UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3104 – Aprendizaje por Refuerzo

Ing. Javier Josué Fong Guzmán

Proyecto Final

Jugador Galaxian

Diego Leiva

Guatemala, 21 de noviembre de 2025

Descripción del Problema y Objetivos

El objetivo de este proyecto es desarrollar y entrenar agentes de aprendizaje por refuerzo profundo capaces de jugar al entorno ALE/Galaxian-v5 utilizando la librería Gymnasium en Python.

Los objetivos específicos fueron:

- Implementar al menos dos prototipos de agentes de RL con algoritmos y arquitecturas distintas.
- Realizar experimentos controlados para poder analizar el efecto de algunos hiperparámetros clave y comparar el desempeño de ambos modelos frente a una política aleatoria.
- Generar evidencia del proceso de entrenamiento y del desempeño final mediante gráficas y videos de juego.

Arquitecturas, algoritmos y configuraciones utilizadas

Entorno

Se trabajó con el entorno Gymnasium ALE/Galaxian-v5 en donde se realizó el registro de los entornos ALE para hacerlos utilizables.

Para seguir las buenas prácticas de DeepMind en Atari, se aplicaron los siguientes wrappers al entorno:

AtariPreprocessing:

- Observaciones en escala de grises.
- Frame skip = 4.
- Redimensionado de la pantalla a 84×84 píxeles.

FrameStackObservation:

- Apilamiento de 4 frames consecutivos, lo que da observaciones tipo (stack_size, 84, 84) o equivalentes.

Esto produce un estado de entrada con información temporal y reduce dimensionalidad, tal como en los trabajos originales de Atari.

Arquitectura DQN clásica

El primer modelo implementado fue una versión estándar del Deep Q-Network (DQN) utilizada originalmente para Atari. La red recibe como entrada un tensor de forma $(C, 84, 84)$, donde C corresponde al número de frames apilados (en este caso 4). Esto permite que la red observe el movimiento entre fotogramas sin necesidad de memoria interna o LSTM.

La arquitectura se divide en dos partes:

Bloque convolucional:

Este bloque extrae características visuales del estado del juego. Está compuesto por:

- **Conv1**: 32 filtros, kernel_size=8, stride=4
- **Conv2**: 64 filtros, kernel_size=4, stride=2
- **Conv3**: 64 filtros, kernel_size=3, stride=1

Después de estas capas, el mapa de características se reduce a un tamaño aproximado de $(64, 7, 7)$, equivalente a 3136 valores al aplanarlo. Esta representación comprimida captura información espacial relevante para decidir acciones.

Bloque denso (Fully Connected):

Sobre los 3136 valores extraídos, la red aplica:

- **fc1**: 3136 → 512 unidades (ReLU)
- **fc_out**: 512 → número de acciones del entorno

La salida final es un vector con los valores Q estimados para cada acción posible. Como en muchas implementaciones de Atari, antes de entrar a la red los píxeles se normalizan a un rango de 0 a 1 dividiendo entre 255.

Arquitectura Dueling DQN

El segundo modelo implementado fue Dueling DQN, que conserva exactamente el mismo bloque convolucional que el DQN clásico, pero modifica la parte final de la red para separar explícitamente dos tipos de información: el valor del estado y la ventaja de cada acción.

Esta separación permite que el agente aprenda mejor en situaciones donde varias acciones son equivalentes o cuando la decisión óptima depende más del estado que de la acción puntual.

Después de aplanar los 3136 valores provenientes del bloque convolucional, la red se divide en dos "*ramas*":

Stream de valor $V(s)$:

Evalúa cuán bueno es el estado actual, independientemente de la acción.

- fc_val: 3136 → 512
- val_out: 512 → 1

Stream de ventaja $A(s,a)$:

Evalúa qué tan buena es cada acción relativa a las demás.

- fc_adv: 3136 → 512
- adv_out: 512 → número de acciones

Finalmente, ambas salidas se combinan mediante la formula $Q(s,a) = V(s) + A(s,a)$

Este ajuste elimina la ambigüedad entre valor y ventaja, y en la práctica hace que el modelo aprenda más rápido y sea más estable, especialmente en juegos donde muchas acciones no tienen efecto inmediato.

