

**Compte rendu IA**

**Algorithme A\* : Taquin**

**Algorithme Negamax : TicTacToe**

Ralaimady SETA

4GEI-IR

Groupe 4

**Compte rendu TP1**

1. **Familiarisation avec le problème du Taquin**

a) Clause permettant de représenter la solution finale du Taquin 4x4 :

final\_state([a,b,c,d],

[e,f,g,h],

[I,j,k,l],

[m,n,o,vide])).

b)



Récupère dans L et C respectivement le numero de la ligne et le numero de la colonne de la pièce d du jeu de taquin à la situation Initiale.



Récupère dans P la pièce située aux coordonnées (3,2) (Ligne, Colonne) du jeu de taquin à la situation finale.

c) Requête permettant de savoir si une pièce donnée (P) est bien placée dans Uo (ici l’état initial) par rapport à l’état final (Fin) :

initial\_state(Uo),

nth1(L, Uo, Ligne),

nth1(C, Ligne, P),

nth1(L, Fin, Ligne),

nth1(C, Ligne, P).

d) Requête permettant de trouver une situation suivante de l’état initial du Taquin 3x3 (3 sont possibles) :

initial\_state(Ini),

rule(R,1,Ini,Suiv).

e) Requête permettant d’avoir les 3 réponses regroupées dans une liste (ici List)

initial\_state(Ini),

findall(S, rule(A,1,Ini,S), List).

f) Requête permettant d’avoir la liste de tous les couples [A,S] tels que S est la situation qui résulte de l’action A en Uo :

initial\_state(Ini),

findall([A,S], rule(A,1,Ini,S), List).

1. **Développement des 2 heuristiques**

\*\*Voir le code source\*\*

1. **Implémentation de A\***

Analyse expérimentale

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Initial state | [b, h, c], [a, f, d], [g,vide,e] | [ a, b, c], [ g, h, d], [vide,f, e] | [b, c, d], [a,vide,g], [f, h, e] | [f, g, a], [h,vide,b], [d, c, e] | [e, f, g], [d,vide,h], [c, b, a] | [a, b, c], [g,vide,d], [h, f, e] |
| H2 | 4 | 2 | 10 | 16 | 24 | Ø |
| F\* | 5 | 2 | 10 | 20 | 30 | Ø |
| Heuristique 1 (ms) | 1 | 2 | 5 | Ø | Ø | Ø |
| Heuristique 2 (ms) | 2 | 2 | 4 | 29 | Ø | Ø |

On remarque que l’heuristique 2 est meilleur lorsqu’on a des problèmes avec une heuristique élevée, un problème donc plus complexe. Quant à l’heuristique 1, elle est plus rapide avec des situations initiales à valeur d’heuristique faible.

A mon avis, on pourra résoudre le Taquin 4x4 de même longueur car le fait de rajouter une ou deux cases ne modifiera pas forcément la capacité de la méthode de résolution modulo le temps de calcul et l’espace mémoire nécessaire qui découle de la complexité de la situation initiale comme dans le cas du taquin 3x3.

On remarque que pour la dernière situation dans le tableau on ne trouve pas de situation.

Pour le Rubik’s Cube, on peut imaginer chaque face du Rubik’s Cube comme étant un Taquin, on marquera donc chaque case de la face.

Les actions possibles ici vont être différentes, car un mouvement va apporter une modification sur 4 faces simultanément.

**Compte rendu TP2**

1. **Familiarisation avec le problème du Tictactoe 3x3**

Situation\_initiale(S), joueur\_initial(J).

Mets dans S et J respectivement la situation initiale et le joueur initial.

Situation\_initale(S), nth1(3,S,Lig), nth1(2,Lig,o)

Vérifie si le joueur o a joué dans la case (3,2).

Requêtes de tests unitaires :

Ali=[o,o,o], alignement\_perdant(Ali,x). TRUE

Ali=[x,\_,x], alignement\_perdant(Ali,x). FALSE

Ali=[\_,\_,\_], alignement\_perdant(Ali,x). FALSE

Ali=[o,o,x], alignement\_gagnant(Ali,x). FALSE

Ali=[x,\_,\_], alignement\_gagnant(Ali,x). TRUE

Ali=[\_,\_,\_], alignement\_gagnant(Ali,x). TRUE

Ali=[\_,\_,\_],possible(Ali,x). TRUE

Ali=[x,\_,\_],possible(Ali,x). TRUE

Ali=[\_,o,x],possible(Ali,x). FALSE

1. **Développement de L’heuristique**

Requête de tests unitaires

M=[[\_,\_,\_],[\_,\_,\_],[\_,\_,\_]],heuristique(x,M,H).

M=[[x,o,x],[o,x,o],[\_,o,\_]],heuristique(x,M,H).

M=[[o,x,o],[o,o,x],[x,x,\_]],heuristique(x,M,H).

M=[[x,o,x],[x,\_,o],[o,o,\_]],heuristique(x,M,H).

1. **Développement de l’algorithme Negamax**

\*\***Voir le code source\*\***

1. **Expérimentation et extensions**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Profondeur | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Coup | [2,2] | [2,2] | [2,2] | [2,2] | [2,2] | [2,2] | [2,2] | Ø | Ø |
| Valeur | 4 | 1 | 3 | 1 | 3 | 1 | 2 | Ø | Ø |

Après test, on peut remarquer que le meilleur coup à jouer est de jouer au centre, c’est-à-dire le coup [2,2] qui nous donne V=4.

Pour une profondeur d’analyse égale à 9, on a soit match nul soit une situation finale où un des joueurs est gagnant.

Pour éviter de développer inutilement des situations symétriques de situations déjà développées, on peut envisager d’effectuer des rotations pour savoir si l’état que nous souhaitons tester n’est pas un état symétrique à un état déjà traité auparavant.

Pour améliorer l’algorithme, on peut mettre en place l’élagage Alpha-Beta, vu en cours. L’élagage alpha-Beta se base sur la comparaison de son meilleur coup avec le meilleur coup des branches déjà explorées.