RAPPORT PUISSANCE 4

EN ÉQUIPE

Auteurs

Alex Lebrun - alex.lebrun.1@ulaval.ca

Walter Bonetti - walter.bonetti.1@ulaval.ca

Isabelle Eysseric - isabelle.eysseric.1@ulaval.ca

François Martineau- francois.martineau.4@ulaval.ca

JEU INTELLIGENT

RAPPORT PUISSANCE 4

EN ÉQUIPE

Sommaire

RAPPORT PUISSANCE 4

1. Description du jeu

1.1 Présentation générale du jeu

***Figure 1** : Plateau de Puissance 4 de la société de jeu Hasbro.*

Puissance 4, ou Connect 4 pour les anglophones, est un jeu de société stratégique. Il a été inventé par Howard Wexler et date de 1974. Il est actuellement commercialisé par la société de jeu Hasbro.

Le premier joueur à aligner une suite de 4 pions d'une même couleur remporte la partie.

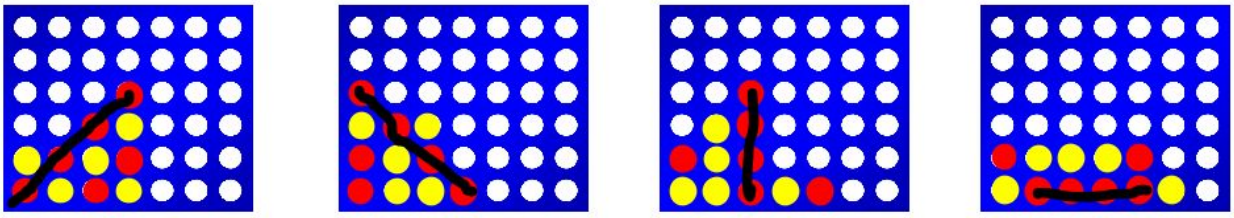


***Figure 1.3** : Exemple d'alignements possibles avec les positions diagonales, verticales et horizontales.*

En 1988, une solution exacte au jeu a été trouvée avec des calculs informatiques par James D. Allen et publie un livre avec cette stratégie infaillible. Le premier joueur peut toujours gagner en fonction d'où il débute le jeu et si ses coups sont parfaits.

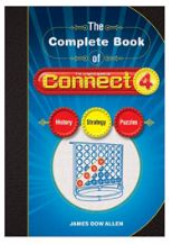


Figure 1.2 : Livre de James D. Allen, The Complete Book of Connect 4

1.2 Description des règles

Le jeu se joue à 2 personnes sur une grille de 6 lignes et 7 colonnes, ce qui donne 42 cases vides pour placer les jetons. Chaque joueur a un total de 21 jetons, ce qui permet de remplir la grille dans sa totalité. Chacun possède une couleur de jeton, par convention rouge et jaune, ce qui permet de le différencier de son adversaire.

Un des deux joueurs au choix commence en plaçant un jeton de sa couleur dans une des colonnes. Le jeton coulisse jusqu'au bas de la grille et c'est au tour du joueur suivant qui fera de même. Il n'est pas autorisé de bouger les jetons une fois posés dans la grille.

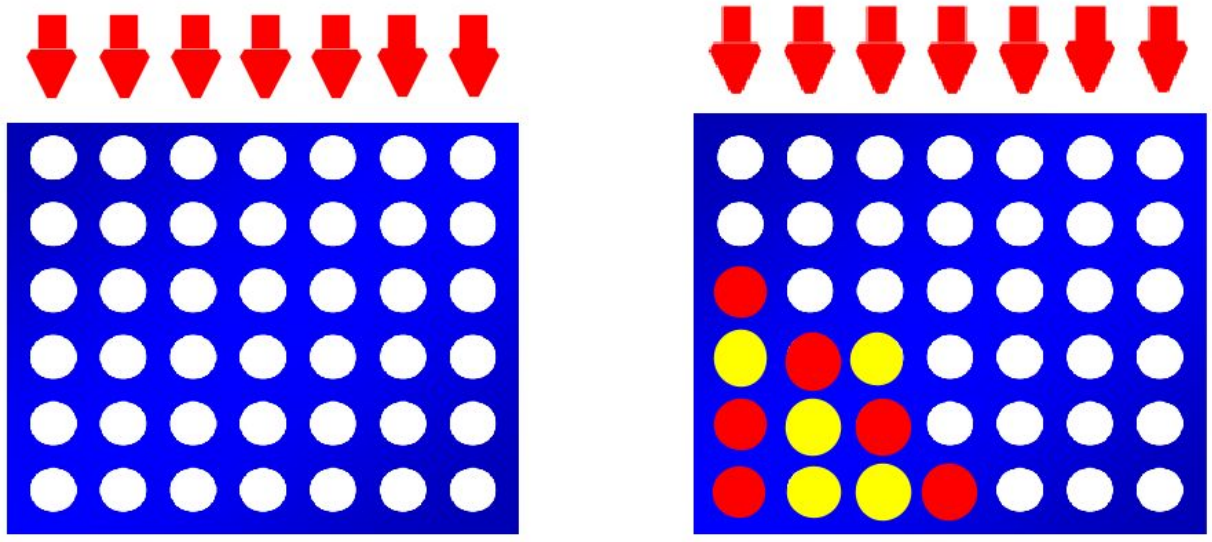


Figure 1.4 *: Exemple de grilles lors d'une partie. La première représente le début d'une partie avec une grille vide et la seconde une partie entamée. Les flèches rouges sur le dessus indiquent le sens d'empilement des jetons.*

Le gagnant sera celui qui arrive à aligner 4 jetons consécutifs de la même couleur. Il est possible d'aligner ses jetons horizontalement, verticalement ou bien horizontalement.

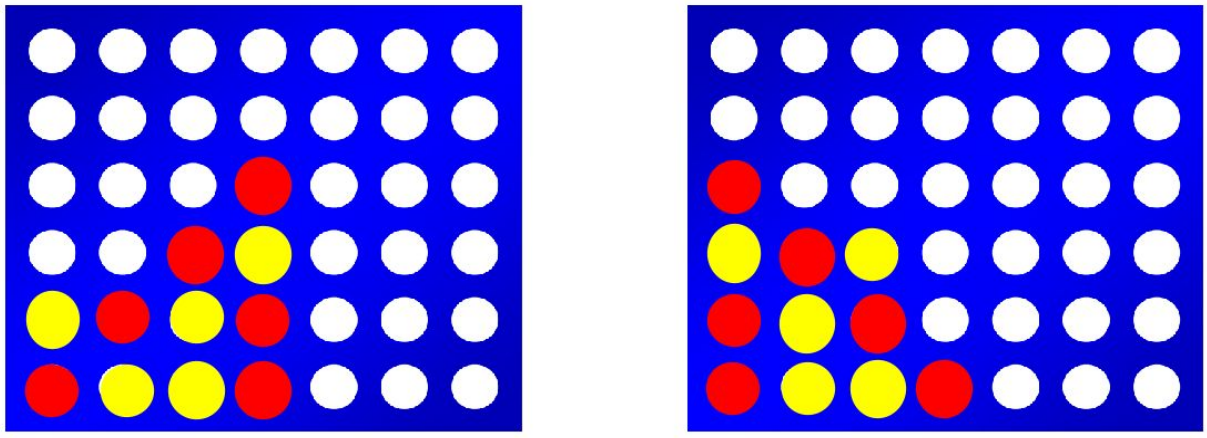


Figure 1.5 *: Exemple de grilles lors d'une partie. La première et la deuxième représentent un alignement vainqueur en diagonale droite ou gauche avec les jetons rouges.*

