Vincent Beaudoin (111 103 778)
Alexandre Picard-Lemieux (111 103 625)
Gabriel Legault (111 089 063)
Clément Spies (111 139 346)

Intelligence artificielle I
IFT-2003

TP #2
Concevoir un jeu en utilisant l'IA

Travail présenté à
Laurence Capus

Département d'informatique et de génie logiciel
Université Laval
Hiver 2016

Introduction

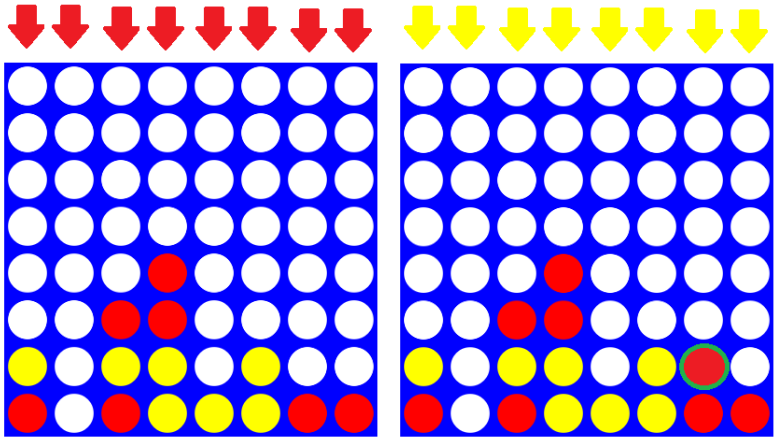
L'intelligence artificielle est utilisée dans de nombreuses applications. Dans ce travail, elle sera utilisée pour créer un adversaire dans le jeu de Puissance 4. Nous voulons donc analyser ce jeu et créer un joueur intelligent à l'aide du langage Prolog. Pour cela, il sera utilisé la démarche de résolution de recherche intelligente par espace d'états. Nous avons voulu implémenter cette solution avec minimax. Par contre, nous n'avons pas été capables de le réaliser dans les temps. L'adversaire prend donc la première colonne qui est disponible.

Description du jeu choisi[1][2]

Présentation du jeu

Puissance 4 est un jeu de stratégie commercialisé en 1974. Le but du jeu est de faire des connexions de quatre jetons dans une grille de sept colonnes et de 6 rangées.

Voici quelques exemples du jeu avec des plateaux de 8 colonnes et 8 rangées:



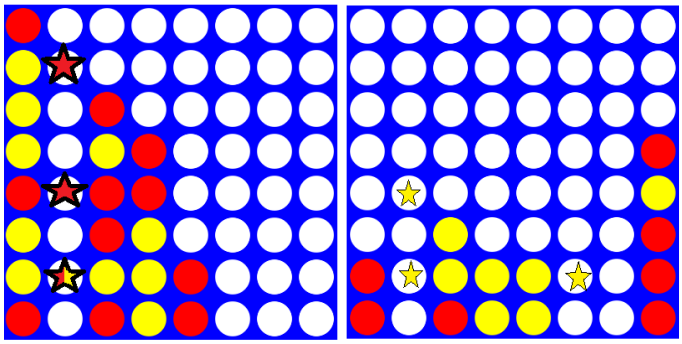
Grille où c'est au tour du joueur rouge *Grille où c'est au tour du joueur jaune*

Description des règles

Chaque joueur possède vingt-et-un jetons d'une couleur différente. Ces couleurs sont en général jaune et rouge. Les joueurs placent un jeton dans une colonne à chaque tour. La pièce va se placer à la rangée la plus basse qu'elle peut aller. Un joueur ne peut pas placer de jeton dans une colonne pleine. Pour gagner, il faut faire un alignement horizontal, vertical ou diagonal de quatre jetons de sa couleur. La partie est déclarée nulle s'il est impossible de mettre d'autres jetons et qu'il n'y a aucun gagnant.