Replay Buffer

Se utilizó un replay buffer para almacenar las transiciones que el agente va experimentando (estado, acción, recompensa, siguiente estado y fin del episodio) y entrenar a partir de lotes aleatorios en lugar de usar únicamente la última jugada, lo cual reduce la correlación entre muestras y estabiliza el aprendizaje.

La memoria tiene un tamaño fijo y funciona de manera circular: cuando se llena, las transiciones nuevas reemplazan a las más antiguas. Los estados se guardan en formato compacto como imágenes en escala de grises de 84×84 con 4 frames apilados, permitiendo almacenar muchas experiencias sin saturar la RAM. Durante el entrenamiento, el modelo actualiza sus parámetros utilizando lotes extraídos de esta memoria.

Algoritmo de aprendizaje

Ambos modelos se entrenan con un esquema de DQN estándar con target network:

Dos redes:

- **online_net**: se actualiza en cada paso de entrenamiento.
- **target_net**: copia periódica de online_net.

Actualización DQN:

- Se muestrea un batch del replay buffer.
- Se calqula el Q value a partir de la online_net
- Se calcula el target
- Se optimiza la perdida con el optimizador Adam

Hiperparámetros y configuraciones

El proyecto tuvo dos fases principales de entrenamiento: una fase de comparación inicial entre DQN y Dueling DQN y una fase de entrenamiento extendido solo para el mejor modelo.

Entrenamiento comparativo

En la fase comparativa se buscaba validar que todo el pipeline funcionara correctamente y que ambos modelos fueran capaces de aprender mejor que una política aleatoria. Para esto se entrenó cada variante durante aproximadamente 300,000 pasos de juego.

Hiperparámetros derivados dinámicamente de max_steps:

- **buffer_capacity** ≈ min(max_steps // 2, 200_000) con mínimo de 50,000.
- **start_learning_after** ≈ máximo entre:
  - 10% de buffer_capacity.
  - 1% de max_steps.
  - 5,000 pasos.
- **target_update_freq** ≈ 5% de max_steps con mínimo de 5,000.
- **epsilon**:
  - epsilon_start = 1.0, epsilon_end = 0.05.
  - epsilon_decay_steps ≈ 0.5 * max_steps.
- **gamma** = 0.99.
- **batch_size** = 32.

Esta configuración se diseñó para experimentar rápidamente con diferentes cantidades de pasos (5k, 50k, 300k) ajustando automáticamente los parámetros.

Entrenamiento completo

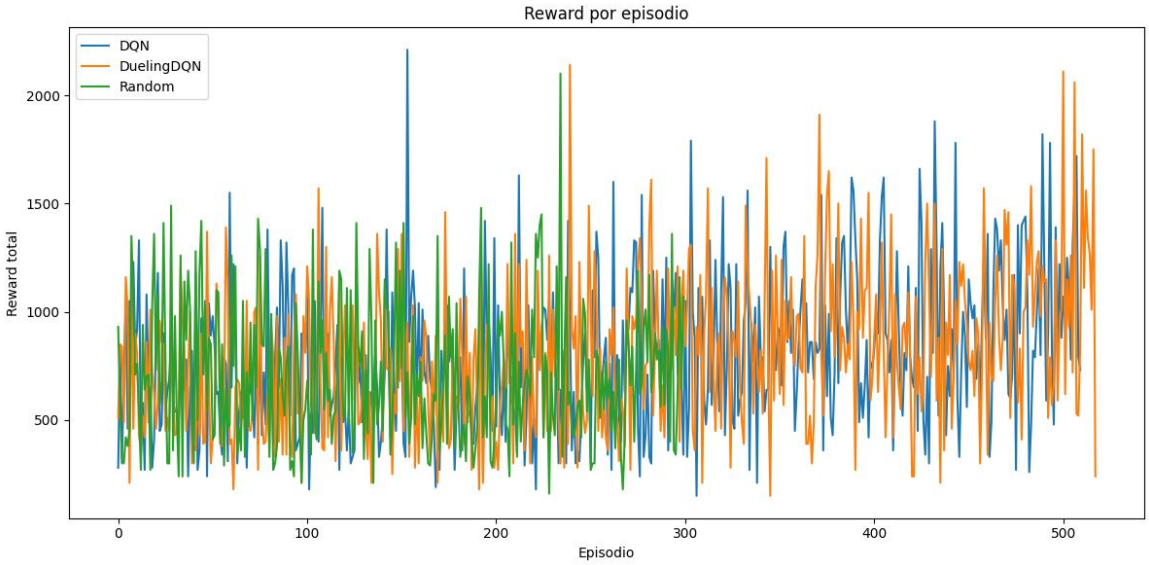
En la segunda fase se tomó como base la arquitectura Dueling DQN y se realizó un entrenamiento más ambicioso, llegando a unos 1.5 millones de pasos. En este caso los hiperparámetros se fijaron manualmente inspirándose más de cerca en la configuración del articulo por Volodymyr Mnih, et.al. *"Human-level control through deep reinforcement learning"*.