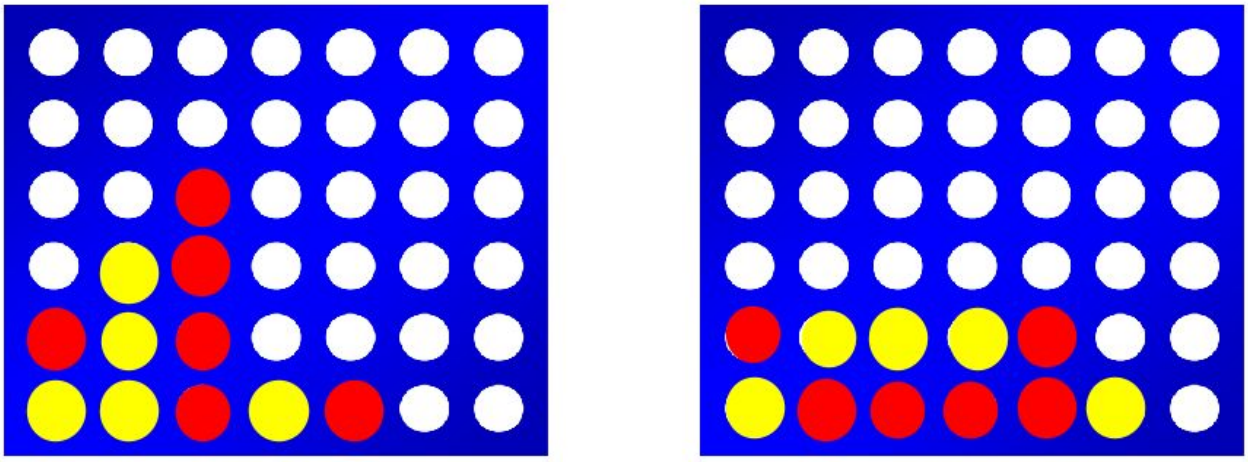


Figure 1.6 *: Exemple de grilles lors d'une partie. Les deux grilles représentent un alignement vainqueur à la verticale ou à l'horizontale avec les jetons rouges.*

Quand il ne reste plus de jetons en jeu, c'est que la grille est remplie et qu'aucun des joueurs n'a réussi à aligner 4 jetons d'affilés. La partie est donc déclarée nulle et il n'y a donc pas de vainqueur.

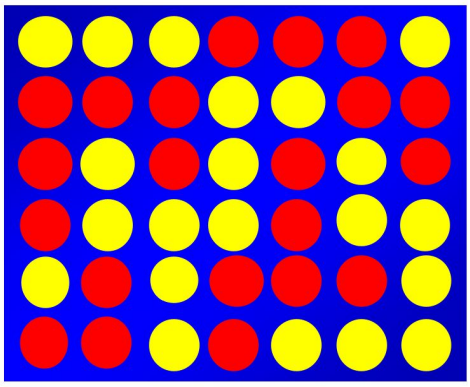


Figure 1.7 *: Exemple de grilles d'une partie déclarée nulle car aucun des deux joueurs n'a réussi à aligner quatre jetons de sa couleur.*

Il est interdit de retirer son coup une fois mis pour finalement le mettre à un autre endroit ou bien modifier celui de son adversaires. Une fois le jeton lâché dans la grille, celui ne peut être en aucun cas modifié. Aussi, le pion doit être sur la ligne la plus en bas ou bien sur un jeton déjà présent mais ne doit pas avoir de cases vide qui le sépare de la ligne la plus basse disponible.

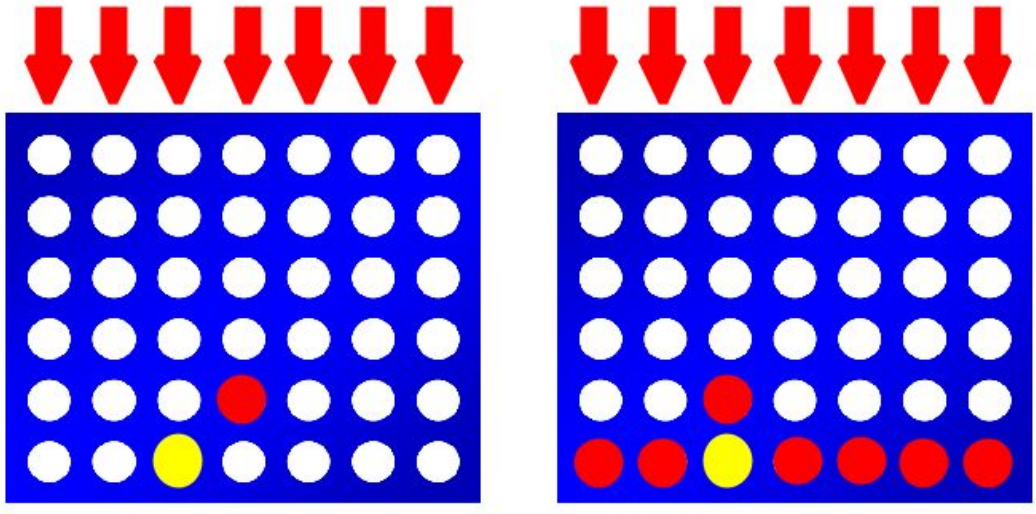


***Figure 1.8** : Exemple d'un coup non valide dans la grille de gauche, et tous les emplacements que le jeton rouge peut prendre dans la grille de droite.*

1.3 Description de l'agent intelligent et son environnement [1]

L'environnement de l'agent est déterministe, complètement observable. Il joue à tour de rôle et contre un adversaire.

- **Entièrement observable**, puisqu'il a accès à la totalité de son environnement.
- **Multiagent concurrentiel**, puisqu'il joue contre un adversaire et tente de le gagner.
- **Déterministe**, puisque l'état suivant est complètement déterminé par l'état courant grâce à la liste de combinaisons possibles.
- **Séquentiel**, puisque la décision que l'agent aura un impacte sur ses actions et les actions de son adversaires dans les tours à venir.
- **Statique** puisque l'agent n'a pas besoin d'observer l'environnement pendant que l'adversaire joue son tour.
- **Discret** puisque le jeu possède un nombre fini d'opérations possibles.