[1] Wikipédia. [En ligne]. https://fr.wikipedia.org/wiki/Puissance_4 (Page consultée le 27 février 2016)
[2] Roadtolarissa. [En ligne]. http://roadtolarissa.com/connect-4-ai-how-it-works/ (Page consultée le 27 février 2016)

Voici quelques illustrations :



Grille avec les positions gagnantes Grille avec les positions gagnantes

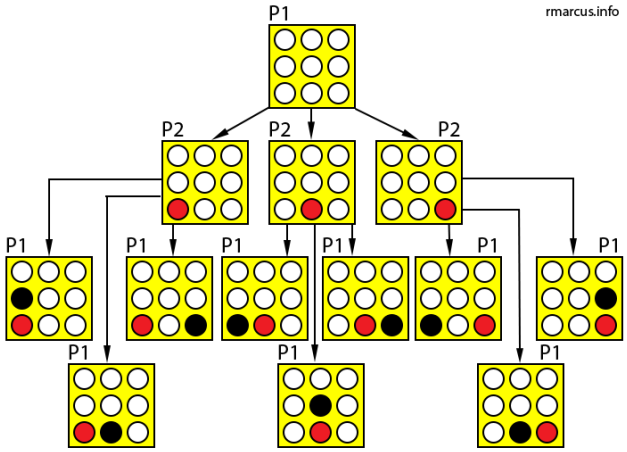
Modélisation du problème et de la solution

Description du problème

L'espace d'états de ce problème est décrit par l'ensemble des grilles m x n, possédant des cases vides, des cases avec des jetons rouges et des cases avec des jetons jaunes. Dans le jeu original, m = 6 et n = 7. Cela signifie donc que chaque état est l'ajout d'un jeton potentiel et que chaque nœud des profondeurs est un tour.

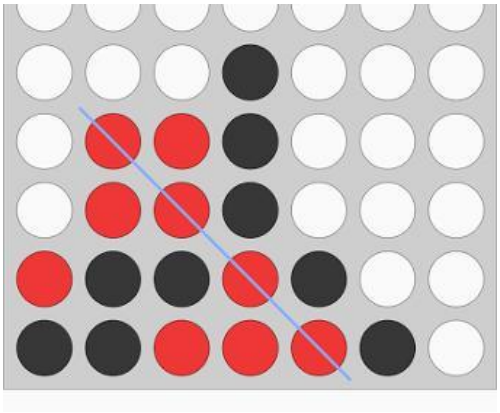
L'état initial est la grille vide. Il n'y a pas de jetons d'ajouter encore, car le jeu n'est pas encore commencé. L'état final est la grille possédant une connexion de quatre jetons ou une grille pleine de jetons. La partie est donc terminée puisqu'il y a un gagnant ou la partie est nulle. L'état initial peut aussi être rempli avec des jetons si on refait le calcul en plein milieu de la partie. L'état final peut aussi être un jeu non terminé si on limite la profondeur de l'espace d'état.

Les mouvements autorisés sont Ci qui signifie qu'on ajoute un jeton dans la colonne i. Il y a donc n mouvements autorisés par état. Il est possible d'en avoir moins s'il y a des colonnes pleines. Par exemple, voici l'espace d'état pour une profondeur de 3 avec une grille 3x3[3] :



[3] RMarcus. [En ligne]. https://rmarcus.info/blog/2014/12/23/connect4.html (Page consultée le 27 février 2016)

L'état initial est la racine de cet arbre. Voici un état final potentiel :

