PRÁCTICA DL – ***DEEP REINFORCEMENT LEARNING APLICADO A SUPERMARIO BROS***

Asahel Hernández Torné

1. JUPYTER NOTEBOOK

Se ha modificado el Notebook original para eliminar las celdas que únicamente servían como tutorial introductorio y facilitar la ejecución de los entrenamientos.

Además, se ha añadido un GIF en la primera celda de texto donde se visualiza el mejor comportamiento obtenido por los entrenamientos.

El notebook se organiza en cuatro secciones principales:

1. Configuración del entorno

2. Modificación y ampliación

- 2.1 Preprocesado de la imagen.
- 2.2 Personalización de la función de recompensa.
- 2.3 Obtención de información temporal.
- 2.4 Creación del entorno de entrenamiento

3. Entrenamiento del agente

- 3.1 Implementación de entornos vectorizados
- 3.2 Creación de *callbacks* durante el entrenamiento
- 3.3 Funciones adicionales
- 3.4 Entrenamiento de la política

4. Evaluación de desempeño

- 4.1 Visualización del comportamiento del agente
- 4.2 Evaluación interactiva del agente (visualización en tiempo real)

Todos los *imports* de librerías se han aglutinado en la primera celda para no tener ningún error de inclusión.

2. WORKSPACE DEL PROYECTO

El *workspace* del proyecto se ha organizado de la siguiente manera:

```
Proyecto Mario RL/
├── .venv/                          → Entorno virtual de Python
├── docs/                           → Documentación del proyecto
├── final_models/                   → Modelos PPO entrenados
│   └── PPO_CNN_2705_1.zip ...
├── mario_logs/                     → Logs de entrenamiento
├── mario_models/                   → Modelos intermedios
├── mario_monitor_dir/              → Información de episodios
├── media/                          → Gifs e imágenes
│   ├── gifs/                       → Render de los agentes jugando
│   │   ├── PPO_CNN_2705_2.gif ...
│   └── images/                     → Output finales de los entrenamientos
│       └── PPO_CNN_2705_2.png ...
├── src/                            → Código de backup
├── mario_final_model.zip           → Modelo PPO final entrenado
├── mario_rl.ipynb                  → Notebook de la práctica
└── requirements.txt                → Dependencias del entorno
```

Para la entrega se simplifica el *workspace,* entregando únicamente el *notebook*, el mejor modelo y la carpeta media con las imágenes y gifs para su visualización en el *notebook*:

```
EntregaRL_AsahelHernandez/
├── media/                          → Gifs e imágenes
│   └── gifs/
│       └── PPO_CNN_3005_3.gif      → Render del agente jugando
├── DeepRL_Memoria.pdf              → Memoria del proyecto
├── mario_final_model.zip           → Modelo PPO final entrenado
└── mario_rl.ipynb                  → Notebook de la práctica
```

3. MARCO TEÓRICO

3.1 ALGORITMO

Primeramente, se debe evaluar la tarea a realizar y los diferentes algoritmos y métodos que pueden ser utilizados para su resolución.

El entorno donde se desarrolla la tarea es un juego de plataformas 2D donde las observaciones realizadas por el agente son *frames* de la imagen, es decir observaciones visuales. Además, el entorno incluye elementos adicionales, físicas, enemigos y colisiones, conformando un entorno no determinista.

Las acciones que puede emprender el agente son discretas: movimientos, saltos, etc. y las recompensas son modificables, pueden ser densas o esparcidas, dependiendo del criterio que se aplique.

Stable Baselines3 integra una serie de algoritmos para RL. A continuación, se analizarán dichos algoritmos y su idoneidad para poder entrenar un agente que sea capaz de superar un nivel de SuperMario Bros.

De la lista de algoritmos disponibles para utilizar en el entrenamiento, se pueden descartar directamente los siguientes:

- **DDPG**, **TD3**, **SAC**, **TQC**, **CrossQ**: Requieren acciones continuas. No sirven directamente para la tarea, donde las acciones son discretas (mover/saltar).
- **ARS**: Aunque funciona para acciones discretas, no permite múltiples acciones.
- **HER**: Está pensado para tareas de tipo *goal-based*. Aunque la tarea tiene un objetivo a cumplir ideal, superar el nivel, tiene otros objetivos intermedios.

Los algoritmos que podrían funcionar, pero no son ideales o requieren procesamiento extra:

- **DQN**: Soporta acciones discretas, pero no es tan robusto con imágenes. Necesita adaptaciones extra.
- **QR-DQN**: Variante de DQN con distribución de valores. Mismo problema: requiere muchos trucos para funcionar bien con imágenes.

Aquellos que sí podrían ser utilizados, obteniendo resultados correctos serían:

- **PPO**: El más estable, ideal para entornos visuales con acciones discretas. Soporta imágenes, *multiprocessing* y recompensas complejas. Muy usado en RL moderno.
- **A2C**: Similar a PPO, pero menos estable. Apto para entornos discretos con imágenes. Puede ser una alternativa ligera.
- **RecurrentPPO**: Igual que PPO, pero con memoria (LSTM). Ideal si el entorno necesita "recordar" lo que pasó (por ejemplo, si la visibilidad es limitada).
- **TRPO**: Similar a PPO, pero más exigente en cálculo. Funciona bien pero menos práctico que PPO para CPU.

ALGORITMO ESCOGIDO: PPO (Proximal Policy Optimization)

Tras analizar los algoritmos disponibles, se ha seleccionado PPO para realizar las pruebas y entrenamientos ya que es el más idóneo para la tarea, ya que:

- Diseñado para entornos con observaciones visuales complejas.
- Estable y robusto, puede trabajar con recompensas mal escaladas o ruidosas.
- Funciona bien para entornos discretos.
- Fácil paralelización.
- Algoritmo muy usado en este tipo de entornos.

3.2 POLÍTICAS

Para entrenar un agente mediante PPO se pueden emplear diferentes políticas, implementadas en *Stable Baselines3*. A continuación, se analizan las políticas disponibles en la librería:

- **CNN**: Es la política más adecuada para este caso ya que su uso está optimizado para entornos con observaciones visuales (formato [C,H,W]).

  Dado que SuperMario Bros es un entorno visual, el uso de una red convolucional (CNN) que extraiga automáticamente características visuales relevantes del estado del juego, monstruos, posición del agente, obstáculos, monedas, etc., resulta adecuado para la tarea. Debe tenerse en cuenta que no capta dependencias temporales, solo interpreta lo que visualiza en el *frame* actual o *frames* apilados.

- **CNNLstm**: Esta política integra una red convolucional CNN para extraer características visuales más una red LSTM para mantener una memoria a largo plazo.

  Podría utilizarse para resolver la tarea siempre y cuando se defina correctamente cómo manejar dicha memoria ya que solventa el problema de las CNN de no poder captar dependencias temporales. No requeriría el uso de *frames* apilados. Dado que integra una capa más (LSTM), aumenta el la carga computacional y el coste temporal del entrenamiento.

- **CNNLnLstm**: Esta política es una ampliación de CNNLstm pero añadiendo una etapa de *Layer Normalization* para mejorar la estabilidad durante el entreno. Sería una opción para utilizar si el uso de CNNLstm en el entrenamiento divergiese o no fuese estable.

- **MLP**: Aunque esta política puede utilizarse en entornos similares al SuperMario, dado que no puede trabajar con imágenes directamente, requiere de procesamiento extra para obtener las características visuales del entorno y codificarlas de manera vectorizada para que MLP pueda trabajar correctamente. Esta política es más útil en otros entornos como articulaciones robóticas.

- **MLPLstm** y **MLPLnLstm**: Al igual que MLP, no procesa información visual.

Además, *Stable Baselines3* permite crear políticas customizadas, pero dado que no se dispone del tiempo suficiente, se descarta la idea.