Environnement de tâche	Observable	Agents	Déterministe	Épisiodique	Statique	Discret
Puissance 4	Entièrement	Multi	Déterministe	Séquentiel	Statique	Discret

***Figure 1.9** : Environnement de tâche d'une partie de jeu Puissance 4 et ses caractéristiques.*

2. Modélisation problème et solution

2.2 Description du problème

Le problème d'une partie de jeu peut être représentée par un espace d'états avec 4 états différents:

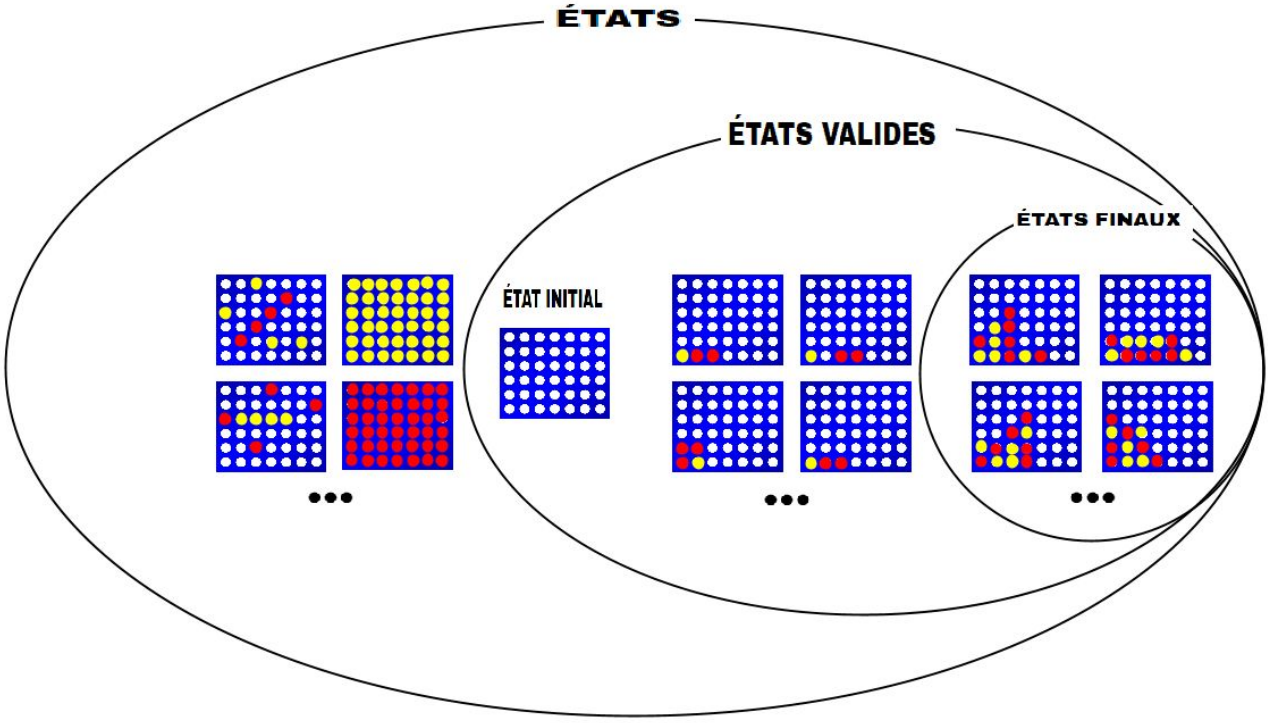


***Figure 2.1**: Représentation visuelle de l'espace d'états avec des combinaisons possibles pour chaque catégorie mais pas toutes énumérées.*

- **États (noeuds)**: l'ensemble de tous les états possibles incluant l'état initial, les états finaux, les opérations possibles ainsi que les états non possibles comme mettre un jeton au-dessus d'une case vide.

- **États valides (transitions)**: Il y a seulement 6 possibilités qui équivaut au nombre de colonnes du jeu. C'est-à-dire jouer la colonne 1, 2, 3, 4, 5, 6 ou 7. Pour le joueur de couleur rouge par exemple, il mettra un de ses jetons dans une des cases vides ou sur un autre jeton, peu importe sa couleur. Il en sera de même pour le joueur de couleur jaune.

- **État initial (racine)**: L'état initial est celui de la grille vide 6 x 7, contenant 42 cases vides au total puisque c'est l'état avant que le jeu commence.

- **États finaux (feuilles)**:
    - Toutes les cases de la grille sont remplies
    - Le but est atteint pour le joueur Rouge ou Jaune. Pour le joueur de couleur rouge par exemple, le tableau contient un jeton rouge sur 4 cases verticales, horizontales ou diagonalement. Il en est de même pour le joueur de couleur jaune.

On peut représenter l'arbre de jeu pour une partie, dont les noeuds sont des états du jeu et les arêtes sont des coups.

