UNIVERSIDAD
DE GRANADA

Inteligencia Artificial

Memoria Practica 3

Antonio Marfil Sánchez

ESTRUCTURA

```
struct celda{
        State estado = State(8,8);
        int best_fitness;
        int best_movement;
        int movimiento = 0;
        int prof;
        bool terminal = false;
        bool nodo_padre = false;
        int padre;
        bool ganador = false;
};
```

La estructura celda se compone:

- **estado** (8,8) que es una variable de tipo State, a la que se le pasara el estado del juego
- **best_fitness**, es un valor que corresponde al mejor coste del movimiento.
- **best_movement**, valor entero que es la columna correspondiente al movimiento del jugador.
- **movimiento**, valor entero que es es lo movimiento actual.
- **prof**, valor entero que representa la profundidad.
- **terminal**, valor condicional para saber si es un nodo terminal.
- **padre**, valor del nodo inicial.
- **ganador**, valor condicional para saber si hay ganador.

MINMAX

- **void minMax(struct celda &celda)**

  Esta función es la que se encarga de hacer todos los cálculos posibles para conocer el mejor movimiento. Se le pasa por referencia la estructura **celda** para que según sus atributos podamos sacar la información que necesita saber para realizar la mejor jugada.

Se crea un **vector<struct celda> arbol** que será el arbol que se analizará en el recorrido de posibilidades del juego, y se inicializan a **0** variables como: **contador, cantidad_nodos, mejor_movimiento**. Además se crea una variable **prof a la que se le asignará el valor de profMax=4**.

Dentro de un bucle **while(proMax>0)** se realizan las siguientes acciones:

- 1º se reduce -1 a prof.
- **Si el contador está 0**, se crea un vector<int> con las columnas disponibles para colocar ficha y se aumenta el contador a 1. Se crean tantos nodos hijos como columnas disponibles y se inicializan cada una de ellas su fitness a 0 y su movimiento al de la columna correspondiente además de asignarlo al padre que tiene valor 0. Si se da el caso de que la profMax=1 será el nodo.hijo un terminal por lo que se le da valor true. Una vez todo está preparado se le añade al árbol de estados.

  Si el contador no esta a 0, Si se da el caso que el nodo padre es ganador se le asigna ganador = true, si no; se seguirá creando nodos hijos y asignándoles al correspondiente padre como en el primer caso donde el contador está a 0.

- Se reduce la profMax a -1.

Ahora toca calcular los max y min de el árbol con todos los nodos posibles generados con un **bucle for que tomara la condición i<size**, donde **size es el número de nodos** que tiene el árbol. este bucle realizará:

- Si el nodo[i] es terminal o ganador se le introducirá en best_fitness el cálculo de la heurística.
- Según el número introducido en la máxima (profundidad máxima auxiliar) se calcular si es par o no. Si es par entonces la profundidad que sea par se le realizará el máximo, si es impar la profundidad se le realizará el máximo. Sirve para diferentes profundidades tanto pares como impares. Para maximizar o minimizar se comparan los best_fitness del nodo padre con el hijo. Si la comparación es correcta entonces el best_fitness del padre tomara el del hijo. Con esto se consigue escalar el mejor resultado para el movimiento que deseamos.

El siguiente paso es recorrer lo 8 primeros nodos sin contar el nodo padre principal y ver si alguno de ellos contiene un valor mínimo muy grande para poder tapar al adversario. Si no hay ningún valor negativo muy alto entonces se comprueba si hay uno muy alto para conseguir la victoria. Este mejor movimiento obtenido se le asigna a nodo, que es el atributo pasado por referencia.

En caso de que no se actualice el nodo.best_movement, es decir que su valor sea 0, de las posibles columnas vacias escogerá una de ellas de manera random.

- **int calcularCoste(State state, int valorJugador)**

Esta función se encarga de calcular el número de hueco que le falta a la ficha de un jugador (coste) para completar una línea de 4. Si a la ficha del jugador 1 le faltan 2 huecos para completar una línea el coste será de 2. Cuenta las posibles líneas tanto verticales, horizontales y diagonales.

- **int heuristica(State state, int player, int prof)**

La heurística comprueba si el state que se le ha pasado tiene ganador y si el ganador corresponde al jugador en cuestión devuelve un valor positivo muy grande y negativo en caso contrario.

En el caso de que no haya ningún ganador, según la profundidad se modifica el valor de player y se obtiene quien sería su enemigo. Con estos valores para los distintos jugadores se calcula la diferencia de los costes entre ello para devolverlo y evaluarlo en minMax.