Lunar Lander - IA (Provisional)

Ivo Raimondi
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
ivorai@alum.us.es HMF2475

Miguel Romero Mateos
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
migrommat@alum.us.es KPH6911

***Resumen*—Escribir aquí dos párrafos indicando el objetivo principal del trabajo, y un resumen de las conclusiones obtenidas. Cabe mencionar que este documento se ha confeccionado siguiendo el formato de conferencias de IEEE (ver la guía para autores para más información, existen plantillas para Word y LaTeX). Este documento se debe emplear como guía, se pueden añadir nuevas secciones según sea necesario. Es importante dotarlo de un número razonable de referencias bibliográficas.**

***Palabras clave*—Inteligencia Artificial, otras palabras clave...**

ÍNDICE GENERAL

I. Introducción Miguel

En esta sección se dedican varios párrafos para dar el contexto en el que se desarrolla el trabajo presentado. Normalmente se va desde lo más general a lo más preciso, para englobar el trabajo dentro de un ámbito concreto. Hay que recordar que se debe referenciar la información que se consiga de fuentes bibliográficas (artículos, libros, capítulos de libros, páginas web, apuntes, etc.). Para ello se añade el ítem correspondiente en el apartado de referencias, dotándolo de un número, y se añade la referencia al final de la frase o párrafo correspondiente [**?**]. Si se realiza en Microsoft Word o LibreOffice, se puede hacer uso de *referencias cruzadas* para actualizar la numeración de las referencias de forma automática en todo el documento.

Después vienen un par de párrafos para dar un poco más de detalle sobre el trabajo realizado. Hay que indicar la problemática o el objetivo que se marca, y cómo se ha enfocado la solución propuesta en este trabajo. Finalmente, el último párrafo se dedica a comentar la estructura del documento por secciones, como el que sigue.

II. Algoritmos Implementados - Miguel

debe incluir explicaciones, dificultades encontradas y las decisiones de diseño adoptadas

III. Experimentación

A continuación, se detallan todas las pruebas realizadas sobre nuestro algoritmo, explorando distintos ajustes en los hiperparámetros más relevantes para afinar nuestros modelos. Entre ellos se incluyen:

- **max_steps**: número máximo de pasos antes de finalizar un episodio.
- **hidden_size**: cantidad de neuronas en la capa oculta de las redes neuronales DQN. En nuestro caso siempre que hablemos de esto será de 2 capas del mismo tamaño.
- **replays_per_episode**: número de veces que se actualiza la red neuronal durante un solo episodio.
- **episodes**: cantidad total de episodios o simulaciones completas que el agente ejecuta durante el proceso de entrenamiento.
- **learning_rate**: tasa de aprendizaje, que determina el tamaño del paso durante la minimización de la función de pérdida.
- **batch_size**: número de experiencias (transiciones estado-acción-recompensa) utilizadas en cada iteración de entrenamiento.
- **memory_size**: capacidad máxima de la memoria de repetición (ReplayBuffer), donde se almacenan experiencias pasadas para el aprendizaje.
- **target_network_update_freq**: frecuencia, en número de episodios, con la que se actualizan los pesos de la red objetivo (target network) para igualarlos a los de la red principal (Q-network).

Ajustando adecuadamente estos hiperparámetros, podemos obtener un modelo más eficiente y efectivo. Existen varios métodos para optimizar estos valores, como la búsqueda en rejilla (*grid search*); sin embargo, dado que esta técnica es más adecuada cuando se trabaja con espacios discretos y que en nuestro caso sería computacionalmente costosa —ya que no contamos con recursos de alto rendimiento—, optamos por una estrategia más práctica: realizar pruebas aleatorias, pero

fundamentadas en criterios razonables y experiencia previa adquirida durante el curso e investigación.

Para no saturar nuestros equipos personales, realizamos las pruebas utilizando la plataforma Google Colab. Inicialmente, los experimentos fueron nombrados como *Colab X* (donde X es el número de prueba). Posteriormente, seleccionamos cinco modelos destacados, a los que asignamos nombres descriptivos según su comportamiento y desempeño.

Antes de analizar cada uno, debemos establecer cómo definimos si un episodio fue exitoso. El entorno de LunarLander proporciona recompensas intermedias a lo largo del episodio que dependen de múltiples factores:

- **Penalizaciones**: alejarse de la plataforma de aterrizaje, moverse demasiado rápido, inclinarse excesivamente, usar los motores (especialmente el principal), o estrellarse.
- **Recompensas**: mantenerse cerca del centro, aterrizar con suavidad, usar poca propulsión, apoyar las patas sin colisión y estabilizar la nave.

Considerando estas señales, definimos que un episodio exitoso es aquel cuya recompensa total es mayor o igual a **100 puntos**, ya que esto implica un aterrizaje controlado, eficiente y seguro, sin abusar de los motores ni desviarse significativamente de la plataforma.

A continuación, se presenta el análisis de los cinco modelos seleccionados, detallando sus configuraciones, el progreso observado durante el entrenamiento, los resultados obtenidos y una interpretación del comportamiento aprendido por el agente en cada caso.

A. ***Colab 1*** - *DopeyLander*

Estos son los hiperparámetros usados en este modelo:

```
#Los parametros más importantes son los marcados con un comentario
agent = DQNAgent(lunar,replays_per_episode=1000,epsilon=1.0, epsilon_decay=0.995, epsilon_min=0.01,
                 episodes=3000, #Episodios de entrenamiento
                 hidden_size=256 ,#Tamaño de la capa oculta
                 gamma=0.99, # Factor de descuento
                 learning_rate=0.001, #Tasa de aprendizaje
                 batch_size=64, #Tamaño minibatch
                 memory_size=10000, #Tamaño replay buffer
                 target_network_update_freq=15 #Actualizacion de la red objetivo
                 )
# agent.load_model("modelos/modelo_DQN.h5") <- si se quiere seguir entrenando el modelo anteriormente guardado
agent.train(nombre_archivo="modelos/modelo_DQN", save_graphs=True, save_every_50=True)
```

Fig. 1. Estos fueron los hiperparametros configurados para Colab 1

Este fue el primer experimento, y funcionó como línea base. Se utilizó una red con una capa oculta muy grande (256 neuronas en ambas capas ocultas), una tasa de aprendizaje alta y una enorme cantidad de actualizaciones por episodio (1000 replays).

A continuación, se muestra una imagen del entorno de trabajo en Google Colab durante el entrenamiento:

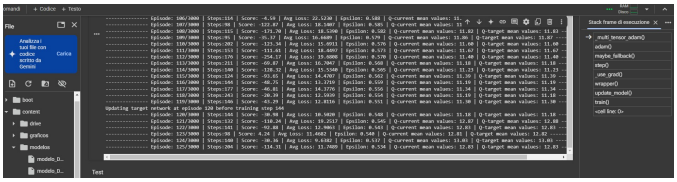


Fig. 2. Progreso durante el entrenamiento - Colab 1

Aunque el agente aprendió que debía encender el motor principal al principio para no caer directamente, nunca logró estabilizarse ni aterrizar correctamente. El entrenamiento fue extremadamente costoso (3000 episodios con 1000 replays cada uno), y no se observaron mejoras significativas a medida que disminuía el valor de epsilon.