Estos hiperparámetros que cambiaron respecto al entrenamiento comparativo son:

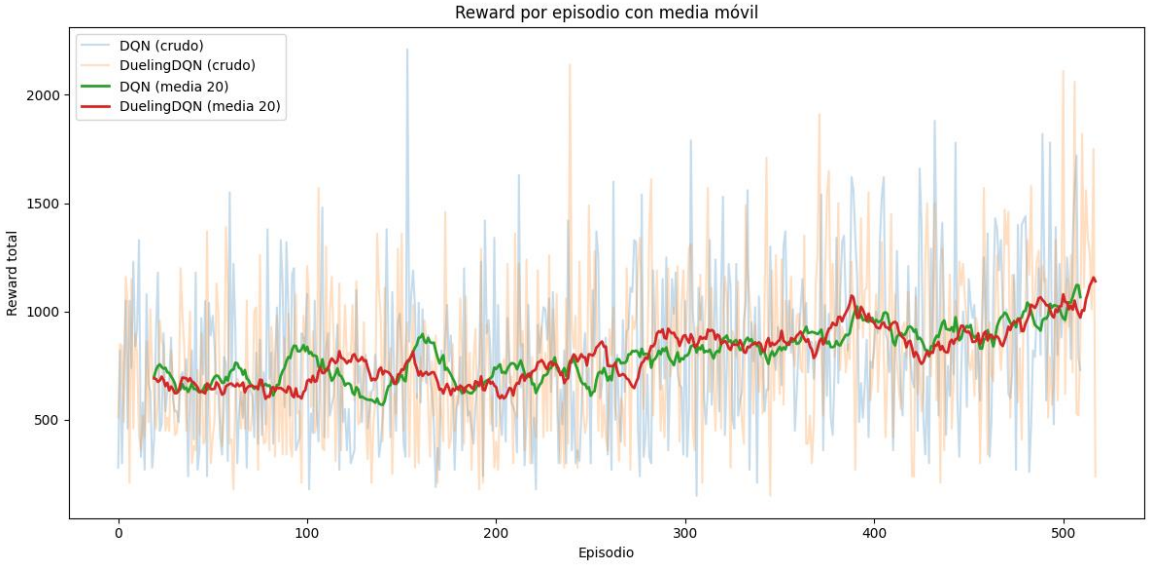
- **buffer_capacity** = 350,000. Inicialmente se consideró 1,000,000, pero esto saturaba la RAM (32GB), por lo que se redujo a 350k, aún demandando ~88% de uso de memoria.
- **start_learning_after** = 50,000 transiciones.
- **target_update_freq** = 10,000 pasos.
- **epsilon_decay_steps** = 1,300,000, Esto implica que la mayor parte del rango de epsilon se recorre en los primeros ~1.3M pasos, dejando ~200k pasos finales principalmente en régimen de explotación.

Evidencia del proceso de entrenamiento

Gráficos de entrenamiento comparativo



En este gráfico se ve la comparación entre las recompensas brutas por episodio para 3 agentes diferentes.



