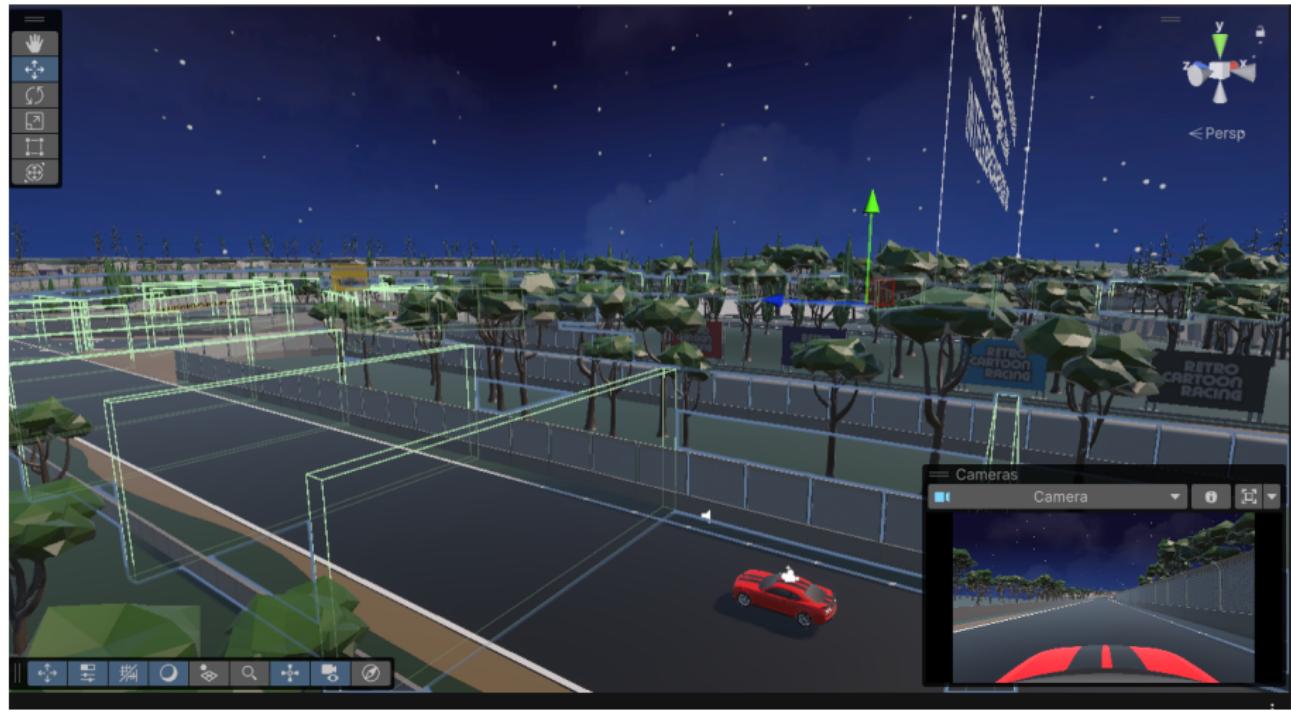
- Técnica de aprendizaje basada en la interacción agente-entorno.
- Objetivo: Maximizar la recompensa acumulada a través de la interacción del agente con el entorno.



