Universidad de La Sabana | Facultad de Ingeniería

Asignatura: Fundamentos de Inteligencia Artificial

Informe de Contribución Individual
Reto Connect-4

Estudiante: Santiago Gavilan **Fecha:** 24 de Noviembre de 2025

1 Resumen de Contribución

Aunque trabajamos en equipo en la lógica del agente, mi parte principal fue asegurarme de que funcionara de verdad y probarlo con datos. Me encargué de armar los experimentos para ver si el agente aprendía y de dejar el código limpio y organizado. Mi objetivo era cumplir con la parte de "Validación" y "Calidad" del reto, demostrando con gráficas que nuestra solución no es solo suerte, sino que realmente mejora.

Básicamente, diseñé el bucle de entrenamiento, generé las curvas de aprendizaje y organicé el notebook final (`entrega.ipynb`) para que fuera fácil de leer y ejecutar.

2 Aportes Destacados

Siguiendo la rúbrica del proyecto, estos fueron mis dos aportes más importantes:

Aporte A: Pruebas y Entrenamiento (Validación)

No bastaba con decir "el agente juega bien". Hice un sistema para entrenarlo miles de veces y graficar su progreso contra un jugador aleatorio. Esto fue clave para pasar de la anécdota a los datos. Implementé el código que entrena al agente usando Q-Learning, aplicando lo que vimos en clase sobre convertir incertidumbre en riesgo. Gracias a esto, pudimos sacar las gráficas del notebook que prueban que el agente aprende con el tiempo.

Archivo: `entrega.ipynb / policy.py`
Evidencia (Commit): `feat: Integración final Híbrida (Q-Learning + Minimax)`

Aporte B: Limpieza y Documentación del Código

Para asegurar una buena nota en implementación, me dediqué a que el código fuera legible. Refactoricé el archivo `policy.py` y actualicé el `README.md` para que cualquier persona entienda rápidamente cuál es nuestra política y cómo funciona la lógica del agente. Además, estructuré el `entrega.ipynb` como una historia completa: explica cómo probamos el agente, lo entrena y muestra los resultados, todo en orden para que sea reproducible.

Archivo: `README.md / policy.py`
Evidencia (Commit): `feat: Cambio en el readme para comprension de cual es la policy`

3 Desafíos y Logros

Desafíos Enfrentados

- **Gráficas Inestables:** Al principio, las curvas de aprendizaje salían con mucho ruido porque el juego tiene azar. Tuve que arreglarlo haciendo que el código promediara varios intentos para suavizar las líneas y que se entendiera el progreso.
- **Guardar el Aprendizaje:** Fue complicado lograr que la Q-Table se guardara y cargara bien (`q_table.pkl`) sin perder datos entre ejecuciones, pero logré configurar la persistencia para que el agente "recordara" lo aprendido.

Principales Logros

- **Demostrar que aprende:** Logré sacar la evidencia numérica de que el agente mejora, cumpliendo con el requisito de validación.
- **Entrega limpia:** Dejé el proyecto listo para entregar, con un código ordenado y un notebook que cualquiera puede correr y entender sin problemas.

4 Reflexión y Propuestas de Mejora

Reflexión

El agente funciona bien resolviendo el problema del "arranque en frío" con la heurística, pero tiene un límite. Como usa una tabla (Q-Table), solo aprende de las posiciones exactas que ha visitado. Si se encuentra con una jugada parecida pero nueva, no sabe generalizar. Es bueno memorizando, pero le falta intuición.

Para mejorar en el futuro

1. **Entrenar contra otros:** En lugar de jugar solo contra sí mismo, debería entrenar contra otros bots (como uno puro Minimax) para que aprenda estrategias más variadas.
2. **Usar Redes Neuronales:** Cambiaría la tabla por una red neuronal (Deep Q-Learning). Así, el agente podría entender patrones generales del tablero y no solo posiciones de memoria, aunque sería más difícil de validar y ajustar.