Los siguientes gráficos muestran el rendimiento del modelo durante su entrenamiento:

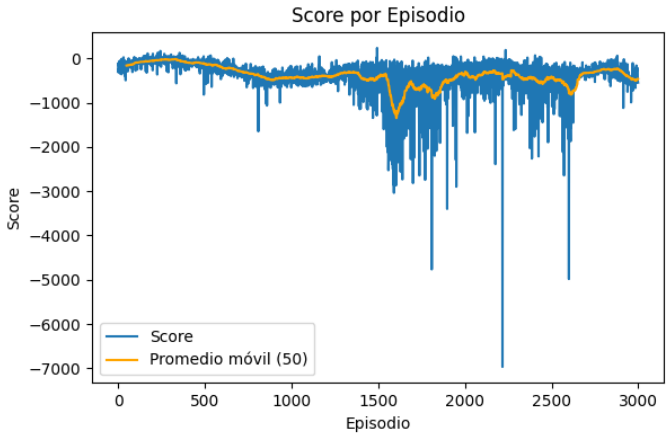


Fig. 3. Recompensa total con promedio móvil de los episodios - Colab 1

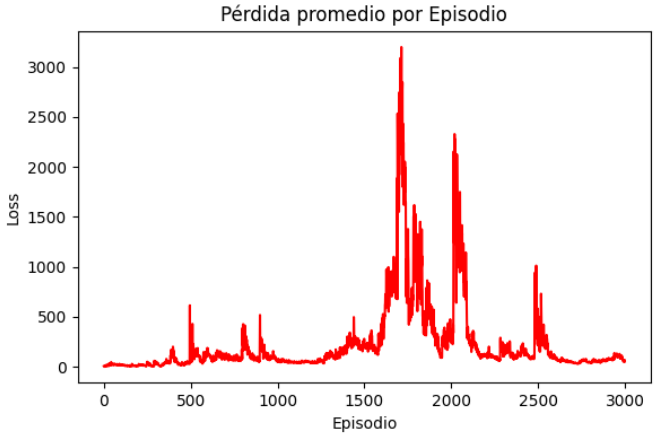


Fig. 4. Pérdida promedio por episodio - Colab 1

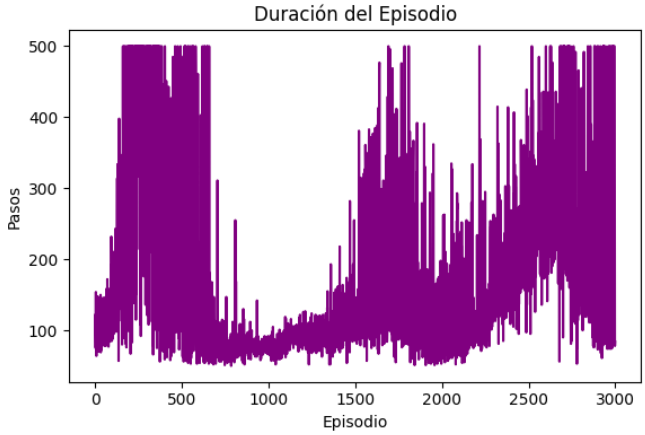


Fig. 5. Duración de los episodios - Colab 1

Como puede observarse, el modelo nunca logró especializarse. Mantuvo recompensas bajas durante todo el entrenamiento, su pérdida promedio fue constantemente elevada y los episodios caóticos.

La siguiente figura muestra la tasa de acierto una vez finalizado el entrenamiento usando $\epsilon = 0.0$, es decir, sin exploración:

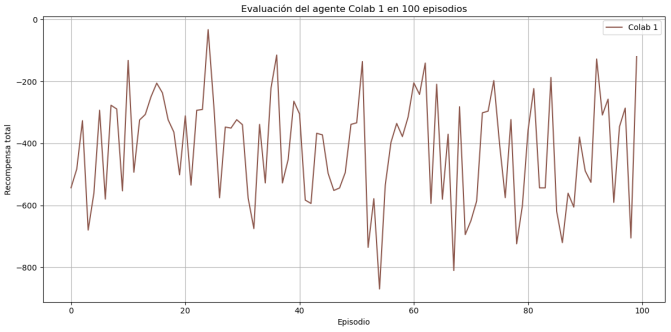
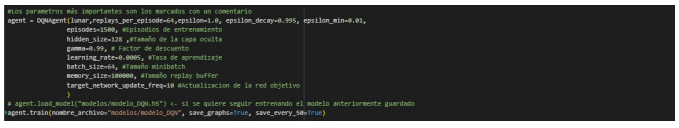


Fig. 6. Tasa de acierto de Colab 1 con el modelo ya entrenado - Colab 1

El resultado fue un rendimiento nulo, con una tasa de acierto del **0.00%**.

*B. **Colab 2** - First Lander*

Estos son los hiperparámetros usados en este modelo:



```
#Los parametros más importantes son los marcados con un comentario
agent = DQNAgent(lunar,replays_per_episode=64,epsilon=1.0, epsilon_decay=0.995, epsilon_min=0.01,
                 episodes=1500, #Episodios de entrenamiento
                 hidden_size=128 ,#Tamaño de la capa oculta
                 gamma=0.99, # Factor de descuento
                 learning_rate=0.0005, #Tasa de aprendizaje
                 batch_size=64, #Tamaño minibatch
                 memory_size=100000, #Tamaño replay buffer
                 target_network_update_freq=10 #Actualizacion de la red objetivo
                 )
# agent.load_model("modelos/modelo_DQN.h5") <- si se quiere seguir entrenando el modelo anteriormente guardado
agent.train(nombre_archivo="modelos/modelo_DQN", save_graphs=True, save_every_50=True)
```

Fig. 7. Hiperparámetros usados en Colab 2

Este modelo representó una mejora significativa respecto al anterior. Se redujo el tamaño de la red a 128 neuronas en ambas capas ocultas, se utilizaron solo 64 replays por episodio y la tasa de aprendizaje se disminuyó a 0.0005. Esto favoreció la estabilidad del entrenamiento y permitió un aprendizaje más fino.