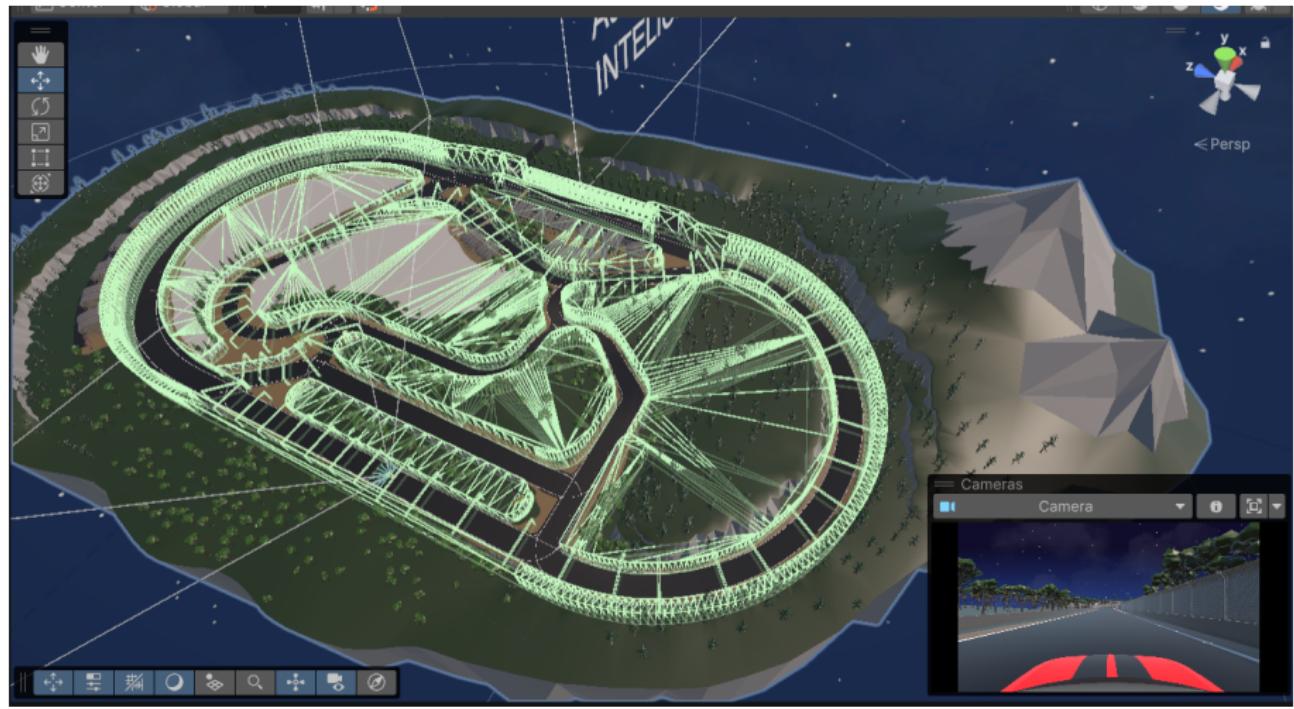
Objetivos

- Crear un vehículo autónomo (Agente) que aprenda a conducir en un entorno 3D mediante una ruta ya establecida.
- Integrar Unity y Python mediante ML-Agents.
- Comparar el rendimiento algoritmos de aprendizaje por refuerzo mediante la herramienta MLagents del entorno de unity6 y/o Deep Q-Learning para tomar decisiones óptimas mediante un algoritmo de python.

Entorno de Unity



Pista con las físicas



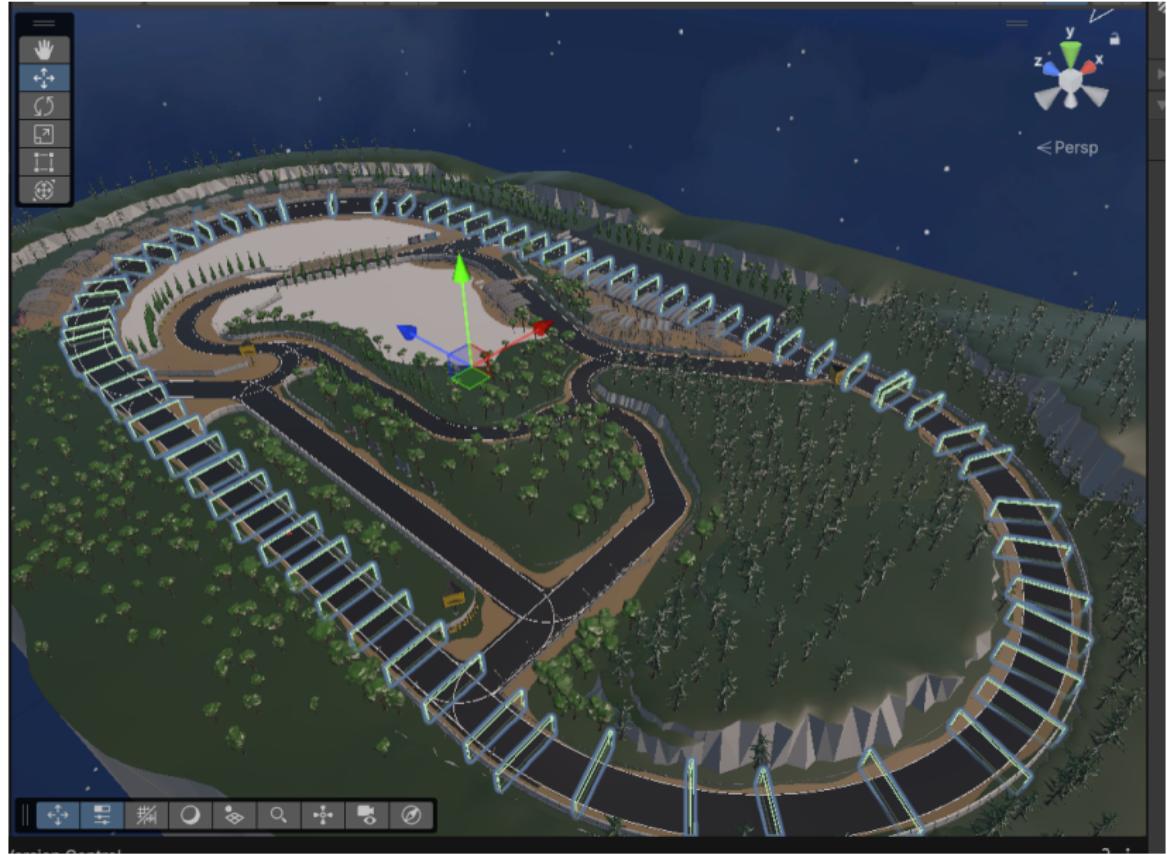
Diseño del Entorno en Unity

- Pista cerrada con obstáculos, cada obstáculo posee una etiqueta.
- Geometrías de muros fijas para evitar salirse del espacio determinado.
- Checkpoints ordenados que deben recorrerse en secuencia.
- Episodio se reinicia al chocar o desviarse.

Sistema de Checkpoints

- Recompensa positiva por cada checkpoint correcto.
- Penalización por omisión o colisión.
- Métricas:
 - ▶ Máximo pasos para completar la pista .
 - ▶ Distancia al siguiente checkpoint.
 - ▶ Recompensa acumulada.