En la gráfica de media móvil se observa que ambos modelos mejoran significativamente respecto a la política aleatoria. Y que Dueling DQN tiende a mantener una ligera ventaja respecto al DQN clásico tras cierto número de episodios, aunque la diferencia no es dramática con solo 300k pasos.

Checkpoints y videos intermedios

Para evidenciar visualmente la evolución de los agentes se generaron videos en los episodios 1, 300 y 500 (a partir de diferentes checkpoints) para ambos modelos.

Los videos se encuentran en los siguientes directorios dentro del repositorio:

- videos/DQN_Iteration/
- videos/DuelingDQN_Iteration/

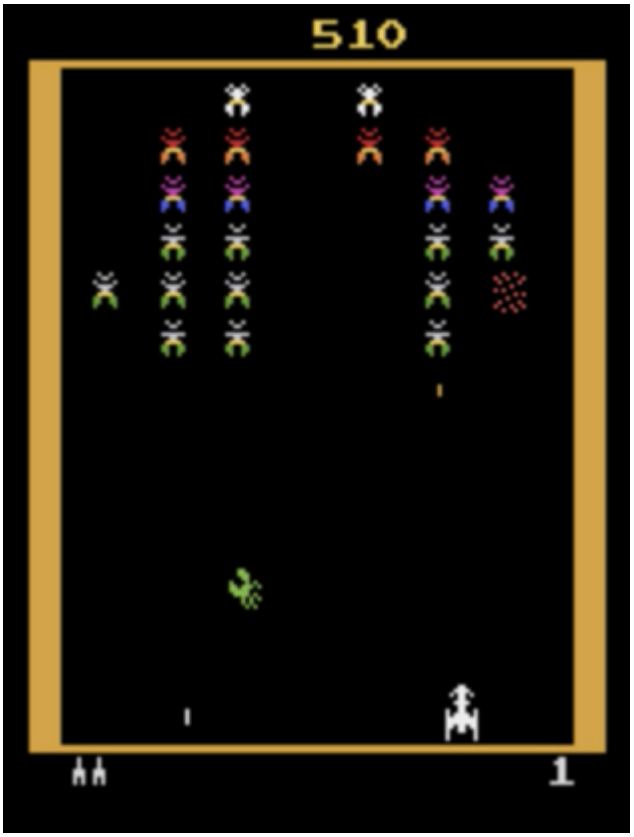
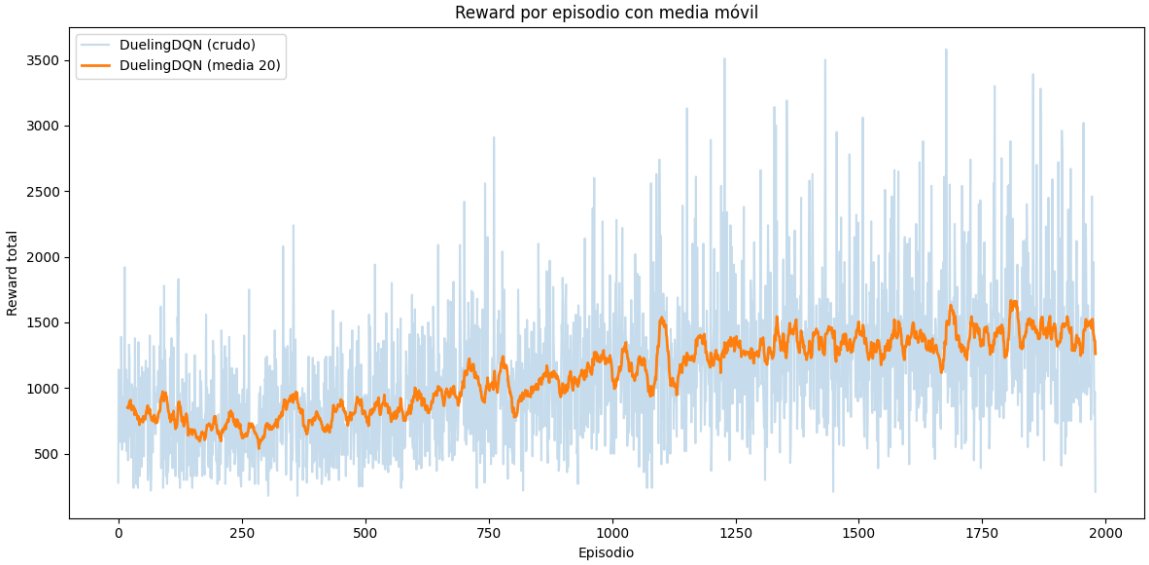


