# Notes TP2

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/d5fbe2ae-a91a-4fec-89ef-59fd59549851/4A_S2_IA_Sujet_TP2.pdf

## Règles

Ils interviennent pour 1/3 de votre note dans cette UF

Vous pouvez n'envoyer qu'un seul document (formats acceptés : pdf, doc, odt) pour les 2 sujets de TP (aetoile et negamax).

Le compte-rendu doit inclure :

- les réponses aux questions posées dans chaque sujet,
- le code source (avec des commentaires pertinents si possible aux endroit les plus intéressants), éventuellement une copie des tests unitaires effectués
- les temps de réponse du programme pour différents problèmes de difficulté variée
- les limitations du programme (peut-on facilement réutiliser le code pour d'autres problèmes ?)
- les extensions réalisées ou entrevues

*n.b : s’il n’est précisé, noter que le temps d’exécution de la requête est quasi-nul*

*n.b : non confondrons les termes de tableau & matrice pour parler de la grille de jeu du morpion*

---

### Question 1 - Familiarisation avec le TicTacToe 3x3

#### 1.2

```
?- situation_initiale(S), joueur_initial(J).
S = [[_42, _48, _54], [_66, _72, _78], [_90, _96, _102]],
J = x.
% initialise le "tableau" du morpion (vide, cf. matrice 3x3 libre), et attribue à J le
"x"
?- situation_initiale(S), nth1(3,S,Lig), nth1(2,Lig,o).
S = [[_206, _212, _218], [_230, _236, _242], [_254, o, _266]],
Lig = [_254, o, _266].
% met un "o" dans le 2° élément (nth1(2,Lig,o)) de la 3° ligne de S (nth1(3,S,Lig))
% ne retourne pas de nouvelle matrice, une variable de S est juste fixée à présent
```

---

## 2.2 - Prédicat *alignement*

Trois types d'alignement possibles :

```
alignement(L, Matrix) :- ligne(L,Matrix).
alignement(C, Matrix) :- colonne(C,Matrix).
alignement(D, Matrix) :- diagonale(D,Matrix).
```

1. **Ligne**

```
ligne(L, M) :-
  nth1(_E,M,L).
% on utilise nth1 pour récupérer les éléments (_E) de la ligne L, dans la matrice M
```

2. **Colonne**

   a. on transpose la matrice du morpion et répète la fonction d'alignement des lignes (on sait déjà le faire et c'est + simple).

```
transpose([[]|_], []).
transpose(Matrix, [Row|Rows]) :- transpose_1st_col(Matrix, Row, RestMatrix),
                                 transpose(RestMatrix, Rows).
transpose_1st_col([], [], []).
transpose_1st_col([[H|T]|Rows], [H|Hs], [T|Ts]) :- transpose_1st_col(Rows, Hs, Ts).

colonne(C,M) :-
  transpose(M,Mt),
  ligne(C,Mt).
% transpose et transpose_1st_col pour transposer M, puis on fait appel à ligne
```

3. **Diagonale**

   a. "concept" opposé pour les deux diagonales *(K++ vs K--)*

```
diagonale(D, M) :- % première diagonale
  premiere_diag(1,D,M).
premiere_diag(_,[],[]).
premiere_diag(K,[E|D],[Ligne|M]) :-
  nth1(K,Ligne,E),
  K1 is K+1,
  premiere_diag(K1,D,M).


diagonale(D, M) :- % seconde diagonale
    seconde_diag(D,M).
seconde_diag(D, M) :- % on récupère la taille de la matrice (on décrémente, ici on che
rche le point de départ pour K)
  length(M,K),
  seconde_diag(K, D, M).
seconde_diag(_,[],[]).
seconde_diag(K,[E|D],[Ligne|M]) :-
  nth1(K,Ligne,E),
  K1 is K-1,
  seconde_diag(K1,D,M).

% test unitaire
% Le comportement présenté dans le sujet de TP est complet (on a bien 8 solutions, cf
 3 lignes, 3 colonnes et 2 diagonales)
?- M = [[a,b,c], [d,e,f], [g,h,i]], alignement(Ali, M).
M = [[a, b, c], [d, e, f], [g, h, i]], % ligne 1
Ali = [a, b, c] ;
M = [[a, b, c], [d, e, f], [g, h, i]], % ligne 2
Ali = [d, e, f] ;
M = [[a, b, c], [d, e, f], [g, h, i]], % ligne 3
Ali = [g, h, i] ;
M = [[a, b, c], [d, e, f], [g, h, i]], % colonne 1
Ali = [a, d, g] ;
M = [[a, b, c], [d, e, f], [g, h, i]], % colonne 2
Ali = [b, e, h] ;
M = [[a, b, c], [d, e, f], [g, h, i]], % colonne 3
Ali = [c, f, i] ;
M = [[a, b, c], [d, e, f], [g, h, i]], % première diagonale
Ali = [a, e, i] ;
M = [[a, b, c], [d, e, f], [g, h, i]], % seconde diagonale
Ali = [c, e, g] ;
false.
```