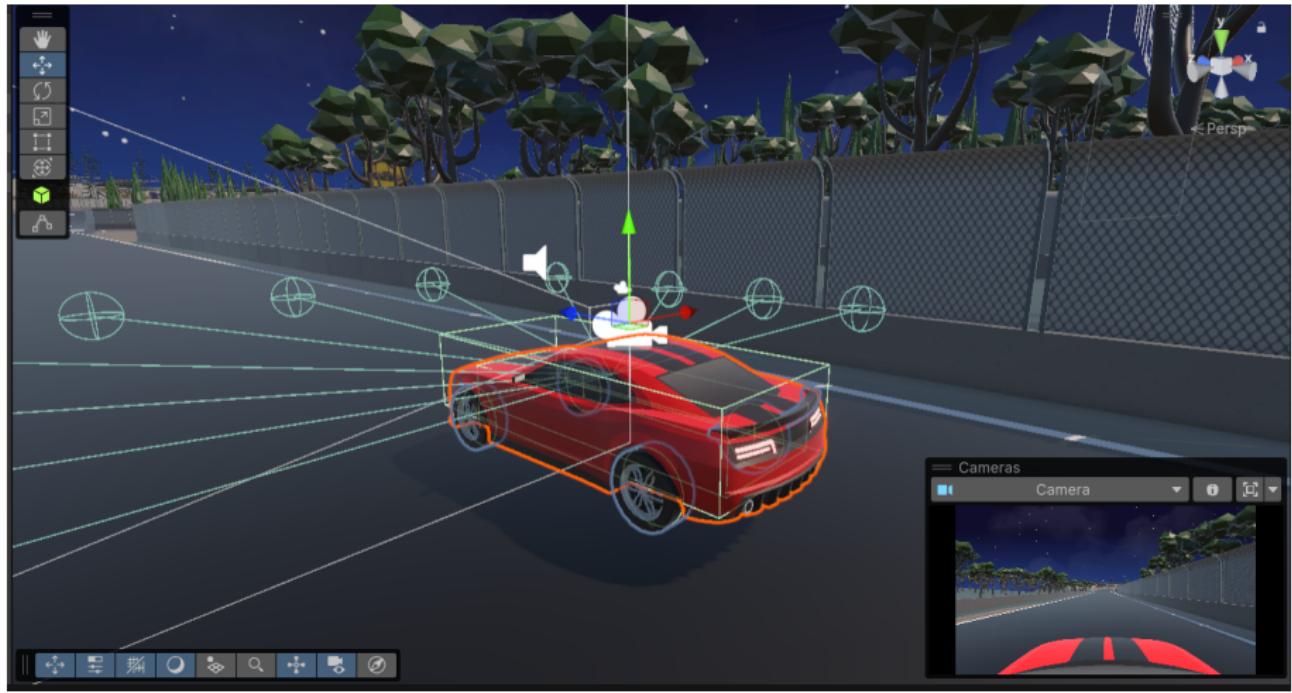
Checkpoints



Observaciones del Agente

- Velocidad y rotación del vehículo.
- Etiqueta hacia muros u obstáculos mediante el Raycast 3d.
- Estado del agente (posición, colisiones).

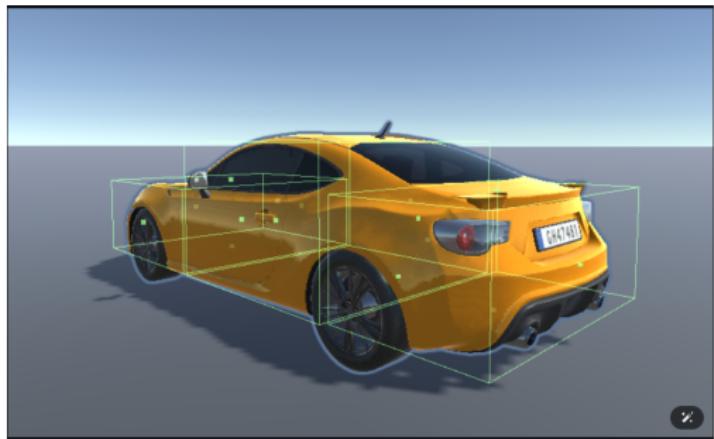
Agente



Car Controller Simple

- Este controlador amplía el controlador de automóvil Unity Standard Asset con: Mejor frenado, Antivuelco, Zonas de impulso (similares a las almohadillas de velocidad de Mario Kart)
- Centro de masa preciso
- Características configurables como fuerza del motor, frenado, ángulo de giro etc.

Car Simple Controller



Simple Car Controller (Script)

Script	SimpleCarController
Max Steer Angle	20
Motor Force	500
Brake Force	2000
Top Speed	245
Speed Type	KPH
Anti Roll	3500
Traction Control	<input checked="" type="checkbox"/>
Slip Limit	0.3
Steering Assist	<input checked="" type="checkbox"/>
Steering Assist Ratio	0.7
Number Of Gears	6
Audio Source	Car (Audio Source)
Minimum Pitch	0.5
Maximum Pitch	2
Boost Zone Multiplier	2

Wheel Colliders

= Element 0	Front Left Wheel Collider (Whe)
= Element 1	Front Right Wheel Collider (Whe)
= Element 2	Back Left Wheel Collider (Whe)
= Element 3	Back Right Wheel Collider (Whe)

Wheel Meshes

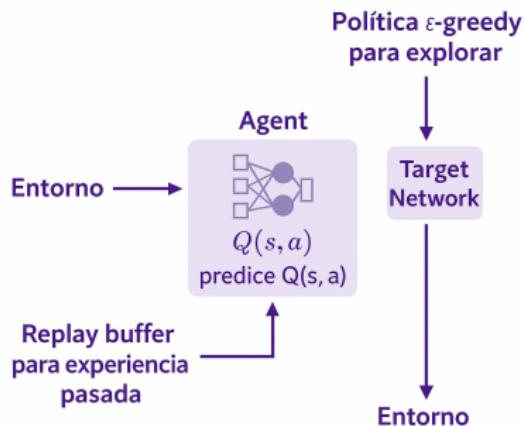
= Element 0	Front Left Wheel (Transform)
= Element 1	Front Right Wheel (Transform)
= Element 2	Back Left Wheel (Transform)
= Element 3	Back Right Wheel (Transform)

Acciones del Agente

- Aceleración: avanzar, frenar, neutro (-1, 0, 1).
- Dirección: izquierda, derecha, recto (-1, 0, 1).
- Espacio de acciones discreto: 9 combinaciones posibles.
- Máximo tiempo de recorrido 80 segundos.

