Heuristique MINMAX

Implémentation PROLOG d'un algorithme MinMax basé sur une heuristique simple d'évaluation d'une position sur la grille

Vidéo YouTube - Algo MinMax
https://www.youtube.com/watch?v=I-hh51ncgDI

Fonctionnement de l'heuristique utilisé par notre Algorithme MinMax

- Score: **Nombre de ligne de longueur 4 possibles qui passent par cette position pour chaque joueur.** -> un pion placé à une position avec un score plus élevé augmente les possibilité de succès et donc de gagner. (Ces valeurs sont pré-calculées)

- On évalue une grille, en regardant chaque position remplie de la grille, et en ajoutant au score si le jeton actuel est de x la valeur positionScore(x,y), sinon si elle appartient à o, on soustrait.

- Une grille a une valeur positive si elle avantage x.

- Une grille a une valeur négative si elle avantage o.

Voici le pseudo code du min max. Dans notre cas, si notre joueur est x, il est le maximizingPlayer donc on cherche à maximiser le score de la grille.
Sinon, o est le Not(maximizingPlayer) et on cherche à minimiser le score de la grille.

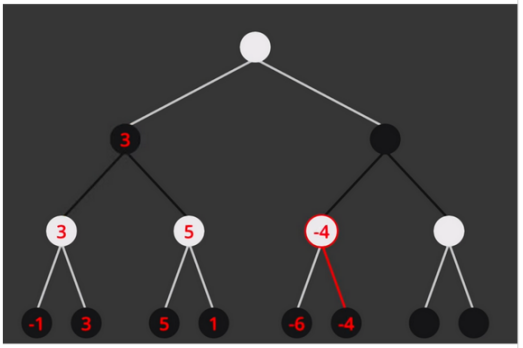
Slides de description de MinMax et implémentation PROLOG

Utilisation de l'algorithme minmax : Description rapide

```
function minimax(position, depth, maximizingPlayer)
   if depth == 0 or game over in position
      return static evaluation of position

   if maximizingPlayer
      maxEval = -infinity
      for each child of position
         eval = minimax(child, depth - 1, false)
         maxEval = max(maxEval, eval)
      return maxEval

   else
      minEval = +infinity
      for each child of position
         eval = minimax(child, depth - 1, true)
         minEval = min(minEval, eval)
      return minEval
```



* Source : https://youtu.be/l-hh51ncgDI

Utilisation de l'algorithme minmax : Avec PROLOG

Pseudo-Code *

```
function minimax(position, depth, maximizingPlayer)
   if depth == 0 or game over in position
      return static evaluation of position

   if maximizingPlayer
      maxEval = -infinity
      for each child of position
         eval = minimax(child, depth - 1, false)
         maxEval = max(maxEval, eval)
      return maxEval

   else
      minEval = +infinity
      for each child of position
         eval = minimax(child, depth - 1, true)
         minEval = min(minEval, eval)
      return minEval
```

PROLOG

```
% init
boardHeuristic(Grille, R) :- boardHeuristic_worker_outer(Grille, 1, 0, R).

% Stopping condition
boardHeuristic_worker_outer([], _, R, R).

boardHeuristic_worker_outer([Colonne|ResteGrille], X, Acc, R2) :-
    NextX is X+1,
    boardHeuristic_worker_inner(Colonne, X, 1, Acc, R),
    boardHeuristic_worker_outer(ResteGrille, NextX, R, R2).

boardHeuristic_worker_inner([], _, _, R, R).
boardHeuristic_worker_inner([x|ResteColonne], X, Y, Acc, R) :-
    NextY is Y + 1,
    positionScore(X, Y, Score),
    Acc1 is Acc + Score,
    boardHeuristic_worker_inner(ResteColonne, X, NextY, Acc1, R).

boardHeuristic_worker_inner([o|ResteColonne], X, Y, Acc, R) :-
    NextY is Y + 1,
    positionScore(X, Y, Score),
    Acc1 is Acc - Score,
    boardHeuristic_worker_inner(ResteColonne, X, NextY, Acc1, R).
```

* Source : https://youtu.be/l-hh51ncgDI

Utilisation de l'algorithme minmax : Heuristique pour l'évaluation de la grille

- Score: **Nombre de ligne de longueur 4 possibles qui passent par cette position pour chaque joueur.** -> un pion placé à une position avec un score plus élevé augmente les possibilité de succès et donc de gagner. (Ces valeurs sont pré-calculées)
- On évalue une grille, en regardant chaque position remplie de la grille, et en ajoutant au score si le jeton actuel est de x la valeur positionScore(x,y), sinon si elle appartient à o, on soustrait.
- Une grille a une valeur positive si elle avantage x.
- Une grille a une valeur négative si elle avantage o.

En PROLOG:

positionScore(X, Y, Score)

positionScore(1,1,3).
positionScore(2,1,4).
...

