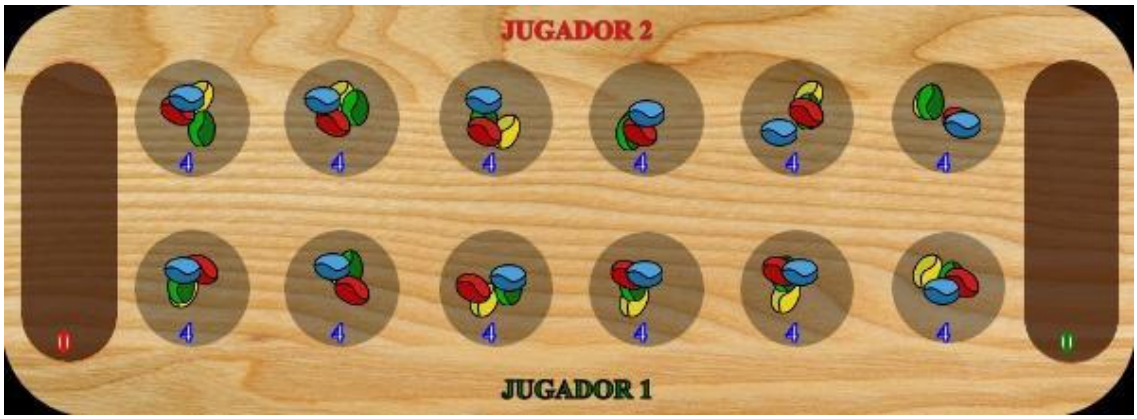
INTELIGENCIA ARTIFICIAL

Práctica 3

Agentes en entornos con adversario



Francisco Ruiz Adán

1. Introducción a la Práctica

Esta práctica consistía en la creación de una Inteligencia Artificial implementada en forma de un agente deliberativo desplegado en un entorno multi-agente competitivo, en el que se encuentran dos agentes con objetivos totalmente opuesto, los cuales realizan una acción por turnos.

En esta práctica se ha desarrollado un agente inteligente capaz de jugar a un juego denominado *Mancala* ya sea contra otro agente o una persona a la cual he nombrado como *Stark*.

2. Métodos Modificados y Nuevos

Se ha modificado el método:

```
Move Stark::nextMove(const vector<Move> &adversary, const GameState
&state)
```

Este método ahora devolverá la casilla que debe elegir nuestro agente para llevar a cabo una partida de manera que este ‘’pueda ganar’’.

```
Move Stark::nextMove(const vector<Move> &adversary, const GameState &state)
{

    Move movimiento = M_NONE;
    GameState estado_actual = state;
    double alfa = -9999999999.0;
    double beta = 9999999999.0;
    int valor, accion;

    valor = Poda_AlfaBeta(estado_actual, 10, accion, alfa, beta);
    movimiento = (Move)accion;

    return movimiento;
}
```

En cuanto a los nuevos métodos se han creado los siguientes:

```
double Stark::Poda_AlfaBeta(const GameState &estado, int prof, int
&accion, double &alfa, double &beta)
```

Calcula el movimiento que debe realizar nuestro agente. Como parámetros le pasamos el estado actual del juego, una profundidad (10 por defecto), una acción (M_NONE por defecto) y dos enteros. Es el algoritmo que utiliza el agente.

```
double Stark::Heuristica(const GameState &estado)
```

Calcula el valor heurístico dado un estado actual del juego. Este método es fundamental para la resolución del problema puesto que según el valor que devuelva este método nuestro algoritmo elegirá una opción u otra.

3. El Algoritmo

Para la elección del mejor movimiento se ha utilizado el algoritmo de la poda alfa-beta, una técnica de búsqueda que reduce el número de nodos evaluados en un árbol de juegos por el algoritmo Minimax. Se trata de una técnica muy utilizada en programas de juegos entre adversarios.

La poda alfa-beta toma dicho nombre de la utilización de dos parámetros que describen los límites sobre los valores hacia atrás que aparecen a lo largo de cada camino:

- α es el valor de la mejor opción hasta el momento a lo largo del camino para MAX, esto implicará por lo tanto la elección del valor más alto.

