Práctica 3

Métodos de Búsqueda con Adversario (Juegos)

DESCONECTA-4 BOOM

Introducción

Para realizar un programa que simule una partida a un juego con adversario de esta complejidad debemos primero aprender a jugar al juego en sí para ver que movimientos son buenos, que situaciones nos benefician o nos perjudican. Luego tenemos que ver como pensamos para ganar, es decir ya valoramos situaciones, pero nuestro rival tambien lo hace, para ser mejor que él y ganarle debemos valorar situaciones más lejanas en jugadas. Si los dos valoramos las situaciones de una manera igual, y yo estoy pensando en lo que va a pasar dentro de 10 turnos mientras que el juega solo a 1 turno, claramente yo voy a salir victorioso, porque tengo en cuenta muchas mas situaciones que él.
En nuestro juego para simular estos comportamientos hemos implementado dos funciones principales, una para valorar tableros/situaciones, llamada **Valoración** y otra para explorar el arbol de posibles movimientos hasta una profundidad indicada, llamada **poda-alfa-beta.**

Algoritmo Minimax Poda Alfa Beta.

Para implementar la elección de acciones en este juego por turnos hemos usado el algoritmo Minimax con poda alfa beta con una profundidad máxima de 8 niveles. La poda nos ayuda a poder usar más profundidad, ya que solo con un algotirmo Minimax clasico, profundidad máxima de 8, y 8 acciones que podemos realizar en cada partida, debería realizar 8^8=16777216 iteraciones por el arbol, lo cual es de un orden O(8^n). Con la poda alfa beta intentamos reducir este número y convertirno en polinomial, aunque en el peor de los casos cuando las valoraciones no sean las más adecuadas podría actual igual.

Básicamente nuestro algoritmo simula 8 turnos de una partida real, 4 de mi jugador y 4 del rival. En los niveles pares 0,2,4 y 6 elegiremos la mejor de las acciones para nosotros, y el rival hara lo mismo en sus niveles. Ha cada nodo le asignamos unas cotas, inferior alfa, y superior beta, inicializadas a -inf y +inf correspondientemente. Cada nodo hoja o terminal se corresponde con la situación del tablero 8 jugadas posteriores, las cuales estan valoradas en función a un algoritmo de valoración que posteriormente explicare. Sabiendo todo esto el algoritmo minimax poda alfa beta, desciente hasta el último nivel donde estan los nivel donde estan los nodos hoja o terminales y valora la situación de los tableros resultantes en cada nodo en base a una función valoración que despues explicaremos. Una vez completadas las valoraciónes asciende al penúltimo nivel, que en nuestro casi sería el nivel 6, nivel máx. En este el nodo en el que estamos elegirá la mejor valoración de entre sus hijos (cada hijo representa un tablero), y en caso de que esta valoración fuera mayor que su cota inferior, cambiaría esta por esa valoración. La explicación de esto es muy sencilla, si llegamos a la situación de ese nodo, la mejor acción que podemos tomar es la que nos lleve al tablero de ese hijo con la mayor valoración. Este proceso se repetirá para todos los nodos ese nivel. Posteriormente el algoritmo ascenderá un nivel, el cual será nodo mín. En este nivel, el criterio es al contrario, ahora se elegiŕa el valor más bajo de entre todas las cotas inferiores de los nodos hijos, y si este es menor que la cota superior de nuestro nodo se intercambiara. (La mejor acción que puede elegir el rival en ese caso). Este proceso se repite hasta terminal el arbol, eso si para no realizar todas las iteraciónes en el momento en el cual la cota inferior de un nodo sea mayor o igual que su cota superior, ese nodo no comprobara más hijos, es decir se podaran sus hijos.

```
double Poda_AlfaBeta(Environment actual_, int jugador_, int profundidad, int
PROFUNDIDAD_ALFABETA, Environment::ActionType &accion, double alpha, double beta) {

   //Elegimos dentro de PROFUNDIDAD_ALFABETA movimientos que tablero es el mas
```

```
beneficioso para nosotros
   if(profundidad==PROFUNDIDAD_ALFABETA || actual_.JuegoTerminado()  ) { // Estado
terminal
          if( jugador_ == 2 )
             return  Valoracion(actual_, jugador_,profundidad);
          else
             return Valoracion(actual_,jugador_,profundidad); //
Valoracion2(actual_,jugador_,profundidad);

   }

   Environment V[8]; // PUT1, PUT2, PUT3, PUT4, PUT5, PUT6, PUT7, BOOM
   int n_gen = actual_.GenerateAllMoves(V); // posibles movimientos del espacio de movimientos
que tenemos, sirve para saber los hijos que tiene el nodo actual
   double aux;


     if(profundidad%2==0) { // Esto es nodo MAX 0 2 4 6
          for(int i=0; i<n_gen; i++) {

aux=Poda_AlfaBeta(V[i],jugador_,profundidad+1,PROFUNDIDAD_ALFABETA,accion,alpha,be
ta); // Llamamos recursivamente al siguiente nodo hijo
             if(aux>alpha) {
                alpha=aux;
                if(profundidad==0) accion=static_cast< Environment::ActionType
>(V[i].Last_Action(jugador_)); // La accion elegida para llegar hasta el nodo hijo que tenia un
valor mayor que alpha
             }
             if(beta<=alpha) return alpha; // Poda, se para la recursividad de este nodo
          }
          return alpha;

       }
       else{ // MIN 1 3 5 7
          for(int i=0; i<n_gen; i++) {

aux=Poda_AlfaBeta(V[i],jugador_,profundidad+1,PROFUNDIDAD_ALFABETA,accion,alpha,be
ta);
             if(aux<beta) {
                beta=aux;
                if(profundidad==0) accion=static_cast<Environment::ActionType
>(V[i].Last_Action(jugador_)); // La accion elegida para llegar hasta el nodo que tenia unv valor
menor que beta
             }
             if(beta<=alpha) return beta; // Poda, se para la recursividad de este nodo
          }
          return beta;
       }

}
```

