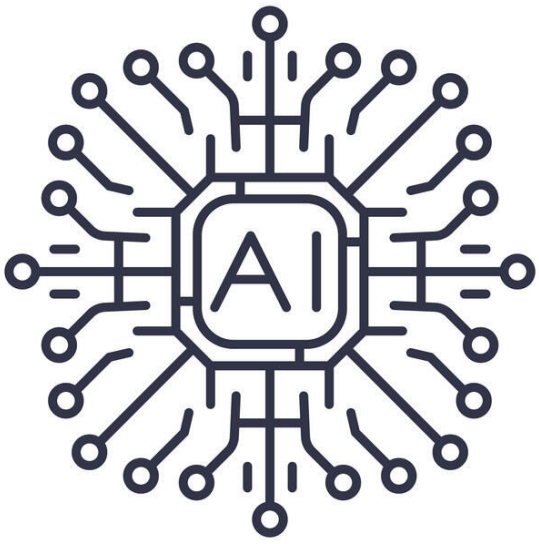
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport Intelligence Artificielle

Dutems Alan | Zolver Benjamin
4ème année IR-SI
Année universitaire 2023/2024



Contents

1 TP 1 - A* Taquin **1**
1.1 Présentation . . . 1
1.2 Réponses aux questions . . . 1
1.3 Remarques . . . 2
1.3.1 Tests unitaires . . . 2
1.3.2 Limite de notre programme . . . 2

2 TP 2 - Negamax TicTacToe **3**
2.1 Présentation . . . 3
2.2 Réponses aux questions . . . 3
2.3 Remaques . . . 4
2.3.1 Tests unitaires . . . 4
2.3.2 Temps de réponses . . . 4

3 GitHub **4**

1 TP 1 - A* Taquin

1.1 Présentation

Ce TP a pour objectif d'évaluer la meilleur serie de coup à jouer sur un plateau de taquin 3x3. Pour cela on utilisera l'algorithme A*. Nous nous sommes permis de changer légèrement la fonction A*, pour passer un paramètre supplémentaire qui donne la taille du taquin. En plus de cette modification nous avons ajouté un prédicat `?- size_tab_2d(T,N).` qui permet de calculer le nombre d'élements dans un tableau à 2 dimensions. Grâce à ces deux ajouts, nous avons maintenant un code générique qui permet de résoudre un taquin de n'importe quelle taille.

1.2 Réponses aux questions

1. **Quelle clause Prolog permettrait de représenter la situation finale du Taquin 4x4 ?**
   La clause prolog qui permet de représenter la sitation finale du Taquin 4x4 est:
   `?- finalstate([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,vide]])`

2. **A quelles questions permettent de répondre les requêtes suivantes:**
   - **?- initial_state(Ini), nth1(L,Ini,Ligne), nth1(C,Ligne, d).**
     Cette requête donne la position de la lettre d dans le taquin initial.
   - **?- final_state(Fin), nth1(3,Fin,Ligne), nth1(2,Ligne,P).**
     Cette requête donne la pièce P à la place (3,2) dans le taquin final.

3. **Quelle requête Prolog permettrait de savoir si une pièce donnée P (ex : a) est bien placée dans U0 (par rapport à F) ?**
   `?- initial_state(Ini), nth1(L, Ini, Ligne), nth1(C,Ligne,P), final_state(Final), nth1(L,F,Ligne), nth1(C,Ligne,P).`

4. **Quelle requête permet de trouver une situation suivante de l'état initial du Taquin 3×3 (3 sont possibles) ?**
   `?- initial_state(Ini),rule(R,1,Ini,S2).`

5. **Quelle requête permet d'avoir ces 3 réponses regroupées dans une liste ? (cf. findall/3 en Annexe).**
   `?- initial_state(Ini), findall(R, rule(R,1,Ini,_),List).`

6. **Quelle requête permet d'avoir la liste de tous les couples [A, S] tels que S est la situation qui résulte de l'action A en U0 ?**
   `?- initial_state(Ini), findall([R,S], rule(R,1,Ini,S),List).` pour tous les avoir dans une liste.
   `?- initial_state(Ini), rule(R,1,Ini,S).` pour les avoir les couples un par un (on peut ajouter Couple = [R,S] pour avoir le résultat sous forme d'un tableau).

7. **Noter le temps de calcul de A* et l'influence du choix de l'heuristique : quelle taille de séquences optimales (entre 2 et 30 actions) peut-on générer avec chaque heuristique (H1, H2) ? Présenter les résultats sous forme de tableau.**
   Ce tableau présente la différence de temps de calcul pour A* en utilisant les deux prédicats différents pour le calcul de l'heuristique. Le test se fait sur un ensemble de taquin de difficulté différente.

heuristique1	heuristique2
1s 66ms	0s 42ms

Table 1: Tableau de comparaison

   On remarque que le test pour : `S4 = [[e, f, g], [d,vide,h], [c, b, a]]` ne fonctionne pas avec le calcul de l'heuristique1 mais fonctionne avec le calcul de l'heuristique2.