POLÍTICA ESCOGIDA: CNN (Convolutional Neural Network)

Se concluye que la mejor política a aplicar es una red convolucional CNN, y sus derivados CNNLstm y CNNLnLstm ya que:

- Es un tipo de red neuronal especializada en procesamiento de imágenes.
- El entorno de la tarea es visual, observaciones visuales mediante *frames*. Se requiere entender el espacio.
- La red CNN extrae **directamente** las características importantes.
- Comparada con una red MLP, CNN tiene menos parámetros ya que los comparte a través de los *frames*. Esto se traduce en un entrenamiento más rápido, una mayor generalización y menos memoria requerida.

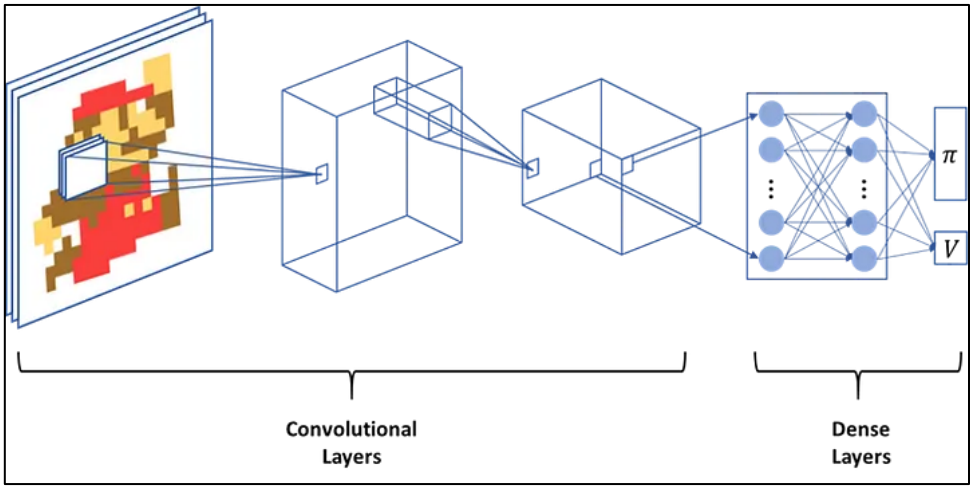


Ilustración 1. Cómo funciona PPO con red CNN para el entorno de SuperMario Bros.

3.3 CONFIGURACIÓN DEL ENTRENAMIENTO Y EL ENTORNO

Existen diferentes parámetros en el entorno y entrenamiento que pueden ser modificados para obtener distintos resultados:

Entorno

Pueden seleccionarse distintos mundos y escenarios en los que entrenar el agente. Por defecto y para todos los entrenamientos se ha seleccionado el mundo 1, escenario 1.

El número de entornos vectorizados que pueden ejecutarse también es un parámetro modificable. Se ha establecido en 8 ya que la capacidad de cómputo del ordenador donde se han realizado las pruebas permitía este valor.

Observaciones

El número de *frames* a apilar, para generar una observación temporal para PPO, es un parámetro importante a tener en cuenta. Se ha mantenido 4 como número de *frames* apilados durante todas las pruebas.

Acciones

De las acciones disponibles para aplicar por parte del agente se pueden seleccionar 3 *subsets:* ***RIGHT_ONLY, SIMPLE_MOVEMENT, COMPLEX_MOVEMENT***. Se ha probado tanto con todas las configuraciones.

Además, se ha añadido un número de *frames* durante los cuales mantener la misma acción. Durante 4 *frames,* el agente mantendrá la misma acción.

Parámetros de entrenamiento

PPO admite diferentes parámetros de entrada a la hora de configurar el modelo y el entrenamiento.

- **Política:** El tipo de arquitectura que usará el agente. “CnnPolicy” para utilizar una red CNN.
- **Learning Rate:** Tasa de aprendizaje, puede mantenerse en un valor fijo o decaer con el tiempo para estabilizar el entrenamiento. Controla cuánto se actualizan los pesos de la red en cada paso de optimización.
- **Number of Steps:** Número de pasos (experiencias) antes de cada actualización de la política.
- **Batch Size:** Tamaño del lote de entrenamiento usado en la optimización. Influye en la estabilidad y eficiencia del entrenamiento:
- **Entropy Coefficient:** Coeficiente de la entropía. Aumentar este valor fomenta la exploración, penalizando políticas muy deterministas.

Se probará con diferentes valores para ***learning rate***, ***number of steps*** *y* ***batch size***.

4. FUNCIÓN DE RECOMPENSA.

La función de recompensa ha sufrido muchas modificaciones a lo largo del desempeño de la práctica, como se puede observar en el *Diario de Pruebas*, pero manteniendo la siguiente estructura básica:

- Penalización por morir: -*death_penalty*
- Recompensa por moverse: +*delta_x_reward*
- Recompensa por llegar a la bandera: +*flag_reward*
- Otras recompensas y penalizaciones: ±*other_values*

Quedando la función de recompensa, de manera simplificada, como:

$$\textbf{Recompensa} = +delta\_x\_reward + (-death\_penalty) + flag\_reward \pm other\_values$$

En el último agente entrenado, 3005_4, puede verse la función de recompensa final.

5. MEJORAS APLICADAS

3.1 MANTENER ACCIÓN DURANTE UN NÚMERO DE FRAMES

Se ha implementado una mejora al *Wrapper CustomStackFrame* que permite mantener la misma acción realizada durante un número de *frames*, ya que hay ciertas acciones que requieren que la acción se mantenga fija, además que se ajusta más al comportamiento realista que tiene un humano al jugar al videojuego, manteniendo los botones pulsados durante un número de *frames*. (**Generado con ChatGPT**).

```
for _ in range(self.repeat):
    obs, reward, done, trunk, info = self.env.step(action)
    processed_obs = process_frame(obs)
    self.frames.append(processed_obs)
    total_reward += reward
    if done or trunk:
          break
```

Las recompensas obtenidas durante la repetición de acciones se suman para evitar suavizar la señal de recompensa a lo largo del tiempo.

3.2 DECRECIMIENTO LINEAL DEL LEARNING RATE

Se ha implementado una función de decrecimiento del *learning rate*, que, partiendo de un valor inicial, va decayendo linealmente hasta llegar a 0 (o un valor infinitamente pequeño). Esto permite que el agente explore más al inicio y explote más al final.

Para aplicar un decaimiento lineal del *Learning Rate* se ha creado una nueva función:

```
def linear_schedule(initial_value: float) -> Callable[[float], float]:
    def func(progress_remaining: float) -> float:
        return progress_remaining * initial_value
    return func
```

La función se ha obtenido de la documentación de *StableBaselines3: Learning Rate Schedule*.

6. PRUEBAS REALIZADAS

PPO con arquitectura CNN

En la siguiente tabla se pueden observar los parámetros aplicados durante los entrenamientos realizados y los resultados obtenidos, en una escala del 1 al 10, donde se pondera el desempeño del agente a lo largo del nivel, y lo lejos que consigue llegar.