---

## Prédicat *possible*

Pour réaliser ***possible***, on définit deux cas pour ***unifiable*** :

- L'élément est libre → var(X) → unifiable
- L'élément n'est pas libre → ground(X) → ~~unifiable~~, reste à vérifier si l'élément "appartient" à J (*x* ou *o*), auquel cas l'alignement est toujours possible

```
possible([X|L],J) :- unifiable(X,J), possible(L,J).
possible([],_).
unifiable(X,_) :- var(X).
unifiable(X,J) :- ground(X), X=J.

% tests unitaires
?- A=[_,_,_], possible(A,x). % yes (ligne libre)
A = [_8, _14, _20] ;
false.

?- A=[x,_,x], possible(A,x). % yes (2 x et 1 variable libre)
A = [x, _14, x] .

?- A=[_,o,x], possible(A,x). % no (présence de o)
false.

?- A=[x,x,x], possible(A,x). % yes (alignement gagnant même)
A = [x, x, x].

?- A=[o,o,o], possible(A,x). % no (alignement de o)
false.
```

---

## Prédicats *alignement_gagnant* et *alignement_perdant*

```
% alignement_gagnant
alignement_gagnant(Ali, J) :- ground(Ali), possible(Ali,J). % l'alignement est ground
 (pas de variable libre) et possible (donc tous les éléments sont ceux de J)

% alignement_perdant
alignement_perdant(Ali, J) :- adversaire(J,J2), alignement_gagnant(Ali,J2). % aligneme
nt_perdant = alignement_gagnant de l'adversaire (cf. prédicat ci-dessus)


% tests unitaires
% alignement_gagnant (le principe est symétrique pour o)
?- A=[x,x,x], alignement_gagnant(A,x). % yes (3 x)
A = [x, x, x].

?- A=[x,x,_], alignement_gagnant(A,x). % no (1 variable libre)
false.

?- A=[o,x,o], alignement_gagnant(A,x). % no (présence de o)
false.

?- A=[o,o,o], alignement_gagnant(A,x). % no (gagnant pour o)
false.

?- A=[_,_,_], alignement_gagnant(A,_). % no (variables libres)
false.

% alignement_perdant
?- A=[o,o,o], alignement_perdant(A,x). % yes (alignement de o)
```

```
A = [o, o, o].

?- A=[o,o,o], alignement_perdant(A,o). % no (alignement_gagnant)
false.

?- A=[o,x,o], alignement_perdant(A,x). % no (présence de x)
false.

?- A=[o,_,o], alignement_perdant(A,x). % no (présence de o mais variable libre)
false.

?- A=[_,_,_], alignement_perdant(A,x). % no (variables libres)
false.
```