Aprendizaje por Refuerzo

- Red neuronal que predice $Q(s, a)$.
- Política ϵ -greedy para explorar.
- Replay buffer para experiencia pasada.
- Target Network para estabilidad.



Entorno Python - UNITY

```
C:\Windows\system32\cmd.e: x + v
C:\Users\Juan David>conda activate unity
(unity) C:\Users\Juan David>mlagents --force
"mlagents" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

(unity) C:\Users\Juan David>mlagents-learn --force

Version information:
 ml-agents: 0.29.0,
 ml-agents-envs: 0.29.0,
 Communicator API: 1.5.0,
 PyTorch: 1.13.1+cpu
[INFO] Listening on port 5004. Start training by pressing the Play button in the Unity Editor.
```

Configuración de ML-Agents

- Uso de PPO (Proximal Policy Optimization): Es un algoritmo de aprendizaje por refuerzo profundo que entrena políticas directamente, limitando los cambios bruscos para mantener la estabilidad y mejorar el rendimiento en entornos complejos.
- YAML de configuración:
 - ▶ `buffer_size`, `batch_size`, `learning_rate`
 - ▶ `epsilon_start`, `epsilon_end`, `epsilon_decay`
- Sincronización con '`time_scale`' y '`decision_interval`'.

Agente

The screenshot shows a code editor with three tabs at the top: 'car_config.yaml' (active), 'Solution.ipynb', and 'Untitled-1.ipynb'. The main area displays the following YAML configuration:

```
C: > Users > Juan David > ! car_config.yaml
  behaviors:
    DQLCarAgent:
      trainer_type: ppo
      max_steps: 5000000
      time_horizon: 64
      summary_freq: 1000
      keep_checkpoints: 5
      checkpoint_interval: 10000
      threaded: true

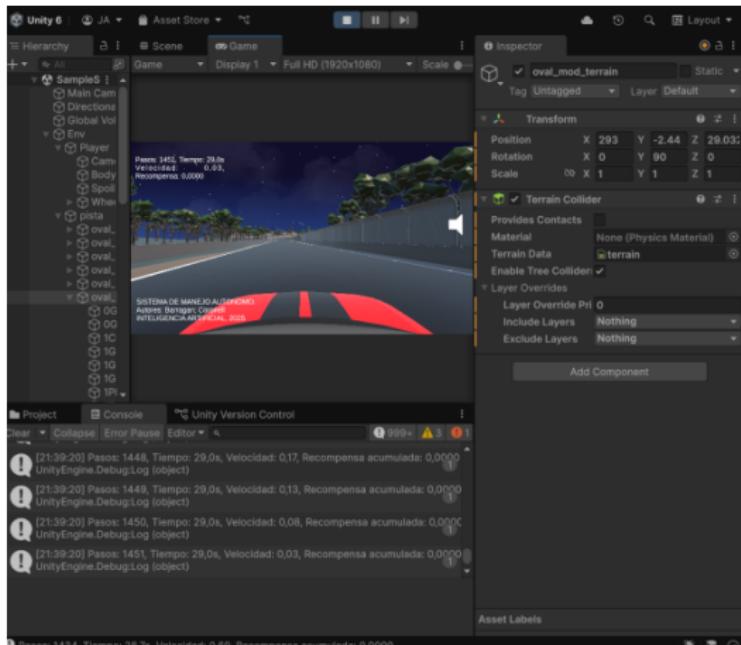
      hyperparameters:
        batch_size: 1024
        buffer_size: 10240
        learning_rate: 0.0003
        beta: 0.005
        epsilon: 0.2
        lambda: 0.95
        num_epoch: 3
        learning_rate_schedule: linear
        beta_schedule: linear
        epsilon_schedule: linear

      network_settings:
        normalize: false
        hidden_units: 128
        num_layers: 2
        vis_encode_type: simple
        deterministic: false

      reward_signals:
        extrinsic:
          gamma: 0.99
          strength: 1.0
```

- Comportamiento NPC (MonoBehaviour o Mono Comportamiento) .
- Se asigna el tiempo de algoritmo (PPO).
- Se ajustan los hyperparametros.
- Aprende de la experiencia.
- Pérdida:
$$\mathcal{L} = (r + \gamma \max Q(s', a') - Q(s, a))^2$$
- Optimización por descenso del gradiente (Adam).

Entrenamiento del agente



```
[INFO] Hyperparameters for behavior name DQLCarAgent:  
trainer_type: ppo  
hyperparameters:  
batch_size: 1024  
buffer_size: 10240  
learning_rate: 0.0003  
beta: 0.005  
epsilon: 0.2  
lambd: 0.95  
num_epoch: 3  
learning_rate_schedule: linear  
beta_schedule: linear  
epsilon_schedule: linear  
network_settings:  
normalize: False  
hidden_units: 128  
num_layers: 2  
vis_encode_type: simple  
memory: None  
goal_conditioning_type: hyper  
deterministic: False  
reward_signals:  
intrinsic:  
gamma: 0.99  
strength: 1.0  
network_settings:  
normalize: False  
hidden_units: 128  
num_layers: 2  
vis_encode_type: simple  
memory: None  
goal_conditioning_type: hyper  
deterministic: False  
init_path: None  
keep_checkpoints: 5  
checkpoint_interval: 500000  
max_steps: 500000  
time_horizon: 64  
summary_freq: 50000  
threaded: False  
self_play: None
```

SARSA: State-Action-Reward-State-Action

SARSA es un algoritmo **on-policy** de aprendizaje por refuerzo. La idea principal de SARSA es el ensayo y error.