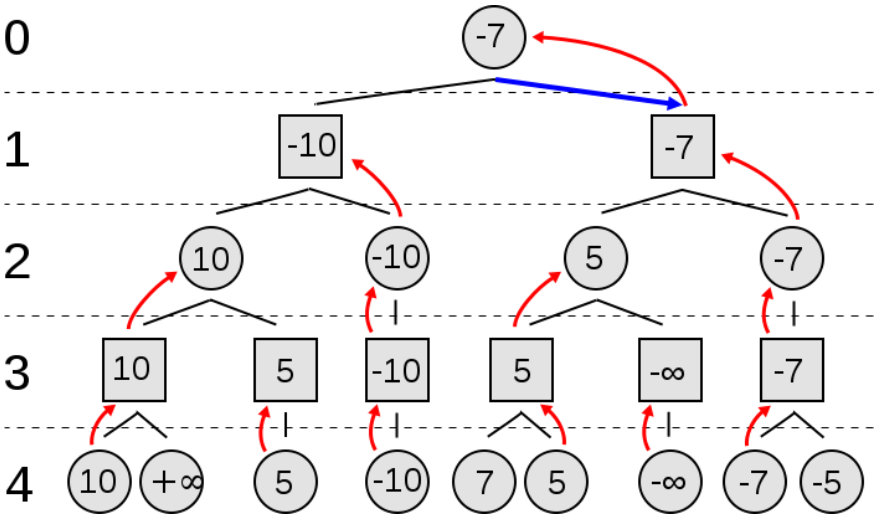


Description de la solution[4]

La procédure minimax sera utilisée pour créer l'adversaire. On nomme les deux joueurs MIN et MAX. MAX représente le joueur qui veut gagner et MIN représente le joueur qui veut empêcher MAX de gagner. Les deux joueurs possèdent les mêmes connaissances concernant l'espace d'état.

Cette procédure consiste à descendre jusqu'à une profondeur donnée et calculer la valeur heuristique de chacun de ces nœuds. Le parent prend la valeur MAX ou MIN de ses enfants tout dépendamment du parent.

Voici un exemple d'un arbre qui utilise cet algorithme. Dans cet exemple, MAX choisit donc son deuxième enfant après le calcul des valeurs.



La fonction heuristique est donc :

[4] Wikipedia. [En ligne]. https://en.wikipedia.org/wiki/Minimax (Page consultée le 27 février 2016)

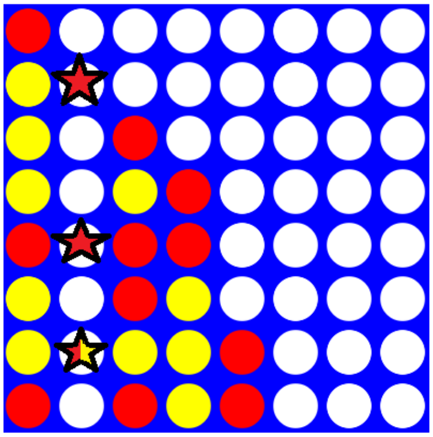
```
01 function minimax(node, depth, maximizingPlayer)
02     if depth = 0 or node is a terminal node
03         return the heuristic value of node

04     if maximizingPlayer
05         bestValue := −∞
06         for each child of node
07             v := minimax(child, depth − 1, FALSE)
08             bestValue := max(bestValue, v)
09         return bestValue

10     else    (* minimizing player *)
11         bestValue := +∞
12         for each child of node
13             v := minimax(child, depth − 1, TRUE)
14             bestValue := min(bestValue, v)
15         return bestValue
```

```
(* Initial call for maximizing player *)
minimax(origin, depth, TRUE)
```

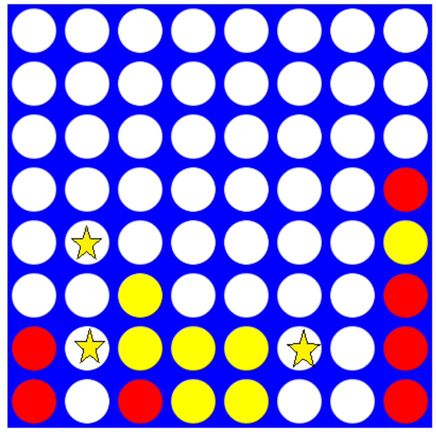
La valeur heuristique est calculée de manière très simple :