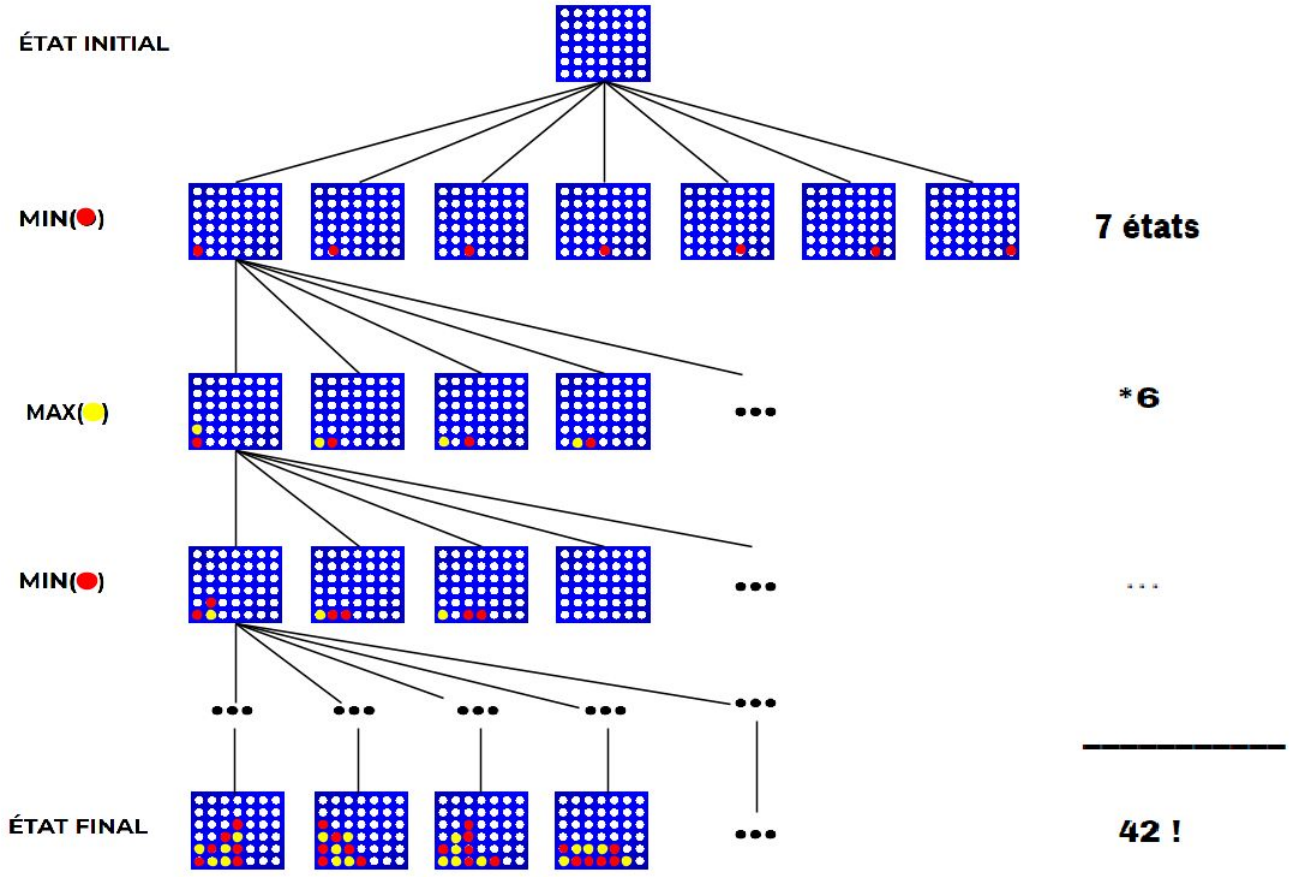


***Figure 2.2**: Représentation visuelle de la taille de l'espace d'états avec des combinaisons possibles mais pas toutes énumérées. Et 42! donne* 42! = 1.4050061e+51 états

Fonctions heuristiques: $f = g + h$

- **G** : coût pour aller du noeud initial au noeud courant dépendamment du nombre de jetons bien placés.
- **H** : coût additionnel pour aller du noeud courant au noeud solution dépendamment de la profondeur du graphe

Test de but: Vérifier si 4 jetons d'une même couleur sont alignés ou bien si la grille est pleine.

2.3 Description de la solution

Notre algorithme s'inspire de la recherche Alpha-Beta, mais étant donné la difficulté à avoir un résultat pour une recherche trop profonde (restrictions de mémoire), nous avons opté pour une recherche en largeur plutôt qu'en profondeur.

Nous fonctionnons avec un système d'état. Il n'y aura que deux types d'états qui seront listés dans le programme.

État_initial : État de la grille de jeu alors qu'aucun coup n'a été joué.

Etat_final : Les états qui entraînent un arrêt du jeu, soit avec un gagnant, soit une partie nulle.

Voici donc une liste des fonctions que nous utiliserons pour résoudre le problème.

Pour travailler avec les états

`etat_initial(E).`
Retourne un état de départ pour le jeu avec des case vides dans la variable E.

`etat_final(ETAT,GAGNANT).`
Retourne "true" si ETAT fait partie d'un état final.

`operation(CASE, ETAT, OP, ETAT_SUIVANT).`
Cette fonction permet de passer d'un état de départ(ETAT) à l'autre (ETAT_SUIVANT) en jouant une CASE.

`n(x,o).  n(x,o).`
Cette fonction fait le changement de joueur. Le premier joueur est "x" et le 2e est "o".

Pour faire l'affichage

`afficher_grille(ETAT).`
Cette fonction affiche la grille de jeu selon l'ETAT passé en paramètre.

`trou(Jeton).`
Cette fonction affiche le contenu de la case dans le tableau.

`afficher_operation(CASE, tracer(Joueur,Case)).`
Cette fonction affiche un message décrivant ce qu'à fait le joueur comme choix de jeu.

`afficher_gagnant(Joueur).`
Cette fonction affiche le message qui donne le joueur gagnant ou un message "Partie nulle".

Diverses fonctions utilitaires

`switch(X, [Val:Goal |Cases]).`
Cette fonction fait le lien entre la valeur (Val) d'une variable "X" donnée en paramètre et une fonction (Goal). Les combinaisons sont stockées dans une liste (Cases) passées en paramètre.

`remplacement(X, [Val:New|Cases], Y).`
Cette fonction remplace une variable (X) par une nouvelle valeur (New) selon la liste (Cases) de combinaisons (Valeur : Nouvelle valeur) qui lui est passée en paramètre.

`update_list(Case, Liste,NouvelleListe).`
Cette fonction servira à modifier la liste de cases disponibles à jouer(Liste) et à retourner une nouvelle liste de cases disponibles (NouvelleListe) après avoir jouer la case (Case).

Pour jouer la partie

connect4.
C'est la fonction qui commence le jeu.

jeu(LIST, ETAT, GAGNANT).
Cette fonction effectue les opération d'un tour de jeu. La liste entrée en paramètre (LIST) est la liste des cases disponibles pour jouer. ETAT représente l'état actuel du jeu. GAGNANT représente les combinaisons gagnantes, c'est ça qui nous permet de vérifier si la partie doit encore jouer un tour ou s'arrêter.

Pour l'IA

choisirOperation (Liste, ETAT, NouvelleListe, ETAT_SUIVANT) .
Cette fonction sert uniquement à commencer la recherche.

On lui fournit l'état actuel du jeu (ETAT) et la liste des noeuds ouverts (Liste) et elle retournera dans ses paramètres le nouvel état du jeu après son choix (ETAT_SUIVANT) ainsi que la nouvelle liste de noeuds ouverts après son mouvement (NouvelleListe).

testerVictoireMAX(ETAT, [X|LISTE])
Cette fonction sert à indiquer si l'un des états de la liste entraînera une victoire du joueur.

On lui donne en paramètre l'état actuel de la grille de jeu (ETAT) et la liste de noeuds ([X|LISTE]). La fonction retourne "true" aussitôt qu'elle teste un X qui l'a fait gagner. Sinon elle retourne "false".

testerVictoireMIN(ETAT, [X|LISTE])
Cette fonction sert à indiquer si l'un des états de la liste entraînera une victoire du joueur.

On lui donne en paramètre l'état actuel de la grille de jeu (ETAT) et la liste de noeuds ([X|LISTE]). La fonction retourne "true" aussitôt qu'elle teste un X qui la fait gagner, sinon elle retourne "false".

choisirMAX(ETAT, LISTE_ORIG, Liste, CHOIX, ETAT_FINAL, NEWLIST)
Cette fonction est la fonction récursive qui prendra la décision.

On lui fournit l'état actuel du jeu (ETAT) et la liste de noeuds ouverts au commencement de la recherche (LISTE_ORIG). La fonction va prendre une liste de noeuds possibles comme paramètre. Cette liste (Liste) sera modifiée à l'intérieure de la fonction pour rechercher la combinaison demandée. Il s'agit principalement d'une liste qui ne contiendra pas les éléments déjà testés.