Criterios valoración.

Para realizar la valoración de los tableros hemos evaluado una seríe de situaciones:

1-El tablero es de VICTORIA, DERROTA o EMPATE:
VICTORIA = +inf
DERROTA = -inf
EMPATE = 0

2-¿Cuantas fichas mias y cuantas del rival tiene el tablero? Deseamos tableros que tengan más fichas del rival que mias, esto es porque con esa situación existen mas posibilidades de que el adversario conecte 4 fichas, ademas esta valoracion esta ponderada por los niveles que estes llenos del tablero, porque deseamos llenar el tablero con mas fichas del rival que nuestras.
Valoracion = (-nºmisfichas +nºrivalfichas)*niveles

3-¿Cuantas fichas mías estan agrupadas?¿Y de mi rival? En este caso buscamos tableros donde el rival tenga sus fichas agrupadas en linea de 3, y nosotros no.
Valoracion = -gruposMiosDe3*V3 - gruposMiosDe2*V2+ gruposRivalDe3*V3 + gruposRivalV2

4- Prefiero tableros en los que muchas de mis fichas esten en la misma fila que mi ficha bomba, esto me dara ventaja para destruir esa fichas y quizas desestabilizar la partida.

5-Prefiero tableros en los que mi ficha bomba este sobre una ficha de mi enemigo.

6-Buscamos tableros en los que no tengamos bloqueda ninguna columna, y el rival si, esto se valora muy alto. La función que se encarga de comprar estas situaciones se llama SituacionesDeseables.

7-Dar importancia a tableros en los que mi bomba en alguna linea de fichas enemigas, quedando posiblidades para ganar, si exploto la bomba y el enemigo coloca las fichas que falten. Esto se comprueba por grupos verticales y diagonales.