Podemos ver un poco de cómo avanzó su aprendizaje:

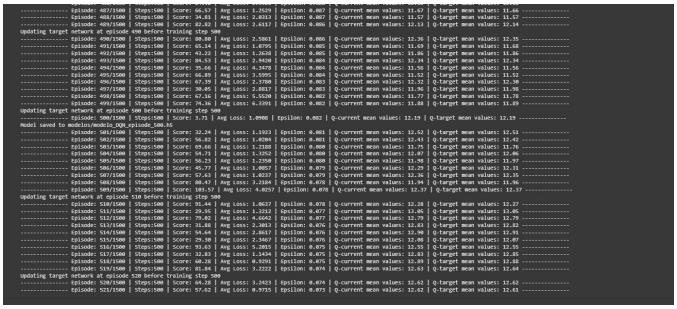


Fig. 8. Progreso durante el entrenamiento - Colab 2

El agente aprendió a dejarse caer al inicio y luego usar el propulsor principal para corregir su trayectoria, estabilizándose con los motores laterales y logrando aterrizajes suaves. El entrenamiento se realizó en 1500 episodios, lo que permitió un buen aprovechamiento de la fase de exploración.

Con los siguientes gráficos se puede observar cómo fue evolucionando el entrenamiento:

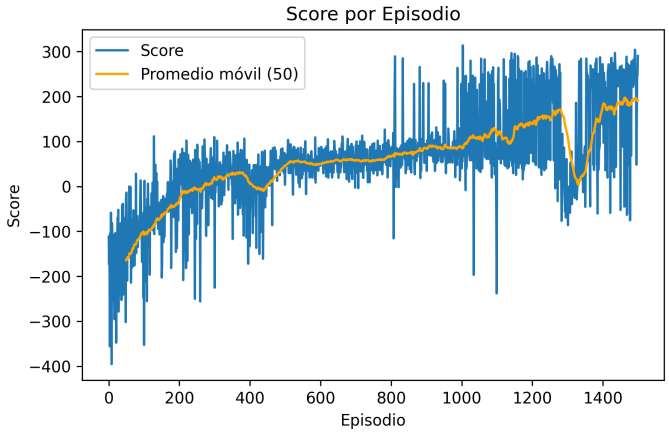


Fig. 9. Recompensa total con promedio móvil - Colab 2

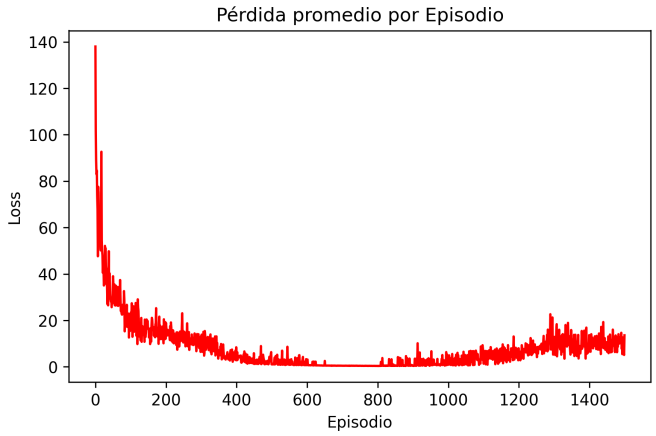


Fig. 10. Pérdida promedio por episodio - Colab 2

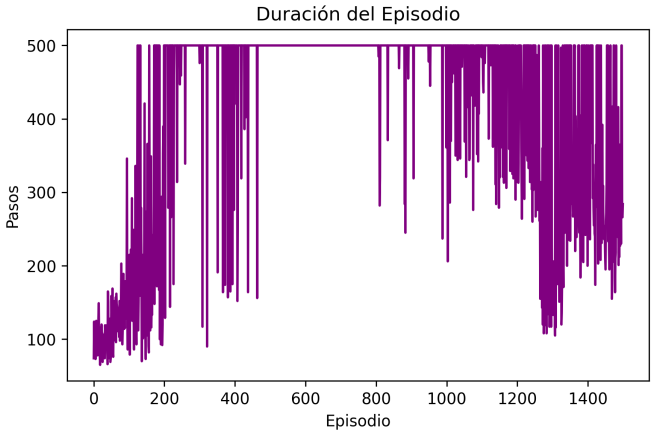


Fig. 11. Duración de los episodios - Colab 2

La evolución de la recompensa muestra una clara tendencia positiva, con una pérdida que se estabiliza gradualmente y una duración de episodios que se ajusta conforme mejora la política aprendida. Estos indicadores reflejan una buena capacidad de generalización y eficiencia en el aprendizaje.

Un aspecto llamativo es la caída abrupta en la recompensa alrededor del episodio 1350 (entre otras que tiene). Esto podría explicarse por el uso de experiencias poco representativas o negativas almacenadas en el replay buffer, que afectaron momentáneamente el proceso de actualización del modelo. Este comportamiento resalta que un mayor número de episodios no garantiza por sí solo un mejor desempeño, especialmente si las muestras recientes no son de buena calidad.

Veremos ahora cómo evolucionó la tasa de acierto del modelo entrenado:

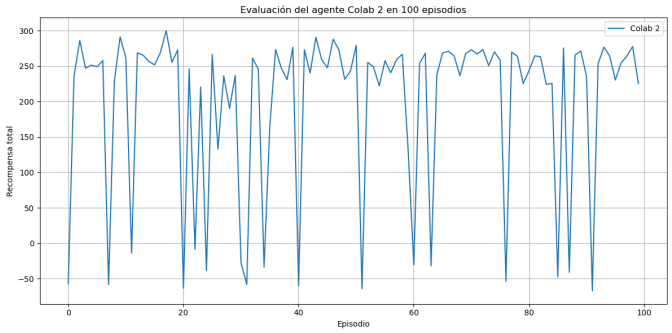


Fig. 12. Tasa de acierto del modelo ya entrenado - Colab 2

Este modelo logró una tasa de acierto del **81.00%**, posicionándose como uno de los más estables y eficientes del conjunto.

*C. **Colab 3** - NO-Lander*

Estos son los hiperparámetros usados en este modelo:

```
#Los parametros más importantes son los marcados con un comentario
agent = DQNAgent(lunar,replays_per_episode=64,epsilon=1.0, epsilon_decay=0.995, epsilon_min=0.01,
                 episodes=1500, #Episodios de entrenamiento
                 hidden_size=128 ,#Tamaño de la capa oculta
                 gamma=0.99, # Factor de descuento
                 learning_rate=0.0005, #Tasa de aprendizaje
                 batch_size=128, #Tamaño minibatch
                 memory_size=100000, #Tamaño replay buffer
                 target_network_update_freq=10 #Actualizacion de la red objetivo
                 )
# agent.load_model("modelos/modelo_DQN.h5") <- si se quiere seguir entrenando el modelo anteriormente guardado
agent.train(nombre_archivo="modelos/modelo_DQN", save_graphs=True, save_every_50=True)
```

Fig. 13. Hiperparámetros usados en Colab 3

Este modelo mostró un comportamiento peculiar. El agente flotaba durante largos períodos y evitaba aterrizar, por mantener el motor principal encendido demasiado tiempo. La mayoría de los episodios se truncaban al alcanzar el límite de pasos (500), priorizando evitar castigos antes que buscar la recompensa de aterrizaje.