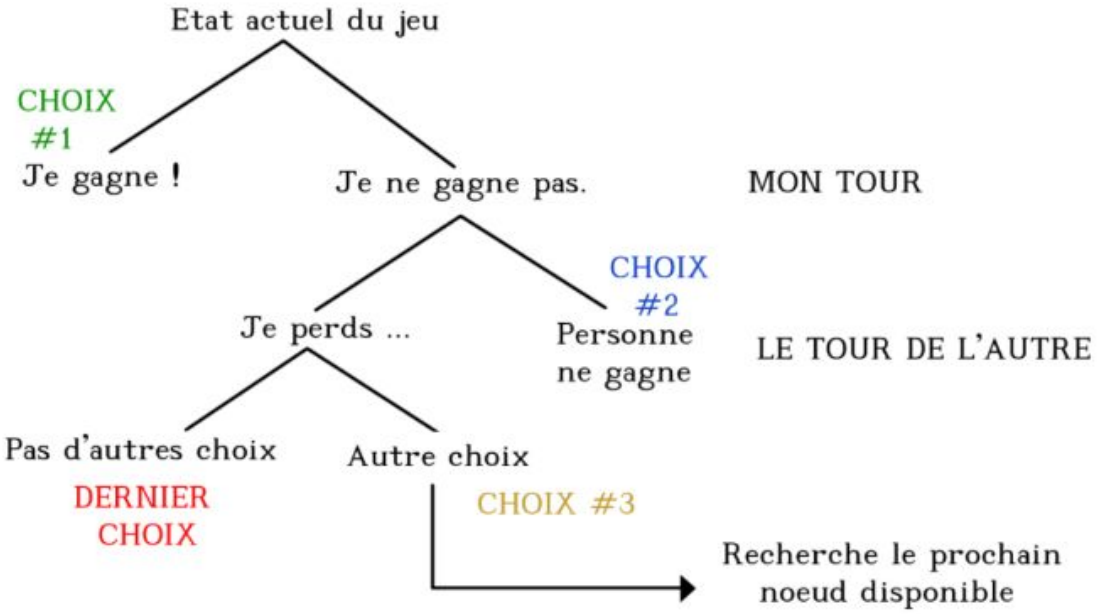
Elle retournera dans ses paramètres, la case qui est son choix de mouvement CHOIX), l'état de la grille de jeu qui suivra ce mouvement (ETAT_FINAL) ainsi que la liste des noeuds qui seront ouverts après ce mouvement (NEWLIST).

Pseudo-code

```
Choisir une mouvement selon un ETAT actuel de jeu et une liste de
choix possibles
   -  On choisit un  noeud ouvert dans la liste Noeuds_Ouverts
   -  On enlève ce noeud de la liste
   -  On tente l'état qui en résulterait (ETAT_SUIVANT)

   ETAT_SUIVANT est gagnant?
      OUI:
        -  On choisit ce noeud sans hésiter.
      NON:
          ETAT_SUIVANT mettrait l'adversaire en position de
          gagner?
              OUI:
                  Avons nous d'autres choix?
                     OUI:
                     -  On  recommence  la  recherche  en  éliminant
                        ce noeud des options.
                     NON:
                     -  On doit prendre cette option de force.
              NON:
                -  Ce noeud est jugé comme "satisfaisant" car il
                   ne permet pas à l'adversaire de gagner
```

Voici à quoi ressemble l'arbre de décision:



3. Implémentation

```
%%%%%%%%%%%%%%%%%%%%%%%%    LES ETATS    %%%%%%%%%%%%%%%%%%%%%%

%  Board:          1    2    3    4    5
%             a   a1   a2   a3   a4   a5
%             b   b1   b2   b3   b4   b5
%             c   c1   c2   c3   c4   c5
%             d   d1   d2   d3   d4   d5
%             e   e1   e2   e3   e4   e5
%             f   f1   f2   f3   f4   f5
%
%  Liste qui représente le board:
%      p4(a1,a2,a3,a4,a5,b1,b2,b3,b4,b5,c1,c2,c3,c4,c5,
%         d1,d2,d3,d4,d5,e1,e2,e3,e4,e5,f1,f2,f3,f4,f5, JOUEUR_ACTIF)
%
%   _ :  La valeur de la case peut être n'importe quoi
%   J :  La case doit être occupée par une pièce du joueur

%Etat initial
etat_initial(p4(v,v,v,v,v,v,v,v,v,v,v,v,v,v,v,v,v,v,v,v,v,v,v,v,x)).

% Combinaisons gagnantes
etat_final(p4(J,J,J,J,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,J,J,J,J,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,J,J,J,J,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,J,J,J,J,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,_,J,J,J,J,_,_,_,_,_,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,_,_,J,J,J,J,_,_,_,_,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,J,J,J,J,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,J,J,J,J,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,J,J,J,J,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_),J) :- not(J=v).
etat_final(p4(_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,J,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,J,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_),J) :- not(J=v).
etat_final(p4(J,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,J,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J,_),J) :- not(J=v).
etat_final(p4(_,_,_,_,_,_,_,_,_,_,_,_,_,_,J,_,_,_,_,J,_,_,_,_,J),J) :- not(J=v).

% Combinaison de la partie nulle
etat_final(p4(  A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,
                D1,D2,D3,D4,D5, E1,E2,E3,E4,E5,_),J) :-
                                not(A1=v), not(A2=v), not(A3=v), not(A4=v), not(A5=v),
                                not(B1=v), not(B2=v), not(B3=v), not(B4=v), not(B5=v),
                                not(C1=v), not(C2=v), not(C3=v), not(C4=v), not(C5=v),
                                not(D1=v), not(D2=v), not(D3=v), not(D4=v), not(D5=v),
                                not(E1=v), not(E2=v), not(E3=v), not(E4=v), not(E5=v).
% Changement de joueur
n(x,o).
n(o,x).
```

```
%%%%%%%%%%    LES OPERATIONS   %%%%%%%%%%%%%%%%

% Changement d'un ETAT à un ETAT suivant
%  PARAM     1er -Nom de la case (pour faire un choix)
%             2e  -ETAT de départ (avant de jouer)
%             3e  -Commande à faire entre ETAT et ETAT_SUIVANT
%             4e  -ETAT du board après avoir joué la case (1er param)
%
%  Lorsque les noms sont en lettres majuscules dans les ETATS, ce sont des
%  variables qui peuvent prendre 3 valeurs:     x: Joueur1
%                                               o: Joueur2
%                                               v: vide


% Opérations sur ligneA
operation(a1, p4(v,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,A1),
          p4(J1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(a2, p4(A1,v,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5, J1),
          tracer(J1,A2),
          p4(A1,J1,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(a3, p4(A1,A2,v,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,A3),
          p4(A1,A2,J1,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(a4, p4(A1,A2,A3,v,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,A4),
          p4(A1,A2,A3,J1,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(a5, p4(A1,A2,A3,A4,v,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,A5),
          p4(A1,A2,A3,A4,J1,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).


% Opérations sur la ligneB
operation(b1, p4(A1,A2,A3,A4,A5,v,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,B1),
          p4(A1,A2,A3,A4,A5,J1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(b2, p4(A1,A2,A3,A4,A5,B1,v,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,B2),
          p4(A1,A2,A3,A4,A5,B1,J1,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(b3, p4(A1,A2,A3,A4,A5,B1,B2,v,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,B3),
          p4(A1,A2,A3,A4,A5,B1,B2,J1,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(b4, p4(A1,A2,A3,A4,A5,B1,B2,B3,v,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,B4),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,J1,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(b5, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,v,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,B5),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,J1,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
```

```
% Opérations sur la ligneC
operation(c1, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,v,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,C1),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,J1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(c2, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,v,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,C2),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,J1,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(c3, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,v,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,C3),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,J1,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(c4, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,v,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,C4),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,J1,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(c5, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,v,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,C5),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,J1,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).

% Opérations sur la ligneD
operation(d1, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,v,D2,D3,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,D1),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,J1,D2,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(d2, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,v,D3,D4,D5,E1,E2,E3,E4,E5, J1),
          tracer(J1,D2),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,J1,D3,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(d3, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,v,D4,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,D3),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,J1,D4,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(d4, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,v,D5,E1,E2,E3,E4,E5,J1),
          tracer(J1,D4),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,J1,D5,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(d5, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,v,E1,E2,E3,E4,E5,J1),
          tracer(J1,D5),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,J1,E1,E2,E3,E4,E5,J2))
          :- n(J1,J2).

% Opérations sur la ligneE
operation(e1, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,v,E2,E3,E4,E5,J1),
          tracer(J1,E1),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,J1,E2,E3,E4,E5,J2))
          :- n(J1,J2).
operation(e2, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,v,E3,E4,E5,J1),
          tracer(J1,E2),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,J1,E3,E4,E5,J2))
          :- n(J1,J2).
operation(e3, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,v,E4,E5,J1),
          tracer(J1,E3),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,J1,E4,E5,J2))
          :- n(J1,J2).
operation(e4, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,v,E5,J1),
          tracer(J1,E4),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,J1,E5,J2))
          :- n(J1,J2).
operation(e5, p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,v,J1),
          tracer(J1,E5),
          p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,J1,J2))
          :- n(J1,J2).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE %%%%%%%%%%%%%%%%%%%%%%%%%

% Affichage de la grille
afficher_grille(p4(A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,E1,E2,E3,E4,E5,_))
:-
    write("   "), write(" 1  "), write(" 2  "), write(" 3  "), write(" 4  "), write(" 5  "), nl,
    write("a "), trou(A1),trou(A2),trou(A3),trou(A4),trou(A5),writeln('|'),
    write("b "), trou(B1),trou(B2),trou(B3),trou(B4),trou(B5),writeln('|'),
    write("c "), trou(C1),trou(C2),trou(C3),trou(C4),trou(C5),writeln('|'),
    write("d "), trou(D1),trou(D2),trou(D3),trou(D4),trou(D5),writeln('|'),
    write("e "), trou(E1),trou(E2),trou(E3),trou(E4),trou(E5),writeln('|'),
    nl.

%Affiche la case
trou(Jeton) :-
    write('| '),
    switch(Jeton, [v: write(' '), x: write('○'), o: write('●')]),
    write(' ').

%Affiche le choix du joueur (MOVE effectué)
afficher_operation(CASE, tracer(Joueur, Case)) :-
    write('Joueur '),
    switch(Joueur, [v: write(' '), x: write('○'), o: write('●')]),
    write(' joue à '),
    writeln(CASE).

%Affiche le joueur gagnant
afficher_gagnant(Joueur) :-
    switch(Joueur, [v: write("La partie est nulle, personne ne"), x: write('○'), o: write('●')]),
    write(" gagne la partie!"), nl.


%%%%%%%%%%%%%%%%   FONCTIONS UTILITAIRES    %%%%%%%%%%%%%%%%

% switch/2 simule le switch procédural
% PARAM     X:      Variable qui correspond a la fonction a faire
%           Val:    La valeur de la variable
%           Goal:   La fonction à effectuer
%
switch(X, [Val:Goal|Cases]) :-
    ( X=Val -> call(Goal) ; switch(X, Cases) ).

%Sert à remplacer une valeur par une autre
% PARAM     X:    Variable à changer
%           Val: Valeur de la variable
%           New: Valeur qui remplacera Val
%
remplacement(X, [Val:New|Cases], Y) :-
    ( X=Val -> Y=New ; remplacement(X, Cases, Y) ).

%Met à jour la liste de cases disponibles
% PARAM    CASE:  La case qui sera jouée
%          Liste: La liste des choix des cases avant la modification
%          NouvelleListe: La liste des choix après que le coup soit joué
%
update_list( Case, Liste, NouvelleListe):-
La nouvelle case que l'on peut atteindre    %On cherche
    remplacement(Case, [e1: d1, d1: c1, c1: b1, b1: a1, a1: rien,
                        e2: d2, d2: c2, c2: b2, b2: a2, a2: rien,
                        e3: d3, d3: c3, c3: b3, b3: a3, a3: rien,
                        e4: d4, d4: c4, c4: b4, b4: a4, a4: rien,
                        eb: db, db: cb, cb: bb, bb: ab, ab: rien],
                 NouvelleCase),

    %On ajoute la nouvelle case à la nouvelle liste, si on est rendu au maximum, on ne rajoute
rien
    (   not(NouvelleCase=rien)
        -> append([NouvelleCase],Liste, LISTE1)
         ; append([], Liste, LISTE1) ),
    %On enlève la case qui vient d'être jouée
    delete(LISTE1, Case, NouvelleListe).
```

```
%%%%%%%%%%%%%%%%%%%% PARTIE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

connect4 :-
   %Les cases permises au départ
   LIST -[e1,e2,e3,e4,e5],
   etat_initial(E), jeu(LIST,E,GAGNANT).

jeu(LIST,ETAT, GAGNANT) :-
    etat_final(ETAT, GAGNANT),!,
    afficher_grille(ETAT),
    afficher_gagnant(GAGNANT).

jeu(LIST, ETAT, GAGNANT) :-
    afficher_grille(ETAT),
    choisir_operation(LIST, ETAT, NEWLIST, ETAT_SUIVANT),
    jeu( NEWLIST, ETAT_SUIVANT, GAGNANT ).

%%%%%%%%%%%%%%%%%%%% ALGORITHME %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

choisir_operation( Liste, ETAT, NouvelleListe, ETAT_SUIVANT):-
    choisirMAX(ETAT, Liste, Liste, CASE, ETAT_SUIVANT, NouvelleListe),  sleep(0.5).

%Test si on peut gagner en un coup
testerVictoireMAX(ETAT, []):-false.
testerVictoireMAX(ETAT, [X|LISTE]):-
    operation(X,ETAT,OP,ETAT_SUIVANT),
    (   etat_final(ETAT_SUIVANT,GAGNANT) -> true ; testerVictoireMAX(ETAT, LISTE)).

%Test si ce coup permet à l,adversaire de gagner
testerVictoireMIN(ETAT, []):-false.
testerVictoireMIN(ETAT, [X|LISTE]):-
    update_list(X, [X|LISTE], NouvelleListe),
    operation(X,ETAT,OP,ETAT_SUIVANT),
    (   testerVictoireMAX(ETAT_SUIVANT, NouvelleListe) -> true; testerVictoireMIN(ETAT, LISTE)).

% Fin de la recherche, affichage, operation, etc.
choisirMAX(ETAT,LISTE_ORIG,[], CHOIX,ETAT_FINAL, NEWLIST):-
                    operation( CHOIX, ETAT, OP, ETAT_FINAL ),
                    afficher_operation(CHOIX, OP),
                    update_list( CHOIX, LISTE_ORIG, NEWLIST).

%Recherche d'un coup gagnant
choisirMAX(ETAT, LISTE_ORIG,Liste, CHOIX,ETAT_FINAL,NEWLIST):-

        %on choisit aléatoirement un élément de la liste pour rendre ça plus intéressant
        random_member(X,Liste),
        delete(Liste,X,LISTE),
        update_list(X,Liste,NouvelleListe),
        operation(X,ETAT,OP,ETAT_SUIVANT),

     % on analyse l'état obtenu après ce mouvement
     (    testerVictoireMAX(ETAT, NouvelleListe)

            % Si oui, on a trouvé notre choix
            -> choisirMAX(ETAT, LISTE_ORIG, [], X, ETAT_FINAL, NEWLIST)
            % Sinon, on vérifie que ce mouvement ne fera pas gagner MIN le tour suivant
            ;  % Teste si ce movement peut permettre à MIN de gagnr rendu a son tour
               (   testerVictoireMIN(ETAT_SUIVANT, NouvelleListe)
                   % Si la case permet à MIN de gagner,
                   % on recommence la recherche en l'enlevant des choix
                   ->  (   length(NouvelleListe,1)

                           % Si on a qu'un choix, on le prend....
                           -> choisirMAX(ETAT, LISTE_ORIG, [], X, ETAT_FINAL, NEWLIST)
                           % Sinon on cherche avec une autre possibilité
                           ; choisirMAX(ETAT, LISTE_ORIG, LISTE, CHOIX, ETAT_FINAL, NEWLIST))

                   ;% Sinon on prend ce choix car il ne permet pas à MIN de gagner au moins
                    choisirMAX(ETAT, LISTE_ORIG, [], X, ETAT_FINAL, NEWLIST)
               )
     ).
```

