1. **Implementación de Módulo Táctico para MCTS**

Link commit:
https://github.com/ValeRuizTo/connect4-aljuri-ruiz/commit/9b749776926e9331e27e686d3708b73240b91fba

Desarrollé un módulo de tácticas inmediatas para corregir fallos críticos que el MCTS de 50 iteraciones no podía evitar debido a su poca profundidad. Antes de esta mejora, el agente fallaba en jugadas básicas: no aprovechaba victorias directas y tampoco bloqueaba amenazas simples del oponente. Implementé la función immediate_tactics, que revisa todos los movimientos legales y detecta si alguno gana de inmediato o evita una derrota en el siguiente turno. Esta capa se ejecuta antes del MCTS, garantizando decisiones tácticas correctas sin aumentar el costo computacional y manteniendo compatibilidad con los límites de Gradescope.

2. **Arquitectura para el funcionamiento de la policy (Evaluation utils, policy hello y training_env)**

Link commit policyHelloDefensiva:
https://github.com/ValeRuizTo/connect4-aljuri-ruiz/commit/9a6b602c446593c3f81fee15d943c0562f9833ca

Link commit training:
https://github.com/ValeRuizTo/connect4-aljuri-ruiz/commit/c725029049ae5df6f1d558d2b58c5d15a27a8b3c

Mi aporte se centró en construir todo el ecosistema de auto-juego que permitió entrenar a los agentes mediante miles de partidas simuladas. Este sistema usa el training_env:

- La ejecución completa de una partida entre dos agentes, gestionando turnos, estados intermedios y detección del estado final.
- La asignación de recompensas según el resultado de la
- El registro estructurado de episodios completos, necesarios para aplicar el método de aprendizaje Monte Carlo.

Este entorno permitió generar una gran diversidad de estados gracias a miles de partidas entre agentes de distintos niveles. Sin este ecosistema, no habría sido posible entrenar la Q-table o evaluar el desempeño de las políticas evolutivas.

Configuración del entrenamiento masivo y generación de la Q-table final (evaluation_utils.py):

Además desarrollé el sistema que entrena a múltiples agentes de forma paralela y colaborativa. Organicé un conjunto agentes 8 inteligentes y 1 defensivo (helloPolicy) diseñados para maximizar la diversidad de estados observados.

Todos estos agentes compartieron la misma Q-table, lo que aceleró significativamente el aprendizaje colectivo: la experiencia obtenida por un agente se convertía inmediatamente en conocimiento disponible para los demás, lo que significa mejora de policy.

Evolución de helloPolicy

En la implementación inicial, la política Hello era completamente aleatoria, lo cual servía como línea base pero producía partidas con muy poca información útil para el aprendizaje. Decidí extender esta política para convertirla en un agente defensivo, capaz de detectar amenazas inmediatas del oponente (como una posible conexión en 3) y bloquearlas así generar de datos más variados y estables, contribuyendo una Q-table más sólida.

Limitaciones enfrentadas :El principal límite del sistema fue que la Q-table inicial se construyó a partir de enfrentamientos donde uno de los agentes era aleatorio y el otro un MCTS limitado. Como consecuencia, los retornos FVMC aprendieron algunos patrones subóptimos o inconsistentes.

Impacto Mi implementación contribuyo al aprendizaje real basado en episodios y memoria compartida entre agentes. Gracias a este ecosistema, el agente dejó de ser una política estática y se transformó en uno capaz de acumular experiencia, integrando esa memoria en la Q-table final. Esto permitió que el agente híbrido (Q (FVMC) + tácticas + MCTS) superara consistentemente al MCTS puro en múltiples enfrentamientos. En resumen, mi aporte estableció la infraestructura que hizo posible el aprendizaje colectivo y generó la Q-table que alimenta directamente al agente final del proyecto.

Reflexión de la solución y mejoras

A lo largo del proyecto el agente evolucionó desde un enfoque basado en reglas hacia una arquitectura híbrida que combina tácticas inmediatas, MCTS y una Q-table aprendida por FMVC y el auto-juego. Este proceso sigue la lógica de Generalized Policy Iteration, donde primero estimamos valores con Monte Carlo y luego refinamos la política con decisiones Q-greedy. El entrenamiento offline/local, apoyado en múltiples agentes y una política defensiva que generó partidas más útiles, permitió construir una Q-table que funciona como memoria estable y depurada. La política final integra tres niveles tácticos, experiencia almacenada y MCTS como fallback haciéndola mucho más robusta, logrando un balance sólido entre razonamiento rápido, aprendizaje previo y exploración estratégica. En conjunto, el agente resultante es más consistente, evita errores básicos y se adapta mejor a estados nuevos, reflejando una solución robusta y bien justificada.

Para las mejoras podría ser bueno aumentar el número de iteraciones en MCTS para obtener estimaciones de valor más estables. En cuanto al aprendizaje, ampliar la cantidad de episodios ayuda a mejorar la precisión de los valores Monte Carlo y disminuir la varianza. Incluir oponentes más variados también fortalecería la robustez del agente frente a distintos estilos de juego. Finalmente, aplicar una limpieza periódica de la memoria permite descartar estados con pocas visitas, reduciendo ruido y manteniendo únicamente conocimiento estadísticamente confiable. Y por último añadir al MCTS del entrenamiento el uso de la q_tables para que se tomen decisiones más informadas