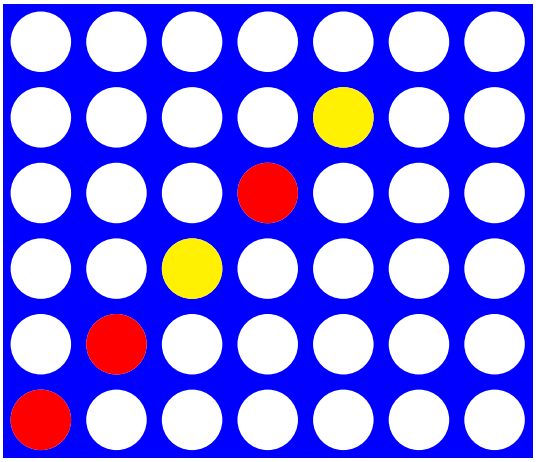
Conecta 4

Marc Plana i Adrián Borrego

November 2024



Índex

1 Introducció

L'objectiu principal d'aquest projecte és estudiar l'algorisme de Minimax, centrant-nos especialment en les seves variants amb poda alfa-beta i l'impacte de l'ordenació dels fills en el rendiment. Per posar en pràctica aquests conceptes, s'ha triat com a cas d'estudi el joc de Connecta 4, un entorn ideal per avaluar l'eficiència d'aquestes tècniques degut a la seva estructura de joc estratègic pero no masa complex a l'hora de definir una heurística.

Aquest document aborda des del disseny d'una heurística que guia el jugador fins a una anàlisi exhaustiva de l'impacte de la poda alfa-beta en el número de heurisitques a jugades finals explorades i com l'ordenació prèvia dels fills pot influir en la seva eficàcia.

2 Heurística i Implementació

2.1 Heurística dissenyada pel jugador

L'heurística dissenyada pel jugador busca avaluar les posicions del tauler amb l'objectiu de maximitzar les opcions de guanyar, alhora que es minimitzen les oportunitats de l'oponent. Per a cada cel·la del tauler, la funció d'heurística avalua la presència de fitxes pròpies i de l'oponent en totes les direccions possibles: horitzontal, vertical i diagonal (tant principal com inversa).

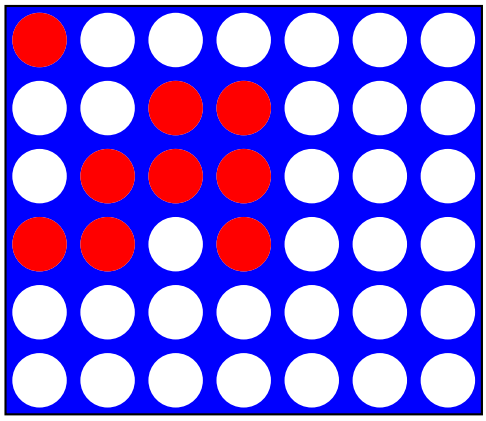
Cada conjunt consecutiu de fitxes d'un jugador rep una puntuació proporcional a la seva longitud:

1 fitxa consecutiva: `+1`

2 fitxes consecutives: `+10`

3 fitxes consecutives: `+100`

4 fitxes consecutives: `+1000` **(GUANYA!)**



En canvi, si aquestes fitxes consecutives són de l'oponent, es resta la puntuació corresponent per fer més realista laposibilitat de perdre. Això garanteix que l'heurística prioritzi tant l'atac com la defensa, en funció de l'estat del tauler.

Aquest disseny fa que el jugador sigui capaç de reconèixer situacions on estigui aprop de guanyar i bloquejar moviments perillosos de l'oponent.

2.2 Detalls d'implementació

L'heurística es divideix en tres funcions principals:

- `heuristic(Tauler board, int color)`: Aquesta funció principal recorre tot el tauler i calcula la puntuació global. Per a cada cel·la, determina el color actual (propi o de l'oponent) i crida la funció `evaluatePosition` per obtenir la puntuació corresponent. També te en compte el domini a les columnes centrals.
- `evaluatePosition(Tauler board, int row, int col, int color)`: Avalua la puntuació d'una cel·la en funció de les fitxes consecutives en les quatre direccions (horitzontal, vertical, diagonal principal i diagonal inversa). La puntuació final és la suma de totes les direccions.
- `evaluateDirection(Tauler board, int row, int col, int color, int direccioRow, int direccioCol)`: Examina una direcció específica des d'una cel·la donada. Si es detecten fins a quatre fitxes consecutives del mateix color, retorna una puntuació específica, depenent del nombre de fitxes consecutives i el numero de forats en blanc que hi hagi.