A continuación, se puede observar el progreso general del modelo durante su entrenamiento:

```
Episodio 1245 | Steps: 500 | Score:   80.47 | Avg Loss: 0.1603 | Epsilon: 0.010 | Q-current mean values: 17.04 | Q-target mean values: 17.04
Episodio 1246 | Steps: 500 | Score:   32.28 | Avg Loss: 0.1607 | Epsilon: 0.010 | Q-current mean values: 17.33 | Q-target mean values: 17.33
Episodio 1247 | Steps: 500 | Score:   37.80 | Avg Loss: 0.1624 | Epsilon: 0.010 | Q-current mean values: 16.96 | Q-target mean values: 16.96
Episodio 1248 | Steps: 500 | Score:  106.85 | Avg Loss: 0.1624 | Epsilon: 0.010 | Q-current mean values: 16.34 | Q-target mean values: 16.34
Episodio 1249 | Steps: 500 | Score:   32.13 | Avg Loss: 0.1576 | Epsilon: 0.010 | Q-current mean values: 16.46 | Q-target mean values: 16.46
Episodio 1250 | Steps: 500 | Score:   67.50 | Avg Loss: 0.1640 | Epsilon: 0.010 | Q-current mean values: 17.20 | Q-target mean values: 17.20
Episodio 1251 | Steps: 500 | Score:   92.94 | Avg Loss: 0.1751 | Epsilon: 0.010 | Q-current mean values: 17.21 | Q-target mean values: 17.21
Episodio 1252 | Steps: 500 | Score:   70.27 | Avg Loss: 0.1602 | Epsilon: 0.010 | Q-current mean values: 16.95 | Q-target mean values: 16.94
Episodio 1253 | Steps: 500 | Score:   37.94 | Avg Loss: 0.1511 | Epsilon: 0.010 | Q-current mean values: 16.28 | Q-target mean values: 16.29
Episodio 1254 | Steps: 500 | Score:   38.05 | Avg Loss: 0.1627 | Epsilon: 0.010 | Q-current mean values: 17.24 | Q-target mean values: 17.23
Episodio 1255 | Steps: 500 | Score:   51.45 | Avg Loss: 0.1591 | Epsilon: 0.010 | Q-current mean values: 17.11 | Q-target mean values: 17.11
Episodio 1256 | Steps: 500 | Score:   83.38 | Avg Loss: 0.1587 | Epsilon: 0.010 | Q-current mean values: 16.08 | Q-target mean values: 16.09
Episodio 1257 | Steps: 500 | Score:   74.79 | Avg Loss: 0.1548 | Epsilon: 0.010 | Q-current mean values: 16.79 | Q-target mean values: 16.79
Episodio 1258 | Steps: 500 | Score:   96.56 | Avg Loss: 0.1597 | Epsilon: 0.010 | Q-current mean values: 17.20 | Q-target mean values: 17.20
Episodio 1259 | Steps: 500 | Score:   70.25 | Avg Loss: 0.1644 | Epsilon: 0.010 | Q-current mean values: 17.37 | Q-target mean values: 17.37
Episodio 1260 | Steps: 500 | Score:   75.02 | Avg Loss: 0.1543 | Epsilon: 0.010 | Q-current mean values: 16.81 | Q-target mean values: 16.81
Episodio 1261 | Steps: 500 | Score:   53.83 | Avg Loss: 0.1572 | Epsilon: 0.010 | Q-current mean values: 17.09 | Q-target mean values: 17.09
Episodio 1262 | Steps: 500 | Score:   73.85 | Avg Loss: 0.1600 | Epsilon: 0.010 | Q-current mean values: 17.24 | Q-target mean values: 17.24
Episodio 1263 | Steps: 500 | Score:   92.79 | Avg Loss: 0.1551 | Epsilon: 0.010 | Q-current mean values: 17.19 | Q-target mean values: 17.19
Episodio 1264 | Steps: 500 | Score:   56.31 | Avg Loss: 0.1564 | Epsilon: 0.010 | Q-current mean values: 17.33 | Q-target mean values: 17.33
Episodio 1265 | Steps: 500 | Score:   91.36 | Avg Loss: 0.1523 | Epsilon: 0.010 | Q-current mean values: 16.38 | Q-target mean values: 16.38
Episodio 1266 | Steps: 500 | Score:   92.31 | Avg Loss: 0.1605 | Epsilon: 0.010 | Q-current mean values: 17.04 | Q-target mean values: 17.04
Episodio 1267 | Steps: 500 | Score:   76.92 | Avg Loss: 0.1575 | Epsilon: 0.010 | Q-current mean values: 17.24 | Q-target mean values: 17.24
Episodio 1268 | Steps: 500 | Score:   77.84 | Avg Loss: 0.1656 | Epsilon: 0.010 | Q-current mean values: 17.22 | Q-target mean values: 17.22
```

Fig. 14. Progreso durante el entrenamiento - Colab 3

A pesar de su política evasiva, el entrenamiento fue relativamente estable. Se utilizó un batch de tamaño mayor (128), lo que ralentizó el aprendizaje, pero permitió actualizaciones más suaves.

Los siguientes gráficos permiten analizar su entrenamiento con mayor detalle:

