UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Inteligencia Artificial

Proyecto:

Agente que juega Othello

Equipo:

Anzures Reyes Miguel Alejandro

Montaño Bautista Marco Alan

Rivera López Jorge Erick

INTRODUCCIÓN:

La *Inteligencia Artificial* (IA) nos ayuda a resolver problemas que a una persona le llevaría un tiempo considerable y en caso de resolverlo correctamente tendría un amplio margen de error, al ser la IA un área de las *Ciencias de la Computación*, es de su interés el tratar de minimizar el tiempo en el cual resuelve dichos problemas, así como, el espacio en memoria que requiere para dicha solución, si consideramos un enfoque racional en la IA una de las formas para resolver dichos problemas es analizar y categorizar todas las posibles acciones que nos lleven a una conclusión satisfactoria, sin embargo, seguir este enfoque puede ser computacionalmente imposible depende del problema, además de un gran consumo en memoria, una solución a este problema es el uso de una serie de estrategias para tener un mejor y más efectivo manejo de los recursos, siendo el caso particular de las heurísticas aplicadas a algoritmos bien definidos como minimax o poda alfa-beta.

Adelante se detalla una implementación de un agente oponente al jugador humano en el juego de Othello haciendo uso del algoritmo Minimax con su correspondiente heurística.

EL PROBLEMA

Se considerará un modelo del juego de Othello además de dos jugadores, uno controlado por un Humano y otro por el Agente que estamos desarrollando, el problema consiste en este caso en diseñar al agente tal que sea capaz de jugar contra un adversario humano y realice acciones congruentes con el objetivo de ganar, haciendo uso del algoritmo de Minimax y una heurística.

EL AGENTE

Nuestra propuesta para resolver el problema, es un agente basado en modelos pues analiza cómo cambia el mundo alrededor de él y en base a esto decide qué resultado va a producir cierta acción, el agente en todo momento será capaz de ver todo el tablero, pues el entorno es finito y tiene que analizar cada posible jugada para determinar la mejor opción, el tablero es de 8x8 y se utilizará una

representación de árbol que dará al agente la capacidad de ver las jugadas hasta cierto nivel por adelantado.

A continuación se presenta el REAS del agente en cuestión

RENDIMIENTO	**ENTORNO**	**ACTUADORES**	**SENSORES**
Ganar el juego	Tablero de 8x8	Colocar fichas, Escoger mejor jugada	Visor de posiciones

EL ALGORITMO MINIMAX

Minimax es un método de decisión para minimizar la pérdida máxima esperada en juegos con adversario y con información perfecta, generalmente es implementado en un Árbol.

Un breve análisis del código de Minimax nos permite identificar lo siguiente:

1. Los parámetros necesarios son tres: un nodo raíz (de un árbol con las posibles configuración del juego), un entero positivo que indica la profundidad a buscar en el árbol y un valor booleano para indicar los estados MIN y MAX según el caso.
2. Se trata de una función recursiva.
3. Se distinguen tres casos en el código: el caso base cuando se desea obtener el valor de utilidad de un nodo hoja, el caso cuando se desea obtener el valor máximo de las nodos hijos (MAX) y el caso cuando se desea obtener el valor mínimo de los nodos hijos (MIN).
4. Se requiere de una función que evalúe los estados hoja (heurística).

El algoritmo explorará los nodos del árbol asignándoles un valor numérico mediante una función de evaluación (heurística), empezando por los nodos terminales y subiendo hacia la raíz. La función de utilidad definirá lo buena que es la posición para un jugador cuando la alcanza.

En la práctica el método minimax es impráctico excepto en supuestos sencillos pues realizar la búsqueda completa requerirían cantidades excesivas de tiempo y memoria, sin embargo, existen variaciones del algoritmo Minimax que son mucho

más manejables en nuestro caso particular usaremos el Minimax restringido a nivel 4 de profundidad.

LA HEURÍSTICA

La heurística que se utilizó fue simple, consiste en contar el número de fichas a favor que se obtienen al hacer una jugada.

COMPLEJIDAD

Minimax tradicional realiza una búsqueda DFS completa, ahora consideremos a **m** como la máxima profundidad del árbol de juego, teniendo **m** entonces hay **b** movimientos válidos, por lo tanto minimax tiene una complejidad en tiempo de $O(b^m)$ y en espacio O(bm) si se genera en una vez todas las posibles acciones, si solo se genera una es O(m).

COMPORTAMIENTO RESPECTO AL OBJETIVO DEL JUEGO

Considerando nuestra función heurística la cual se base en contar por cada posible acción la cantidad de fichas favorables menos las fichas no favorables, podemos afirmar que nuestro agente “juega” siempre a ganar pues escoge la opción donde haya más fichas favorables de su color, esta aunque es una estrategia simple y más apegada a cómo jugaría un oponente humano es una buena forma de ilustrar el funcionamiento del algoritmo minimax y un agente basado en modelos en general, además de que como se mencionó anteriormente es una buena simulación de un oponente humano que no tiene algún conocimiento previo del juego de othello.

EFICIENCIA DE RECURSOS

Como ya ha sido mencionado construir el árbol Minimax completo usaría demasiados recursos, tantos que tal vez inclusive vuelva nuestra propuesta no-implementable, por esto mismo nosotros consideramos una variación de Minimax acotado a cierta profundidad, en nuestro caso particular es de profundidad cuatro de esta forma se reduce grandemente la cantidad de recursos

de memoria y procesamiento que se utiliza para determinar cada acción, es decir reducimos nuestra complejidad en tiempo a O(b^4) y espacio O(b*4).

CONCLUSIONES.

Para este proyecto se tomó en consideración el algoritmo minimax en su forma original. Pero debido a que el árbol de las configuraciones del tablero tiene una profundidad de 60 y un factor de ramificación de entre 5 a 15 que representan los movimientos legales del jugador (según el libro) O(15”60) se descarto la idea.

Consideramos como mejora el algoritmo de poda alfa-beta que es la mejora al algoritmo minimax pero seguimos teniendo árboles de una gran profundidad.

Como solución definitiva optamos por utilizar el algoritmo minimax original pero creando una restricción en la construcción del árbol. Dicha restricción limita que un árbol tenga una profundidad superior a 4 niveles, lo que construye un árbol de profundidad de 4 y con un número de nodos 15¨4 = 50,625 que puede manejar una computadora en un tiempo razonable.

Esta solución requiere de una heurística que ya mencionamos anteriormente y consideramos es eficiente y representa un reto para el jugador.

En conclusión utilizamos el algoritmo de minimax o alguna de sus variantes debido a que es una solución óptima para juegos de 2 jugadores con un entorno totalmente observable, pero tuvimos que limitar su uso para poder procesar los cálculos necesarios, lo que influye en cómo juega el computador y también repercute en el hecho de que al no construir todo el árbol de decisiones no realiza un juego perfecto como debería ocurrir si se implementara minimax con el árbol completo.

BIBLIOGRAFÍA

- Russel S. & Norvig P.. (2010). Adversarial Search. En *Artificial Intelligence : A Modern Approach*(pp.162-164-165). Ciudad de México: Prentice Hall.
- Allis V.. (1994). *Searching for Solutions in Games and Artificial Intelligence.* octubre 11, 2017, de Universidad de Limburg Sitio web: https://project.dke.maastrichtuniversity.nl/games/files/phd/SearchingForSolutions.pdf