Inteligencia Artificial Proyecto 2: Othello

Integrantes del equipo:

- Nava Kopp Mariano Rafael:
  Correo: normanment@gmail.com
  Cuenta: 305186942
- Gaspar Velasco René Jesus:
  Correo: renejesusgv@ciencias.unam.mx
  Cuenta: 310285580
- Moreno de la Rosa Alan:
  Correo: c_m_a1990@ciencias.unam.mx
  Cuenta: 307235943

Repositorio del proyecto:

https://github.com/firecold100/Othello-IA-2018-1

Introducción:

El problema principal de este proyecto es el desarrollo de un agente que tiene con finalidad jugar el juego de mesa “Othello”, también conocido como “Reversi”, de manera inteligente. Para ello, aunque no hay forma de garantizar una victoria, se eligió desarrollar el agente basándonos en el algoritmo de MiniMax tradicional, para encontrar la mejor tirada posible computacionalmente. El agente se desarrolló tomando en cuenta dos heurísticas de juego, las cuales entregan cómputos diferentes y permite comparar las diferencias entre ellas.
La primera y menos eficaz otorga puntos aleatoriamente a las configuraciones dadas, la segunda, y que se utiliza por defecto consiste en elegir aquella jugada que le permita al agente tener el mayor número de fichas sobre el tablero, y minimizar el número de fichas del contrincante, adicionalmente se otorgan bonus si se tira cerca de una esquina (por estrategia). La estructura del proyecto se divide en 5 clases, siendo la principal la clase Reversi.pde que hace uso de las otras 4 clases; Tablero.pde, Casillas.pde, MyTreeNode.pde, Configuración.de, Tablero.pde; en orden de aparición.
En la clase Reversi.pde la función principal es la función mouseClicked() que inicia por ver si la casilla está vacía; de ser así el usuario pone una ficha y luego establece que es turno del agente intercambiando jugador. Posteriormente se

dibuja dibuja el tablero haciendo uso de la función draw(), después es turno del agente (jugador2) que verifica si es posible una jugar y usando la función MiniMax decide qué jugada es aquella que le permite tener más posibilidad de ganar y delimita el menor número de posibles jugadas a su contrincante.
En cualquier caso, si no existe una tirada válida, se hace uso de la función determinaGanador(); la cual cuenta el puntaje obtenido por ambos jugadores.

Definición Del Problema:

Se requiere desarrollar un agente que juegue Othello de forma inteligente; que sea capaz de tomar decisiones que lo guíen a una victoria frente al usuario.
El agente deberá ejecutar sus movimientos de forma rápida y sin utilizar recursos innecesarios, para esto se hace uso del algoritmo MiniMax tradicional, el cual garantiza cierta eficiencia en memoria a la hora de jugar, ya que la estructura de datos construida puede ser modificada a conveniencia.
El agente construye árboles de búsqueda de forma que se inspeccionen todas las posibles jugadas de forma ordenada dentro de un rango (profundidad); que debe buscar un equilibrio entre la eficiencia y uso de recursos, y la garantía de búsqueda apropiada para ejecutar una tirada inteligente.
Para ello sólo considera como profundidad el número de jugadas que se anticipan en cada decisión; lo que significa que un árbol de profundidad 1 sólo es útil para analizar una jugada a futuro.

Descripción de la propuesta e implementación:

Tipo de agente: El agente está definido para resolución de problema, en este caso, definido para derrotar a sus oponentes en el juego de Othello, está basado en utilidad ya que con la información de su entorno, usa sobre posibles escenarios la métrica(heurística) para calificar que tan agusto estará tomando dicha decisión, por lo tanto al comparar el resultado de posibles acciones, se tiene una calificación de utilidad que define el comportamiento del agente.

Tipo de agente: Jugador de Othello
Medida de rendimiento: determinaGanador(), juegoAcabo();
Ambiente: Tablero.
Actuadores: flip(), flipPorCopia(), getMovimiento()
Sensores: Acceso a la configuración del tablero en todo momento.