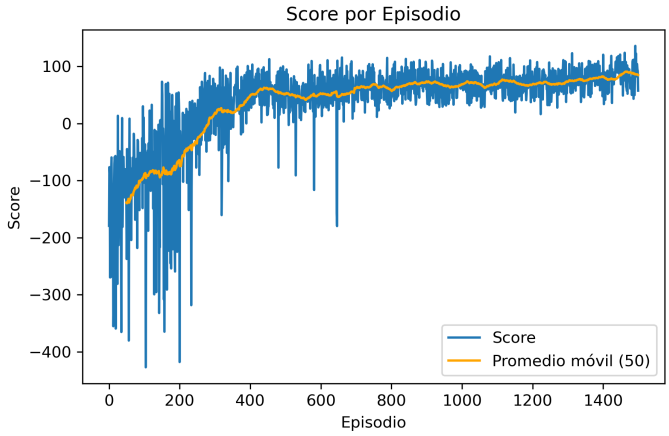


Fig. 15. Recompensa total con promedio móvil - Colab 3

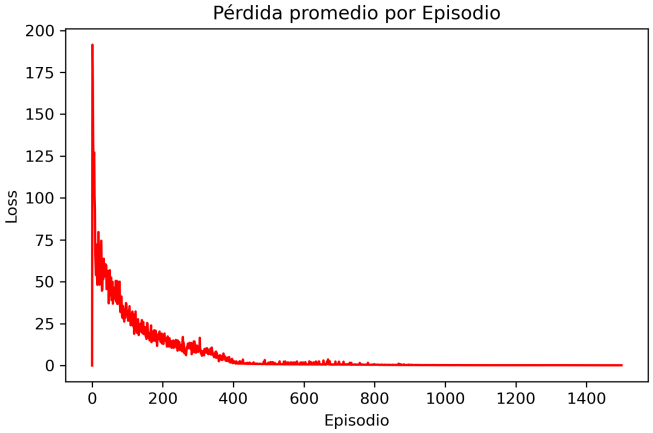


Fig. 16. Pérdida promedio por episodio - Colab 3

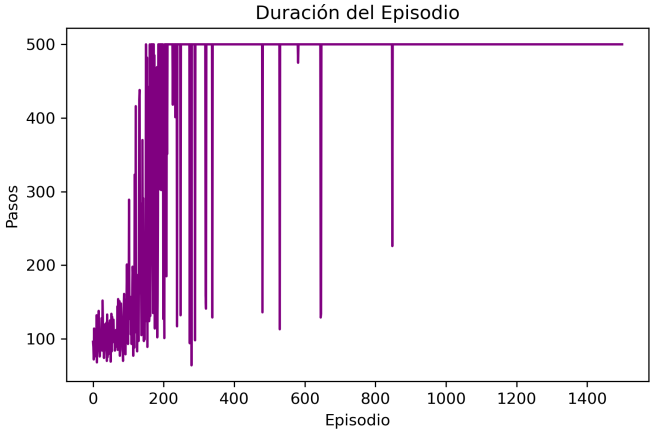


Fig. 17. Duración de los episodios - Colab 3

Aunque la recompensa y la pérdida muestran cierta mejora, el aumento sostenido en la duración de los episodios sugiere que el agente aprendió a evitar terminar la tarea, en lugar de completarla de forma óptima.

Finalizado el entrenamiento, se evaluó la política aprendida sin exploración:

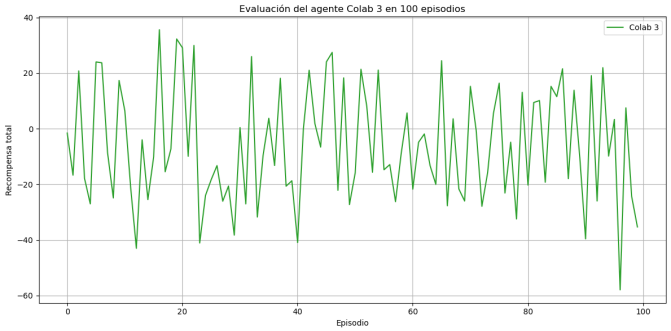


Fig. 18. Tasa de acierto del modelo ya entrenado - Colab 3

Este modelo no logró aterrizar correctamente en ninguna prueba final, obteniendo una tasa de acierto del **0.00%**.

*D. **Colab 4** - The King Lander*

Estos son los hiperparámetros usados en este modelo:

```
#Los parametros más importantes son los marcados con un comentario
agent = DQNAgent(lunar,replays_per_episode=32,epsilon=1.0, epsilon_decay=0.995, epsilon_min=0.01,
                 episodes=1500, #Episodios de entrenamiento
                 hidden_size=256 ,#Tamaño de la capa oculta
                 gamma=0.99, # Factor de descuento
                 learning_rate=0.0005, #Tasa de aprendizaje
                 batch_size=64, #Tamaño minibatch
                 memory_size=100000, #Tamaño replay buffer
                 target_network_update_freq=10 #Actualizacion de la red objetivo
                 )
# agent.load_model("modelos/modelo_DQN.h5") <- si se quiere seguir entrenando el modelo anteriormente guardado
agent.train(nombre_archivo="modelos/modelo_DQN", save_graphs=True, save_every_50=True)
```

Fig. 19. Hiperparámetros usados en Colab 4

Este fue el modelo con mejor desempeño general. Utilizó una red más profunda (256 neuronas en cada capa oculta), pero con una cantidad reducida de actualizaciones por episodio (32 replays), lo cual ayudó a prevenir el sobreajuste. La combinación de una tasa de aprendizaje moderada y una política de exploración bien balanceada permitió un aprendizaje rápido, estable y generalizable.

El progreso del modelo fue el siguiente:

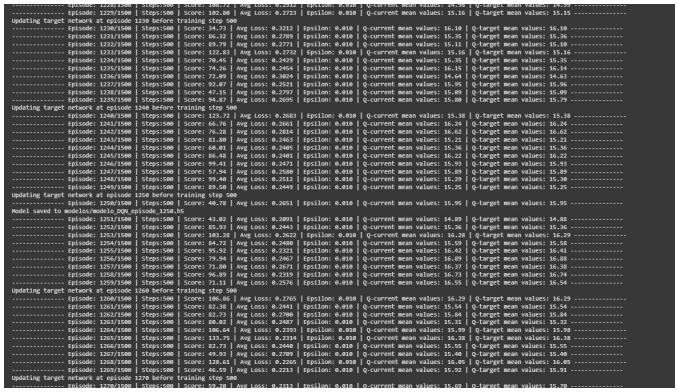


Fig. 20. Progreso durante el entrenamiento - Colab 4

Durante el entrenamiento, el agente fue capaz de ejecutar maniobras complejas desde estados iniciales desfavorables, logrando una mejora sostenida en el desempeño. Se observó una clara estabilidad y consistencia en los resultados obtenidos a lo largo de los episodios. Con los gráficos generados en la fase de entrenamiento, se puede observar lo siguiente:

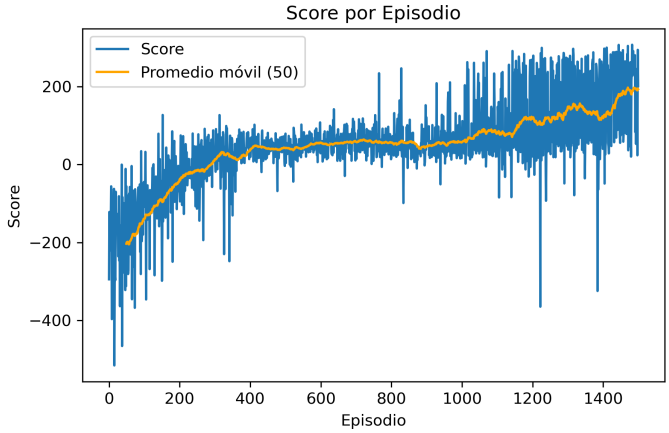


Fig. 21. Recompensa total con promedio móvil - Colab 4

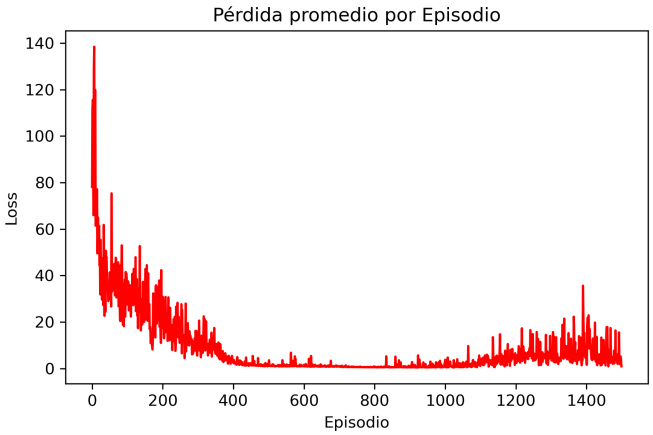


Fig. 22. Pérdida promedio por episodio - Colab 4

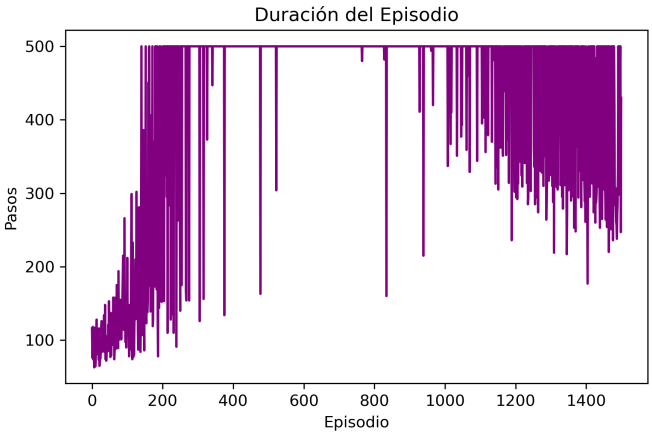


Fig. 23. Duración de los episodios - Colab 4

Los resultados muestran una recompensa creciente y sostenida con una variabilidad significativamente menor respecto a otros modelos. La pérdida disminuye de forma progresiva hasta estabilizarse, indicando que el modelo alcanza una convergencia adecuada. Además, la duración de los episodios se va acortando, lo que sugiere que el agente aprende a completar la tarea de manera más eficiente con el tiempo. Este comportamiento refleja una política bien afinada, capaz de generalizar correctamente a lo largo del entrenamiento.

En la fase de prueba (con $\epsilon = 0$) se obtuvo el siguiente rendimiento:

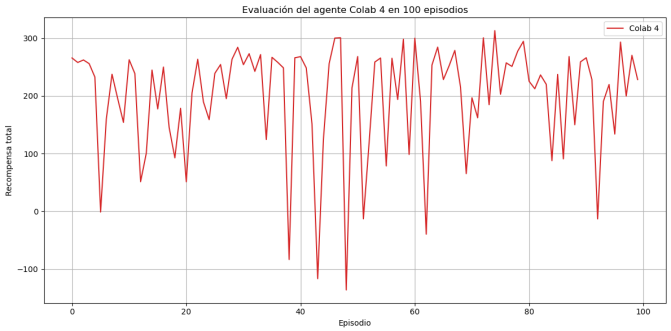


Fig. 24. Tasa de acierto del modelo ya entrenado - Colab 4

Con una tasa de acierto del **91.00%**, este modelo demostró ser el más eficiente y consistente entre todos los evaluados, validando así la combinación de arquitectura y parámetros elegidos.

*E. **Colab 5** - Cheaty Lander*

Estos son los hiperparámetros usados en este modelo:

```
#Los parametros más importantes son los marcados con un comentario
agent = DQNAgent(lunar,replays_per_episode=64,epsilon=1.0, epsilon_decay=0.995, epsilon_min=0.1,
                 episodes=2000, #Episodios de entrenamiento
                 hidden_size=64 ,#Tamaño de la capa oculta
                 gamma=0.99, # Factor de descuento
                 learning_rate=0.00005, #Tasa de aprendizaje
                 batch_size=64, #Tamaño minibatch
                 memory_size=100000, #Tamaño replay buffer
                 target_network_update_freq=20 #Actualizacion de la red objetivo
                 )
# agent.load_model("modelos/modelo_DQN.h5") <- si se quiere seguir entrenando el modelo anteriormente guardado
agent.train(nombre_archivo="modelos/modelo_DQN", save_graphs=True, save_every_50=True)
```

Fig. 25. Hiperparámetros usados en Colab 5

Este experimento intentó maximizar la estabilidad. Se usó una red muy liviana (64 neuronas en cada capa oculta), una tasa de aprendizaje extremadamente baja (0.00005), se limitaron los pasos por episodio a 300, se fijó un valor mínimo de epsilon más alto (0.1) y se acortó el máximo de pasos a 300.

Su progreso puede observarse en la siguiente figura:

```
------------ Episode: 936/2000 | Steps:300 | Score: 19.52 | Avg Loss: 1.9683 | Epsilon: 0.100 | Q-current mean values: 4.51 | Q-target mean values: 4.52 ------------
------------ Episode: 937/2000 | Steps:300 | Score: 14.83 | Avg Loss: 2.0640 | Epsilon: 0.100 | Q-current mean values: 4.66 | Q-target mean values: 4.66 ------------
------------ Episode: 938/2000 | Steps:300 | Score: 8.48 | Avg Loss: 1.8848 | Epsilon: 0.100 | Q-current mean values: 5.04 | Q-target mean values: 5.04 ------------
------------ Episode: 939/2000 | Steps:300 | Score: 27.75 | Avg Loss: 1.9333 | Epsilon: 0.100 | Q-current mean values: 4.82 | Q-target mean values: 4.82 ------------
updating target network at episode 940 before training step 300
------------ Episode: 940/2000 | Steps:300 | Score: 24.14 | Avg Loss: 2.1228 | Epsilon: 0.100 | Q-current mean values: 4.83 | Q-target mean values: 4.83 ------------
------------ Episode: 941/2000 | Steps:300 | Score: 22.39 | Avg Loss: 2.3628 | Epsilon: 0.100 | Q-current mean values: 5.03 | Q-target mean values: 5.03 ------------
------------ Episode: 942/2000 | Steps:300 | Score: 1.60 | Avg Loss: 2.4715 | Epsilon: 0.100 | Q-current mean values: 5.09 | Q-target mean values: 5.11 ------------
------------ Episode: 943/2000 | Steps:300 | Score: 22.06 | Avg Loss: 2.5833 | Epsilon: 0.100 | Q-current mean values: 4.92 | Q-target mean values: 4.94 ------------
------------ Episode: 944/2000 | Steps:300 | Score: 24.58 | Avg Loss: 2.5334 | Epsilon: 0.100 | Q-current mean values: 5.03 | Q-target mean values: 5.02 ------------
------------ Episode: 945/2000 | Steps:300 | Score: 63.01 | Avg Loss: 2.0426 | Epsilon: 0.100 | Q-current mean values: 5.21 | Q-target mean values: 5.21 ------------
------------ Episode: 946/2000 | Steps:300 | Score: 6.91 | Avg Loss: 2.4288 | Epsilon: 0.100 | Q-current mean values: 5.03 | Q-target mean values: 5.03 ------------
------------ Episode: 947/2000 | Steps:300 | Score: 45.74 | Avg Loss: 2.2600 | Epsilon: 0.100 | Q-current mean values: 5.18 | Q-target mean values: 5.18 ------------
------------ Episode: 948/2000 | Steps:300 | Score: 4.48 | Avg Loss: 2.0394 | Epsilon: 0.100 | Q-current mean values: 5.07 | Q-target mean values: 5.07 ------------
------------ Episode: 949/2000 | Steps:300 | Score: 49.78 | Avg Loss: 2.3055 | Epsilon: 0.100 | Q-current mean values: 5.10 | Q-target mean values: 5.11 ------------
------------ Episode: 950/2000 | Steps:300 | Score: 34.19 | Avg Loss: 2.0871 | Epsilon: 0.100 | Q-current mean values: 4.92 | Q-target mean values: 4.93 ------------
Model saved to modelos/modelo_DQN_episode_950.h5
------------ Episode: 951/2000 | Steps:300 | Score: -20.47 | Avg Loss: 2.0228 | Epsilon: 0.100 | Q-current mean values: 4.96 | Q-target mean values: 4.97 ------------
------------ Episode: 952/2000 | Steps:300 | Score: -11.33 | Avg Loss: 1.5430 | Epsilon: 0.100 | Q-current mean values: 5.09 | Q-target mean values: 5.11 ------------
------------ Episode: 953/2000 | Steps:300 | Score: 9.46 | Avg Loss: 2.1431 | Epsilon: 0.100 | Q-current mean values: 5.11 | Q-target mean values: 5.09 ------------
------------ Episode: 954/2000 | Steps:300 | Score: 45.77 | Avg Loss: 2.1783 | Epsilon: 0.100 | Q-current mean values: 5.06 | Q-target mean values: 5.05 ------------
------------ Episode: 955/2000 | Steps:300 | Score: 51.32 | Avg Loss: 1.9175 | Epsilon: 0.100 | Q-current mean values: 5.14 | Q-target mean values: 5.15 ------------
------------ Episode: 956/2000 | Steps:300 | Score: 14.78 | Avg Loss: 2.0135 | Epsilon: 0.100 | Q-current mean values: 4.85 | Q-target mean values: 4.86 ------------
------------ Episode: 957/2000 | Steps:300 | Score: 6.07 | Avg Loss: 1.9022 | Epsilon: 0.100 | Q-current mean values: 5.00 | Q-target mean values: 5.01 ------------
------------ Episode: 958/2000 | Steps:300 | Score: 35.23 | Avg Loss: 1.9165 | Epsilon: 0.100 | Q-current mean values: 5.24 | Q-target mean values: 5.24 ------------
------------ Episode: 959/2000 | Steps:300 | Score: 12.28 | Avg Loss: 2.0018 | Epsilon: 0.100 | Q-current mean values: 5.24 | Q-target mean values: 5.23 ------------
```