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]$$

- Aprende de la acción realmente tomada (a').
- Balancea exploración y explotación mediante una política ϵ -greedy.
- Se actualiza tras cada paso del agente.

Aproximación de Q con Fourier

La función $Q(s, a)$ se aproxima mediante una combinación lineal:

$$Q(s, a) \approx \sum_{i=0}^n \theta_{i,a} \cdot \phi_i(s)$$

Donde:

- $\phi_i(s) = \cos(\pi c_i^T s)$ son las funciones base de Fourier.
- c_i es un vector de coeficientes enteros.
- $\theta_{i,a}$ son los pesos aprendidos por acción.
- Los espacios de estado pueden ser **continuos y de alta dimensión**.
- Q-Table tradicional no escala bien en entornos complejos.
- Fourier permite representar funciones de valor como combinación de senos y cosenos.

Entrenamiento del Agente

- ① Obtener estado s_t desde Unity.
- ② Calcular $\phi(s_t)$ con Fourier.
- ③ Seleccionar acción a_t con política ϵ -greedy.
- ④ Aplicar acción en Unity, recibir r_t, s_{t+1} .
- ⑤ Actualizar pesos con:

$$\theta_{a_t} \leftarrow \theta_{a_t} + \alpha \cdot \delta \cdot \phi(s_t)$$

- ⑥ Donde:

$$\delta = r + \gamma Q(s', a') - Q(s, a)$$

Métricas de Evaluación

- Pasos por episodio.
- Recompensa total.
- Checkpoints alcanzados.
- Colisiones detectadas.
- Tiempo promedio por vuelta.

Visualización de Métricas



Metricas de PPO

Environment/Cumulative Reward

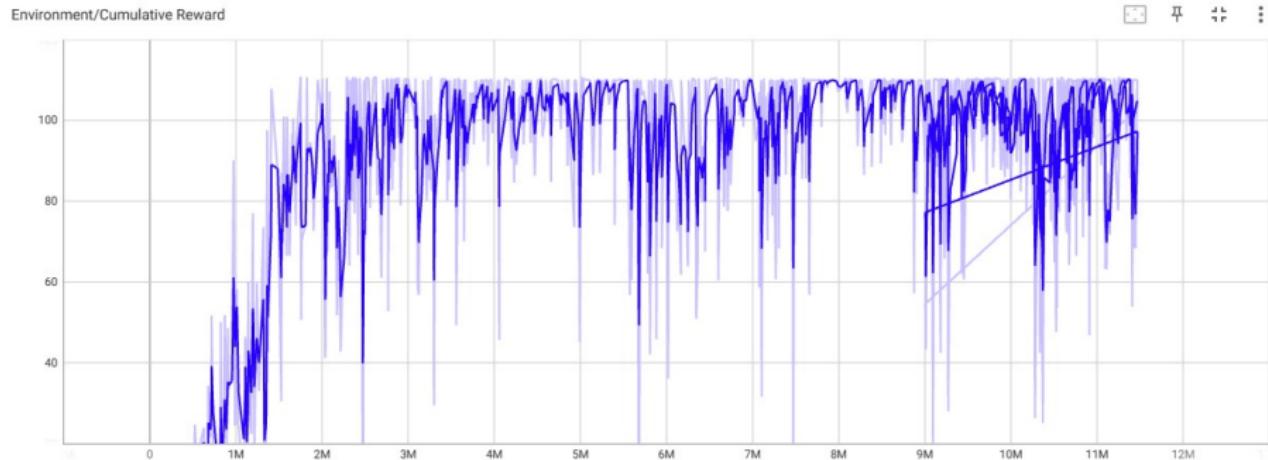


Figura: PPO: Estrategia con reinicio de episodio al alcanzar recompensa acumulada de -25: Recompensa vs pasos totales

