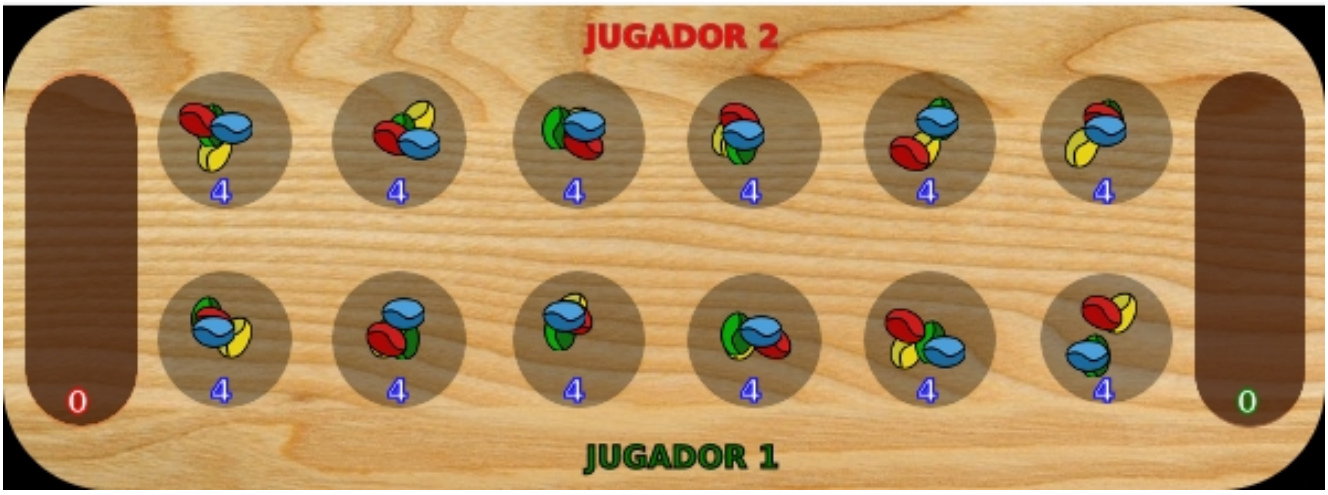
INTELIGENCIA ARTIFICIAL
PRÁCTICA 3



Agentes en entorno con adversario

Memoria de la Práctica

Alumno: Daniel Bolaños Martínez
DNI: 76592621-E
DGIIM.

1. Objetivo

El **objetivo** de esta práctica es desarrollar un agente inteligente capaz de jugar contra otro adversario en el juego *Mancala* e intentar obtener la victoria frente a él.

Se utilizarán las clases proporcionadas para programar el agente pedido y la interfaz gráfica para visualizar las acciones de los jugadores.

2. Técnica de Búsqueda

El problema que se nos presenta, es un **Juego de Adversario Competitivo** donde tendremos que enfrentarnos a un adversario, ya sea humano o no con nuestro agente y hacer todo lo posible para ganarle.

Usaremos la clase *GameState* que proporciona todas las utilidades necesarias para obtener los datos de cada turno en un tablero de *Mancala* y crearemos un árbol de exploración de juegos para hacer que nuestro agente pueda obtener la victoria.

He utilizado una **estrategia recursiva** para crear el árbol de estados del juego, por lo que no he necesitado almacenar los diversos nodos en ninguna estructura de datos específica.

El objetivo de mi implementación es por tanto, aplicar recursivamente un **algoritmo de búsqueda** en un árbol de estados sobre los distintos tableros que serán de la clase *GameState* y que simularán los distintos nodos de nuestro árbol.

Finalmente, necesitaremos elegir en cada turno (del jugador que nos toque desempeñar), la acción que lleve a mejores soluciones en el árbol de estados, por lo que el algoritmo nos deberá devolver un valor que nos servirá para elegir entre los 6 mejores hijos de nivel 1 en el turno actual y que nos ofrezcan una mejor solución dependiendo de la valoración de sus hijos a una profundidad escogida, en mi caso 11.

3. Algoritmo (Poda Alfa-Beta)

El algoritmo utilizado en la búsqueda de los distintos nodos ha sido la **poda alfa-beta**. Como vimos en teoría, la poda alfa-beta mejora al algoritmo **min-max**, algoritmo por excelencia en la búsqueda en árboles de juegos competitivos donde tenemos que maximizar nuestro resultado y minimizar el de nuestro contrincante.

La **poda alfa-beta** implementa como su propio nombre indica, la poda de ramas del árbol con el objetivo de reducir el coste computacional del algoritmo **min-max**.

He implementado el algoritmo en mi código como una función que tiene como parámetros: el nodo tablero, los valores alfa y beta, la profundidad en el árbol y el jugador del turno actual.

Guardaré en una variable global *jugadorBot*, el turno desempeñado por nuestro agente en la partida que se esté ejecutando en ese momento, con el fin de llamar a la función alfa-beta con el

parámetro del jugador actual simulado por nuestro agente.

Si el nodo introducido es un nodo final (que lo sabremos con la función *isFinalState()*) o es un nodo hoja (la *profundidad* es 0), obtendremos una valoración de ese nodo utilizando una **heurística** que explicaré más adelante. En caso contrario, aplicaremos de nuevo para cada hijo del nodo pasado como parámetro, el algoritmo de poda alfa beta (con *profundidad -1*) maximizando el valor de alfa si el jugador es el mismo que el de nuestro agente o minimizando beta si es el del oponente en ese turno.

Podaremos las ramas del árbol haciendo un *break* dentro de la función cuando el valor de alfa sea mayor o igual que el de beta. De esta forma, eliminaremos esa parte del árbol haciendo que ese nodo, no siga expandiendose sobre sus hijos.

Finalmente en la función *nextMove()*, elegiremos el mejor movimiento en cada turno haciendo una búsqueda del mejor nodo valorado sobre los 6 primeros por el algoritmo. Haremos un bucle iterando sobre los 6 posibles movimientos y elegiremos el de mejor valoración.

4. Heurística

Para asociar un valor a cada nodo cuando uno de estos es nodo hoja o final, he realizado **dos heurísticas** diferentes dependiendo de si somos el jugador 1 o 2 con el objetivo de mejorar el puesto en la liga.

En ambos casos la función heurística devuelve como resultado **heurística a favor** menos la **heurística en contra**, lo único que varía es la definición de estos dos valores según el turno de nuestro agente.

Si nuestro agente juega como **jugador 1**, las heurísticas serán la suma de 500 (en el caso de que ese jugador sea ganador del juego en ese nodo), 10 veces el número de fichas del granero y 10 veces el número de casillas vacías del oponente.

De esta forma el agente como **jugador 1**, hará mejor valoración de los nodos que le aporte más fichas a su granero y más casillas vacías a su contrincante con el objetivo de que el rival se quede lo más rápido sin fichas y puedas llevarte todas las restantes a tu granero.

Si nuestro agente juega como **jugador 2**, la heurística intenta además de ganar, perder por el menor número de fichas posibles, para evitar que en caso de empate en la liga, pueda superar a otros bots por menor número de puntos en contra.

Para ello, el agente como **jugador 2**, valorará mejor los nodos que más semillas le ofrezcan a él en todas las casillas y menos al rival en todas las casillas, de esta forma, aunque gane el rival, el número de semillas en contra, será el menor posible. También sumará 500 puntos a la heurística del jugador que gane en el nodo valorado, al igual que la heurística primera.