TRAVAUX PRATIQUES D'I.A. N° 2

ALGO MINMAX - APPLICATION AU TICTACTOE

OBJECTIFS PEDAGOGIQUES

L'objectif est de maîtriser l'algorithme Negamax, être capable de l'implémenter sous une forme générique.
L'algorithme doit être réutilisable pour résoudre tout type de problème modélisable comme la recherche du meilleur coup initial dans un arbre de jeu à 2 joueurs de profondeur maximale donnée, à condition de définir dans un fichier dédié :

- l'état initial du jeu S0
- les règles permettant de définir les coups jouables pour un joueur dans une situation du jeu S donnée
- l'heuristique estimant l'avantage du joueur courant dans une situation quelconque S.

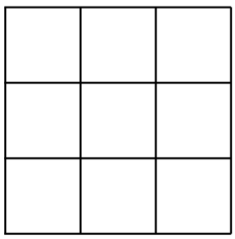
En fonction du temps restant, il pourra être envisagé :

- de résoudre le problème des symétries de jeu (situations équivalentes qu'il est inutile de développer)
- de modéliser un autre jeu (ex : puissance 4, Othello, Dames …)
- d'élaguer l'arbre de recherche en passant à la version AlphaBeta de l'algorithme

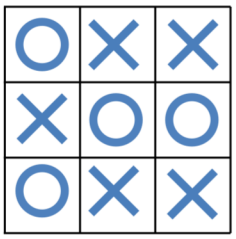
TRAVAIL A EFFECTUER

1 Familiarisation avec le problème du TicTacToe 3×3

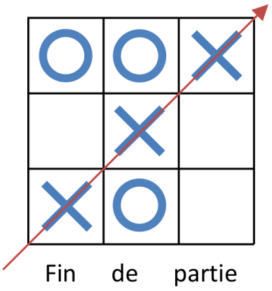
Le problème "*fil rouge*" utilisé pour la mise au point du programme *Negamax* est la recherche du coup optimal dans la situation S_0 pour une profondeur variant entre 1 et 7 coups.



Situation initiale S_0 :
le joueur × doit commencer.



Fin de partie nulle



Fin de partie gagnante pour ×

1.1 Créer un répertoire pour le tp n°2 d'I.A. et y placer le programme **`titactoe.pl`** téléchargé sur moodle (section TP N° 2).

1.2 Compiler ce programme et répondre aux questions suivantes :

Quelle interprétation donnez-vous aux requêtes suivantes :

```
?- situation_initiale(S), joueur_initial(J).

?- situation_initiale(S), nth1(3,S,Lig), nth1(2,Lig,o)
```

2.2 Lire attentivement le fichier et repérer les parties que vous devez compléter. En particulier, compléter le programme pour définir les différentes formes d'alignement retournées par le prédicat **`alignement(Ali, Matrice)`**

Comment accéder à une ligne, une colonne, une diagonale d'une matrice matrice carrée NxN ?
On doit avoir le comportement suivant (il existe 8 alignements dans une matrice 3x3) :

```
?- M = [[a,b,c], [d,e,f], [g,h,i]],   alignement(Ali, M).
Ali=[a,b,c];
Ali=[d,e,f];
Ali=[g,h,i];
Ali=[a,d,g];
Ali=[b,e,h];
Ali=[c,f,i];
Ali=[a,e,i];
Ali=[c,e,g];
no
```

Un alignement étant défini, on veut déterminer s'il est encore possible que J puisse gagner avec celui-ci : c'est vrai si chaque case est *unifiable* à J, c-à-d si elle est encore libre (utiliser le prédicat **`var`** pour le savoir), ou bien si elle est déjà unifiée à J. Attention vérifier qu'une variable X est unifiable à J, n'est pas équivalent à l'unifier (on ne peut pas se contenter d'écrire X = J) !

Définir le prédicat **`possible(Ali, Joueur)`** pour qu'il ait le comportement suivant :

```
?- A=[_,_,_], possible(A,x).
Yes
?- A=[x,_,x], possible(A,x).
Yes
?- A=[_,o,x], possible(A,x).
No
```

Définir enfin les prédicats **`alignement_gagnant(A, J)`** et **`alignement_perdant(A, J)`** qui réussissent si A est un alignement totalement instancié (utiliser le prédicat **`ground`** pour le savoir) ne contenant que des valeurs `J` (respectivement que des valeurs de l'adversaire de `J`).

Proposer des requêtes de tests unitaires pour chaque prédicat.

2. Développement de l'heuristique *h(Joueur, Situation)*

L'heuristique évalue quelle sont les chances de gagner pour un joueur donné dans une situation donnée. Elle a été définie (cf. cours) comme étant la différence entre

- le nombre d'alignements potentiellement réalisables par J dans la situation S, moins
- le nombre d'alignements potentiellement réalisables par l'adversaire de J dans la même situation S

Développer le prédicat **`heuristique(Joueur,Sit,H)`** qui retourne la valeur de l'heuristique pour le joueur J dans une situation donnée.
Proposer une requête permettant de tester votre heuristique dans la situation initiale (elle doit retourner 0 quel que soit le joueur).

Proposer d'autres tests unitaires pour vérifier qu'elle retourne bien les valeurs attendues dans le cas d'une situation gagnante pour J, perdante pour J ou nulle (toutes les cases ont été jouées sans qu'aucune joueur n'ait gagné).

2. Développement de l'algorithme Negamax

3.1 Placer dans votre répertoire du tp n°2 d'I.A. le programme **`negamax.pl`** téléchargé sur moodle (section TP N° 2).

3.2 Lire attentivement le fichier et repérer les parties que vous devez compléter.

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.

Tester ce prédicat en déterminant la liste des couples [Coup, Situation Resultante] pour le joueur X dans la situation initiale.

3.3 Compléter les définitions demandées et tester-les en proposant des tests unitaires.

4. Expérimentation et extensions

4.1 Quel est le meilleur coup à jouer et le gain espéré pour une profondeur d'analyse de 1, 2, 3, 4 , 5 , 6 , 7, 8, 9

Expliquer les résultats obtenus pour 9 (toute la grille remplie).

4.2 Comment ne pas développer inutilement des situations symétriques de situations déjà développées ?

4.3 Que faut-il reprendre pour passer au jeu du puissance 4 ?

4.4 Comment améliorer l'algorithme en élaguant certains coups inutiles (recherche Alpha-Beta) ?