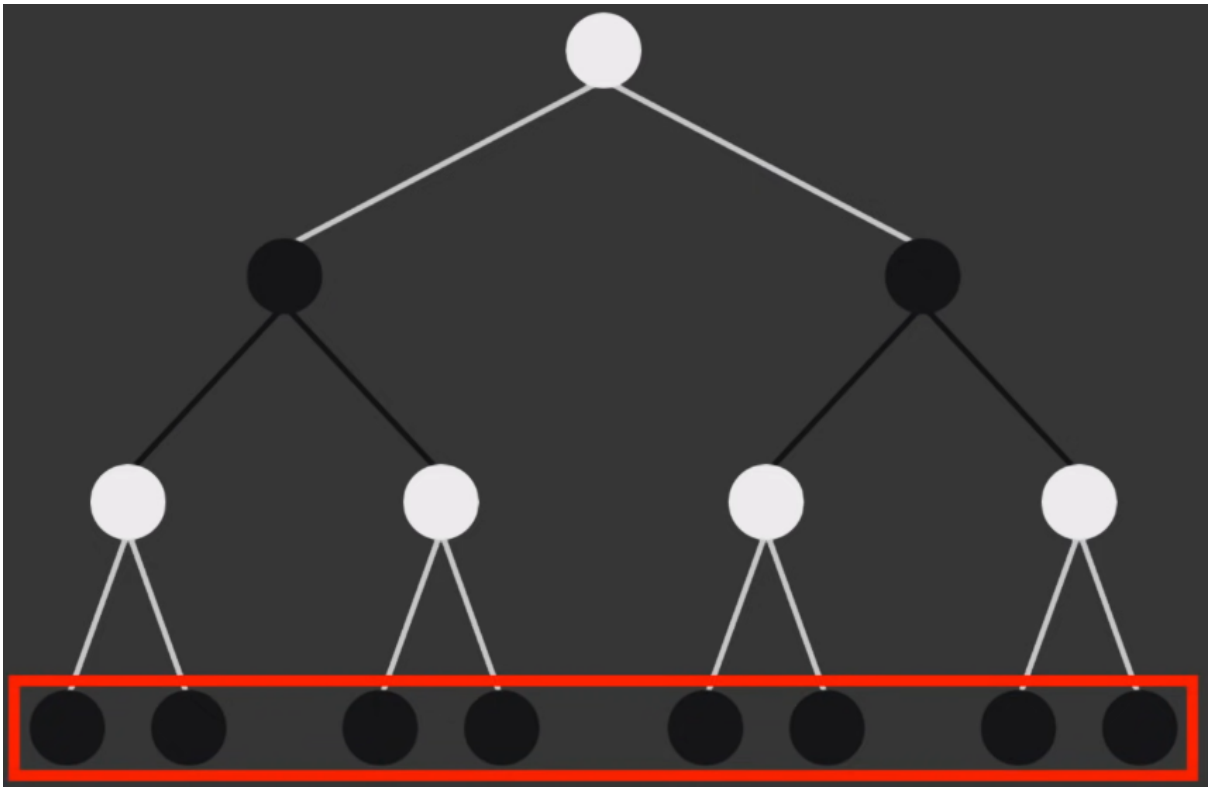
Explication du fonctionnement de l'algorithme MinMax

```
function minimax(position, depth, maximizingPlayer)
    if depth == 0 or game over in position
        return static evaluation of position

    if maximizingPlayer
        maxEval = -infinity
        for each child of position
            eval = minimax(child, depth - 1, false)
            maxEval = max(maxEval, eval)
        return maxEval

    else
        minEval = +infinity
        for each child of position
            eval = minimax(child, depth - 1, true)
            minEval = min(minEval, eval)
        return minEval
```

Les états encadrés en rouge, sont les états où l'on calcule le score de la grille.



PS:

If depth = 1

First step of the game is always the middle column because it provides the highest score for the starting player.

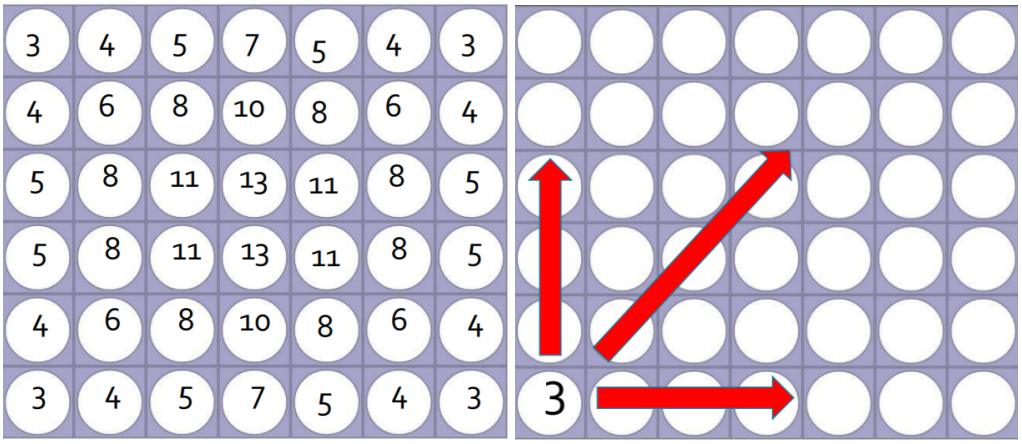
```
1 ?- startSims(1).
Score x : 7
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |x| | | |
joueur : x
```

If Depth > 1

If the starting player plays in middle position, when going deeper in the tree, we find that the opponent can get a score of 10. Hence we don't play the middle position as it favors the opponent in the future.

Statisctics to include:

- **minmax against gauche ou Droite**
- **minmax against minmax with different depth(the AI with higher depth should win)**
- **minmax vs minmax with same depth**
- **Number of steps till victory in case of minmax vs minmax and compare it with number of steps till victory in case of minmax vs gauche ou droit**
  -> Intuitively the second game should not last longer than minmax vs minmax
  -> if we use a higher depth value the number of steps till victory reached should diminish in case of minmax vs gauche ou droite



(Image des valeurs rendus par la Score Function pour un X et un Y donné)