**Memoria**

En esta memoria explicaré mi práctica entregada de Mancala. Comenzaré explicando el diseño del estado para realizar la búsqueda. Seguiré con una explicación del algoritmo implementado y terminaré comentando la heurística utilizada.

El diseño de mi estado ha sido la estructura de datos GameState ya implementada. Decidí mantener esta estructura ya que lo que más tiempo consumía era simular los nuevos estados y no obtener los datos de los estados. Aunque implementase una nueva estructura de datos, está no me ofrecería ventaja ya que los nodos GameState ya han sido creados. Sin embargo, he de comentar que si que he utilizado varias variables auxiliares que me ayuda a lo largo de la implementación. Las declaré como globales para aumentar la rapidez del código (aunque puede que no sea lo más correcto). A lo largo de la memoria las iré explicando.

El algoritmo que he utilizado ha sido poda α-β. He implementado mi algoritmo como una función recursiva a la cual pasamos: el nodo estado, la profundidad, un bool indicando si es nuestro turno, alfa y beta. En el código podemos ver tres partes principales. Un *if* al principio que observa si tiene que devolver el valor, y un *if-else* indicando si es nodo MAX o MIN. Comentaré lo relativo a devolver el valor (primer *if*) en la heurística.

Respecto al *if-else* es la típica estructura que veríamos en cualquier algoritmo de poda. Este tiene una estructura bastante simple. Comenzamos iniciando un valor *best* que será con el que compararemos alfa/beta. Con un bucle *for* sacamos los 6 valores posibles del estado actual (comprobando antes que se puede hacer un movimiento en esa posición). Con este nuevo estado llamamos a la función de forma recursiva. Esta devolverá un valor con el cual actualizaremos los valores de *best* y el alfa/beta actual. Tras esto, comprobamos que los valores de alfa y beta no se crucen y continuamos con la siguiente iteración. En el caso que se crucen, he utilizado un *break* para salir del bucle y devolver el valor correspondiente.

Debo comentar una peculiaridad de la parte de nodo MAX. El objetivo final de esta implementación es conseguir el mejor movimiento posible. Sin embargo la poda α-β devuelve un valor heurístico y no un movimiento. Por esto, ya que cuando estemos en un nodo MAX significa que “nosotros” estamos jugando, he introducido un condicional luego de actualizar los valores de alfa y *best*. Antes de llamar por primera vez a la función, asignamos a una variable global *b* el valor inicial de *best,* esta nueva variable tendrá el valor del nodo padre. Este condicional comprueba lo primero que la profundidad actual sea 0. Entonces, en el caso de que el valor del nodo padre (*b*) haya cambiado, guarda el movimiento actual en una variable global *sig*, la cual será la que devuelve la función *nextMove*. Con esto, conseguimos que cada vez que el nodo inicial cambie de valor (al actualizarse con el valor devuelto de uno de sus sucesores), el movimiento sea guardado para luego poder devolverlo.

Comenzaré ahora a explicar mi heurística, en donde está la verdadera inteligencia del programa.

Como he explicado antes el primer *if* dentro de mi función sería la condición de parada de mi programa. Esta se compone de tres subcondiciones, si cualquiera de ellas se cumple, se calcula el valor del nodo actual y se devuelve en vez de seguir. Una de la básicas es, si el nodo actual es un estado final, es obvio que no podemos seguir obteniendo hijos y debemos calcular el valor. Otra de las subcondiciones es que hayamos explorado más de 170 mil nodos. Con una variable global (*exp*), vamos manteniendo un registro de cuántos nodos hemos explorado. Si pasa este limite, sabemos que no vamos a generar más sucesores en ningún caso. La última subcondición, es si hemos llegado al límite de profundidad. En el primer movimiento ( y solo en el primero ), el límite está en profundidad es 10. Tras este, el límite se actualiza a 11. Además, antes de llamar a la función recursiva por primera vez en cada turno, comprobamos cuántos hijos tiene el nodo inicial. Si este, es menor o igual a 4, aumentamos la profundidad máxima a 12 para este turno. Esto nos permite aumentar un poco la profundidad al saber que el tenemos menos sucesores en el nodo inicial y, por tanto, el tamaño del árbol va a ser mucho menor.

Comentaré al función heurística la cual da valores a los nodos. Comenzar diciendo que utilicé variables globales para ahorrar el gasto en declararlas, siempre asegurando que al comienzo del cálculo las iniciamos a 0. Escribiré lo que devuelve mi función y daré una lista de las variables y su significado:

*Total2 + Total - TotalAdv + Ex - Rob - Ex2*

* *total2* = semillas en mi granero
* *total* = semillas en mi campo
* *totalAdv* = semillas en granero contrario
* *ex* = puedo realizar un movimiento extra
* *rob* = semillas que el contrario me puede robar
* *ex2* = el contrario puede realizar un movimiento extra

Comenzar comentando que los tres primeros valores son los que van a guiar la heurística y serán los que diferencien los valores parecidos. Los últimos tres valores han servido para aumentar el número de nodos podados y aumentar la eficacia de la búsqueda. Una pequeña especificación es que el le he dado un pequeño aumento al valor de las semillas de la posición 1 (más a la izquierda). Quise hacer esto ya que, al intentar acumular las semillas más a la izquierda, hay más posibilidades de que haya más semillas en mi campo al terminar la partida y vayan a mi granero al final. Aunque parezca solo un pequeño cambio, en algunas partidas a cambiado el resultado final de forma radical. También comentar, que para mejorar la eficiencia de los cálculos, ya que se realizan muy a menudo, he desenrollado los bucles. Aunque hace que el código sea menos legible y tenga algo más de repetición, permite al compilador asegurar que no hay dependencias y aumentar la eficiencia.

Por último, decir que para cada uno de los valores de la heurística tiene un coeficiente por el cual multiplicamos para tenerlos más o menos en cuenta. Estos valores modifican la conducta de la inteligencia, haciéndola más/menos agresiva, intentando perder por lo mínimo o ganar por más diferencia, etc..