- Si c'est une victoire (4 jetons connectés de MAX), on retourne 10000;
- Si c'est une défaite (4 jetons connectés de MIN), on retourne -10000;
- Si c'est nul (la grille est pleine), on retourne 0;
- Si la partie est encore en cours (grille non pleine sans victoire), on utilise 10 * (nombre de positions qui mènent à une victoire de MAX - nombre de positions qui mènent à une victoire de MIN) + nombre de positions qui mènent à une connexion de 3 jetons de MAX – nombre de positions qui mènent à une connexion de 3 jetons de MIN.

Il faut aussi considérer que « XOX » et « XXOX » sont aussi des positions possibles pour O. Par exemple, voici les valeurs heuristiques pour ces deux grilles si MAX est jaune. Dans ces exemples, les étoiles représentent les positions pour des connexions possibles de 4 jetons.



Valeur heuristique : -23



Valeur heuristique : 29

La deuxième grille serait donc plus préférable si le jaune qui a mis son dernier jeton et si les deux grilles étaient à la même profondeur.

Implantation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Puissance 4
% La question a poser est : ?- puissance4.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicats necessaires au jeu de puissance 4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Retourne un plateau vide.
plateau_vide([[],[],[],[],[],[],[]]).

% Garde la hauteur d'une colonne plus petite que 6
hauteur(I,Plateau) :- position_liste(I,Plateau,Z), longeur(Z,N), N<6.

% La position dans le plateau peut etre mis a cette postion si sa hauteur est satisfaite
position(1,Plateau) :- hauteur(1,Plateau).
position(2,Plateau) :- hauteur(2,Plateau).
position(3,Plateau) :- hauteur(3,Plateau).
position(4,Plateau) :- hauteur(4,Plateau).
position(5,Plateau) :- hauteur(5,Plateau).
position(6,Plateau) :- hauteur(6,Plateau).
position(7,Plateau) :- hauteur(7,Plateau).

% Les deplacements possibles des jetons
```

```
deplacer(Position,Plateau,o,NouveauPlateau) :- position_liste(Position, Plateau,
PositionColonnePlateau),
                        joindre(PositionColonnePlateau, [o], NouvelleColonne),
                        remplacer_element(Position, NouvelleColonne, Plateau, NouveauPlateau).

deplacer(Position,Plateau,x,NouveauPlateau) :- position_liste(Position, Plateau,
PositionColonnePlateau),
                        joindre(PositionColonnePlateau, [x], NouvelleColonne),
                        remplacer_element(Position, NouvelleColonne, Plateau, NouveauPlateau).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Puissance 4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Fonction principale qui permet de jouer le jeu
puissance4 :- init_jeu(B), jeu_actif(B,cont).

% Initialise le jeu
init_jeu(Plateau) :- plateau_vide(Plateau).

% Fait un tour dans une partie
jeu_actif(Plateau,_) :- egal(Plateau).
jeu_actif(Plateau,joueur_gagne) :- afficher_plateau(Plateau), nl, write('Vous avez gagne!'), nl.
jeu_actif(Plateau,ordinateur_gagne) :-  afficher_plateau(Plateau),nl, write('Vous avez perdu!'), nl.
jeu_actif(Plateau,cont) :- deplacer_joueur(Plateau,NouveauPlateau,Cont1),
                deplacer_ordinateur(NouveauPlateau,NouveauNouveauPlateau,Cont1,Cont2),
                jeu_actif(NouveauNouveauPlateau,Cont2).

% Affiche le plateau
afficher_plateau(Plateau) :- afficher_plateau(Plateau,6).
afficher_plateau(_,0) :- write('+---+---+---+---+---+---+---+'),nl,
                write('  1   2   3   4   5   6   7  '), nl.
afficher_plateau(Plateau,Ligne) :- write('+---+---+---+---+---+---+---+'),nl,
                                        write('| '), afficher_ligne(Plateau,Ligne,1), write(' |'),nl,
                                        NouvelleLigne is Ligne-1,
                                        afficher_plateau(Plateau,NouvelleLigne).

