Agente Inteligente para el Juego de Hex

Javier A. González Díaz
Facultad de Matemática y Computación

12 de abril de 2025

Resumen

Este informe describe el diseño e implementación de un agente inteligente para el juego de Hex, desarrollado como parte de la asignatura de Inteligencia Artificial. El agente utiliza una combinación de algoritmos de búsqueda como Minimax con poda alfa-beta, el algoritmo A* para estimar distancias mínimas, y una heurística basada en conexiones virtuales (puentes seguros) que asegura un juego estratégico y equilibrado.Además se utilizan técnicas de filtrado , conteo y ordenación para lograr mejores tiempos de ejecución.

1. Introducción

El juego de Hex es un juego de estrategia para dos jugadores, cuyo objetivo es conectar lados opuestos del tablero con fichas propias. Debido a su naturaleza determinista y su gran espacio de estados, es un candidato ideal para aplicar técnicas de Inteligencia Artificial y demostrar control de tiempo en tomas de decisiones sin sacrificar efectividad.

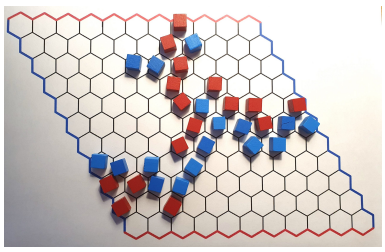

Figura 1: Tablero de HEX.

2. Algoritmo Minimax con Poda Alfa-Beta

El núcleo de decisión del agente es el algoritmo Minimax, que explora los posibles movimientos futuros para elegir el más prometedor. Para mejorar la eficiencia, se implementó poda alfa-beta, lo cual reduce el número de nodos evaluados sin afectar el resultado.

$$\text{Minimax}(estado, profundidad, \text{jugador}) = \begin{cases} \text{valor heurístico} & \text{si profundidad} = 0 \text{ o estado terminal} \\ \text{máx}(\text{Minimax}(hijo)) & \text{si es turno del jugador} \\ \text{mín}(\text{Minimax}(hijo)) & \text{si es turno del oponente} \end{cases}$$

3. Función de Evaluación

La función de evaluación combina múltiples factores para estimar la conveniencia de una posición:

$$f(s) = w_1 \cdot f_1 + w_2 \cdot f_2 + w_3 \cdot f_3$$

donde:

- f_1: diferencia entre el costo mínimo de camino del oponente y del jugador (calculado con A*),
- f_2: diferencia de conexiones virtuales seguras (puentes seguros),
- f_3: diferencia de cantidad de fichas colocadas.

Con los pesos:

$$w_1 = 0{,}8, \quad w_2 = 0{,}3, \quad w_3 = 0{,}2$$

4. A* para Costo del Camino Mínimo

El algoritmo A* se utiliza para estimar el costo mínimo que un jugador necesita para conectar sus lados del tablero. En Hex, el tablero puede modelarse como un grafo donde cada celda es un nodo conectado a sus seis vecinos inmediatos. Esta representación permite aplicar técnicas clásicas de búsqueda en grafos.

4.1. Cálculo del Camino Mínimo

Para cada jugador, se definen las -celdas fuente- como aquellas situadas en el borde inicial (superior o izquierdo, según el jugador), y el objetivo es alcanzar el borde opuesto (inferior o derecho).

El algoritmo A* combina el costo real acumulado $g(n)$ desde el inicio hasta el nodo actual, con una heurística $h(n)$ que estima la distancia restante hasta el objetivo. En este caso:

$$h(r, c) = \begin{cases} n - 1 - c, & \text{si el jugador conecta de izquierda a derecha} \\ n - 1 - r, & \text{si conecta de arriba a abajo} \end{cases}$$

Esta heurística representa una estimación optimista de la distancia, que asume que el jugador puede avanzar sin obstáculos hasta su meta. Esto garantiza que A* sea **admisible**, es decir, que nunca sobreestima el costo real.

Cada celda se evalúa con el siguiente criterio de costo:

- Si está ocupada por el jugador: costo = 0
- Si está vacía: costo = 1
- Si está ocupada por el oponente: costo = ∞

Este enfoque prioriza caminos ya parcialmente controlados por el jugador , interrumpe caminos construidos por el oponente y penaliza severamente las obstrucciones de este.

4.2. Utilidad en la Evaluación

El valor devuelto por A* representa la cantidad mínima de movimientos necesarios (en el mejor de los casos) para que el jugador conecte sus lados. Se compara con el costo estimado del oponente, y esta diferencia se utiliza como componente principal en la función de evaluación:

$$f_1 = \text{costo}_{\text{oponente}} - \text{costo}_{\text{jugador}}$$

Cuanto menor sea el costo del jugador respecto al del oponente, más prometedora es la posición para el agente.

5. Puentes Seguros (Safe Bridges)

Los *safe bridges* son patrones de conexión virtual que, aunque no estén ocupados, representan enlaces estratégicos que solo el jugador puede completar en futuras jugadas. Se utilizan para detectar ventaja posicional y se cuentan dinámicamente durante la evaluación de los nodos.

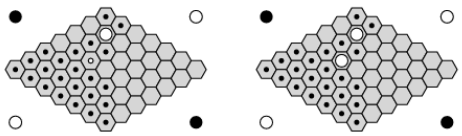

Figura 2: Ejemplo de puente seguro entre las dos fichas blancas.

El uso de *safe bridges* o conexiones virtuales es una técnica reconocida en entornos profesionales de Hex. Estos patrones permiten prever futuras conexiones seguras que el oponente no puede bloquear completamente en un solo turno. Esta capacidad de anticipación estratégica distingue a jugadores avanzados, y su inclusión en un agente automático refuerza la calidad de su juego posicional.

6. Filtrado y Selección Eficiente de Jugadas

El agente genera un conjunto reducido de movimientos relevantes en cada turno, evitando recorrer todo el tablero. Esto se logra mediante filtros centrados en las zonas activas del juego y en estructuras estratégicas como los puentes.Esta estrategia permite evitar considerar movimientos irrelevantes o lejanos al contexto estratégico actual, mejorando así el rendimiento en cada iteración de `minimax`.

La función `get_relevant_moves` combina:

- Vecinos vacíos de las fichas del jugador (para expandir conexiones).
- Vecinos vacíos de las fichas del oponente (para bloquear).
- Celdas que podrían formar *puentes seguros* con fichas propias.

Los puentes seguros se identifican cuando dos fichas no adyacentes comparten exactamente dos vecinos vacíos, lo cual permite prever conexiones virtuales resistentes a bloqueos. Las funciones `get_potential_bridges` y `bridge_neighbors` encapsulan esta lógica.

Para la apertura, se prioriza el centro del tablero y, si está ocupado, se elige un vecino aleatorio.

Gracias a esta estrategia de filtrado:

- Se reduce drásticamente el espacio de búsqueda.
- Se mantiene un enfoque táctico, tanto ofensivo como defensivo.
- Se mejora la eficiencia del algoritmo `minimax`, permitiendo mayor profundidad sin penalizar el rendimiento.