A nivell tècnic:

- La funció `evaluateDirection` utilitza un bucle per comprovar fins a quatre cel·les consecutives, assegurant-se que aquestes no surtin dels límits del tauler.
- Es fa ús de variables com `row` i `col` per representar la posició actual, mentre que `direccioRow` i `direccioCol` permeten especificar la direcció que s'està avaluant.
- Per optimitzar, només es calculen direccions a partir de cel·les que tenen potencial per formar línies guanyadores.

3 Estudi de les Estrategies

- Sense poda
- Amb poda alfa-beta
- Amb poda alfa-beta i ordenació de moviments

A continuació, mostrarem les proves que em realitzat en cadascun dels casos i com millora cadascun a l'anterior, amb els resultats corresponents en termes de càlculs heurístics i farem la comparativa.

3.1 Algoritme MinMax Sense Poda

L'algoritme MinMax sense poda explora totes les jugades possibles fins a la profunditat màxima, sense intentar reduir el nombre de nodes a explorar. La quantitat de nodes fills per a un arbre de MinMax complet, amb una ramificació de b i una profunditat d, és b^d. Això vol dir que, en el pitjor dels casos, l'algoritme explora tots els nodes fins arribar a la profunditat especificada, el que pot ser molt costós a mesura que augmenten b i d.

3.2 Algoritme MinMax Amb Poda Alfa-Beta

L'algoritme MinMax amb poda alfa-beta redueix significativament el nombre de nodes a explorar, millorant l'eficiència del càlcul. Mitjançant la poda alfa-beta, es poden evitar certes ramificacions de l'arbre de decisions que no afectaran al resultat final. Això fa que el nombre de nodes explorable es redueixi de b^d a aproximadament $O(b^{d/2})$. Aquesta millora es fa evident a mesura que augmenta la profunditat, ja que la poda elimina ràpidament ramificacions innecessàries.

3.3 Algoritme MinMax Amb Poda Alfa-Beta + Ordenació de Moviments

La combinació de poda alfa-beta amb l'ordenació de moviments permet una optimització addicional, ja que s'exploren primer les jugades més prometedores. Amb aquesta tècnica, es redueix encara més el nombre de nodes explorats, aprofitant el fet que les millors opcions es processen primer i la poda es més eficaç.

4 Comparativa entre els Casos

Em fet una gràfica (Pagina 9) que compara el nombre d'heurístiques calculades per cada tirada en cada cas (per a MyJugador VS Profe). Això ens ajuda a veure clarament la millora del rendiment a través de la poda alfa-beta i l'ordenació de moviments.