1e6 timesteps							
AGENTE	**TIEMPO**	**L. RATE**	**BATCH**	**NSTEPS**	**# ENVS**	**IMG**	**EXITO**
2705_1	190m 48.7s	2.5e-4	256	512	1	84x84	1
2705_2	157m 22.5s	L(0.001)	64	512	8	84x84	**5**
2805_1	149m 15.0s	L(0.001)	128	1024	8	84x84	1
2805_2	157m 06.4s	L(0.0001)	128	1024	8	84x84	2
2805_3	148m 43.3s	0.000001	64	512	8	84x84	1
2805_4	147m 56.8s	L(0.001)	128	1024	8	84x84	1
2805_5	153m 23.9s	L(0.0005)	64	512	8	84x84	4
2905_1	149m 37.8s	L(0.0005)	64	512	8	84x84	**5**
2905_2	167m 19.3s	L(0.0007)	64	512	8	84x84	2
2905_3	161m 53.1s	L(0.0005)	64	512	8	84x84	6
2905_4	154m 42.1s	L(0.0005)	64	512	8	84x84	1
2905_5	158m 50.7s	L(0.0005)	64	512	8	84x84	8
3005_1	151m 23.5s	L(0.00035)	64	512	8	84x84	6
3005_2	149m 9.7s	L(0.0007)	64	512	8	84x84	6
3005_3	**156m 13.8s**	**L(0.0005)**	**64**	**512**	**8**	**84x84**	**10**
1.5e6 timesteps							
3005_4	238m 20.1s	L(0.00035)	64	512	8	84x84	10

DIARIO DE PRUEBAS

Además de la tabla donde se numeran las pruebas realizadas, los principales parámetros de entrenamiento y configuración y el resultado obtenido se ha elaborado un diario de pruebas donde se ha ido registrando los cambios realizados entre las pruebas.

El formato empleado para el diario es secuencial, anotándose prueba tras prueba, e indicando entre que agentes se ha realizado el cambio, como indica cada título de subsección.

En cada entrada del diario se describen los resultados obtenidos, visualizaciones gráficas en los entrenamientos más relevantes, y los cambios significativos en la función de recompensa, la estructura del entorno y el agente y en algunos parámetros relevantes. También se describen las observaciones, deducciones y motivaciones que han llevado a realizar los cambios aplicados entre pruebas.

0. Modificaciones previo agente 2705_1:

Se parte de una función de recompensa muy simple inicial y unos parámetros de entreno sugeridos por **ChatGPT** como inicio de las pruebas.

1. Conclusiones agente 2705_1 y modificaciones previo agente 2705_2:

Se mejora la función de recompensa en varios aspectos: se añaden penalizaciones por quedarse quieto o retroceder, se añade penalización por tiempo transcurrido, se añade penalización por morir.

Se descubre la posibilidad de implementar un *linear_schedule* al *learning rate (Learning Rate Schedule)*. **(Sugerencia de ChatGPT).**

2. Conclusiones agente 2705_2 y modificaciones previo agente 2805_1:

El último *callback* de información del agente 2705_2 arroja la siguiente información:

--------------------------------------- | rollout/ | | | ep_len_mean | 168 | | ep_rew_mean | 211 | | time/ | | | fps | 106 | | iterations | 245 | | time_elapsed | 9435 | | total_timesteps | 1003520 | | train/ | | | approx_kl | 2.2559121e-05 | | clip_fraction | 0 | | clip_range | 0.2 | | entropy_loss | -0.155 | | explained_variance | 0.931 | | learning_rate | 5.76e-07 | | loss | 13.2 | | n_updates | 2440 | | policy_gradient_loss | 2.45e-05 | | value_loss | 29.2 | ---------------------------------------	**ep_len_mean:** El agente dura de media 168 pasos por episodio. Esto indica que no muere inmediatamente, pero tampoco termina niveles muy largos. **ep_rew_mean:** Recompensa media por episodio. Indica que el agente no está completando el nivel ya que la recompensa por bandera es +1000. **approx_kl, entropy_loss, learning_rate**: Valores esperados dado que es el último *callback, y* se entiende que ya ha acabado de actualizar políticas y aprender. **explained_variance:** El modelo "entiende" bien lo que pasa en el entorno. **policy_gradient_loss:** Política ha convergido. **loss, value_loss:** Se debe comprobar que sean estables.

Las explicaciones de algunos de los parámetros anteriores se han obtenido utilizando **ChatGPT** y contrastando con la documentación disponible. Sólo se han añadido al informe los últimos *callbacks* de los entrenamientos significativos.

El resultado obtenido en el entrenamiento de 2705_2 mejora con creces el obtenido en la prueba anterior, pero Mario muere por un *Goomba* tras recorrer una distancia considerable, por lo que se aumenta la penalización por morir:

- *Death penalty:* -50 → -500

Se aumenta el número de pasos a actualizar y el tamaño del lote de entrenamiento.

3. Conclusiones agente 2805_1 y modificaciones previo agente 2805_2:

Se observa mal comportamiento en el agente 2805_1 ya que no consigue superar el primer *Goomba*. Se prueba a modificar la función de recompensa:

- *Death penalty*: -500 → -100 (más cercana al valor del agente 2705_2)
- *Time penalty*: -0.0005 → -0.001 (se aumenta penalización por inacción)
- *Δx Reward*:
  - Avanzar: +0.1 → +1.0
  - Retroceder bruscamente: -2.0 → -10.0
  - Retroceder suavemente: -0.02 → -0.1
  - Quedarse quieto: -0.2 → -1.0

4. Conclusiones agente 2805_2 y modificaciones previo agente 2805_3:

El último *callback* de información obtenido del entrenamiento del agente 2805_2 refleja lo siguiente:

--	
| rollout/ | |	**ep_len_mean:** El agente dura de media 121 pasos por episodio.
| ep_len_mean | 121 |	**ep_rew_mean:** Aparece un número superior ya que la recompensa por avanzar aumentó considerablemente. No significa que funcione mejor.
| ep_rew_mean | 902 |	**approx_kl, entropy_loss, learning_rate**: Valores esperados dado que es el último *callback, y* se entiende que ya ha acabado de actualizar políticas y aprender.
| time/ | |	**explained_variance:** El modelo predice moderadamente.
| fps | 106 |	**policy_gradient_loss:** Política ha convergido y apenas cambia.
| iterations | 245 |	**loss, value_loss:** Un valor tan alto indica que las recompensas son muy dispares.
| time_elapsed | 9420 |	
| total_timesteps | 1003520 |	
| train/ | |	
| approx_kl | 1.5255573e-05 |	
| clip_fraction | 0 |	
| clip_range | 0.2 |	
| entropy_loss | -0.838 |	
| explained_variance | 0.516 |	
| learning_rate | 5.76e-08 |	
| loss | 3.38e+03 |	
| n_updates | 2440 |	
| policy_gradient_loss | -0.000118 |	
| value_loss | 6.32e+03 |	
--	

La interpretación de **ChatGPT** respecto a estos resultados no era del todo correcta, ya que el LLM desconocía que la función de recompensa había cambiado y consideraba que el agente estaba avanzando mucho con respecto a anteriores entrenamientos.

Se cambia el tipo de movimiento a *COMPLEX_MOVEMENT* para permitir un rango mayor de acciones.

Se prueba con un *learning rate* fijo para permitir que el agente aprenda de manera constante.

5. Conclusiones agente 2805_3 y modificaciones previo agente 2805_4:

Dado que los resultados siguen siendo malos, se revierte el cambio al *learning rate* lineal y se aumenta notablemente su valor, se aumenta el espacio de entrenamiento.

No se hacen cambios a la función de recompensa.

Se añade la posibilidad de visualizar gráficamente la evolución de la recompensa durante el entrenamiento, lo que facilita el análisis temporal de lo sucedido.

6. Conclusiones agente 2805_4 y modificaciones previo agente 2805_5:

Los resultados siguen sin ser prometedores y por ello cambia completamente el paradigma, simplificando el agente y el entorno.

Primero, se modifica la función de recompensa totalmente, simplificándola notablemente, eliminando la penalización por tiempo y las penalizaciones por retroceder o quedarse quieto.