Metricas de PPO

Environment/Episode Length

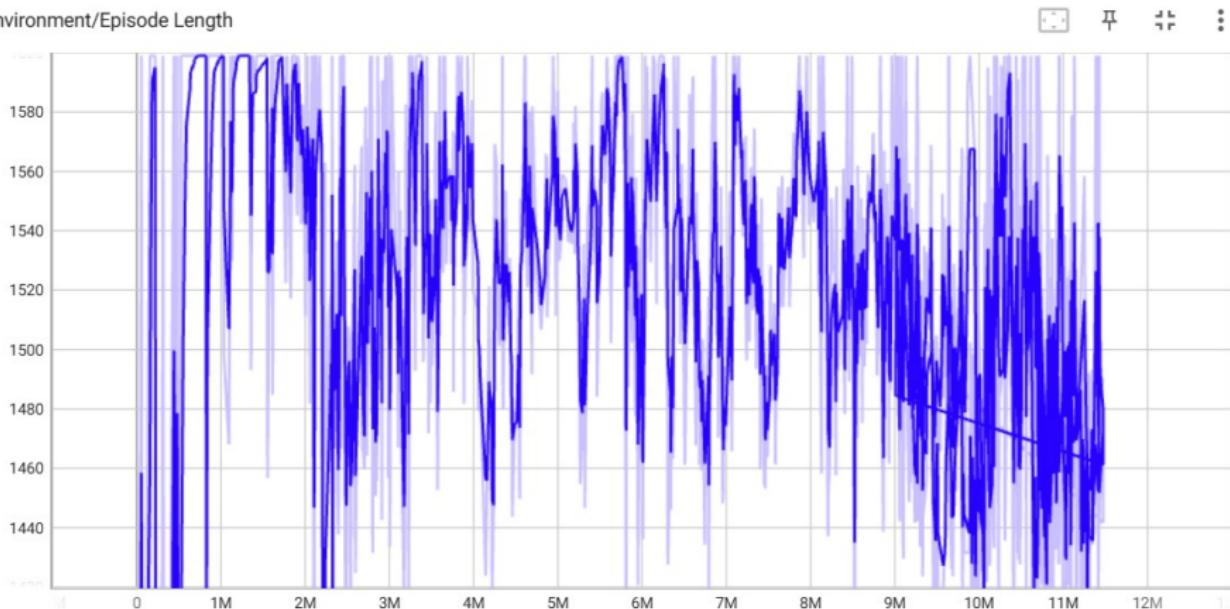


Figura: PPO: Estrategia con reinicio de episodio al alcanzar recompensa acumulada de -25: Pasos totales vs Episodios.

Metricas de PPO

Environment/Cumulative Reward

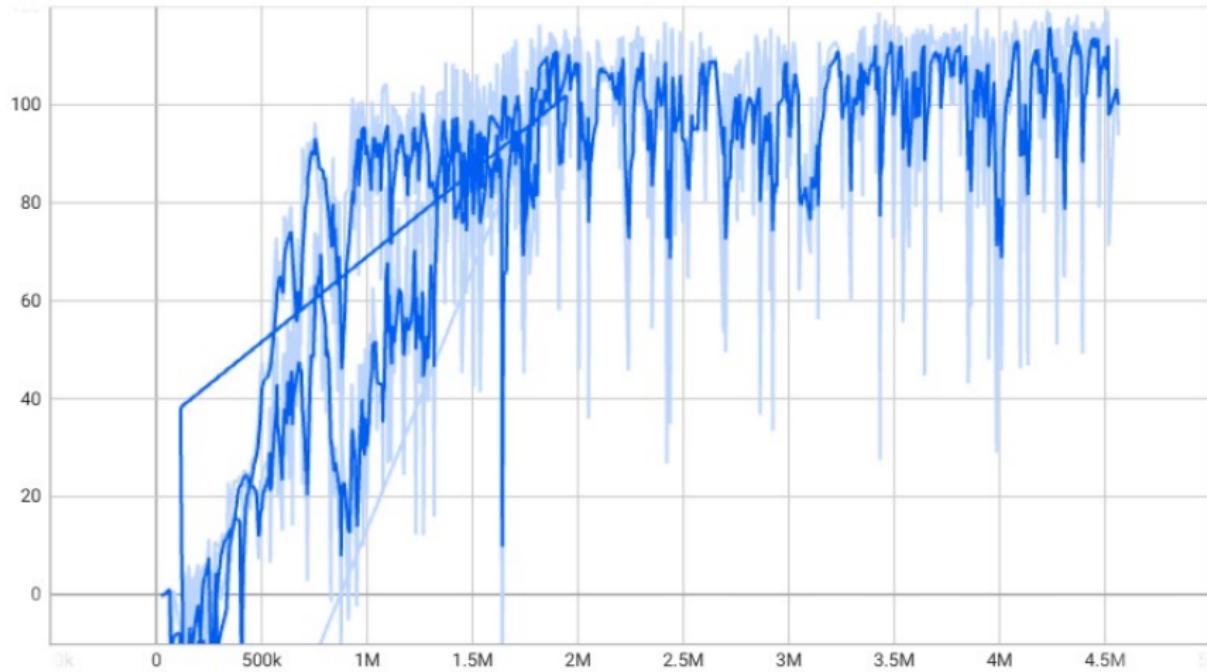


Figura: PPO: Sin reinicio de episodio al alcanzar recompensa negativa de -25:
Recompensa vs pasos totales