---

# Question 2 - Développement de l'heuristique

```
heuristique(J,Situation,H) :-   % cas 1, alignement gagnant pour J
   H = 10000,       % grand nombre approximant +infini
   alignement(Alig,Situation),
   alignement_gagnant(Alig,J), !. % cut pour ne pas rentrer dans le cas 3

heuristique(J,Situation,H) :-   % cas 2, alignement perdant pour J
   H = -10000,        % grand nombre approximant -infini
   alignement(Alig,Situation),
   alignement_perdant(Alig,J), !. % cut pour ne pas rentrer dans le cas 3

heuristique(J,Situation,H) :- % cas 3, alignement ni gagnant ni perdant
    % coups possibles pour J
    findall(Alig_g,(alignement(Alig_g,Situation),possible(Alig_g,J)),Lg),
    length(Lg,Cg),
    % coups possibles pour l'adversaire
    adversaire(J,J2),
    findall(Alig_p,(alignement(Alig_p,Situation),possible(Alig_p,J2)),Lp),
    length(Lp,Cp),
    % H = nb de coups possibles pour J - nb de coups possibles pour J2
    H is Cg-Cp.

% tests unitaires
% situation initiale, autant de coups possibles pour les deux donc H=0
?- situation_initiale(S), heuristique(J,S,H).
S = [[_56, _62, _68], [_80, _86, _92], [_104, _110, _116]],
J = x,
H = 0 ;
S = [[_56, _62, _68], [_80, _86, _92], [_104, _110, _116]],
J = o,
H = 0.

% alignement perdant
?- situation(S_perdante), J=x, heuristique(J,S_perdante,H).
S_perdante = [[o, _912, a], [o, b, _942], [o, _960, _966]],
J = x,
H = -10000.
```

```
% alignement gagnant
?- situation(S_gagnante), J=o, heuristique(J,S_gagnante,H).
S_gagnante = [[o, _920, a], [o, b, _950], [o, _968, _974]],
J = o,
H = 10000.

% aucune case libre, alignement gagnant
?- S_null=[[x,o,x],[o,x,o],[x,o,x]], J=x, heuristique(J,S_null,H).
S_null = [[x, o, x], [o, x, o], [x, o, x]],
J = x,
H = 10000.

% aucune case libre et aucun alignement gagnant / perdant
?- S_null=[[x,o,x],[o,x,o],[o,x,o]], J=x, heuristique(J,S_null,H).
S_null = [[x, o, x], [o, x, o], [o, x, o]],
J = x,
H = 0.

% J pose "x", il a plus de coups possibles que son adversaire, situation avantageuse H
>0
?- S_null=[[_,_,_],[_,x,_],[_,_,_]], J=x, heuristique(J,S_null,H).
S_null = [[_402, _408, _414], [_426, x, _438], [_450, _456, _462]],
J = x,
H = 4.

% Adversaire pose "o" au premier tour, J a moins de coups possible, H négatif (désavan
tageux)
?- S_null=[[_,_,_],[_,o,_],[_,_,_]], J=x, heuristique(J,S_null,H).
S_null = [[_402, _408, _414], [_426, o, _438], [_450, _456, _462]],
J = x,
H = -4.
```

---

# Question 3 - Développement de l'algorithme Négamax

## 3.2

"*Quel prédicat permet de connaître sous forme de liste l'ensemble des couples [Coord,Situation_Resultante] tels que chaque élément (couple) associe le coup d'un joueur et la situation qui en résulte à partir d'une situation donnée.*"

Il s'agit du prédicat *successeurs(J,Etat,Succ)*

```
?- situation_initiale(S), joueur_initial(J), successeurs(J,S,Succ).
S = [[_276, _282, _288], [_300, _306, _312], [_324, _330, _336]],
J = x,
Succ = [[[1, 1], [[x, _1322, _1328], [_1340, _1346, _1352], [_1364, _1370, _1376]]],
 [[1, 2], [[_1214, x, _1226], [_1238, _1244, _1250], [_1262, _1268|...]]], [[1, 3],
 [[_1112, _1118, x], [_1136, _1142|...], [_1160|...]]], [[2, 1], [[_1010, _1016|...],
 [x|...], [...|...]]], [[2, 2], [[_908|...], [...|...]|...]], [[2, 3], [[...|...]
```

```
|...]], [[3|...], [...|...]], [[...|...]|...], [...|...]].
% Succ = liste des coups possibles, avec la matrice résultante, depuis la situation de
départ = 9 coups possibles (toutes les cases sont libres)
```