Fig. 26. Progreso durante el entrenamiento - Colab 5

El agente aprendió a estabilizarse en el aire, pero nunca llegó a aterrizar correctamente. La política aprendida fue demasiado conservadora y no incentivaba a tomar riesgos para completar la tarea.

Durante la fase de entrenamiento se generaron los siguientes gráficos:

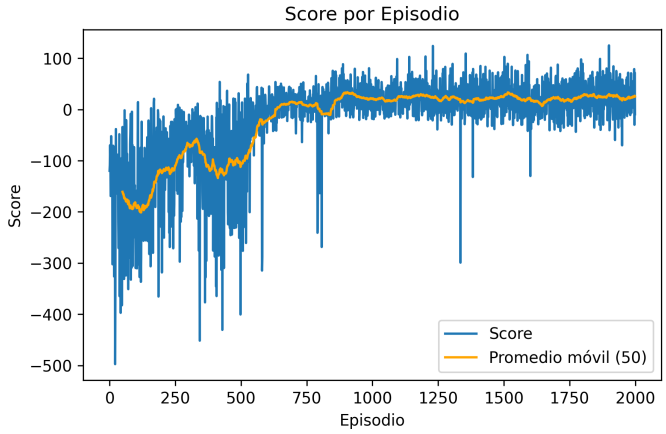


Fig. 27. Recompensa total con promedio móvil - Colab 5

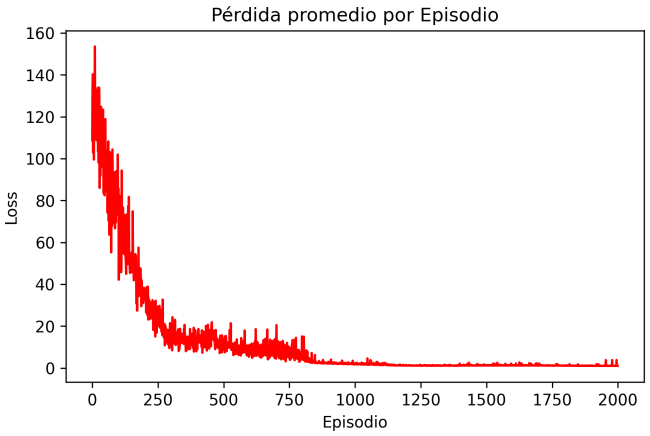


Fig. 28. Pérdida promedio por episodio - Colab 5

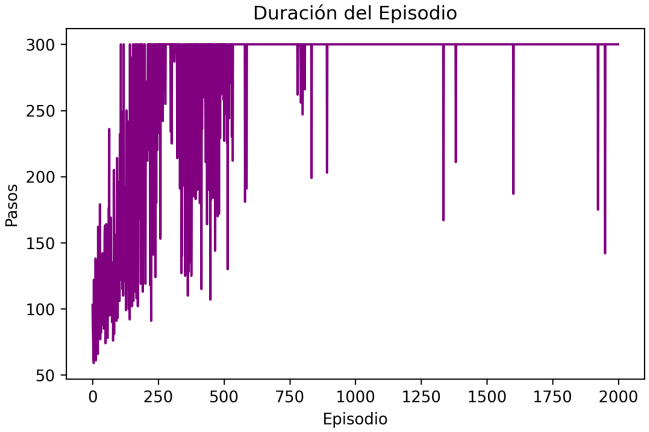


Fig. 29. Duración de los episodios - Colab 5

Se observa un aprendizaje extremadamente lento, con recompensas que apenas mejoran a lo largo del tiempo. Las pérdidas si se van reduciendo. La duración de los episodios fue artificialmente corta por el límite de pasos, lo que impidió desarrollar políticas más completas.

Durante la fase de prueba sin exploración, se obtuvo la siguiente tasa de acierto:

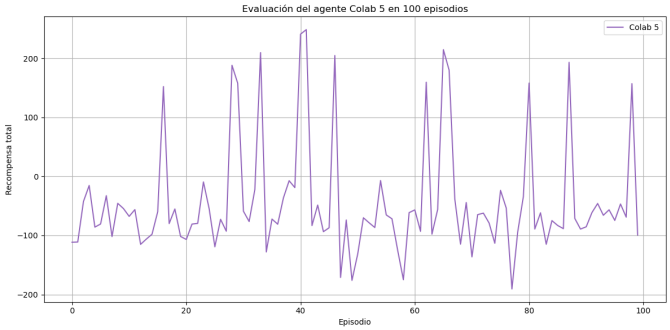


Fig. 30. Tasa de acierto del modelo ya entrenado - Colab 5

Este modelo alcanzó una tasa de acierto muy baja: solo **4.00%**, lo que evidencia que la política aprendida no incentivaba a completar la tarea correctamente, a excepción de casos particulares en los que funciona por no usar lo suficiente el motor principal y terminar "aterrizando por error".