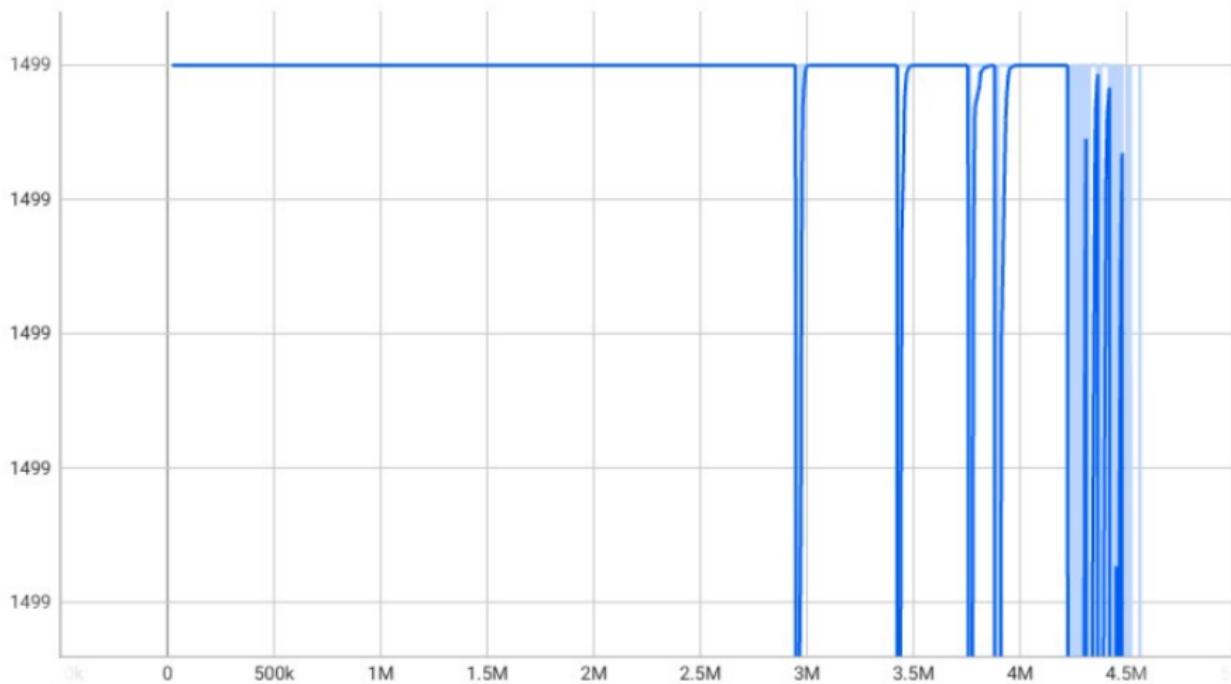


Figura: PPO: Estrategia sin opción de reinicio anticipado al alcanzar recompensas negativas: Pasos Totales vs Episodios

Metricas de SARSA

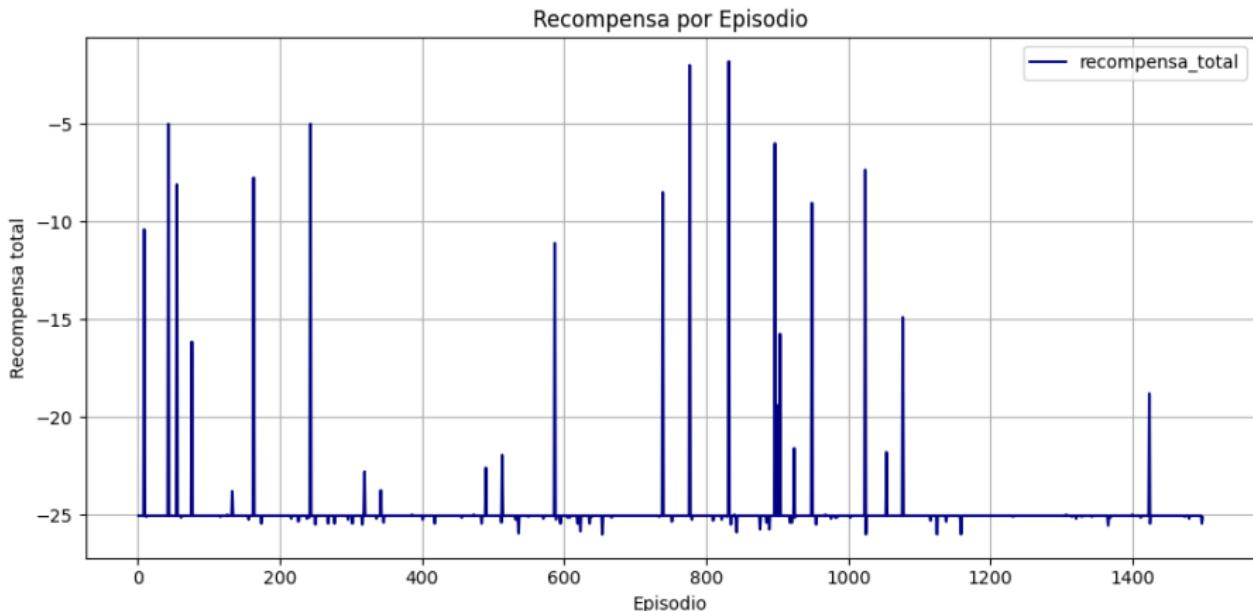


Figura: SARSA con la optimización de tiempo, orden de Fourier 3 y 1000 aproximaciones. Recompensa vs Episodios

Metricas de SARSA

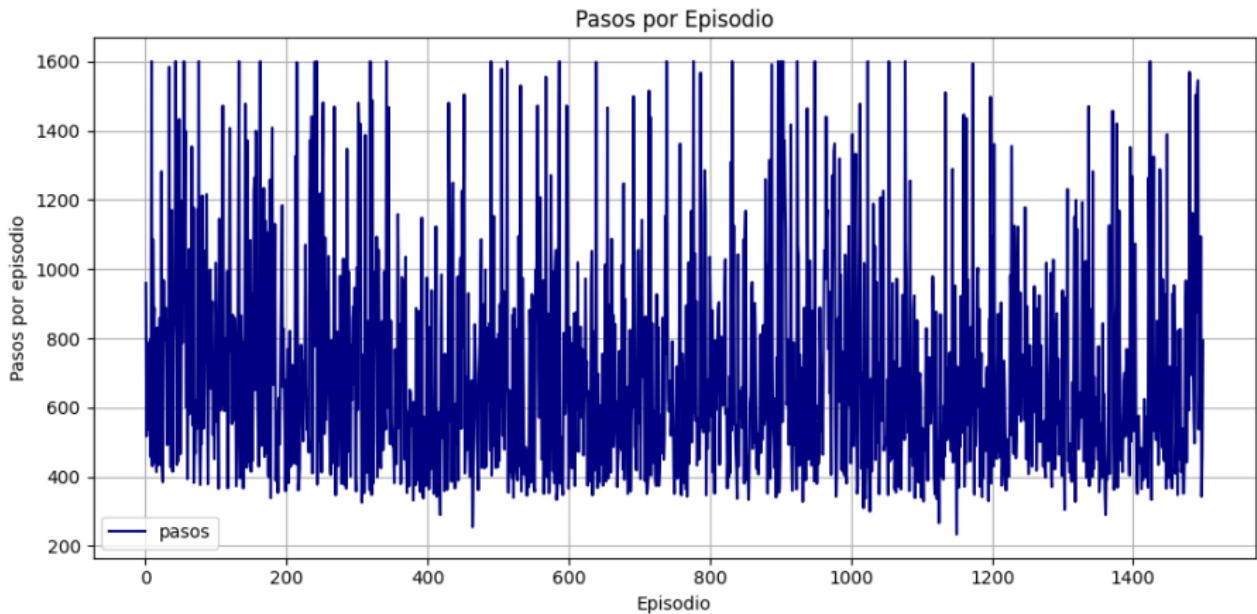


Figura: SARSA con la optimización de tiempo, orden de fourier 3 y 1000 aproximaciones.: Pasos alcanzados en cada episodio vs episodios totales.