% Affiche une ligne
afficher_ligne(Plateau,Ligne,7) :- position_liste(7,Plateau,Liste), position_liste(Ligne,Liste,Symbole),
write(Symbole).
afficher_ligne(Plateau,Ligne,Colonne) :- position_liste(Colonne,Plateau,Liste),
position_liste(Ligne,Liste,Symbole),
            write(Symbole), write(' | '),
            NouvelleColonne is Colonne+1, afficher_ligne(Plateau,Ligne,NouvelleColonne).
```

```
% Permet de deplacer le joueur
deplacer_joueur(Plateau,NouveauPlateau,Cont) :- afficher_plateau(Plateau), nl, write('Choisir un
deplacement'),nl,
                                repeat, obtenir_deplacement(Position), (position(Position,Plateau)-> true
;
                                nl, write('Cette colonne est pleine'), nl, fail), !,
                                deplacer(Position,Plateau,x,NouveauPlateau),
                                gagne(Position,NouveauPlateau,joueur,Cont).

% Obtenir le deplacement
obtenir_deplacement(Position) :- repeat,
          afficher_obtenir_deplacement(X),
          X>=49,
          X=<55,
         !,
         Position is X-48.

% Afficher le message pour obtenir le deplacement
afficher_obtenir_deplacement(X) :- nl,write('Choisir 1-7.'),nl,get(X).

% Permet de deplacer l'ordinateur
deplacer_ordinateur(Plateau,Plateau,joueur_gagne,_).
deplacer_ordinateur(Plateau,NouveauPlateau,_,Cont2) :- calcul_deplacement(Plateau,Position),
                                   deplacer(Position,Plateau,o,NouveauPlateau),
                                   gagne(Position,NouveauPlateau,ordinateur,Cont2).

% Calcule le deplacement de l'ordinateur
calcul_deplacement(Plateau,Position) :- position(Position,Plateau).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Code de victoire
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Est-ce que le plateau est plein?
egal(Plateau) :- not(position(_,Plateau)), nl, nl, write('La partie est nulle!'), nl.

% Permet de veifier si il y a une victoire verticallement
gagne_verticalement(Position,Plateau,W) :- !,
                        position_liste(Position,Plateau,Colonne),
                        longeur(Colonne,N),
                        N >= 4,
                        N1 is N-1,
                        position_liste(N1,Colonne,W),
```

```
                    N2 is N-2,
                    position_liste(N2,Colonne,W),
                    N3 is N-3,
                    position_liste(N3,Colonne,W).

% Permet de veifier si il y a une victoire horizontalement ou diagonalement
gagne_horizontalement_ou_diagonalement(Position,Plateau,W) :-
position_liste(Position,Plateau,Colonne),
                                longeur(Colonne,N),
                                somme(1,1,0,N,Plateau,W,0,SommeHorizontale),
                                somme(1,Position,-1,N,Plateau,W,0,SommeDiagonaleBas),
                                somme(1,Position,1,N,Plateau,W,0,SommeDiagonaleHaut),
                                !,
                                (SommeHorizontale >= 4 ; SommeDiagonaleBas >= 4 ;
SommeDiagonaleHaut >=4 ).

% Valeur des positions dans le plateau
valeur_position(X,Y,Plateau,_,M1,M2) :-position_liste(X,Plateau,Colonne),
                                        longeur(Colonne,N),
                                        N <Y,
                                        (M1 >=4 -> M2 is M1; M2 is 0).
valeur_position(X,Y,Plateau,W,M1,M2) :- position_liste(X,Plateau,Colonne),
                                        position_liste(Y,Colonne,W),
                                        M2 is M1+1.
valeur_position(_,_,_,_,M1,M2) :- (M1 >=4 -> M2 is M1; M2 is 0).