8. **Quelle longueur de séquence peut-on envisager de résoudre pour le Taquin 4x4 ?**
   Si on considère un taquin 4x4 avec toutes les pièces mal placées, on peut au mieux le résoudre en 15 coups. Mais en moyenne il faut au minimum 52 coups [1] pour le résoudre à partir d'une situation initial résolvable.

9. **A* trouve-t-il la solution pour la situation initiale suivante ?**
   Non il est impossible pour A* de trouver la solution pour cette situation initiale car cette situtation n'est pas atteignable depuis la situation finale.

10. **Quelle représentation de l'état du Rubik's Cube et quel type d'action proposeriez-vous si vous vouliez appliquer A*?**
    On ferait quelque chose de similaire avec des listes de listes et une solution finale. On établirait le calcul de l'heuristique avec la distance de Manhattan en 3 dimentions. On pourrait également définir des prédicats qui déplace les lignes, les colonnes et les faces. Cependant on aurrait pas une case vide à déplacer mais uniquement des faces à tourner avec une couleur centrale qui reste toujours identique.

1.3 Remarques

1.3.1 Tests unitaires

L'ensemble des tests unitaires sont développés dans notre code en fin du fichier solveur.pl et après chaque fonction dans taquin.pl.

1.3.2 Limite de notre programme

Nous avons remarqué qu'avec la plupart des situations initiales 4x4 "bien mélangées", notre algorithme A* n'est pas capable de le résoudre. Une erreur du type `Stack limit` est relevée.

[1] https://mpechaud.fr/scripts/parcours/testtaquin.html

2 TP 2 - Negamax TicTacToe

2.1 Présentation

Ce TP a pour objectif d'évaluer le meilleur coup à jouer joueur par joueur dans une partie de TicTacToe à l'aide de l'algoritme Negamax.

2.2 Réponses aux questions

1. **Quelle interprétation donnez-vous aux requêtes suivantes :**

   - **?- situation_initiale(S), joueur_initial(J).**
     Cette requête donne la situation initial ainsi que le nom du premier joueur.
   - **?- situation_initiale(S), nth1(3,S,Lig), nth1(2,Lig,o).**
     Cette requête permet d'attribuer à une case du jeu en l'occurence la case (3,2) la valeur o. Contrairement au premier TP où les valeurs étaient fixées par état et donc on ne pouvait que tester la présence ou non d'un élément, dans ce cas, on peut fixer une valeur puisque les cases sont "disponibles".

2. **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 ?**
   Il semblerait qu'à partir de la profondeur 8, il y ait un revirement de situation. En effet, de 1 à 7 l'algo nous invite à jouer au centre ce qui semble en effet le plus intuitif. Néanmoins, à partir de la profondeur 8, l'algo change d'avis, et nous dit de jouer dans un coin.

3. **Comment ne pas développer inutilement des situations symétriques de situations déjà développées?**
   Afin de ne pas developper inutilement des situations symétriques déjà développées, on conserve en mémoire chaque situation déjà parcourue et son évaluation. De fait, en cas de rencontre d'une situation similaire, on récupère la valeur précédemment stockée ce qui évite de refaire les calculs et permet d'obtenir un algorithme d'exploration plus efficace.

4. **Que faut-il reprendre pour passer au jeu du puissance 4?**
   Pour transformer ce jeu en jeu de puissance 4 nous pouvons garder l'algorithme de recherche du meilleur coup car il faut changer de joueur après chaque coup. Cependant il faut redefinir la taille de la matrice par une matrice de 6 lignes et de 7 colonnes. Il faut également redéfinir les différentes situations de victoire.

5. **Comment améliorer l'algorithme en élaguant certains coups inutiles (recherche Alpha-Beta)?**
   Une façon de le faire serait simplement d'ajouter des arguments alpha et beta qui seraient verifiés pendant l'élagage dans loop-negamax et qui évitent ainsi de parcourir inutilement certaines branches en ayant la borne supérieure et inférieure pour chaque noeud.

2.3 Remaques

2.3.1 Tests unitaires

L'ensemble des tests unitaires sont développés dans notre code en fin du fichier negamax.pl et tictactoe.pl.

2.3.2 Temps de réponses

Ce tableau présente les temps de réponses en fonction de la profondeur d'analyse choisit pour un tictactoe initialement vide de notre algo Negamax :

Profondeur	Temps de réponse
1	0s 10ms
2	0s 15ms
3	0s 18ms
4	0s 40ms
5	1s 50ms
6	5s 40ms
7	14s 80ms
8	28s 10ms
9	44s 60ms

Table 2: Temps de réponse en fonction de la profondeur

3 GitHub

Lien vers notre dépot GitHub avec l'ensemble de notre code : `https://github.com/Ascyam/TP_IA`