8- Si juego como jugador 2, adoptare una actitud mas defensiva explotando cuantas mas bombas pueda mejor.

Los criterios 4,5,6 y 7 se valoraran solo si existe bomba del jugador en juego, y se valoraran positivamente para nosotros y negativamente para el rival.

El criterior número 7 intenta, predecir buenas situaciones, y en función de lo buenas que sean recibira una valoración u otra, pudiendo hasta alcanzar una valoración de +inf o -inf. Esto es porque es capaz de predecir las jugadas anteriores a VICTORIA o DERROTA, y esto nos dara una ventaja respecto a los compañeros que no lo tengan en cuenta. ¿Por que no dara ventaja esto? La respuesta es sencilla, ambos mi rival y yo, usaremos un algoritmo poda alfa beta que explore como máximo 8 niveles, lo cual nos deja en igualdad de condiciones, pero si yo consigo predecir algunas de las jugadas del nivel 8 que me llevan a VICTORIA o DERROTA en el nivel 9, me podre anticipar a mi rival en algunas ocasiones.

Finalmente para la entrega se han utilizado algunos de estos criterios, concretamente el 1, 3, 6 y 7.

Optimización de la heurística

Todo estas heurísticas planteadas, se pueden combiar dando más importancia a unas que ha otras, activando unas en alguna situación de la partida. Yo para elegirlas y ponderarlas he hecho pruebas contra heuristicas sencillas como 1, o la 1 y la 3 (que creo que son las que implementaran mis compañeros). Comprobando el resultado de las posibles heuristicas que he planteado contra las anteriores dichas, me he dado cuenta de que una heuristica sencilla puede dar muy buenos resultados, y es muy dificil controlar los movimientos con heuristicas complicadas, que pueden incluso arruinar nuestra partida. Por ese motivo finalmente solo elegí 3 de las herusticas planteadas, dado que me estaban dando mejores resultados.

Las heurística 7 no la elegí porque creo que debe tener algun tipo de fallo, ya que cuando la uso mi jugador puede perder en situaciones en las que tiene mas opciones que no le hacen perder, he revisado el codigo y creo que es correcto, finalmente la he dejado aparte porque al explorar 8 niveles puede la podaAlfaBeta no creí que la ventaja fuera signiticativa aunque funcionara bien.

```
double Valoracion(const Environment &estado, int jugador,int prof){
   double valoracion=0;
   int ganador = estado.RevisarTablero();
   int m, s;

   if (ganador==jugador)
         return 99999999.0-prof; // Gana el jugador que pide la valoracion
   else if (ganador!=0)
         return -99999999.0+prof; // 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



   // A partir de aqu\ED comienza las valoraciones de las jugadas que no son victoria, derrota o
empate
   // Valoramos la cantidad de fichas y las agrupaciones entre ellas
  // valoracion += ValoracionAgrupacionesFichas(estado, jugador);
    ContarFichas(estado,m,s,jugador);
    int niveles = NivelesCompletos(estado);

    valoracion = ValoracionContarFichas(estado, jugador)*niveles*s/4; // Esta valoraci\F3n
aumenta conforme mas niveles hay completos

   // Valoramos situaciones en las que nuestro rival tiene bloquedas columnas
   valoracion += SituacionesDeseables(estado,jugador);


   valoracion += ValoracionAgrupacionesFichas(estado,jugador);
/*
   if(estado.Have_BOOM(jugador)){
      valoracion += JugadasMaestras(estado,jugador);
```

```
    }
     if(estado.Have_BOOM((jugador+1)%3)){
        valoracion -= JugadasMaestras(estado,(jugador+1)%3);
    }*/
    if(jugador==2)
        if(estado.N_Jugada()%4==0 && estado.Have_BOOM(jugador)==0)
            valoracion += EXPLOTARBOMBA*1000;// Porque hemos explotado la bomba, y vamos a
colocar otra


    return valoracion;
}
```