PROP
Activitat 2

Autors:

Explicació Heurística

Al començar a plantejar la heurística el primer en el que vam pensar era en comptar els "punts" que guanyes segons on tires la fitxa. Aquest punts van determinats pel nombre de combinacions en qualsevol direcció (excepte cap amunt) de 4, 3 i 2 que es formaven després d'aquesta tirada. El problema era que per com teníem programat el minimax i els paràmetres de la funció heurística no podiem fer-ho per una tirada en concret.

Així que vam decidir aplicar aquest recompte de punts a tot el tauler i sumar el punts de cada fitxa. En aquest cas el problema va ser que no teníem en compte els fitxes del rival, per tant en un tauler on els dos tenen situacions semblants pero el rival te avantatge ens podria donar una bona heurística quan no és el cas.

Per solucionar això hem fet que la funció compti la puntuació dels dos jugadors i després que es resti la del rival a la nostra. D'aquesta forma podem tenir una visió més global i acertada del tauler.

Les puntuacions assignades son:

* -Si està en una de les files centrals +4 punts.
* -Si té una peça aliada adjacent +2 punts.
* -Si té dos peces aliades adjacents seguides en la mateixa direcció +5 punts.

Com es pot veure no hi ha diferència entre els punts que val una diagonal i una horitzontal, encara que la diagonal sigui normalment més difícil d'aconseguir, sobretot al principi. Però no es tracta de que sigui perfecte.

Un mica més tard vam afegir el punts extres per les files centrals, que al principi no vem tenir en compte.

A nivell de la puntuació els valors podrien ajustarse i com a resultat donar una millor predicció però el resultat amb els actuals ja ens sembla correcte.

Codi Minimax amb poda alpha-beta

```java
/**
* Funció que calcula/busca la millor heuristica entre el nostres moviments posibles
* @param t Tauler on estem jugant la partida
* @param color Color del jugador adversari
* @param alpha Millor valor trobat fins al moment
* @param beta Pitjor valor trobat fins al moment
* @param jugades Profunditat de jugades que ens queda per explorar
* @return Retorna la major heuristica trobada, per tant la millor jugada possible per a Random4
*/
public int MaxValor(Tauler t, int color, int alpha, int beta, int jugades){
  if (jugades == 0 || !t.espotmoure()){ //Si no hi han moviment possibles o jugades restants miram la heuristica del tauler final
      int heuristica = getHeuristica(t, color);
      ++comptadorTaulers;
      System.out.println("Heuristica torn "+comptadorTaulers+": "+heuristica); //<- Comentar per millorar el rendiment
      return heuristica;
  }
  int valor = -999;
  for(int i=0; i < 8; i++){ //Un moviment per columna
      if(t.movpossible(i)){ //Si el moviment no es possible no fa falta probar aquesta tirada
          Tauler aux = new Tauler(t); //Utilitzem aux per no modificar el tauler de la partida
          aux.afegeix(i, color);
          if (aux.solucio(i, color)){ //Si el moviment és guanyador no fa falta expandir, hem acabat
              valor = 999; //Valor molt gran perque és la nostra victoria
              return valor;
          }
          valor = Math.max(valor, MinValor(aux, color*(-1), alpha, beta, jugades-1)); //Ens quedem amb el valor més gran
          alpha = Math.max(valor,alpha); //Modifiquem el valor de alpha si es necessari i apliquem la poda
          if (beta <= alpha) return valor;
      }
  }
  return valor;
}
/**
* Funció que calcula/busca la millor heuristica entre el moviments possibles del adversari
* @param t Tauler on estem jugant la partida
* @param color Color del jugador al que representa Random4
* @param alpha Millor valor trobat fins al moment
* @param beta Pitjor valor trobat fins al moment
* @param jugades Profunditat de jugades que ens queda per explorar
* @return Retorna la menor heuristica trobada, per tant la millor jugada possible per a l'adversari
*/
public int MinValor(Tauler t, int color, int alpha, int beta, int jugades){
  if (jugades == 0 || !t.espotmoure()){ //Si no hi han moviment possibles o jugades restants miram la heuristica del tauler final
      int heuristica = getHeuristica(t, color);
      ++comptadorTaulers;
      System.out.println("Tauler num "+comptadorTaulers+": "+heuristica); //<- Comentar per millorar el rendiment
      return heuristica;
  }
  int valor = 999;
  for(int i=0; i < 8; i++){ //Un moviment per columna
      if(t.movpossible(i)){ //Si el moviment no es possible no fa falta probar aquesta tirada
          Tauler aux = new Tauler(t); //Utilitzem aux per no modificar el tauler de la partida
          aux.afegeix(i, color);
          if (aux.solucio(i, color)){ //Si el moviment és guanyador no fa falta expandir, hem acabat
              valor = -999; //Valor molt petit perque és la nostra derrota
              return valor;
          }
          valor = Math.min(valor, MaxValor(aux, color*(-1), alpha, beta, jugades-1)); //Ens quedem amb el valor més petit
          beta = Math.min(valor, beta); //Modifiquem el valor de beta si es necessari i apliquem la poda
          if (beta <= alpha) return valor;
      }
  }
  return valor;
}
```

```java
/**
* Funció que calcula la columna més óptima per fer la nostra tirada
* @param t Tauler on estem jugant la partida
* @param color Color del jugador al que representa Random4
* @return Retorna un int indicant la columna on realitzarem la nostra tirada
*/
public int moviment(Tauler t, int color){
  comptadorTaulers=0;
  int valor = -999;
  int col = 0;
  int alpha = -999;
  int beta = 999;
  for(int i=0; i < 8; i++){ //Un moviment per columna
      if(t.movpossible(i)){ //Si el moviment no es possible no fa falta probar aquesta tirada
          Tauler aux = new Tauler(t); //Utilitzem aux per no modificar el tauler de la partida
          aux.afegeix(i, color);
          if (aux.solucio(i, color)){ //Si el moviment Ã©s guanyador no fa falta expandir, hem acabat
              return i;
          }
          int valor_nou = MinValor(aux, color*(-1), alpha, beta, jugades-1); //ComenÃ§em a explorar el minimax
          if (valor < valor_nou){ //Si el valor trobat Ã©s millor que l'actual, actualitzem la columna a retornar i el valor per comparar
              valor = valor_nou;
              col = i;
          }
      }
  }
  return col;
}
```

Incidència de la poda alpha-beta en els nodes explorats

On b és el factor de ramificació i on n és el nivell màxim d'exploració

En el minimax cal visitar b^n nodes.

En el nostre cas el valor per a b és 8 (Perquè tenim 8 columnes) i el valor de n és 8 (Perquè la profunditat fins la que explorem és 8). Per tant el nombre de "nodes" que explorem és 8^8, 16.777.216 nodes.

Quan fem els prints podem veure que el nostre programa només visita 800.000 "nodes" o taulers aproximadament.

Per tant podem dir que la nostra poda elimina el 95% del "nodes" ha visitar.