La función de recompensa queda como:

- *Death penalty* : -100
- *Δx Reward*:
  - Avanzar: +1.0
- *Flag Reward*: +1000

Además, se simplifican las acciones que puede realizar el agente de manera drástica, pasando de movimiento *COMPLEX_MOVEMENT* a *RIGHT_ONLY.*

7. Conclusiones agente 2805_5 y modificaciones previo agente 2905_1:

A continuación, puede observarse el último *callback* de información del entrenamiento de 2805_5 y la gráfica de evolución de la recompensa.

rollout/	
ep_len_mean	187
ep_rew_mean	766
time/	
fps	109
iterations	245
time_elapsed	9196
total_timesteps	1003520
train/	
approx_kl	0.00010532548
clip_fraction	2.44e-05
clip_range	0.2
entropy_loss	-0.0324
explained_variance	0.97
learning_rate	2.88e-07
loss	45.6
n_updates	2440
policy_gradient_loss	-0.000201
value_loss	98.8

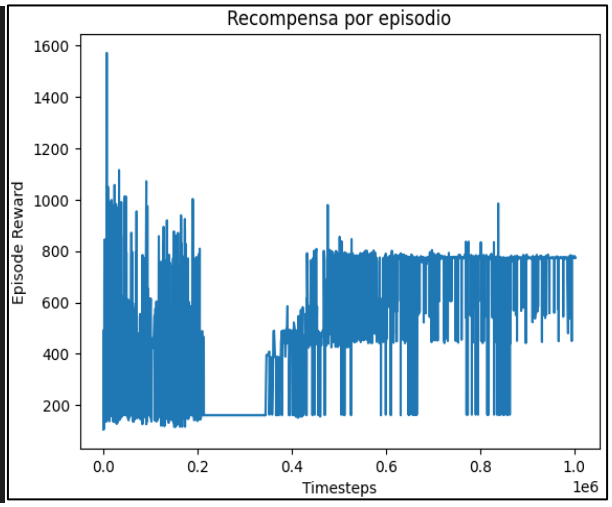


Ilustración 2. Resultado entrenamiento 2805_2

Como puede observarse, la política entrenada se ha estabilizado en un subóptimo, ya que se mantiene bastante constante en los valores obtenidos a partir del *timestep* ~8.5e6.

Se interpreta que debe suceder ya que la penalización por muerte no resultaba muy gravosa frente a la recompensa por avanzar.

Para corregir este error se mantiene la penalización por muerte, pero se disminuye recompensa por avanzar un orden de magnitud por debajo, para asegurar que tanto la muerte como llegar a la bandera son eventos significativos.

Además, se recompensa el conseguir puntuación, ya sea recolectando monedas o matando enemigos.

La función de recompensa queda:

- *Death penalty* : -100
- *Δx Reward*:
  - Avanzar: +1.0 → +0.1
- *Flag Reward*: +1000
- *Score reward*: Δscore * (+0.01)

En cuanto a parámetros de entrenamiento, se mantienen exactamente como el entrenamiento anterior, como puede visualizarse en la tabla.

8. Conclusiones agente 2905_1 y modificaciones previo agente 2905_2:

El último *output* del entrenamiento de 2905_1 arroja la siguiente información:

	--	**ep_len_mean:** El agente sobrevive o avanza mucho más tiempo que en entrenamientos anteriores. Está aprendiendo una política más duradera. **ep_rew_mean:** La recompensa media es moderada. Si llegar a la meta otorga +1000, este valor sugiere que aún no llega regularmente, pero explora bien y avanza. **approx_kl, entropy_loss, learning_rate**: Valores esperados dado que es el último *callback, y* se entiende que ya ha acabado de actualizar políticas y aprender. **explained_variance:** La red de valores predice con gran precisión las recompensas. **policy_gradient_loss, loss, value_loss:** El modelo ha convergido.
	| rollout/ | |	
	| ep_len_mean | 580 |	
	| ep_rew_mean | 107 |	
	| time/ | |	
	| fps | 111 |	
	| iterations | 245 |	
	| time_elapsed | 8971 |	
	| total_timesteps | 1003520 |	
	| train/ | |	
	| approx_kl | 4.3602297e-05 |	
	| clip_fraction | 0 |	
	| clip_range | 0.2 |	
	| entropy_loss | -0.285 |	
	| explained_variance | 0.995 |	
	| learning_rate | 2.88e-07 |	
	| loss | 0.288 |	
	| n_updates | 2440 |	
	| policy_gradient_loss | -0.000129 |	
	| value_loss | 0.988 |	
	--	

Además, en la siguiente gráfica de evolución de la recompensa puede observarse que el agente ha tenido varios picos de recompensa altos y posteriormente se ha estabilizado.

Podemos inferir que el agente ha estado en varias ocasiones cerca de alcanzar el final del nivel ya que la recompensa en esos picos es alta y la media de longitud episodio también.

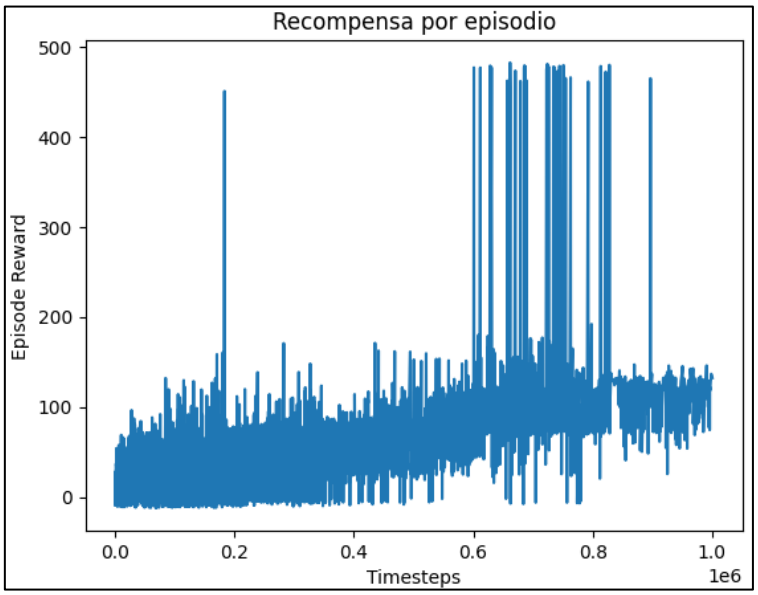


Ilustración 3. Evolución recompensa prueba 2905_1

Se observa que 2905_1 mejora el comportamiento, siendo el mejor desempeño desde el agente 2705_2, pero sigue muriendo más adelante, de manera similar a dicha prueba, por lo que se intenta penalizar más la muerte del agente antes de alcanzar el final del nivel.

Se aumenta penalización por muerte

- *Death penalty* : -100 → -500

Los demás parámetros quedan iguales al agente 2905_1.

9. Conclusiones agente 2905_2 y modificaciones previo agente 2905_3:

El agente se queda atrapado en un tubo ya que en la función de recompensa no se penaliza el quedarse quieto y, al aumentar la penalización por morir, los mejores desempeños para el agente se encuentran cuando este no muere, pero ha avanzado un tramo durante el nivel (matando un *Goomba* por el camino y aumentando su *score* también).

Para paliar el que el agente tome la decisión de mantenerse atrapado se añadió una penalización de -0.05 por no mover el personaje.

La función de recompensa queda como:

- *Death penalty* : -500
- *Δx Reward*:
  - Avanzar: +1.0 → +0.1
  - No avanzar: -0.05
- *Flag Reward*: +1000
- *Score reward*: Δscore * (+0.01)

10. Conclusiones agente 2905_3 y modificaciones previo agente 2905_4:

El resultado del entrenamiento del agente 2905_3 resulta bastante prometedor, ya que consigue superar los *Goombas* que en los entrenamientos de 2705_1 y 2805_2 suponían la muerte de los agentes. Se considera el mejor modelo hasta la fecha.

