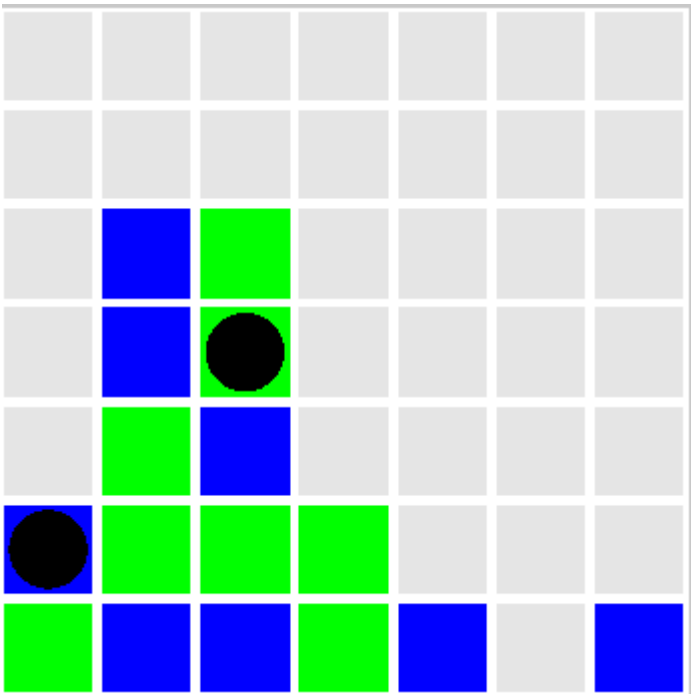
Practica 3 IA

CONECTA-4 BOOM



Realizada por: Pablo Borrego Megías, subgrupo A1

1.Análisis del problema

El problema planteado aquí es el siguiente, se quiere desarrollar un agente que pueda jugar al conecta-4 BOOM según algún algoritmo de exploración en juegos(Minimax o Poda alfa-beta) y se plantea el siguiente reto, el agente debe de ser capaz de vencer a los 3 “ninjas” que ya existen en el juego. Para ello como se ha comentado ya será necesario desarrollar tanto el algoritmo de exploración como una heurística, la cual será necesaria para poder vencer a los ninjas.

2.Descripción de la solución obtenida

Empezaré explicando mi algoritmo, el cual he decidido que sea poda alfa-beta, ya que por lo general es más rápida que minimax.
Pues, según el propio funcionamiento del programa, la poda alfa-beta se encargará de escoger el tablero con la mejor puntuación posible tras explorar todos los posibles tableros que se pueden escoger, y mediante una función valoración (la cual se explicará más tarde) se le asignará una puntuación en base a unos criterios que yo he considerado(mi heurística). Pues bien, la estructura de la poda alfa-beta es muy parecida al esquema visto en clase de teoría, como se puede observar.

```
331     if (profundidad==profundidad_max || tablero.JuegoTerminado()){ //Si hemos llegado al nodo final o si ha acabado el juego
332
333       return Valoracion(tablero, jugador_);
334
```

Como se observa en la foto, nada más empezar se comprobará si se ha llegado al nodo final, y entonces se llamará a la función valoracion() para finalizar el algoritmo.

```
}else {

    int last_act=-1;
    Environment hijo = tablero.GenerateNextMove(last_act);

    if (jugador_==tablero.JugadorActivo()){// Con esto indicamos que el nodo es MAX

        while (n_act!=0){

            aux = Poda_AlfaBeta(hijo, jugador_, profundidad+1, profundidad_max, act_anterior, alpha, beta);

            if (aux>alpha){
               alpha=aux;
               action = static_cast <Environment::ActionType> (last_act); // El movimiento a arrastrar
            }

            if (alpha >= beta){
               return beta; //Poda beta
            }

            hijo = tablero.GenerateNextMove(last_act); //Generamos el siguiente tablero hijo
            n_act--;
        }

      return alpha;
```

Después, si no es el nodo final, si el jugador es activo será por tanto un nodo MAX. Si es así, se llamará recursivamente a la poda de nuevo para dar un valor a aux la cual es una variable auxiliar de tipo double. Se comprueba esa variable con el valor de alfa actual y solo si es mayor se cambiará el valor de alfa por el de este. Si alfa es mayor o igual que beta devolverá beta (De esta manera se realiza la poda) y por último se genera el tablero hijo del actual. Una vez finalice devolverá alfa. Para los nodos beta ocurre lo mismo pero si se da que beta es menor o igual a alfa se podará alfa.

```
361         } else { //Nodo MIN
362             while (n_act!=0){ //Idem
363 
364                 aux = Poda_AlfaBeta(hijo, jugador_, profundidad+1, profundidad_max, act_anterior, alpha, beta);
365 
366                 if (aux<beta){
367                    beta=aux;
368 
369                 }
370 
371                 if (beta <= alpha){
372                    return alpha; //Poda alpha
373                 }
374 
375              hijo = tablero.GenerateNextMove(last_act); //Generamos el siguiente tablero hijo
376                n_act--;
377             }
378           return beta;
```

Proseguimos pues con la heurística que he desarrollado.