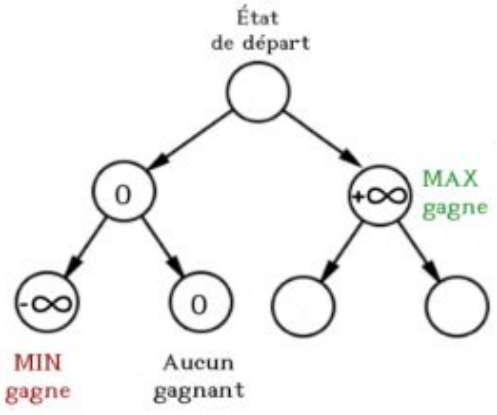
4. Résultats et discussion

4.1 Les résultats obtenus

L'algorithme cherche en premier si l'un des coups suivants peut le faire gagner. S'il en trouve un, il la choisit sans chercher plus loin. Notre Alpha ici est absolu. Aussitôt qu'une combinaison est gagnante en un coup, elle est prise. Si aucun coup ne même à une victoire, on regarde ce que MIN pourra jouer. Le but ici sera de l'empêcher d'atteindre un état gagnant. Notre agent ne cherchant pas plus loin que ce niveau, prendra la première case qu'il rencontre permettant de ne pas faire gagner l'adversaire.

La première difficulté rencontrée fut de comprendre les messages d'erreurs de SWI-Prolog qui ne sont pas toujours évidents. Lors de la réalisation de plusieurs versions différentes de l'algorithme, la mémoire devenait problématique. Les algorithmes plus simples devenaient alors totalement inutilisables et les recherches trop profondes finissaient en "hors pile". Les algorithmes de nature Minimax furent donc rapidement écartés des options. Il a donc fallu faire des sacrifices. Comme vous pourrez le remarquer en exécutant le code, notre espace de jeu ne fait pas 6 cases par 7 comme le jeu classique, mais 5 cases par 5 cases, cela limite les possibilités de recherche.

L'heuristique que nous avons choisi fut simplement est-ce que MIN gagne (+infini) ou MAX gagne (-infini). Cette méthode simple mais efficace simplifie de beaucoup la recherche et évite que l'agent intelligent se mette lui même dans une situation risquée pour assouvir une stratégie qui sera rapidement bloquée par l'adversaire.



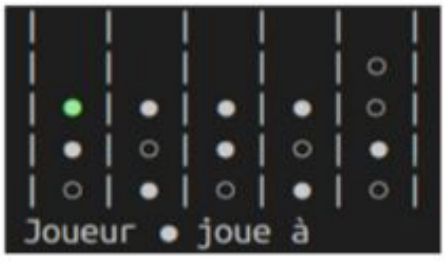
4.2 Les buts fixés sont-ils atteints

Bien que nous avions viser un agent intelligent rapide et efficace cherchant très profond dans l'arbre pour trouver une solution la plus proche possible de la meilleure, nous avons dû réviser cette version en raison des limitations de mémoires imposées par le programme. De plus notre début en programmation logique a demander un certain temps d'adaptation qui a malheureusement eu un impact sur le temps nécessaire à la production du programme.

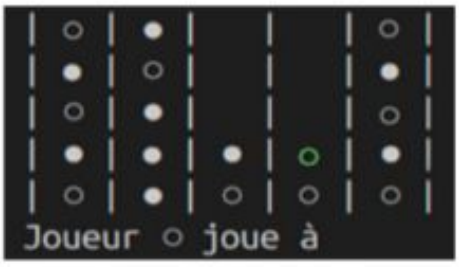
Nous avons cependant respectée la méthode choisie au départ. C'est-à-dire la recherche par Alpha-Bêta. La profondeur a considérablement été réduite par rapport à la vision de départ, mais la recherche reflète cette méthode dans ses grandes lignes.

4.3 Les limites de l'heuristique utilisée

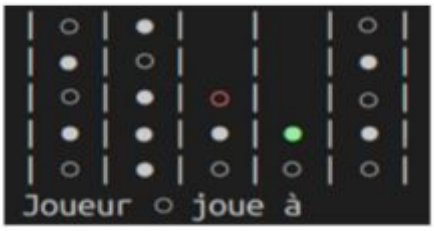
Notre agent ne recherche que quelques niveaux de profondeur et devra faire un choix dit "suffisant" faute de trouver une réponse facile et immédiate. Le niveau actuel de l'agent, bien que nous espérions plus, remplis la tâche qui lui est donnée. Elle permet au joueur ordinateur de choisir un mouvement qui le fera gagner s'il y en a un de disponible et empêcher l'adversaire de gagner s'il en a la possibilité.