% Somme du nombre de jeton
somme(8,_,_,_,_,_,Somme,Somme).
somme(I,Position,Pente,N,Plateau,W,Somme1,Somme) :- Z is I-Position,
                                Y is Pente*Z+N,
                                ((Y =<6, Y>0) -> valeur_position(I,Y,Plateau,W,Somme1,Somme2)
                                ;Somme2 is Somme1),
                                J is I+1,
                                somme(J,Position,Pente,N,Plateau,W,Somme2,Somme).

% Trouve une victoire si il y en a une
gagne(Position,Plateau,ordinateur,ordinateur_gagne) :- gagne_verticalement(Position,Plateau,o).
gagne(Position,Plateau,ordinateur,ordinateur_gagne) :-
gagne_horizontalement_ou_diagonalement(Position,Plateau,o).
gagne(Position,Plateau,joueur,joueur_gagne) :- gagne_verticalement(Position,Plateau,x).
gagne(Position,Plateau,joueur,joueur_gagne) :-
gagne_horizontalement_ou_diagonalement(Position,Plateau,x).
gagne(_,_,_,cont).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicats de service
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% joindre(L1,L2,L3) := Permet de joindre la liste 1 (L1) et liste 2 (L2)
%                      dans la liste 3 (L3).

joindre([],L,L).
joindre([X|L1],L2,[X|L3]) :- joindre(L1,L2,L3).

% position_liste(I,L,Z) := Z est l'élément de la liste L
%          := on L est une liste vide retourne une espace vide.

position_liste(_,[],' ').
position_liste(1,[Y|_],Z) :- Y=Z.
position_liste(I,[_|W],Z) :- J is I-1, position_liste(J,W,Z).

% queue_liste(I,L,L2) := La liste L2 est la liste qui se retrouve après le I ième élément de L

queue_liste(0,L,L2):- L=L2.
queue_liste(I,[_|W],L2) :- J is I-1, queue_liste(J,W,L2).

% tete_liste(I,L,L2) := List L2 is the first I elts of L

tete_liste(I,L,L2) :- queue_liste(I,L,L3), joindre(L2,L3,L).

% remplacer_element(I,E,L1,L2) := L2 est la liste obtenu en
%                      remplaçant la valeur à la position I avec
%                      la valeur E.
remplacer_element(I,E,L1,L2) :- J is I-1,
                      tete_liste(J,L1,L3),
                      joindre(L3,[E],L4),
                      queue_liste(I,L1,L5),
                      joindre(L4,L5,L2).

% longeur(L,N) := N is longueur de la liste L

longeur([], 0).
longeur([_|Q], N) :- longeur(Q, N1), N is N1 + 1.
```

Résultats et discussion (20%)

Discussion des jeux d'essais

Nous n'avons pas réussi à implémenter Minimax dans notre programme. En ce qui concerne l'affichage, nous nous sommes inspirés de quelques exemples. Pour le joueur intelligent, notre implémentation choisit la première colonne qui n'est pas pleine pour jouer. Cela signifie qu'il n'est pas si intelligent. Par exemple, voici un jeu où le joueur gagne en choisissant simplement la deuxième colonne.

```
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7

Choisir un deplacement

Choisir 1-7.
|: 2
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | x |   |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7
```

```
Choisir un deplacement

Choisir 1-7.
|: 2
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | x |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | x |   |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7

Choisir un deplacement

Choisir 1-7.
|: 2
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | x |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | x |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | x |   |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7

Choisir un deplacement

Choisir 1-7.
```

```
|: 2
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   | x |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | x |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | x |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | x |   |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7

Vous avez gagne!
```

On peut alors bien voir que puisque c'est le joueur qui joue en premier, c'est le joueur qui gagne. Si on place nos jetons dans la première colonne, il ira dans la deuxième colonne lorsque la première sera pleine.