---

## 3.3 - Développement des prédicats de négamax

- Il a d'abord fallu développer le prédicat *successeur* dans "tictactoe.pl"

```
successeur(J, Etat,[L,C]) :- nth1(L,Etat,Lig), nth1(C,Lig,J).

% tests unitaires
?- joueur_initial(J), situation_initiale(S), successeur(J,S, [2,2]). % on joue en [2,
2] comme premier coup, matrice résultante S = [[_,_,_],[_,x,_],[_,_,_]]
J = x,
S = [[_158, _164, _170], [_182, x, _194], [_206, _212, _218]].

?- joueur_initial(J), situation_initiale(S), successeur(J,S, [L,C]). % on ne précise p
as [L,C], donc accepte l'ensemble des coups possibles(9).
J = x,
S = [[x, _192, _198], [_210, _216, _222], [_234, _240, _246]],
L = C, C = 1 ;
J = x,
S = [[_186, x, _198], [_210, _216, _222], [_234, _240, _246]],
L = 1,
C = 2 ;
J = x,
S = [[_186, _192, x], [_210, _216, _222], [_234, _240, _246]],
L = 1,
C = 3 ;
J = x,
S = [[_186, _192, _198], [x, _216, _222], [_234, _240, _246]],
L = 2,
C = 1 ;
J = x,
S = [[_186, _192, _198], [_210, x, _222], [_234, _240, _246]],
L = C, C = 2 ;
J = x,
S = [[_186, _192, _198], [_210, _216, x], [_234, _240, _246]],
L = 2,
C = 3 ;
J = x,
S = [[_186, _192, _198], [_210, _216, _222], [x, _240, _246]],
L = 3,
C = 1 ;
J = x,
S = [[_186, _192, _198], [_210, _216, _222], [_234, x, _246]],
L = 3,
C = 2 ;
J = x,
S = [[_186, _192, _198], [_210, _216, _222], [_234, _240, x]],
L = C, C = 3.

?- joueur_initial(J), S=[[o,_,_],[_,_,_],[_,_,_]], successeur(J,S, [1,1]). % on ne peu
```

```
t pas écrire si la variable n'est pas libre
false.
```

---

- Ensuite, le prédicat *meilleur* pour renvoyer le coup le plus avantageux parmi tous ceux possibles (retournés par le prédicat *successeurs*, cf. développement du prédicat *successeur* précédemment

```
meilleur([X],X). % si on n'a qu'un coup possible, c'est forcément le meilleur
meilleur([[CX,VX]|Liste_Couples], Meilleur_Couple):-
  Liste_Couples\=[],
  meilleur(Liste_Couples,[CY,VY]),
  (VX<VY-> Meilleur_Couple=[CX,VX]; Meilleur_Couple=[CY,VY]). % on garde le coup dont
 l'heursitique est la plus basse
```

---

- Les trois cas possibles pour le prédicat *negamax* sont :
    - La profondeur de l'arbre est atteinte, on renvoie l'heuristique pour l'état actuel.

    ```
    negamax(J, Etat, P, P, [Coup, Val]) :-
      Coup = rien,
      heuristique(J,Etat,Val).
    ```

    - Le tableau est complet, on renvoie l'heuristique de l'état actuel. (en d'autres termes, J ne peut pas jouer)

    ```
    negamax(J, Etat, _P, _Pmax, [Coup, Val]) :-
      situation_terminale(J,Etat),
      Coup = rien,
      heuristique(J,Etat,Val).
    ```

    - On calcule les successeurs de l'état actuel et on appelle *negamax* récursivement pour chacun de ces successeurs. On renvoie le minimum des valeurs négatives renvoyées. (i.e le tableau n'est pas complet et la profondeur maximale n'est pas atteinte)

    ```
    negamax(J, Etat, P, Pmax, [Coup, (Val)]) :-
      P=<Pmax,
      successeurs(J,Etat,Succ),
      loop_negamax(J,P,Pmax,Succ,Liste_Couples),
      meilleur(Liste_Couples,[Coup,V2]),
      Val is -V2.
    ```

- Enfin, le prédicat main sera appelé et exécutera l'algorithme *negamax* en retournant C et V le coup optimal et sa valeur d'heuristique associée. Pmax peut aller de 0 à 9.

```
main(C,V, Pmax) :-
  Pmax=<9,
  joueur_initial(J), % main considéré pour le joueur x
  situation_initiale(S), % main sur un état initial (tableau vide)
  P is 0, % on commence à une profondeur nulle
  negamax(J, S, P, Pmax, [C, V]).
```

