Informe de Contribución Individual: Diseño Algorítmico y Robustez

Nombre: Diego Andrés Suárez Obando **Proyecto:** Agente Autónomo para Connect-4

Curso: Fundamentos de Inteligencia Artificial - 2025.2

Rol: Arquitectura Algorítmica y Gestión de Recursos

1. Resumen de Contribuciones Técnicas

Mi responsabilidad principal en el proyecto fue el diseño de la lógica de toma de decisiones del agente, enfocándome específicamente en la estabilidad del algoritmo Monte Carlo Tree Search (MCTS) y su adaptabilidad a las restricciones de ejecución de la plataforma Gradescope. A continuación, detallo mis dos aportes más significativos (Highlights).

Aporte 1: Estrategia de Simulación Híbrida (Hybrid Rollout)

Contexto y Desafío:Durante las pruebas iniciales, observamos que el MCTS estándar sufría de una convergencia lenta. La fase de simulación (*rollout*), al ser puramente aleatoria, introducía una alta varianza estadística ("ruido"). Esto provocaba que el agente, aunque robusto a largo plazo, pasara por alto amenazas tácticas inmediatas (como no bloquear un 3-en-línea del oponente) si no se realizaban miles de simulaciones, lo cual era inviable dado el límite de tiempo.

Implementación Técnica Para mitigar esto, diseñé e implementé una estrategia de simulación de dos fases dentro del método simulated hybrid. En lugar de una ejecución estocástica pura, el algoritmo ahora evalúa el estado del tablero con una heurística determinista durante los primeros 7 niveles de profundidad.

En esta fase "inteligente", el agente verifica activamente si existe un movimiento que otorgue la victoria inmediata (win_found) o si es obligatorio bloquear una victoria inminente del oponente (block_found). Solo si el estado no presenta condiciones críticas, el algoritmo transiciona a una selección pseudo-aleatoria ponderada. Esta arquitectura híbrida reduce drásticamente el horizonte de búsqueda efectiva, permitiendo al agente detectar victorias forzadas en menos de 0.1 segundos.

Evidencia de Código

- **Archivo:** policy.py
- **Método:** _simulate_hybrid
- **Commit:** feat: implement-hybrid-heuristic-rollout https://github.com/Diegosuar/Proyecto-Final/commit/3bdeff108a76f40c1fc5a42652edade66275ef03

Aporte 2: Gestión Dinámica de Tiempo y Tolerancia a Latencia

Contexto y Desafío: El desafío más crítico para la calificación fue el límite estricto de 600 segundos globales impuesto por el autograder. La implementación inicial utilizaba un bucle

fijo (for _ in range(N)), lo que resultaba en tiempos de ejecución impredecibles. Si el servidor de evaluación presentaba latencia, las iteraciones fijas excedían el tiempo asignado por turno, resultando en la descalificación del agente por *Timeout*.

Implementación Técnica Reestructuré el núcleo del bucle de decisión para que fuese "consciente del tiempo" (Time-Bounded). En el método mount, implementé una lógica de cálculo de margen de seguridad (buffer), donde el agente lee el time_out proporcionado por el servidor y se auto-asigna un límite interno reducido (ej. val - 0.05s).

Posteriormente, reemplacé el bucle de iteración fija por un control de flujo basado en el reloj del sistema dentro de _run_mcts, utilizando la condición while (time.time() - start_time) < self.time_limit. Esto transformó al agente en un sistema elástico capaz de maximizar el uso de recursos computacionales sin violar jamás las restricciones temporales, garantizando la entrega de una jugada válida incluso bajo condiciones de alta carga del servidor.

Evidencia de Código

- **Archivo:** policy.py
- **Métodos:** mount y act
- **Commit:** fix: dynamic-time-bounding-loop https://github.com/Diegosuar/Proyecto-Final/commit/c82f61987e146e15e1e4154f482878d7f434ffd2

2. Reflexión y Análisis de la Solución

Logros y Estabilidad:La combinación de la simulación híbrida con la gestión dinámica del tiempo resultó en un agente excepcionalmente robusto. En las pruebas de validación, el agente logró una tasa de victorias del 100% contra oponentes aleatorios y mantuvo un comportamiento estable sin excepciones de tiempo, obteniendo una calificación perfecta en las pruebas automatizadas. La heurística de "pánico" (verificación pre-búsqueda) actúa como una red de seguridad efectiva, eliminando errores no forzados en las etapas tempranas del juego.

Limitaciones Identificadas A pesar de la eficiencia lograda, el agente reconstruye el árbol de búsqueda desde cero en cada turno (root = Node(game)). Esto implica que toda la información estadística recolectada en el turno anterior se descarta, desperdiciando cómputo valioso. En partidas avanzadas, esto limita la profundidad máxima que el agente puede alcanzar, ya que debe "redescubrir" las mejores ramas repetidamente.

Propuesta de Mejora: Reutilización de Árboles (Tree Reuse) Para una futura iteración, la mejora de mayor impacto sería implementar la persistencia del árbol de búsqueda. En lugar de descartar la raíz, el agente debería identificar el nodo hijo correspondiente a la jugada realizada y promoverlo como la nueva raíz para el siguiente turno. Esto conservaría las estadísticas de visitas y victorias (visits, wins), permitiendo que la búsqueda se profundice progresivamente a lo largo de la partida, mejorando el rendimiento en el *endgame* sin costo adicional de tiempo.