A continuación, se puede observar el último *callback* de información y la gráfica de recompensas por episodio:

rollout/	
ep_len_mean	431
ep_rew_mean	155
time/	
fps	103
iterations	245
time_elapsed	9704
total_timesteps	1003520
train/	
approx_kl	0.000683444
clip_fraction	0.000952
clip_range	0.2
entropy_loss	-0.273
explained_variance	0.922
learning_rate	2.88e-07
loss	60.8
n_updates	2440
policy_gradient_loss	-0.000173
value_loss	128

ep_len_mean: El agente sobrevive o avanza durante bastante tiempo. Está aprendiendo una política más duradera.
ep_rew_mean: La recompensa media es baja. Si se observa la gráfica puede observarse que, durante una gran parte del entrenamiento, la recompensa se movía en los números negativos.
approx_kl, entropy_loss, learning_rate: Valores esperados dado que es el último *callback, y* se entiende que ya ha acabado de actualizar políticas y aprender.
explained_variance: La red de valores tiene un gran acierto.
policy_gradient_loss, loss, value_loss: Se observa con callbacks anteriores que los valores oscilan. Recompensas muy oscilantes.

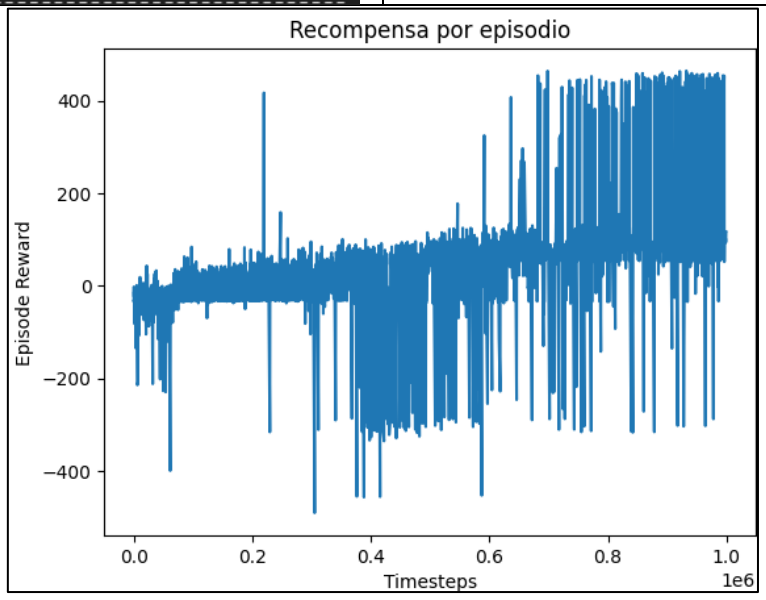


Ilustración 4. Recompensa por episodio para entrenamiento 2905_3.

Dado que se penaliza el no avanzar hacia adelante, el agente puede inferir que realizar saltos (desplazamiento en Y) es negativo, por lo que no es capaz de superar ciertos obstáculos. Esto es lo que puede haber penalizado gravemente al agente durante la fase intermedia del entrenamiento.

Para solventar este problema, pero que el agente siga entendiendo que quedarse quieto no es una opción correcta, se va a penalizar el tiempo transcurrido y se reduce la penalización por no desplazamiento en el eje X. De esta manera, los episodios donde el agente se quede atascado penalizarán igualmente, pero no se penalizará tanto directamente el hecho de no avanzar en el eje X.

Se va a reducir la penalización por morir para permitir que el agente explore situaciones más arriesgadas.

La función de recompensa queda como:

- *Death penalty* : -500 → -100
- *Δx Reward*:
  - Avanzar: +1.0 → +0.1
  - No avanzar: -0.05 → -0.01
- *Time penalty*: -(Δt * 0.0005)
- *Flag Reward*: +1000
- *Score reward*: Δscore * (+0.01)

Además, se aumenta el número de pasos, el *learning rate* de partida y el número de entornos implicados ya que se pretende aumentar la capacidad de aprendizaje del agente, al haber introducido una penalización nueva.

11. Conclusiones agente 2905_4 y modificaciones previo agente 2905_5:

El resultado observado empeora considerablemente al entrenamiento de 2905_4, por lo que se descartan los cambios anteriores fruto de una suposición errónea o, posiblemente, que dicha suposición no estuviese afectando tan gravemente al desempeño del agente. Se elimina la penalización por tiempo transcurrido ya que no había aportado una mejora significativa.

Revisando la función de recompensa, y pidiendo a **ChatGPT** que sugiera alguna mejora para poder aplicar, el LLM sugiere modificar cómo se estaba considerando que el agente estaba muriendo, y la recompensa asociada a este evento, así como la recompensa por avanzar.

En lugar de recompensar por acción de avance, se prueba a recompensar por la distancia avanzada.

La penalización por morir se aplica al descontar el número de vidas del agente con el número de vidas totales.

La función de recompensa queda:

- *Death penalty* : -100 por muerte
- *Δx Reward*:
  - Avanzar: Δx * (+0.1)
  - No avanzar: -0.05
- *Flag Reward*: +1000
- *Score reward*: Δscore * (+0.01)

12. Conclusiones agente 2905_5 y modificaciones previo agente 3005_1:

El agente 2905_5 obtiene le mejor resultado hasta el momento, quedándose Mario atascado en el último tramo antes de alcanzar la bandera.

Además, según se puede observar en la gráfica de recompensa por episodio el agente ha conseguido alcanzar la meta en varias ocasiones, los picos de +1000:

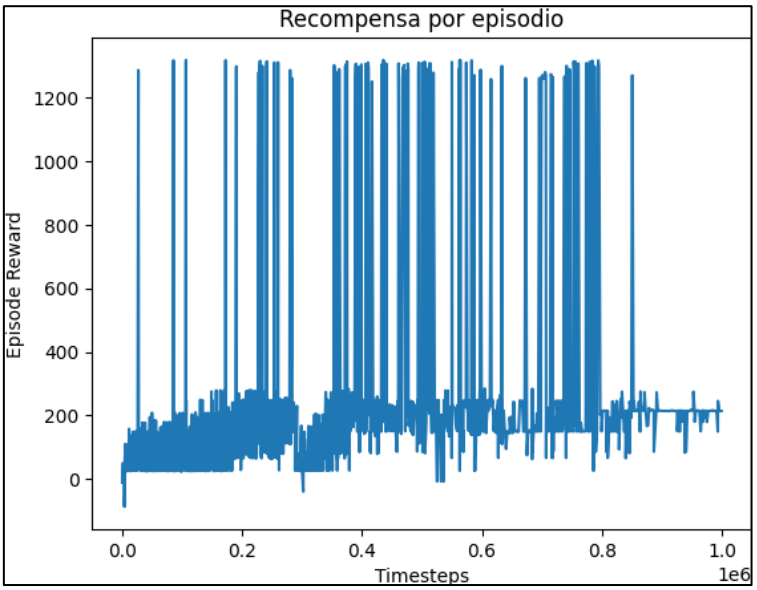


Ilustración 5. Output del entrenamiento 2905_5.

El hecho de que posteriormente no consiga alcanzar la meta indica que no ha afianzado la mejor política que le llevaba al éxito y finalmente ha caído en un subóptimo muy por debajo de lo deseable, aunque estabilizando la política. Esto puede deberse al ruido de la recompensa o a que el valor de *learning rate* no es el óptimo.

Se va a probar a reducir el ruido de la señal de recompensa para que pueda ser interpretada mejor por el agente, reduciendo los picos introducidos por la señal de alcanzar bandera y los picos producidos por muerte del agente:

- *Death penalty* : -100 → -50
- *Δx Reward*:
    - No avanzar: -0.05 → 0
- *Flag Reward*: +1000 → +500

También se reduce el *learning rate* de partida para que deje de explorar y explote las mejores opciones antes, ya que ha conseguido alcanzar la meta en múltiples ocasiones durante el entrenamiento. Es posible que esta decisión haga caer al agente en subóptimos antes de alcanzar la meta, pero primero se pretende observar si el agente es capaz de afianzar la política con una señal de recompensa menos ruidosa.