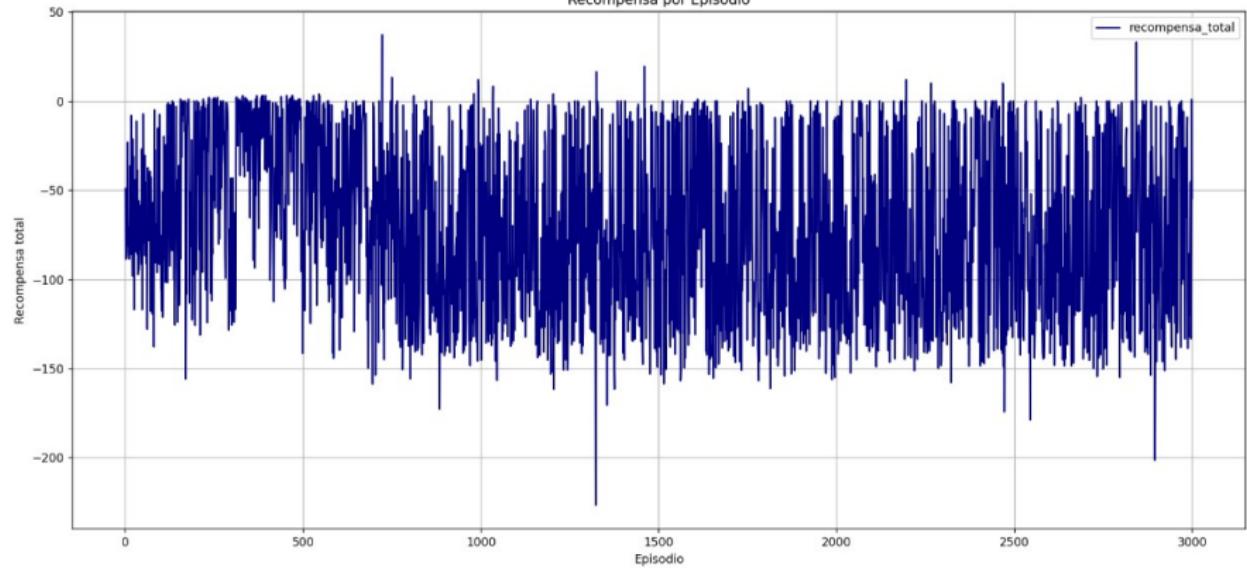


Figura: SARSA sin optimización de tiempo: orden de Fourier 5 y 5
aproximaciones: Recompensa vs episodios totales.

Pasos por Episodio

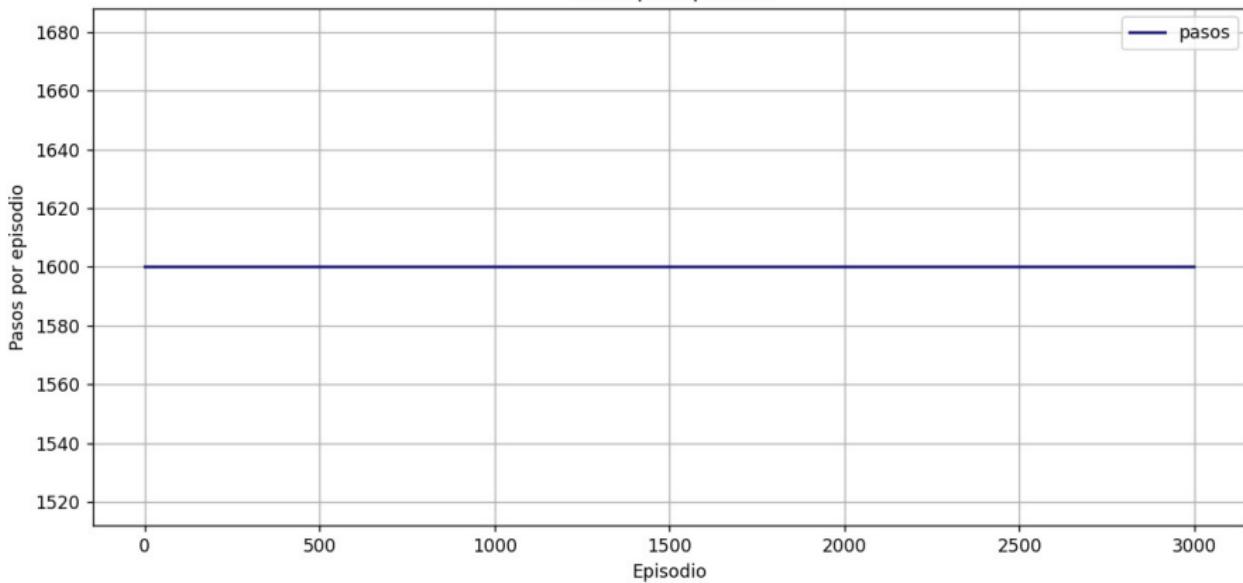


Figura: SARSA sin optimización de tiempo: orden de Fourier 5 y 5
aproximaciones: Pasos vs Episodios totales