- β es el valor de la mejor opción hasta el momento a lo largo del camino para MIN, esto implicará por lo tanto la elección del valor más bajo.

Esta búsqueda alfa-beta va actualizando el valor de los parámetros según se recorre el árbol. El método realizará la poda de las ramas restantes cuando el valor actual que se está examinando sea peor que el valor actual de α o β para MAX o MIN, respectivamente.

La eficacia de la poda alfa-beta depende del orden en el que se examinan los sucesores, es decir, el algoritmo se comportará de forma más eficiente si examinamos primero los sucesores que probablemente serán los mejores. Si esto pudiera hacerse, implicaría que la alfa-beta solo tendría que examinar O ($b^{d/2}$) en lugar de O (b^d) de Minimax

```
double Stark::Poda_AlfaBeta(const GameState &estado, int prof, int &accion, double
&alfa, double &beta)
{
    GameState hijo;
    double mejorValor;
    bool poda = false;
    Player stark = this->getPlayer();

    //Si la profundidad es 0 o es un nodo final devolver la heuristica
    if (prof == 0 || estado.isFinalState())
    {
        int valor;
        valor = Heuristica(estado);
        return valor;
    }

    //Si somos el jugador 1 jugamos como MAX
    if (estado.getCurrentPlayer() == stark)
    {

        mejorValor = alfa;

        //Vamos obteniendo todas las opciones posibles del estado actual
        for (int i = 1; i < 7 && !poda; ++i)
        {
            hijo = estado.simulateMove((Move)i);
            int otra = i;
            double valorHijo = Poda_AlfaBeta(hijo, prof - 1, otra, mejorValor, beta);

            //Si el valor del hijo es mejor que el que ya llevamos elegiremos esa
opcion (el hijo es mas viable)
            if (valorHijo > mejorValor)
                accion = i;

            //Alfa sera por tanto el maximo entre el mejorValor que llevamos y el
obenido en el hijo
            mejorValor = max(mejorValor, valorHijo);

            //En el caso de que beta sea menor a nuestro alfa realizamos la poda
(break)
            if (beta <= mejorValor)
                poda = true;
        }
    }
    else //En caso contrario jugaremos como MIN
    {

        mejorValor = beta;
```

```
        //Vamos obteniendo todas las opciones posibles del estado actual
        for (int i = 1; i < 7 && !poda; ++i)
        {
            hijo = estado.simulateMove((Move)i);
            int otra = i;

            double valorHijo = Poda_AlfaBeta(hijo, prof - 1, otra, alfa, mejorValor);

            //Si el valor del hijo es mejor que el que ya llevamos elegiremos esa
opcion (el hijo es mas viable)
            if (valorHijo < mejorValor)
                accion = i;

            //Beta sera por tanto el minimo entre el mejorValor que llevamos y el
obenido en el hijo
            mejorValor = min(mejorValor, valorHijo);

            //Si beta ess menor a alfa realizamos la poda (break)
            if (mejorValor <= alfa)
                poda = true;
        }
    }

    return mejorValor;
}
```

3.1. La Heurística

La elección de la mejor opción posible se realiza a través de un valor heurístico. Para ello se ha creado una función que va calculando este valor dado un estado actual del juego. Este valor se determina a través de una serie de comprobaciones las cuales van sumando puntos o restando puntos a una variable que ira guardando el resultado.
En concreto se realizan estas comprobaciones:

```
Primero comprobar si obtengo turno extra
      Si obtengo sumar puntuación por 10

Si no
Comprobar si puedo dar una vuelta
      Si doy la da comprobar si ultima ficha cae en casilla vacía
            Cae en vacía: sumar puntación por 2
            Sino comprobar si cae en una que de turno extra
                  Si da turno extra sumar puntuación por 2
                  Sino restar puntuación
Si no da vuelta
      comprobar si le evita al oponente un turno extra
            Si lo hace: sumar puntuación por 2
      Comprobar si mi movimiento le da al oponente un turno extra
            Si lo hace: restar puntuación
```