Preguntas:

1. ¿Qué es algoritmo minimax? ¿Cuáles son sus limitaciones?
   Es un algoritmo para toma de decisiones el cual se construye a partir de árboles, Y busca maximizar la ganancia obtenida en un escenario y minimizar la ganancia del oponente en escenarios subsecuentes.

   Sus limitaciones son, entre otras, el uso de recursos, ya que construir un árbol para cada escenario conlleva a factores muy altos de ramificación, y por lo tanto al uso de bastantes recursos. Otra limitación es que la eficiencia de las decisiones tomadas recaen completamente en las funciones de evaluación (heurísticas), por lo que es de suma importancia que dichas funciones evalúen de forma muy acertada los escenarios.

2. ¿Qué tipo de agente se propone utilizar? Se deben especificar las características del ambiente de trabajo, entorno y el agente. Indica sí utilizaste minimax tradicional, poda alfa-beta o alguna otra variante.

El agente utilizado se definió en la sección anterior, en este caso se tomo la decisión de usar minimax tradicional ya que el factor de ramificación de un tablero de 8 x 8 y con una profundidad acotada de 4 niveles, es suficiente para tener un agente bastante competitivo y los recursos computacionales no son tan altos.

3. De acuerdo con el agente propuesto ¿cuál es la complejidad teórica de tu propuesta (espacio y tiempo)?

Implementando minimax tradicional, la complejidad en tiempo está acotada por una función de ramificación que tiene como valores, el número promedio de tiros válidos elevado a la profundidad del árbol, denotado por O(t^p) donde t, es el valor promedio de tiros posibles y p es la profundidad.
Por defecto, la profundidad la hemos configurado para tener 4 niveles, y tomando en cuenta que en promedio se tienen 5 posibles tiradas, nos da una complejidad promedio de O(5^4), tomando en cuenta un tablero 8 x 8.

Respecto a la complejidad en espacio, está acotada por el número de tiros válidos multiplicando a la profundidad del árbol, ya que cada turno almacenamos la estructura, para cada posible tiro válido, y mantenemos el árbol hasta la profundidad definida, denotado por O(t*p),donde t, es el valor promedio
de tiros posibles y p es la profundidad.

Tomando en cuenta los valores comentados anteriormente, en promedio la complejidad en espacio nos da una evaluación de O(5*4).

4. Tomando como referencia la elección que hace tu agente ¿cómo es su comportamiento con respecto al objetivo del juego (juega a ganar o a no perder)? ¿su comportamiento siempre es adecuado?

El comportamiento del agente está definido para jugar a ganar, ya que siempre intenta maximizar su evaluación, se encarga de mantener siempre la delantera en el puntaje. No podemos garantizar que su comportamiento siempre es el adecuado, sin embargo si cumple con las expectativas definidas.

5. ¿Cuantos recursos consume tu propuesta para poder tomar una decisión?

Tomando en cuenta el análisis de complejidad, este enfoque conlleva a un consumo alto de recursos, sin embargo la definición de la profundidad de los árboles a crear permite que la mayoría de sistemas de cómputo actuales puedan tomar una decisión en un tiempo muy aceptable. Con algunas mejoras como poda alfa-beta se conseguiría disminuir el consumo de recursos, pero dado que el espacio de configuraciones no es tan grande, las mejoras no son muy significativas, sin embargo si el espacio crece dada la profundidad definida en los árboles,si se notará una mejora sustancial.

Referencias bibliográficas:

Russell, S., Norvig, P., Corchado Rodríguez, J. & Joyanes Aguilar, L. (2011). Inteligencia artificial. 2a. Edición. Madrid: Pearson Educación. Capítulo 2. REDcientifica - Ciencia, Tecnología y Pensamiento. [online] Redcientifica.com. http://www.redcientifica.com [Leído el 5 Sep. 2017].

-Minimax Complexity Analysis
https://cis.temple.edu/~vasilis/Courses/CIS603/Lectures/l7.html