Algorithm 1 Heurística per avaluar la situació del tauler

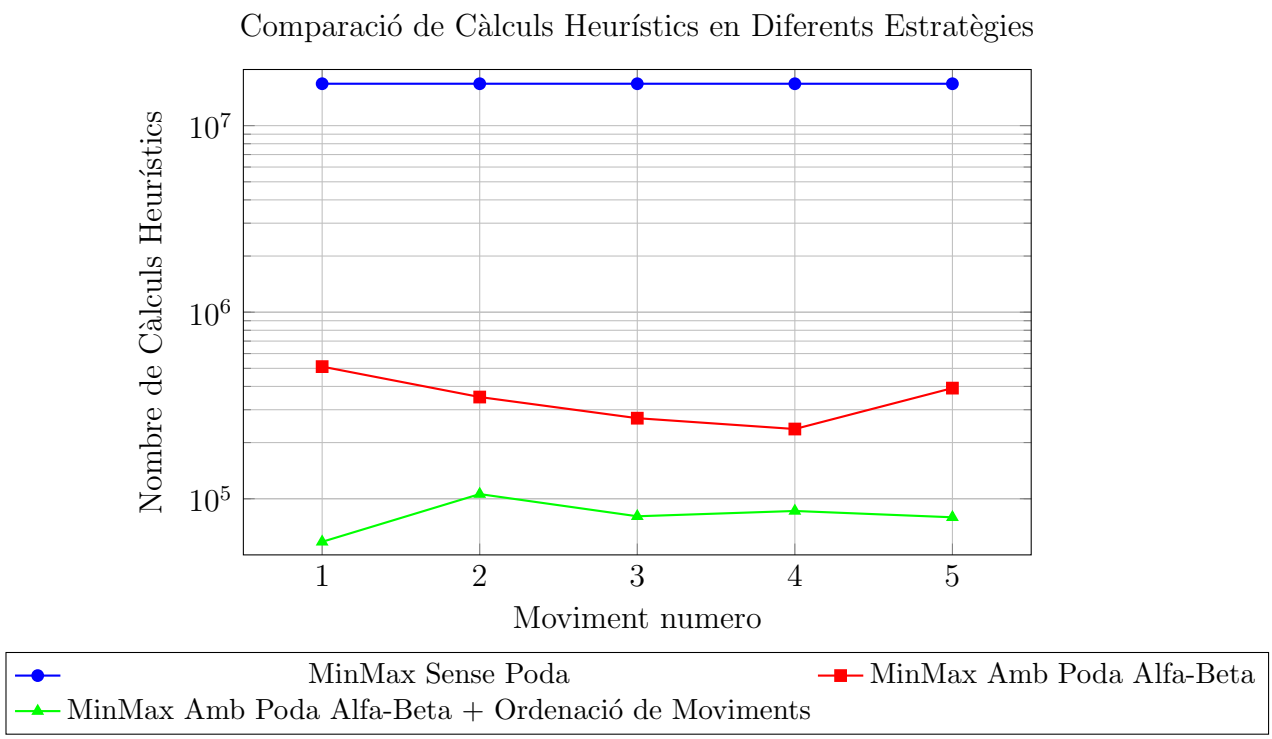
Funció `heuristic(tauler, color)`
  $oponentColor \leftarrow -color$
  $score \leftarrow 0$
for $fila = 0$ **fins a** $tauler.mida - 1$ **do**
  for $columna = 0$ **fins a** $tauler.mida - 1$ **do**
    $currentColor \leftarrow tauler.color(fila, columna)$
    if $currentColor = color$ **then**
      $score \leftarrow score +$ `evaluatePosition`$(tauler, fila, columna, color)$
    else if $currentColor = oponentColor$ **then**
      $score \leftarrow score -$ `evaluatePosition`$(tauler, fila, columna, oponentColor)$
    end if
  end for
end for
$score \leftarrow score +$ `controlCenter`$(tauler, color)$
return $score$

Algorithm 2 Avaluar la posició en el tauler

Funció `evaluatePosition(tauler, fila, columna, color)`
  $score \leftarrow 0$
  $score \leftarrow score +$ `evaluateDirection`$(tauler, fila, columna, color, 1, 0)$
  $score \leftarrow score +$ `evaluateDirection`$(tauler, fila, columna, color, 0, 1)$
  $score \leftarrow score +$ `evaluateDirection`$(tauler, fila, columna, color, 1, 1)$
  $score \leftarrow score +$ `evaluateDirection`$(tauler, fila, columna, color, 1, -1)$
return $score$

Algorithm 3 Avaluar la direcció d'una posició

```
1: Funció evaluateDirection(tauler, fila, columna, color, direccioFila, direccioColumna)
2:     size ← tauler.mida
3:     count ← 0
4:     buitsCount ← 0
5:     score ← 0
6: for i = 0 fins a 3 do
7:     newFila ← fila + i × direccioFila
8:     newColumna ← columna + i × direccioColumna
9:     if newFila ≥ 0 and newFila < size and newColumna ≥ 0 and newColumna < size then
10:        cellColor ← tauler.color(newFila, newColumna)
11:        if cellColor = color then
12:            count ← count + 1
13:        else if cellColor = 0 then
14:            buitsCount ← buitsCount + 1
15:        else
16:            break
17:        end if
18:    else
19:        break
20:    end if
21: end for
22: if count + buitsCount ≥ 4 then
23:    if count = 4 then
24:        score ← score + 1000
25:    end if
26:    if count = 3 and buitsCount > 0 then
27:        score ← score + 100
28:    end if
29:    if count = 2 and buitsCount > 0 then
30:        score ← score + 10
31:    end if
32:    if count = 1 and buitsCount > 0 then
33:        score ← score + 1
34:    end if
35: end if
36: return score
```


Comparació de Càlculs Heurístics en Diferents Estratègies
Nombre de Càlculs Heurístics
10^7
10^6
10^5
1
2
3
4
5
Moviment numero
MinMax Sense Poda
MinMax Amb Poda Alfa-Beta
MinMax Amb Poda Alfa-Beta + Ordenació de Moviments