Universidad de Granada

GRADO EN INGENIERÍA INFORMÁTICA

Práctica 3
Inteligencia Artificial

Memoria de la realización del bot Mancala

Fernández Fernández, Sergio

GRUPO B2

Profesor: Romero Zaliz, Rocio Celeste

Granada, mayo de 2018

En esta memoria voy a proceder a realizar una explicación acerca de como he realizado mi bot jugador del juego mancala.

Para comenzar en el fichero Wilson.h he declarado tres variables en la clase Wilson que hereda de la clase bot. Estas variables que he declarado son **profundidad**, variable donde se almacena un entero, el cual es el limite de la profundidad del árbol que recorre el algoritmo de la poda alfa-beta, inicialmente su valor es 9, inicial izado en el constructor. La segunda variable que he declarado es **current_player**, la cual almacena el jugador actual que es mi bot. La última variable declarada es **opponent_player**, la cual almacena el jugador contrario si actúa como jugador 1 o como jugador 2.

Para calcular los movimientos posibles que puede realizar el jugador en un estado determinado del juego he creado una función privada llamada **movimientosPosibles**, la cual recibe como parámetros el estado actual del juego. Esta función devuelve un vector de Move, el cual almacena los movimientos posibles que puede realizar el jugador en este determinado estado. Si una casilla no tiene semillas se descarta y no se introduce en el vector de movimientos.

La segunda función que he creado es la llamada **heuristics**, la cual recibe como parámetros el estado actual del juego y un jugador de tipo player. Esta función lo que realiza es sumar 25 (número de puntos mínimos para ganar una partida) con el número de puntos del jugador pasado como parámetro. La función devuelve dicha suma. Posteriormente en el algoritmo de poda alfa-beta, llamo a la función con current_player y con opponent_player y realizo la diferencia. Esta es mi forma de calcular la heurística de mi bot, maximizando mis puntos y minimizando los del contrario.

En el algoritmo de poda **alfa-beta**, si es un estado final del juego o la profundidad es 0, se calcula la heurística como he explicado en el párrafo anterior. En caso contrario se realiza el algoritmo explicado en teoría, en el cual creo una lista de movimientos posibles llamando a la función anterior movimientosPosibles, y en un bucle recorriendo todos los movimientos posibles, creo un estado hijo del juego llamando a la función simulateMove con el movimiento de la lista escogido en esa iteración. Con el algoritmo seleccionamos el mejor movimiento y llamamos de nuevo recursivamente a la función poda alfa-beta disminuyendo la profundidad utilizada. La función recibe como parámetros un estado del juego, la profundidad, un movimiento y los valores de alfa y beta, que son los valores que devuelve la función dependiendo del caso.

Por último la función donde se llama al siguiente movimiento del bot es la función **nextMove**, que recibe como parámetros un vector de movimientos y un estado del juego. En esta función inicializo los valores de alfa y beta a menos infinito y más infinito respectivamente. El current_player lo inicializo llamando a la función getCurrentPlayer y el opponent_player lo inicializo con el valor contrario al que devuelva la función anterior. Aquí se realiza una llamada a la función podaAlfaBeta y dicha función devuelve el movimiento seleccionado por el algoritmo.