1. **Implementación del MCTS base del agente**

Link commit:
https://github.com/ValeRuizTo/connect4-aljuri-ruiz/commit/e79b4ca9026b368fcdd3e71e1f5af5d5291e0089

Mi contribución principal al proyecto fue la implementación del agente utilizando Monte Carlo Tree Search (MCTS), construyendo las cuatro etapas esenciales del algoritmo (selección con UCB1, expansión implícita, simulación mediante rollouts y backpropagation), lo cual permitió obtener un agente funcional y capaz de generar jugadas legales desde cero. Sin embargo, la primera versión utilizaba 800 iteraciones y producía tiempos de ejecución elevados que causaban timeouts en Gradescope. Tras experimentación progresiva, reduje las iteraciones a 300 y luego a 50, encontrando este último valor como punto óptimo que preservaba un desempeño competitivo sin exceder los límites de tiempo. Esta optimización permitió estabilizar el agente y habilitó la posibilidad de seguir mejorando el agente y que pudiera pudiera evolucionar hacia versiones más tácticas y eficientes.

2. **Implementación de la nueva política híbrida (Q-learning + MCTS + tácticas) y policy externa**

Link commit politica local:
https://github.com/ValeRuizTo/connect4-aljuri-ruiz/commit/4c5318fa89e67bf9d14794ba5dc964cc4272c0a4

Link commit politica externa:
https://github.com/ValeRuizTo/connect4-aljuri-ruiz/commit/da47957d8d7ee15505b10efad7dac47c1e6be5d3

Mi contribución principal consistió en extender la arquitectura del agente original para transformarlo en un agente híbrido, capaz de combinar tácticas inmediatas, MCTS y un módulo de aprendizaje basado en Monte Carlo First-Visit (FVMC). Este nuevo módulo permitió integrar aprendizaje real dentro de un agente que inicialmente era solo táctico y de búsqueda. Para lograrlo, desarrollé un sistema que:

- Recolecta episodios completos de juego, generados a partir de auto-juego entre múltiples versiones del agente
- Genera identificadores hash únicos para cada estado del tablero, asegurando un almacenamiento compacto y eficiente.
- Actualiza los valores Q(s,a) siguiendo el esquema First-Visit Monte Carlo, lo que permite asignar créditos correctos al primer encuentro de cada estado-acción en un episodio.
- Mejora la política del agente reutilizando la memoria acumulada, almacenada en una Q-table persistida en Q-table.json.

El agente híbrido resultante funciona bajo un esquema de fallback inteligente:

- Si un estado aparece suficientemente entrenado en la Q-table, el agente utiliza directamente los valores aprendidos.

- Si el estado no tiene visitas suficientes, recurre a la política base (tácticas + MCTS), evitando así decisiones basadas en estimaciones ruidosas.

Esta arquitectura permite mantener la robustez del MCTS incluso con pocas iteraciones, mientras que al mismo tiempo capitaliza conocimiento previamente adquirido. La integración fue posible gracias a la infraestructura previa desarrollada por mi compañera, quien implementó la base del entorno de entrenamiento. Sobre esa base añadí el módulo de aprendizaje, la lógica de consulta controlada a la Q-table y el mecanismo híbrido completo.

Limitaciones enfrentadas: Durante el desarrollo surgieron varios desafíos, la Q-table inicial contenía muchos estados de baja calidad, generados por agentes aleatorios, lo que introducía ruido en la política. Para mitigarlo, incorporé un filtro que elimina las consultas Q(s,a) que tuvieran pocas visitas, evitando sobreajuste a valores poco fiables.

El entrenamiento previo entre un MCTS limitado (10 ITER) y un agente aleatorio produjo patrones subóptimos, lo que afectó algunos primeros valores Monte Carlo. Aun así, el filtrado y el fallback a MCTS redujeron este impacto.

Impacto en el agente final: El agente híbrido permitió entrenar offline/local un conjunto de 8 agentes inteligentes y 1 aleatorio defensivo (generado por mi compañera), cuyos episodios se utilizaron para generar la Q-table final, sin los estados basuras previos.

Gracias a esto, el agente hereda patrones útiles descubiertos en miles de partidas simuladas, sin necesidad de aprender durante el torneo. El resultado es un agente mucho más consistente: Evita errores básicos mediante tácticas inmediatas. Se apoya en MCTS en estados nuevos. Aprovecha experiencia acumulada donde existe suficiente evidencia mediante FVMC. La principal limitación es que estados raros o nunca visitados dependen únicamente de MCTS, pero el equilibrio general entre búsqueda, memoria y tácticas produce un agente más fuerte, estable y con decisiones mejor informadas que un simple MCTS.

Reflexion y mejoras posibles

El agente que desarrollamos combina tácticas inmediatas, MCTS, y una Q-table aprendida por FVMC mediante el autojuego, lo cual le da un balance entre intuición rápida (win/block), búsqueda limitada (MCTS) y una política mejorada mediante la experiencia (Q / FVMC). La arquitectura funciona bien: MCTS cubre lo que la Q-table aún no ha visto, y la Q-table evita repetir errores del pasado. Con varias rondas de entrenamiento y limpieza, la tabla fue quedando con estados cada vez más significativos, esto hace que:

- El agente sea estable y no comete errores básicos.
- La Q-table crezca con estados relevantes, no basura.
- El entrenamiento con agentes buenos + agentes defensivos genera variedad.
- Una política mejorada gracias a la competencia entre agentes competentes que también iban mejorando su policy, debido a la Q table compartida.

Para mejorar el desempeño del agente se puede mejorar la generación de la Q-table actual para evitar que la memoria crezca con ruido. También sería importante explorar una integración más profunda entre la búsqueda MCTS y los valores de Q: permitir que MCTS utilice la Q-table para guiar la exploración haría que el agente tome decisiones más informadas incluso en estados poco comunes.