# Question 4 - Expérimentation et extensions

# 4.1 - Meilleur coup pour des profondeurs de 1 à 9

```
% tests pour différentes valeurs de pmax
?- main(C,V,1). % tps d'exécution : NaN
C = [2, 2],
V = 4 .
% valeur de V ok, en jouant au centre on peut faire 1 ligne, 1 colonne, et les 2 diago
nales = 4 alignements possibles, 0 pour l'adversaire (pas de o).

?- main(C,V,2). % tps d'excécution : NaN
C = [2, 2],
V = 1 .
% intuitivement, le meilleur coup pour J2 serait un coin, avec 2 alignements possible
s, et 3 (une diagonale de perdue) pour J, soit V=1 semble correct.

?- main(C,V,3). % tps d'exécution : NaN
C = [2, 2],
V = 3 .

?- main(C,V,4). % tps d'exécution : 0.5s
C = [2, 2],
V = 1 .

?- main(C,V,5). % tps d'exécution : 2s
C = [2, 2],
V = 3 .

?- main(C,V,6). % tps d'exécution : 11s
C = [2, 2],
V = 1 .

?- main(C,V,7). % 7 et +, out of global stack. Programme exécuté sur un ordinateur per
sonnel, il aurait fallu le tester sur un autre. Une solution pour réduire le tps / poi
ds de calcul, serait d'utiliser la symétrie du morpion (e.g, dans un tableau vide, jou
```

```
er en [1,1], [1,3], [3,1] et [3,3] (les coins) est équivalent).
ERROR: Out of global stack
```

Dans tous les cas de profondeur, le meilleur coup à jouer est en [2,2]. C'est le seul coup qui permet de gagner à coup sûr. Pour une profondeur maximale de 9, si le calcul est possible (*cf. pas Out of global stack*), le résultat devrait être [2,2] car à nouveau, c'est le seul qui permet à coup sûr de gagner ou de faire match nul.

---

# Question dans le fichier negamax.pl

```
/*
A FAIRE : commenter chaque litteral de la 2eme clause de loop_negamax/5,
  en particulier la forme du terme [_,Vsuiv] dans le dernier
  litteral ?
*/
```

- J → Joueur pour lequel on exécute négamax (le meilleur coup possible pour lui sera retourné)
- P → Profondeur actuelle (entre 1 et 9 au morpion),
- Pmax → Profondeur maximale de l'arbre (9 au morpion = nb d'éléments du tableau).
- [Coup, Suiv] → Premier élément de *Succ* (ci-dessous) = [coup joué, état du tableau en conséquence]
- Succ → liste des successeurs [Coup, Etat_Suivant], retourné par le prédicat *successeurs* depuis l'état actuel
- Reste_Couples → Reste des [Coup, heuristique associée]
- A → Adversaire de J
- Pnew → Profondeur pour l'itération suivante de negamax (P + 1)
- [_, Vsuiv] → Vsuiv = valeur de l'heuristique associée au coup joué. "_" car l'association se fait directement dans l'appel de loop_negamax ([Coup,Suiv] et [Coup,Vsuiv])

```
% Pour l'exécution de main(C,V,1), on affiche [Coup,Suiv] et Vsuiv. On a bien Vsuiv =
 heuristique associée au Coup (e.g en [2,2], Vsuiv=-4 (pour l'adversaire, 4 pour J)
[[3,3],[[_852,_858,_864],[_876,_882,_888],[_900,_906,x]]]
-3
```

```
[[3,2],[[_954,_960,_966],[_978,_984,_990],[_1002,x,_1014]]]
-2
[[3,1],[[_1056,_1062,_1068],[_1080,_1086,_1092],[x,_1110,_1116]]]
-3
[[2,3],[[_1158,_1164,_1170],[_1182,_1188,x],[_1206,_1212,_1218]]]
-2
[[2,2],[[_1260,_1266,_1272],[_1284,x,_1296],[_1308,_1314,_1320]]]
-4
[[2,1],[[_1362,_1368,_1374],[x,_1392,_1398],[_1410,_1416,_1422]]]
-2
[[1,3],[[_1464,_1470,x],[_1488,_1494,_1500],[_1512,_1518,_1524]]]
-3
[[1,2],[[_1566,x,_1578],[_1590,_1596,_1602],[_1614,_1620,_1626]]]
-2
[[1,1],[[x,_1674,_1680],[_1692,_1698,_1704],[_1716,_1722,_1728]]]
-3
```