Gagner au prochain tour



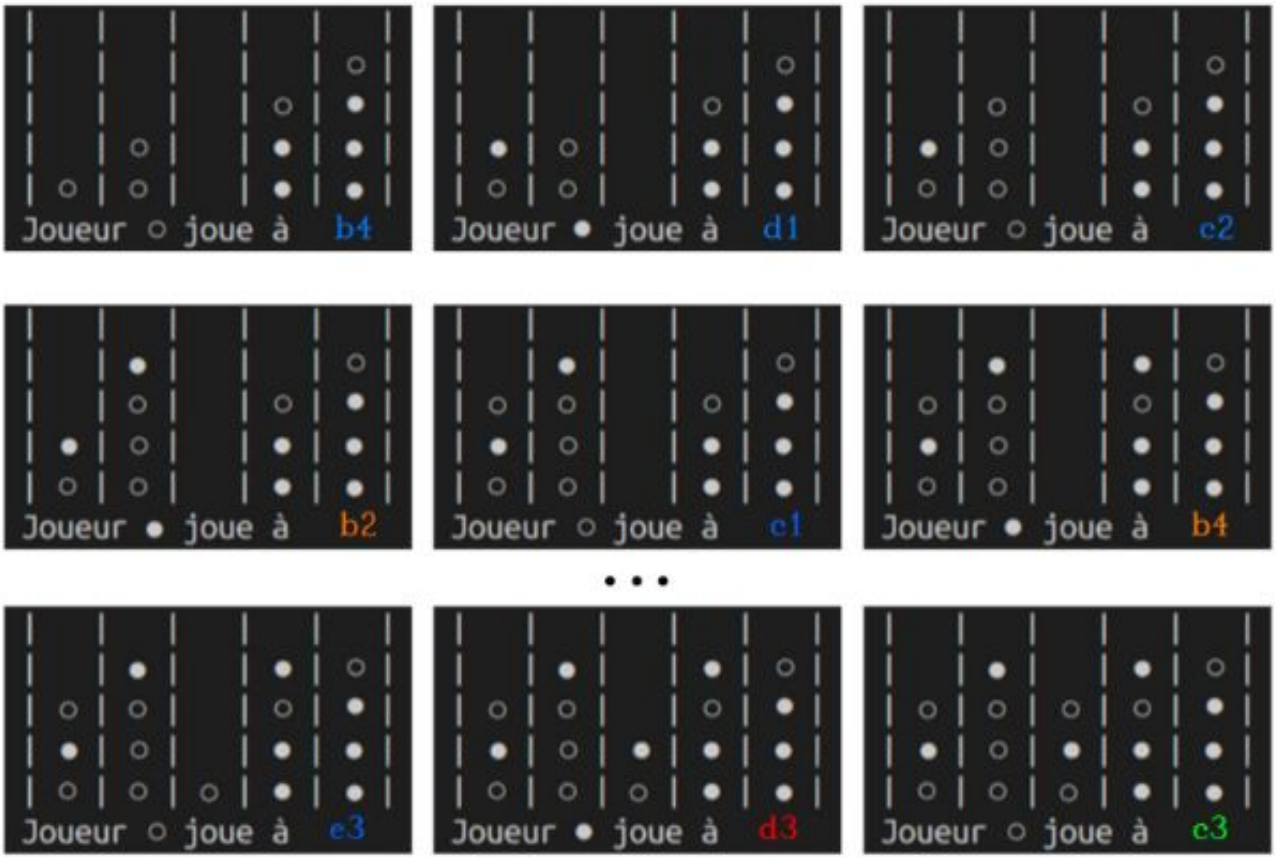
Empêche l'adversaire de gagner ensuite



Les mouvements qu'il fera ne placeront pas l'adversaire dans une position où le prochain coup qu'il fera sera gagnant.

Un joueur qui utilise un minimum de stratégie pourra facilement battre l'agent et c'est d'ailleurs un élément souvent négligé dans les jeux. Un agent trop avancé, par contre, ne laissera aucune chance au joueur. En limitant la recherche, le joueur peut mettre l'agent dans une position ou plus qu'un coup peut le faire gagner et l'agent devra faire un choix sur la combinaison gagnante qu'il bloquera.

Voyons ici une stratégie qui va gagner contre notre algorithme de recherche:



Joueur2 joue à b4	-Limite de profondeur de recherche. (Au moins Joueur1 ne gagne pas)
Joueur1 joue à d1	-Limite de profondeur de recherche. (Au moins Joueur2 ne gagne pas)
Joueur2 joue à b4	-Limite de profondeur de recherche. (Au moins Joueur1 ne gagne pas)
Joueur1 joue à b2	-Attention: e3 fait gagner Joueur2
	Choix alternatif. (Au moins Joueur2 ne gagne pas)
Joueur2 joue à c1	-Limite de profondeur de recherche. (Au moins Joueur1 ne gagne pas)
Joueur1 joue à b4	-Attention: e3 fait gagner Joueur2
	Choix alternatif. (Au moins Joueur2 ne gagne pas)

Possibilité 1:	Joueur2 joue le premier coup dans la colonne 3 (illustrée)
Joueur2 joue à e3	-Limite de profondeur de recherche. (Au moins Joueur1 ne gagne pas)
Joueur1 joue à d3	-Il n'y a plus d'autres coups possibles...
Joueur2 joue à c3	-Coup gagnant !!! (4 sur la rangée d)

Possibilité 2:	Joueur1 joue le premier coup dans colonne 3 (non illustrée)
Joueur1 joue à e3	-Il n'y a plus d'autres coups possibles...
Joueur2 joue à d3	-Coup gagnant !!! (diagonale e2 à b5)

4.3 Les améliorations commencées et à faire

Nous avons porté notre réflexion sur les axes d'amélioration de notre intelligence artificielle.

L'amélioration de l'heuristique est un élément important pour une prochaine version. Aujourd'hui, notre heuristique est capable seulement de nous retourner une réponse binaire (position gagnante, position non gagnante). Dans une nouvelle version, il faut mettre en place un système de gestion du nombre d'éléments reliés.

L'amélioration de la gestion de la **mémoire** et de la **réduction** du code source, se trouve en annexe. Cette deuxième version inspiré de https://www.metalevel.at/various/conn4/ est une meilleur approche pour gérer la mémoire et la réduction de redondance du code.

5. Bibliographie

[0] Puissance 4, Wikipédia, disponible : https://fr.wikipedia.org/wiki/Puissance_4 , consulté : 10/03/2018.

[1]. (fr) Stuart Russel et Peter Norvig, *Intelligence artificielle* 3e édition, Pearson Education France, (2010), Chap. 2 *les agents intelligents*, page 45-49.

[2] (fr) Stuart Russel et Peter Norvig, *Intelligence artificielle* 3e édition, Pearson Education France, (2010), Chap. 2 *les agents intelligents*, page 51-55.

[3] (fr) Stuart Russel et Peter Norvig, *Intelligence artificielle* 3e édition, Pearson Education France, (2010), Chap. 5 *Exploration en situation d'adversité*, page 184-185.

6. Annexe

Implémentation d'essaie

Vous pouvez consulter de manière permanente le code sur ce lien.
https://pastebin.com/ppwAdRSX