13. Conclusiones agente 3005_1 y modificaciones previo agente 3005_2:

El resultado obtenido por el agente 3005_1 es mejor que la media de las pruebas anteriores pero peor que en el agente 2905_5 ya que se queda atascado antes.

El *output* del último *callback* de información es el siguiente:

rollout/	
ep_len_mean	839
ep_rew_mean	405
time/	
fps	108
iterations	245
time_elapsed	9225
total_timesteps	1003520
train/	
approx_kl	0.00011346885
clip_fraction	4.88e-05
clip_range	0.2
entropy_loss	-0.151
explained_variance	0.954
learning_rate	2.02e-07
loss	257
n_updates	2440
policy_gradient_loss	-3.73e-05
value_loss	709

ep_len_mean: El agente sobrevive o avanza mucho más tiempo, también puede indicar que está atascado.
ep_rew_mean: Recompensa media alta, está llegando lejos y alcanzando la meta en algunos momentos.
approx_kl, entropy_loss, learning_rate: Valores esperados dado que es el último *callback, y* se entiende que ya ha acabado de actualizar políticas y aprender.
explained_variance: La red de valores tiene un gran índice de predicción.
policy_gradient_loss, loss, value_loss: El modelo ha convergido. Comparando con *outputs* anteriores, puede verse que la política no está estabilizada del todo puesto que el *value_loss* oscila bastante.

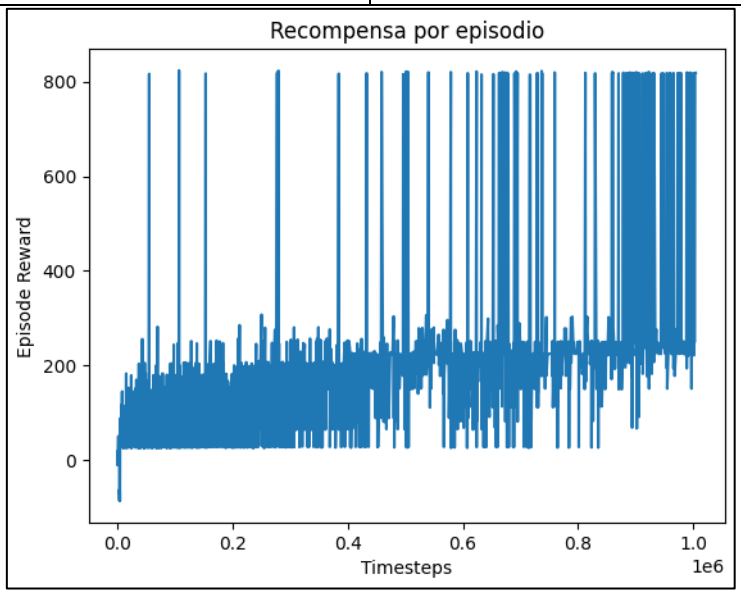


Ilustración 6. Recompensa por episodio entrenamiento 3005_1.

Del *output* final y la gráfica de Recompensa por episodio se puede inferir que, aunque la política es bastante exitosa, no consigue estabilizarse alcanzando la bandera. Esto puede deberse a diversos motivos o la conjunción de varios de ellos:

- Señal de recompensa ruidosa, recompensa por alcanzar la bandera muy grande, desestabiliza.
- *Learning rate* debe afinarse un poco más.
- Se debe penalizar más el quedarse atascado.

Para poder determinar que factores están influyendo más en el comportamiento del agente, se va a probar a aumentar el *learning rate* sustancialmente para permitir que explore más acciones al principio que le lleven a la bandera antes, como lograba el agente 2905_5.

El resto de los factores a monitorizar, recompensa por alcanzar la bandera y penalización por quedarse quieto, no se van a modificar para poder analizar aisladamente que efecto tiene el aumento del *learning rate*.

14. Conclusiones agente 3005_2 y modificaciones previo agente 3005_3:

El resultado del agente 3005_3 es similar al entrenamiento anterior, Mario se queda atascado en una zona similar y no consigue sobrepasar el obstáculo.

Con todo, se puede observar que el efecto predicho para el aumento del *learning rate* se ha cumplido, aumentando la densidad de episodios que alcanza la bandera al principio del entrenamiento:

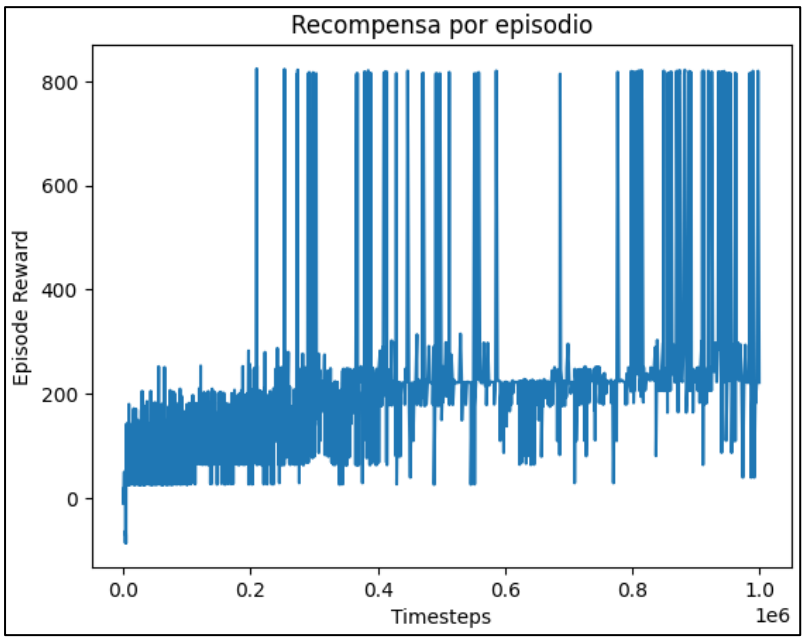


Ilustración 7. Recompensa por episodio del entrenamiento 3005_2.

Aun así, no se ha conseguido afianzar la política de manera consistente para alcanzar la bandera, por lo que se va a intentar suavizar la señal de recompensa todavía más para favorecer que el agente pueda estabilizar más fácilmente la política correcta. Se disminuye la recompensa por alcanzar la bandera mientras que el resto quedan igual:

- *Death penalty* : -50
- *Δx Reward*:
  - Avanzar: Δx * (+0.1)
  - No avanzar: 0.0
- *Flag Reward*: +500 → +100
- *Score reward*: Δscore * (+0.01)

Además del cambio en la función de recompensa, se va a reducir el *learning rate* a un punto intermedio entre los empleados en los agentes 3005_1 y 3005_2. Además, este valor intermedio se hace coincidir con el valor empleado en el entrenamiento con mejor resultado hasta la fecha, el agente 2905_5.

15. Conclusiones agente 3005_3 y modificaciones previo agente 3005_4:

Finalmente, se consigue el comportamiento óptimo, Mario consigue avanzar hasta alcanzar la bandera. El resultado puede observarse en forma de GIF al inicio del Notebook entregado.

El *output* final del entrenamiento es el siguiente:

rollout/	
ep_len_mean	277
ep_rew_mean	405
time/	
fps	107
iterations	245
time_elapsed	9365
total_timesteps	1003520
train/	
approx_kl	3.052807e-05
clip_fraction	0
clip_range	0.2
entropy_loss	-0.136
explained_variance	0.964
learning_rate	2.88e-07
loss	0.485
n_updates	2440
policy_gradient_loss	-0.000359
value_loss	0.838

ep_len_mean: El agente sobrevive o avanza durante un tiempo más pequeño que en anteriores pruebas. Puede indicar muerte prematura o que no se queda atascado.
ep_rew_mean: Recompensa media alta, teniendo en cuenta que se ha reducido la recompensa por alcanzar la bandera. Indica que el agente tiene mucho éxito.
approx_kl, entropy_loss, learning_rate: Valores esperados dado que es el último *callback, y* se entiende que ya ha acabado de actualizar políticas y aprender.
explained_variance: La red predice con gran éxito las recompensas.
policy_gradient_loss, loss, value_loss: El modelo ha convergido correctamente, el *value_loss* es muy reducido, indicando que la política está estabilizada.

