Practica 3

MEMORIA DE PRACTICAS

YUNHAO LIN

INTRODUCCION

El objetivo de esta práctica es diseñar e implementar un agente deliberativo para el juegodesconecta-4 boom en la cual nuestro objetivo es hacer que nuestro adversario haga un conecta 4.

Para ello emplearemos la teoría vista en clase y los conocimientos de c++ previos que tengamos.

IMPLEMENTACION

Para empezar, el guion de practicas nos recomendaban implementar primero el minimax. Siguiendo el seudocódigo que se puede encontrarse fácilmente en la Wikipedia, he hecho una adaptación de ese seudocódigo para crear la implementación final.

Una vez logrado el minimax, el siguiente paso era lograr una poda alfa beta, para ello seguí el mismo proceso anteriormente descrito.

No me voy a extender en el funcionamiento de ambos métodos ya que se ha visto extensivamente en clase.

Los problemas con los que me he encontrado en esta primera fase es navegar hasta los siguientes hijos/movimientos. Esta parte se resolvió con el método GenerateAllMoves en el cual nos devolvía el numero de hijos/movimientos y cuales eran (almacenados en un vector Enviroment)

IMPLEMENTACION DE LA HEURISTICA

En esta parte, el objetivo era darles una puntuación a los diferentes movimientos, para ello valoramos las fichas seguidas que tienen en las casillas adyacentes (izquierda, derecha, arriba, abajo y diagonales).

Para esta parte, he intentado dos maneras, la primera, dada una posición i, j valoraba cuantas de las mías tenia cerca, y dependiendo de las que tenia cerca, le daba una puntuación u otra. Si es el enemigo le doy mas puntos, y si es mía le doy menos puntos, para que no valore ese movimiento. Sin embargo, mis intentos para hacer funcionar este método fueron infructuosos. Por lo cual me decante por un nuevo acercamiento. Esta vez, divido cada parte en funciones. Por lo tanto, tengo una parte que me devuelve las fichas que tengo juntas en una fila.

```
//Funcion que devuelve el valor de una fila del tablero para un jugador x
double heuristica_filas(const Environment &estado, int jugador, int fila){
    double resultado = 0.0;
    int fichas_juntas;


    // Para la fila dada, veo cuantas fichas juntas tiene
    // Para eso recorro las columnas, si veo que es casilla mia (del jugador) aumento fichas_juntas
    for(int j = 0; j < 7; j++){
        // Si
        if(estado.See_Casilla(fila, j) == jugador || estado.See_Casilla(fila, j) == jugador + 3){
            fichas_juntas++;
        }
        else{
            if(fichas_juntas > resultado)
                resultado = fichas_juntas;
            fichas_juntas = 0;
        }
    }

    return resultado; // resultados posibles 0 1 2 3, que nos indica cuantas fichas juntas tenermos, por lo tanto
    // cuanto mas, peor

}
```

Para el caso de las columnas seria idéntico, solo que se recorre las filas.

Una vez implementado estas dos partes, los casos que nos faltarían por valorar seria las diagonales:

```
double heuristica_diagonal_izq(const Environment &estado, int jugador, int col, int fila){
    int fichas_juntas = 0, resultado = 0, contador = 0;
    for( int i = fila, j = col; i < 7 && j >= 0 ; i++,j-- ){
        if( estado.See_Casilla(j,i) == jugador || estado.See_Casilla(j,i) == jugador + 3 ){
            fichas_juntas++;
            contador++;
        }
        else if( estado.See_Casilla(j,i) == 0 ){
            contador++;
        }
        else{
            if ( contador >= 4)
                resultado = fichas_juntas;
            contador = 0;
            fichas_juntas = 0;
        }
    }
    if(contador >= 4)
      resultado = fichas_juntas;


    return resultado;
}
```

Para valorar las diagonales lo que hago es comprobar si las adyacentes son mías, pero además cuento los espacios en blanco que hay. Si el contador supera 4 eso quiere decir que puede haber movimientos en los que puedo hacer un 4 en raya, y por tanto me perjudique, pero si no hay “espacio” para que haga cuatro en raya, no me perjudicaría, y devuelvo 0.

Para el caso de las diagonales a la derecha seguiríamos una lógica muy parecida.

Finalmente reúno todas las puntuaciones previas de la siguiente manera:

```
double heuristica(int jugador, const Environment &estado){
    double puntuacion = 0.0;
    for(int i = 0; i < 7; i++){
        puntuacion = puntuacion + heuristica_filas(estado, jugador,i) + heuristica_diagonal_izq(estado, jugador,7-i-1,0)
        + heuristica_cols(estado, jugador,i) + heuristica_diagonal_dch(estado, jugador,0,i)
        + heuristica_diagonal_izq(estado, jugador,0,7-i-1) + heuristica_diagonal_dch(estado, jugador,i,0);
    }
    return puntuacion;
}
```

Por lo tanto, según esta heurística, una puntuación mas alta, significa un tablero más malo.

VALORACION DE LAS PUNTUACIONES

```
// Funcion heuristica (ESTA ES LA QUE TENEIS QUE MODIFICAR)
//Devuelve el valor heuristico del nodo aka devolver nodo
double Player::miValoracion(const Environment &estado, int jugador){
    int ganador = estado.RevisarTablero();

    if (ganador==jugador)
       return 99999999.0; // Gana el jugador que pide la valoracion
    else if (ganador!=0)
            return -99999999.0; // Pierde el jugador que pide la valoracion
    else if (estado.Get_Casillas_Libres()==0)
            return 0;  // Hay un empate global y se ha rellenado completamente el tablero
    else{
        int otro_jugador;
        if(jugador == jugador_)
            otro_jugador = (jugador_ +1)%2;

        double suma = 0.0; //= heuristica(estado, 0, 0);
        // for (int i=0; i<7; i++)
        //   for (int j=0;j<7; j++){

        //      if(jugador == jugador_){
        //       if (estado.See_Casilla(i,j)!=0 && suma > heuristica(estado,i,j) )
        //         suma = heuristica(estado,i,j);
        //      }else{
        //          if (estado.See_Casilla(i,j)!=0 && suma < heuristica(estado,i,j) )
        //          suma = heuristica(estado,i,j);
        //      }

        // }

        suma = heuristica(otro_jugador, estado) - heuristica(jugador, estado);
        return suma;
    }

}
```

Si no se hay ganador y además quede espacios libres en el tablero, devuelvo la suma de las heurísticas.

Si la suma sale negativa significa que jugador tiene peor tablero, y si positivo, al contrario.

CONCLUSION

En esta practica he aprendido a implementar la poda alfa beta (además del minimax), con la cual, a partir de una serie de valores me da el mejor de ellos. Por lo tanto, nuestro agente, en cada paso intenta hacer la mejor acción posible entre las disponibles. Y por lo tanto creo que he cumplido con los objetivos por la cual estaba diseñada esta práctica.