Heurística

La heurística que he desarrollado consiste en puntuar los tableros en base al número de pares, tríos y cuartetos de fichas consecutivas que se encuentren el cada tablero. Una vez se obtienen las fichas de cada jugador, se puntuarán de esta manera:

```
219     // Calculamos el valor heurístico del tablero
220     total_jugador=(fichas_4*100 + fichas_3*10 + fichas_2*1);
221     total_opuesto=(fichas_opuesto_4*100 + fichas_opuesto_3*10 + fichas_opuesto_2*1);
```

Esas variables lo que indican es el número de pares, trios, etc que ha encontrado de cada jugador, y finalmente devolverá la diferencia del total de ambos jugadores. Pasaré ahora a explicar como se ha hecho ese recuento de casillas consecutivas.

```
161 double Valoracion(const Environment &estado, int jugador){
162     double total_jugador, total_opuesto;
163     // Obtenemos el judador opuesto
164     int jugador_opuesto;
165     int jugador_;
166     if (jugador == 1){
167         jugador_=1;
168         jugador_opuesto = 2;
169     }else{
170         jugador_=2;
171         jugador_opuesto=1;
172     }
173 
174     // Comprobamos las consecutivas del jugador actual
175     int fichas_4 = checkConsecutivas(estado, jugador_, 4);
176     int fichas_3 =  checkConsecutivas(estado, jugador_, 3);
177     int fichas_2 =   checkConsecutivas(estado, jugador_, 2);
178 
179     // Comprobamos las consecutivas del jugador opuesto
180     int fichas_opuesto_4 = checkConsecutivas(estado, jugador_opuesto, 4);
181     int fichas_opuesto_3 =  checkConsecutivas(estado, jugador_opuesto, 3);
182     int fichas_opuesto_2 =   checkConsecutivas(estado, jugador_opuesto, 2);
183 
```

El la propia función valoración se comprobará quién es cada jugador y se llamará para cada una de las variables a la función “checkConsecutivas” a la cual se le pasaŕa como parámetro el tablero actual, el jugador y el número de fichas consecutivas que tiene que buscar.

```
134  int checkConsecutivas(const Environment &estado, int player, int n_casillas){
135      int contador = 0;
136
137
138      for (int i=0; i<7;i++) { //Comprobamos todas las fichas del tablero
139          for (int j=0; j<7;j++) {
140
141              if (estado.See_Casilla(i,j) == (char)player){// Si es el jugador
142
143                  // Comprueba las casillas consecutivas verticales
144                  contador += consecutivasVertical(estado, i, j, player, n_casillas);
145
146                  // Comprueba las casillas consecutivas horizontales de la fila
147                  contador += consecutivasHorizontal(estado, i, j, player, n_casillas);
148
149                  // Comprueba las diagonales
150                  contador += consecutivasEnDiagonal(estado, i, j, player, n_casillas);
151              }
152          }
153      }
154
155
156      return contador;
157
158  }
```

“checkConsecutivas” lo que va realizando es, para cada una de las casillas del tablero, comprueba a que jugador pertenece, y si esa casilla tiene una ficha del jugador pasado por parámetro, se llamarán a las funciones “consecutivasVertical”, “consecutivasHorizontal” y “consecutivasEnDiagonal” a las cuales, se les pasa como parámetros tanto la posición i, j de la casilla como el tablero, el jugador u la cantidad de fichas consecutivas que debe encontrar. Finalmente esta función devolverá la cantidad de pares, tríos o cuartetos de fichas que haya encontrado.

```
57  int consecutivasVertical (const Environment &estado, int fil, int col, int player, int n_casillas){
58      int consecutivas = 0;
59  //cout << "VERTICAL<----------------------"<< endl;
60
61          for (int i= fil ; i<7; i++) {
62              if (estado.See_Casilla(i,col) == player || estado.See_Casilla(i, col)== (player + 3))
63                  consecutivas += 1;
64              else
65                  break;
66          }
67
68      if (consecutivas >= n_casillas)
69          return 1;
70      else
71          return 0;
72  }
```

La función “consecutivasVertical” recorrerá la columna donde está situada la ficha y comprobará si las fichas que tiene por encima son del mismo jugador. Si ha encontrado la cantidad de fichas pedida (2, 3 o 4) o más, devolverá 1.

```
74  int consecutivasHorizontal (const Environment estado, int fil, int col, int player, int n_casillas){
75      int consecutivas = 0;
76  //cout << "HORIZONTAL<----------------------"<< endl;
77  int jugador=estado.JugadorActivo();
78
79          for (int i= col ; i<7; i++) {
80              if (estado.See_Casilla(fil,i) == player || estado.See_Casilla(fil, i)== (player + 3))
81                  consecutivas += 1;
82              else
83                  break;
84          }
85
86      if (consecutivas >= n_casillas)
87          return 1;
88      else
89          return 0;
90  }
```

La función “consecutivasHorizontal” funciona exactamente igual que “consecutivasVertical” con la diferencia de que ahora se explora la fila donde está situada la ficha.

Para “consecutivasEnDiagonal” ocurre lo mismo, las únicas diferencias son que se recorre la diagonal de forma ascendente y descendente, y en vez de devolver 1 o 0, se devolverá el total de consecutivas que haya encontrado. No adjunto fotos ya que es muy extenso el código y me pasaría del límite de páginas.

Resultados

Pues esta es la heurística que he usado para resolver la práctica, vamos ahora a hablar de los resultados obtenidos.

En general la heurística funciona bien, hace movimientos sensatos, de hecho gana fácilmente al ninja 1 tanto con el jugador verde como el azul, el problema es que al ninja 2 solo lo vence cuando juega en el lado verde, y al 3 no lo sé, ya que como tarda tanto en responder no me ha dado tiempo nunca a acabar una partida contra él.
No he conseguido mejorar más la heurística pero el caso es que lo he intentado todo lo que he podido y no he obtenido resultados. He probado a cambiar la valoración de los tableros, a recorrer de más maneras las filas y las columnas, también he probado a añadir la función puntuación al total calculado, a cambiar la poda alfa-beta y también a modificar la heurística y no consigo que venza a más ninjas.

En resumen, mi heurística vence fácilmente al ninja 1 y al ninja 2 cuando juega como jugador verde.