```
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7

Choisir un deplacement

Choisir 1-7.
|: 1
+---+---+---+---+---+---+---+
```

```
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7
```

Choisir un deplacement

Choisir 1-7.
|: 1

```
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7
```

Choisir un deplacement

Choisir 1-7.
|: 1

```
+---+---+---+---+---+---+---+
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
```

```
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7
```

Choisir un deplacement

Choisir 1-7.
|: 2

```
+---+---+---+---+---+---+---+
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | o |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x | x |   |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7
```

Choisir un deplacement

Choisir 1-7.
|: 3

```
+---+---+---+---+---+---+---+
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x | o |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | o |   |   |   |   |   |
```

```
+---+---+---+---+---+---+---+
| x | x | x |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7

Choisir un deplacement

Choisir 1-7.
|: 4
+---+---+---+---+---+---+---+
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x | o |   |   |   |   |   |
+---+---+---+---+---+---+---+
| o | o |   |   |   |   |   |
+---+---+---+---+---+---+---+
| x | x | x | x |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7

Vous avez gagne!
```

Dans ce dernier exemple, le joueur est capable de gagner quand même en jouant au travers de l'ordinateur.

Avantages et limites

Nous n'avons donc pas atteint nos objectifs puisque cet adversaire est trop facile à battre. Cette heuristique est limitée au niveau de l'intelligence. Par contre, il est très performant, parce qu'il prend la première solution possible.

En ce qui concerne minimax, il est très difficile de battre un programme qui l'utilise, car celui-ci pense plusieurs tours à l'avance. Par contre, le temps de réflexion de l'intelligence augmente exponentiellement pour chaque tour qu'on permet à l'heuristique de prévoir. C'est pour cela qu'on limite souvent l'heuristique à une profondeur courte. Cette profondeur courte cause donc l'heuristique à être limitée en début de partie et permet de trouver une colonne rapidement. Par contre, cet algorithme fait très peu d'erreurs en fin de partie.

Améliorations possibles

Il aurait aussi été intéressant de l'implémenter avec AlphaBêta pour voir les différences au niveau des limites et des performances. Cette amélioration, tant au niveau de Minimax et d'AlphaBêta, aurait permis de jouer contre un adversaire de compétition. AlphaBêta est une amélioration de Minimax qui permet de réduire le nombre de nœuds dans l'algorithme de minimax. Cette amélioration permettrait donc d'augmenter la performance en diminuant le temps d'exécution.

Conclusion

Ce que nous avons accompli :

Nous avons trouvé une procédure adaptée pour le jeu Puissance 4 à savoir minimax et sa valeur heuristique calculée à partir de 4 règles (cas de victoire, cas de défaite, cas d'une partie nulle, cas de la partie en cours). Au niveau de la programmation, nous avons réussi à afficher la grille et permis la jouabilité entre un joueur réel et un ordinateur. Par contre, nous n'avons pas réussi à implémenter minimax.

Ce que nous aurions voulu faire :

Nous aurions voulu implémenter la procédure minimax dans le programme et, une fois faite, utiliser d'autres règles pour les comparer et voir lesquelles sont les plus efficaces. Cela aurait été d'autant plus intéressant avec AlphaBêta.

Bibliographie

Wikipédia. [En ligne]. https://fr.wikipedia.org/wiki/Puissance_4 (Page consultée le 27 février 2016)

Roadtolarissa. [En ligne]. http://roadtolarissa.com/connect-4-ai-how-it-works/ (Page consultée le 27 février 2016)

RMarcus. [En ligne]. https://rmarcus.info/blog/2014/12/23/connect4.html (Page consultée le 27 février 2016)

Wikipedia. [En ligne]. https://en.wikipedia.org/wiki/Minimax (Page consultée le 27 février 2016)

Computer Science 440 Nonnumeric Computation. [En ligne]. http://www.cs.sjsu.edu/faculty/pollett/440.1.97f/ (Page consultée le 27 février 2016)