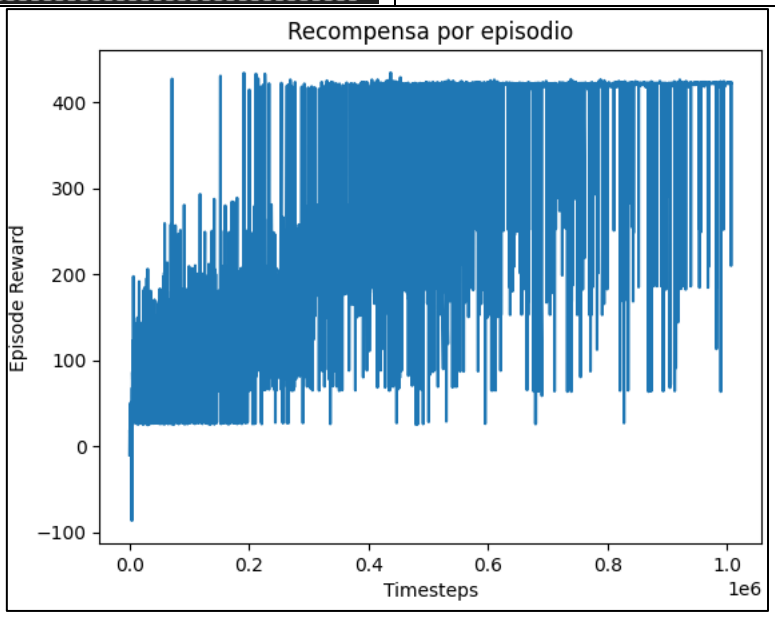


Ilustración 8. Recompensa por episodio para el entrenamiento 3005_3.

Como puede observarse en la gráfica, el agente tiene un índice de éxito en una cantidad alta de episodios. La densidad de episodios donde se alcanza la recompensa máxima es muy alta, estabilizándose hacia el final.

Aun así, hay muchos episodios en los que dicha recompensa cae considerablemente, probablemente debido a que hacia el final del episodio, aunque ha conseguido puntos, se ha quedado atascado en algún obstáculo.

Se confirma que la suposición de reducir el ruido de la señal de recompensa, otorgando una recompensa menor al alcanzar la bandera, favorece la estabilidad del agente, consiguiendo que sea capaz de reproducir y explotar las mejores opciones.

La reducción del *learning rate* a un valor intermedio, 5e-4, permite que el agente comience a explotar las mejores opciones antes y, por lo tanto, refinar el comportamiento óptimo más rápidamente.

Para mejorar la política, se va a realizar un último entrenamiento para conseguir una política más refinada, reduciendo el *learning rate* para permitir que el agente explote más, y aumentando el número de *timesteps* a 1.5e6 para darle más tiempo de mejora.

Además, se vuelve a incluir la penalización por no avanzar, para que el agente no explore los escenarios en los que consigue una alta puntuación por llegar lejos, pero se queda atascado al final, ya que terminarán penalizando negativamente.

16. Conclusiones agente 3005_4:

Tras ejecutar el entrenamiento, se observa que la política ha tenido una mayor estabilidad hacia el final, pero siguen apareciendo episodios esporádicos en los que el agente explora opciones subóptimas. Además, puede verse claramente donde el agente comenzó a afianzar la política exitosa, cerca de los 0.4e6 *timesteps:*

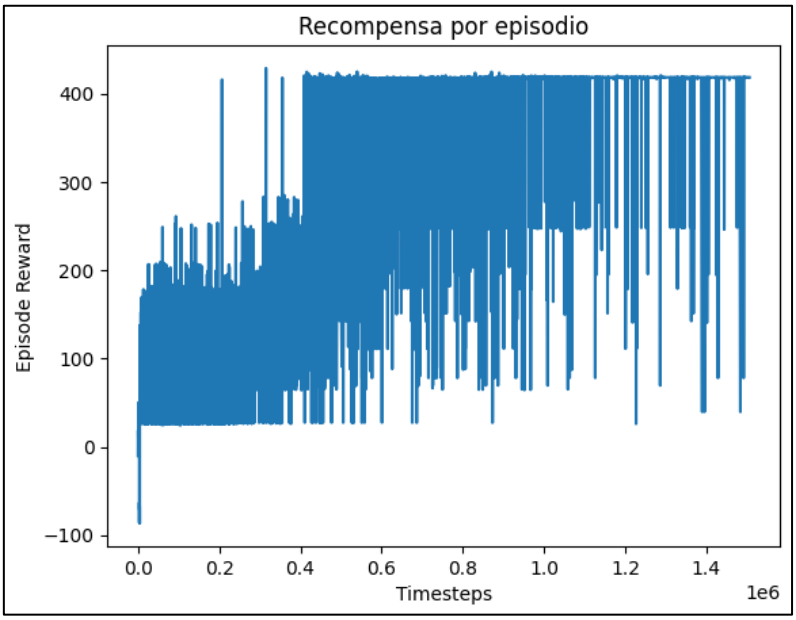
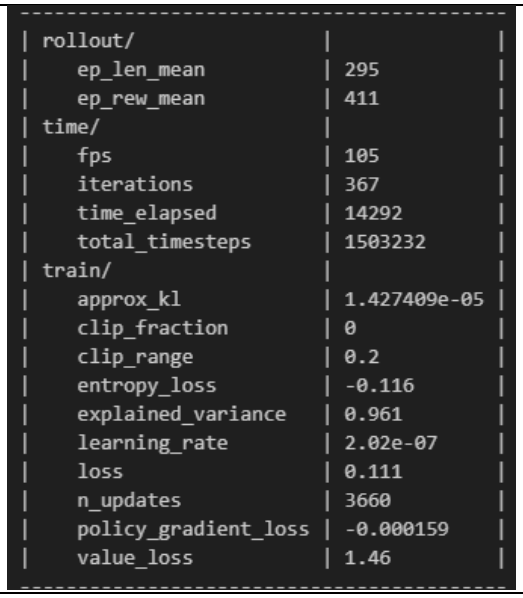


Ilustración 9. Recompensa por episodio entrenamiento 3005_4.

Se considera exitoso el entrenamiento, y tanto el agente de esta prueba, como el de la prueba anterior, 3005_3, se consideran dos agentes capaces de cumplir la tarea.

	**ep_len_mean:** El agente sobrevive o avanza durante un numero de pasos similar a la prueba anterior, 3005_3. **ep_rew_mean:** Recompensa media es ligeramente más alta que en el entrenamiento anterior. **approx_kl, entropy_loss, learning_rate**: Valores esperados dado que es el último *callback, y* se entiende que ya ha acabado de actualizar políticas y aprender. **explained_variance:** La red predice con gran éxito las recompensas. **policy_gradient_loss, loss, value_loss:** El modelo ha convergido correctamente, el *value_loss* es muy reducido, indicando que la política está estabilizada. Sin embargo, comparado con el entrenamiento anterior, tiene un poco más de *value_loss*.

En términos comparativos, tanto el entrenamiento 3005_3 como el 3005_4 alcanzan la meta, aunque a la hora de comparar ambos agentes, el agente 3005_3 obtiene mayor puntuación, pero unos segundos más en alcanzar la meta que 3005_4.