---

## 4.2 - Symétrie

Évoqué dans le 3.3

---

## 4.3 - Puissance 4

Pour développer un algorithme pour le puissance 4, il faudrait prendre en compte un tableau de taille différente (6x7), et les règles du jeu :

- On ne peut que remplir par les colonnes (la ligne sera la première case non remplie)
- Victoire par alignement de 4 jetons en ligne, colonne ou diagonale (similaire morpion)
    - Attention, cela ne correspond pas à une ligne / colonne complète (pas comme au morpion)
    - Modification de l'heuristique

---

## 4.4 - Alpha-Beta

L'idée de l'algorithme alpha-bêta serait de reprendre l'algorithme negamax en ajoutant deux paramètres (alpha et beta)

- alpha : conserve la valeur pour le *Joueur* → max(J)
- beta : conserve la valeur pour l'*Adversaire → min(A*

Ces deux valeurs seront conservées pendant la récursion de l'arbre de recherche. Si on est en dehors de [alpha, beta], on peut *cut* la branche, c'est-à-dire ne plus l'étudier, car la solution optimale ne viendra pas de ladite branche.

Nous n'avons pas mené le développement de cet algorithme à son terme, mais une idée pour *alphaBeta* serait :

```
% pour alphabeta (on reprend negamax), on rajoute les deux paramètres et on modifie
 l'appel de loop_negamax
alphabeta(J, Etat, P, Pmax, Alpha, Beta, [Coup, (Val)]) :-
  P=<Pmax,
  successeurs(J,Etat,Succ),
  loop_negamax_alphabeta(J,P,Pmax,Alpha,Beta,Succ,Liste_Couples),
  meilleur(Liste_Couples,[Coup,V2]),
  Val is -V2.

% pour loop_negamax, l'idée serait de prendre en compte dans la recherche
loop_negamax_alphabeta(_,_,_,_,_,[],[]).
loop_negamax_alphabeta(J,P,Pmax,Alpha,Beta,[[Coup,Suiv]|Succ],[[Coup,Vsuiv]|Reste_Coup
les]) :-
    loop_negamax(J,P,Pmax,Alpha,Beta,Succ,Reste_Couples),
    adversaire(J,A),
    Pnew is P + 1,
    negamax_alpha_beta(A,Suiv,Pnew,Pmax,Alpha,Beta,[_,Vsuiv]),
    test_valeurs(Alpha,Beta,V,Coup,[[Coup,Suiv]|Succ],[[Coup,Vsuiv]|Reste_Couples]),
    cut(J,Suiv,P,Alpha,Beta,Reste_Succ,V,Acc2,Resultat).

% test_valeurs pour mettre à jour au besoin la valeur de alpha ou de beta
  % cas 1/
test_valeurs(Alpha,_Beta,Vsuiv,Coup,[[_C,V]|Succ],[[Coup,Vsuiv]|_Reste_Couples]) :-
    Vsuiv > V,
    Vsuiv < Alpha.
  % cas 2/
test_valeurs(_Alpha,_Beta,Vsuiv,Coup,[[_C,V]|Succ],[[Coup,V]|_Reste_Couples]) :-
    Vsuiv =< V.
  % cas 3/ on n'est pas dans les bornes, on cut
test_valeurs(Alpha,Beta,Vsuiv,Coup,[],[[Coup,Vsuiv]]) :-
    Vsuiv >= Alpha,
    Vsuiv =< Beta.

% cut pour couper les branches inutiles
cut(_,_,_,Alpha,Beta,_,Liste_Couples,Liste_Couples) :-
    Beta < Alpha.
cut(J,Suiv,P,Alpha,Beta,Reste_Succ,V,Accumulateur,Resultat) :-
    maj(Alpha,Beta,V,Suiv,Accumulateur,Acc2),
    Beta2 is min(Beta,V),
    loop_negamax_alpha_beta(J,P,Alpha,Beta2,Reste_Succ,Acc2,Resultat).
```

Notez que le développement complet de l'algorithme n'a pas été mené à son terme, et que les prédicats *test_valeurs* et *maj* ne sont pas définis ici.