F. Comparativa final

La siguiente tabla resume las configuraciones clave y los comportamientos observados en cada modelo. En un principio, el modelo Colab 2 se perfilaba como el mejor, pero fue finalmente superado por Colab 4.

Tras repetir entrenamientos con los mismos hiperparámetros, Colab 4 demostró mayor consistencia, mientras que Colab 2 era más impredecible: a veces obtenía mejores resultados, pero otras quedaba por debajo. Por ello, Colab 4 fue seleccionado como el modelo ganador por su estabilidad, eficiencia computacional y capacidad de generalización, incluso considerando la aleatoriedad inherente del entorno.

TABLA I
COMPARACIÓN DE HIPERPARÁMETROS USADOS EN LOS DISTINTOS ENTORNOS DE GOOGLE COLAB

Parámetro	Colab 1	Colab 2	Colab 3	Colab 4	Colab 5
`replays_per_episode`	1000	64	64	32	64
`episodes`	3000	1500	1500	1500	2000
`hidden_size`	256	128	128	256	64
`learning_rate`	0.001	0.0005	0.0005	0.0005	0.00005
`batch_size`	64	64	128	64	64
`memory_size`	10,000	100,000	100,000	100,000	100,000
`target_network_update_freq`	15	10	10	10	20
`max_steps`	500	500	500	500	300

Análisis de parámetros:

1) `replays_per_episode`: 1000 actualizaciones por episodio (Colab 1) tienden al sobreajuste. 32 (Colab 4) promueven mayor variabilidad y eficiencia, compensado con una red más compleja.
2) `episodes`: Con 1500 episodios bien aprovechados se logra convergencia. Más episodios no siempre implican mejor desempeño si la política no mejora.
3) `hidden_size`: 256 neuronas permiten maniobras complejas. Colab 4 aprovecha esto sin caer en sobreajuste, a diferencia de Colab 1. Tamaños pequeños (Colab 5) no son suficientemente configurables para tareas exigentes.
4) `learning_rate`: Tasa moderada (0.0005) proporciona aprendizaje estable. En Colab 5, una tasa muy baja impide progreso rápido y en Colab 1 una tasa más alta hace que no logre aprender.
5) `batch_size`: Un tamaño mayor (128 en Colab 3) suaviza el aprendizaje pero reduce agilidad. Colab 4 mantiene buen equilibrio.
6) `memory_size`: Un buffer pequeño (Colab 1) limita la diversidad de experiencias. Colab 4 utiliza un tamaño grande que mejora generalización.
7) `target_network_update_freq`: Actualizar cada 10 episodios (Colab 4) ofrece mayor estabilidad. Colab 5 lo hace con menor frecuencia, dificultando la adaptación.
8) `max_steps`: Valores bajos (Colab 5) impiden completar episodios. Esto favorece políticas que evitan el aterrizaje para mantener estabilidad en el aire.

Resumen del comportamiento observado en cada Colab:

- **Colab 1**: Mal aterrizaje, red costosa y sobreentrenada.
- **Colab 2**: Buen equilibrio entre exploración y estabilidad, pero más variante cuando se ejecuta varias veces.
- **Colab 3**: Flota en exceso y evita aterrizar, muchos episodios truncados.
- **Colab 4**: Gran control general, con una muy buena estabilidad y una tasa de aciertos muy alta incluso probandose repetidamente.
- **Colab 5**: Demasiado conservador, no desarrolla una política clara de descenso, simplemente intenta no chocar.

Resolución: Colab 4 logra el mejor balance general. Aprende rápido, es estable y eficiente. Su arquitectura y parámetros están mejor afinados al entorno, siendo este el modelo seleccionado en esta investigación.

IV. CONCLUSIONES

Concluimos que hicimos alto proyecto.

BIBLIOGRAFÍA

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., online book, 2018. Disponible en: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[2] PyTorch, "Optimization tutorial," 2023. Disponible en: https://docs.pytorch.org/tutorials/beginner/basics/optimization_tutorial.html.

[3] Farama Foundation, "Lunar Lander environment," *Gymnasium*, 2023. Disponible en: https://gymnasium.farama.org/environments/box2d/lunar_lander/.

[4] S. Zolna, "Deep Q-Networks explained," *LessWrong*, 2019. Disponible en: https://www.lesswrong.com/posts/kyvCNgx9oAwJCuevo/deep-q-networks-explained.

[5] IBM, "Hyperparameter tuning," 2023. Disponible en: https://www.ibm.com/es-es/think/topics/hyperparameter-tuning.

[6] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015. Disponible en: https://www.nature.com/articles/nature14236.

[7] OpenAI, "Spinning Up in Deep RL," 2023. Disponible en: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[8] Hugging Face, "Large Language Models Course, Chapter 7," 2023. Disponible en: https://huggingface.co/learn/llm-course/chapter7/1.

[9] Pandas Development Team, "pandas.DataFrame.rolling," 2023. Disponible en: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rolling.html.

[10] Lazy Programmer, "What is the replay buffer in DQN (Deep Q-Learning)?", 2019. Disponible en: https://lazyprogrammer.me/what-is-the-replay-buffer-in-dqn-deep-q-learning/.

[11] PyTorch, "torch.optim.Adam," 2023. Disponible en: https://docs.pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam.

[12] J. Code, "gym_solutions," GitHub repository, 2023. Disponible en: https://github.com/johnnycode8/gym_solutions.

[13] D. Silver, "Reinforcement Learning Introduction," YouTube video, 2019. Disponible en: https://www.youtube.com/watch?v=YAfS8-BXp8Q.

[14] H. V. Hasselt, "Deep Q-Networks Explained," YouTube video, 2020. Disponible en: https://www.youtube.com/watch?v=EgklwkyieOY.

[15] A. Ng, "Deep Reinforcement Learning Tutorial," YouTube video, 2018. Disponible en: https://www.youtube.com/watch?v=aircAruvnKk.

[16] S. Walters, “PyTorch Basics Tutorial,” YouTube video, 2021. Disponible en: https://www.youtube.com/watch?v=EUrWGTCGzlA&t=7s.
[17] PyTorch, “Saving, Loading, and Running Models Tutorial,” 2023. Disponible en: https://docs.pytorch.org/tutorials/beginner/basics/saveloadrun_tutorial.html.
[18] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” arXiv preprint arXiv:1412.6980, 2014. Disponible en: https://arxiv.org/abs/1412.6980.