La función de recompensa finalmente ha quedado como:

$$\boldsymbol{R} = \boldsymbol{S} + \boldsymbol{X} + \boldsymbol{D} + \boldsymbol{F}$$

Donde:

$\boldsymbol{S}$: Recompensa por puntuación obtenida:

$$\boldsymbol{S} = +0.01 \cdot \Delta score$$

$\boldsymbol{X}$: Recompensa por desplazamiento horizontal:

$$\boldsymbol{X} = \begin{cases} +0.1 \cdot \Delta x, & si\ \Delta x > 0 \\ -0.01, & en\ otro\ caso \end{cases}$$

$\boldsymbol{D}$: Penalización por morir:

$$\boldsymbol{D} = \begin{cases} -50.0, & si\ v_{act} < v_{prev} \\ 0.0, & en\ otro\ caso \end{cases}$$

$\boldsymbol{F}$: Recompensa por alcanzar la bandera:

$$\boldsymbol{F} = +100.0 \cdot 1_{\{f=1\}}\ ^{1}, f \in \{0,1\}$$

[1] La función $1_{\{f=1\}}$ es la función indicadora que vale 1 si se cumple $f = 1$, y 0 en caso contrario. $\boldsymbol{f}$ es la variable booleana que indica si se ha alcanzado o no la bandera.

7. RESULTADOS Y CONCLUSIONES

En conclusión, se considera exitosa la realización de la práctica, consiguiendo entrenar dos agentes RL con capacidad de superar el primer nivel de *SuperMario Bros*.

Dado que se considera más relevante la puntuación que el tiempo transcurrido, ya que ambos alcanzan la bandera, el agente **3005_3** se considera ligeramente mejor que el agente **3005_4**, al menos en su desempeño en el entorno de evaluación.

Queda demostrado que la función de recompensa, y las ponderaciones otorgadas, resulta clave para el desempeño del agente, ya que en los primeros entrenamientos se utilizaba una función de recompensa muy compleja que introducía demasiada variabilidad a la señal y no favorecía el aprendizaje del agente. Cuando se simplificó dicha función de recompensa se consiguió mejorar el desempeño del agente. Se observa que otorgar una gran recompensa al éxito puede resultar contraproducente, ya que, al reducir la recompensa por alcanzar la bandera, se logró suavizar la señal de recompensa para que el agente pudiera aprender más fácilmente.

La evolución de mejora de los agentes entrenados ha seguido una tendencia ascendente, aun con algunos mínimos en etapas tardías del flujo de implementación y entrenamientos, como puede observarse en la siguiente gráfica (**Obtenida con ChatGPT**):

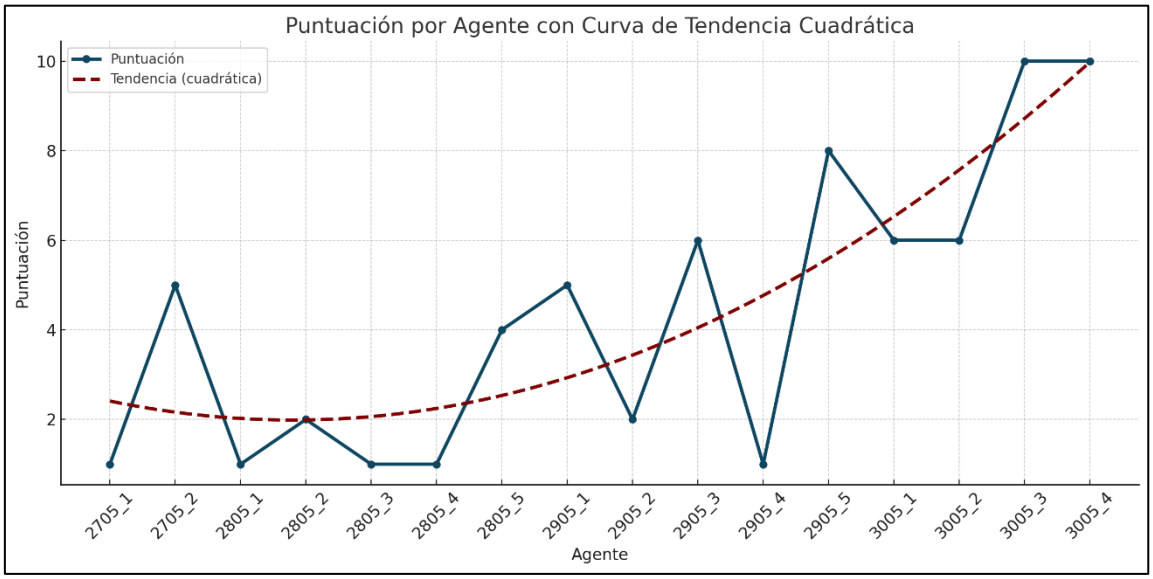


Ilustración 10. Puntuación otorgada a cada entrenamiento en función del rendimiento y la tendencia de mejora observada.

La comprensión obtenida por parte del alumno a lo largo del desarrollo del a práctica también ha ido incrementando notablemente, como puede verse reflejado en el diario de prácticas y en los éxitos obtenidos con los cambios realizados entre pruebas, mejorando notablemente hacia el final de la práctica. También debe mencionarse la mejora que supuso la introducción de gráficas que mostrasen información a lo largo del entrenamiento para mejorar las observaciones y modificaciones derivadas.

8. USO DE LLMs (CHATGPT) Y OTRAS FUENTES

Como se ha ido anotando a lo largo de la práctica se ha utilizado ChatGPT para obtener algunas sugerencias y bloques de código que utilizar como punto de partida. También se ha utilizado para obtener algunas definiciones y ampliaciones al conocimiento teórico.

Otro uso que se le ha dado a ChatGPT es la modificación visual del Notebook de Jupyter, añadiendo bloques HTML e iconos.

Juntamente con ChatGPT se han utilizado otras fuentes de consulta para contrastar la información y mejorar la comprensión, como la documentación oficial de StableBaselines 3, del entorno de gym-super-mario-bros y de la librería de Gymnasium:

- https://stable-baselines3.readthedocs.io/en/master/index.html
- https://github.com/Kautenja/gym-super-mario-bros/tree/master
- https://gymnasium.farama.org/

Se han utilizado también diferentes artículos en línea que resolvían problemas similares a esta práctica como inspiración y fuente de información adicional:

- https://medium.com/@eisukeh/mario-please-jump-an-informal-blog-detailing-my-journey-and-struggles-with-deep-reinforcement-a045cbd51a17
- https://medium.com/@eodor02/building-an-intelligent-mario-bot-using-reinforcement-learning-and-python-1b9ac63ba4be
- https://medium.com/@Yaga987/mastering-super-mario-bros-with-gym-and-ppo-a-deep-reinforcement-learning-approach-a74a5b44fffd
- https://sohum-padhye.medium.com/playing-super-mario-bros-with-reinforcement-learning-81ee0c235372
- https://medium.com/@darioprawarateh/playing-super-mario-with-ppo-reinforcement-learning-agent-e9f61fd04b32
- https://www.statworx.com/en/content-hub/blog/using-reinforcement-learning-to-play-super-mario-bros-on-nes-using-tensorflow
- https://docs.pytorch.org/tutorials/intermediate/mario_rl_tutorial.html
- https://paperswithcode.com/paper/super-reinforcement-bros-playing-super-mario
- https://www.sourish.dev/projects/super-mario-bros-rl/
- https://github.com/vietnh1009/Super-mario-bros-PPO-pytorch
- https://github.com/nikhilgrad/Reinforcement-Learning-Model-for-Super-Mario
- https://github.com/yumouwei/super-mario-bros-reinforcement-learning
- https://www.kaggle.com/code/aaryanpalit/mario-reinforcement-learning
- https://www.reddit.com/r/reinforcementlearning/comments/g6sg39/ppo_and_learning_rate/
- https://liorshilon.hashnode.dev/reinforcement-learning-ppo-vs-dqn-who-is-a-better-mario-player