Gráfico de entrenamiento extendido



Para la etapa de entrenamiento extendido se grafico la recompensa por episodio con media móvil de ventana 20. Aunque el entorno es ruidoso, la curva suavizada muestra una tendencia creciente hasta estabilizarse en un nivel de desempeño superior al alcanzado en la fase de 300k pasos.

Comparación entre prototipos implementados

Se compararon 3 modelos diferentes, DQN clásico, Dueling DQN y Baseline Aleatorio, y posteriormente se realizó un entrenamiento extendido para DuelingDQN.

A continuación, la tabla que resume la recompensa media obtenida por cada modelo durante el entrenamiento comparativo con un epsilon greedy para evaluar la política post entreno:

Agente	Frames aprox.	Recompensa media (eval)
RandomPolicy	N/A	~707.9
DQN clásico	300k	~1090.0
Dueling DQN (comparativo)	300k	~1144.4
Dueling DQN (entrenamiento ext.)	1.5M	~1682.4

Discusión de resultados

Los resultados numéricos y las gráficas confirman que el agente está realmente aprendiendo: tanto el DQN clásico como el Dueling DQN superan con claridad a la política aleatoria. La arquitectura Dueling parece aportar una ventaja, aunque esta no es muy marcada cuando se limita el entrenamiento a 300,000 pasos. En esa escala de datos, ambos modelos se comportan de forma similar y las diferencias pueden quedar parcialmente ocultas por la variabilidad del entorno. Sin embargo, cuando se entrena el Dueling DQN con más pasos (1.5 millones), su desempeño mejora de forma significativa y se aleja más de la línea base, lo que coincide con la motivación original de este tipo de arquitectura.

Un aspecto clave fue el cambio de la función de pérdida. Durante las primeras etapas del proyecto se usó el error cuadrático medio (MSE), pero esto se asoció con un estancamiento temprano del aprendizaje, mientras que al pasar a la pérdida tipo Huber (L1 Loss) el entrenamiento se volvió más estable y prolongado. Este detalle, que puede parecer menor, tuvo un impacto claro en la calidad del agente final.

El esquema de exploración ϵ-greedy con un largo intervalo de decay permitió que el agente explorara durante buena parte del entrenamiento y dedicara los últimos cientos de miles de pasos a explotar lo aprendido. El uso de una semilla global ayudó a reproducir los resultados, aunque abre la pregunta de si entrenar con varias semillas distintas podría producir agentes aún mejores al aprovechar trayectorias de experiencia más variadas.

Por último, el proyecto estuvo limitado por los recursos de hardware disponibles, especialmente la memoria RAM. Un replay buffer de un millón de transiciones habría sido más cercano a los experimentos originales en Atari, pero resultó impráctico por cuestiones de consumo de memoria. Este tipo de restricciones influye directamente en el número de experiencias que el agente puede reutilizar y, por tanto, en la calidad máxima esperable del modelo.

Conclusiones

- El mejor modelo del proyecto fue el Dueling DQN entrenado con aproximadamente 1.5 millones de pasos, que alcanzó una recompensa media significativamente más alta que el resto de las variantes.
- Aun con las limitaciones de recursos, el agente final mostró un comportamiento razonablemente competente en Galaxian y dejó claro que, con más tiempo de entrenamiento y mayor capacidad de memoria, sería posible acercarse más a los niveles de desempeño reportados en la literatura.

Repositorio

https://github.com/LeivaDiego/Galaxian-Player.git