

**Compte rendu de TP**

**Intelligence Artificielle**

15 mars 2021

HOK Jean-Rémy

INSA\* Promotion 55, 4ème Année IR\*

135, Avenue de Rangueil

31077 Toulouse Cedex 4

\*INSA : Institut National des Sciences Appliquées

\*IR : Informatique et Réseaux

**Compte rendu de TP**

**Intelligence Artificielle**

15 mars 2021

HOK Jean-Rémy

INSA\* Promotion 55, 4ème Année IR\*

**INSA Toulouse**

135, Avenue de Rangueil

31077 Toulouse Cedex 4

\*INSA : Institut National des Sciences Appliquées

\*IR : Informatique et Réseaux

Table des matières

[Introduction 1](#_Toc66729596)

[TP1 – Algorithme A\* – Application au Taquin 2](#_Toc66729597)

[Familiarisation avec le problème du Taquin 3x3 2](#_Toc66729598)

[Développement des 2 heuristiques 3](#_Toc66729599)

[Heuristique 1 : nombre de pièces mal placées 3](#_Toc66729600)

[Heuristique 2 : distance de Manhattan 4](#_Toc66729601)

[Implémentation de A\* 5](#_Toc66729602)

[Analyse Expérimentale 9](#_Toc66729603)

[Résultat de l’algorithme, taquin 3x3 9](#_Toc66729604)

[Temps de calcul de 1\* et influence du choix de l’heuristique 9](#_Toc66729605)

[TP2 – Algo minmax – Application au TicTacToe 10](#_Toc66729606)

# Introduction

L’objectif des deux TPs a été de mettre en pratique nos connaissances sur les algorithmes de résolution de problèmes basés sur la recherche arborescente avec heuristique, à savoir :

* TP1 : Algorithme A\* appliqué au Taquin

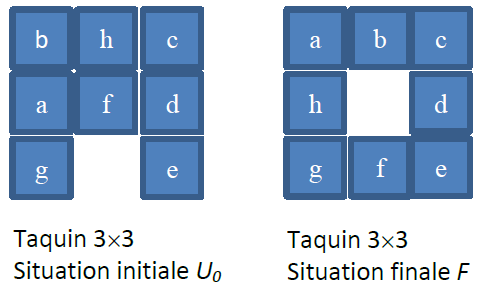


Figure 1 : Contexte Taquin

* TP2 : Algorithme MinMax / Alpha-Beta appliqué au TicTacToe

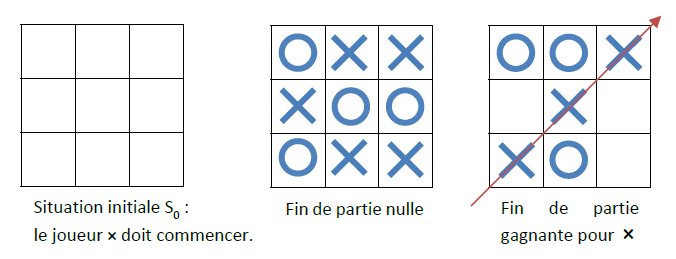


Figure 2 : Contexte TicTacToe

# TP1 – Algorithme A\* – Application au Taquin

## Familiarisation avec le problème du Taquin 3x3

#### Quelle clause Prolog permettrait de représenter la situation finale du Taquin 4x4 ?

* final\_state([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,vide]]).

#### A quelles questions permettent de répondre les requêtes suivantes :

* initial\_state(Ini), nth1(L,Ini,Ligne), nth1(C,Ligne, d).

On récupère la ligne L et la colonne C de la pièce « d » dans l’état initial : à savoir ligne 2 colonne 3.

* final\_state(Fin), nth1(3,Fin,Ligne), nth1(2,Ligne,P).

On récupère la pièce P en position (3,2), ligne 3 colonne 2, dans l’état final : à savoir « f »

#### Quelle requête Prolog permettrait de savoir si une pièce donnée P (ex : a) est bien placée dans U0 (par rapport à F) ?

U0 : état initial et F : état final

initial\_state(Init), nth1(L,Init,Ligne), nth1(C,Ligne, a), final\_state(Fin), nth1(L,Fin,Ligne2), nth1(C,Ligne2,a).

Qui retourne *false* dans notre cas.

#### Quelle requête permet de trouver une situation suivante de l'état initial du Taquin 3×3 (3 sont possibles) ?

initial\_state(Init), rule(\_, \_, Init, Suiv).

#### Quelle requête permet d'avoir ces 3 réponses regroupées dans une liste ?

initial\_state(Init), findall(Suiv, rule(\_,\_,Init,Suiv),Suiv\_Liste).

#### 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(Init),

findall([Action,Suiv], rule(Action,\_,Init,Suiv),Suiv\_Liste).

## Développement des 2 heuristiques

### Heuristique 1 : nombre de pièces mal placées

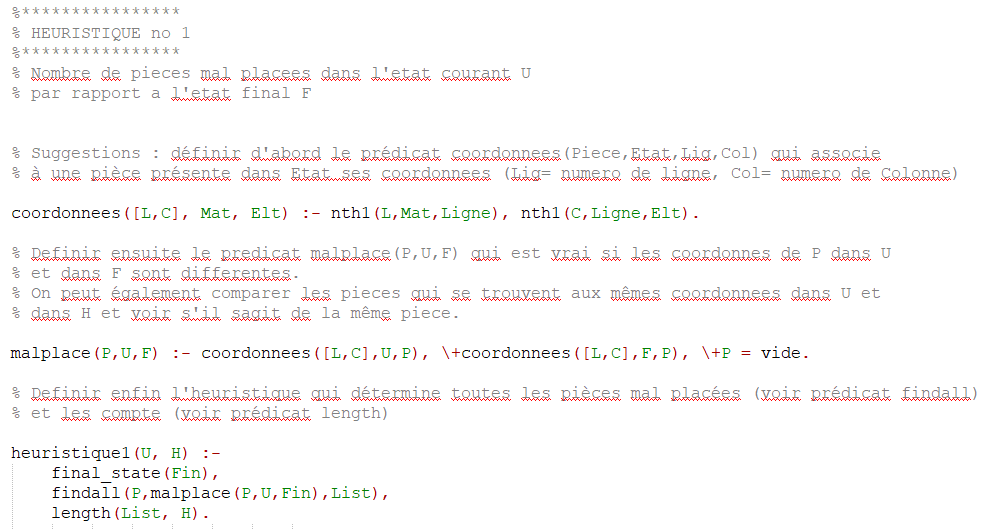


Figure 3 : Heuristique 1

|  |  |
| --- | --- |
| Figure 4 : Enoncé heuristique 1 | Nous trouvons bien 4 lorsque nous appliquons l’heuristique 1 sur l’état initial. |

### Heuristique 2 : distance de Manhattan

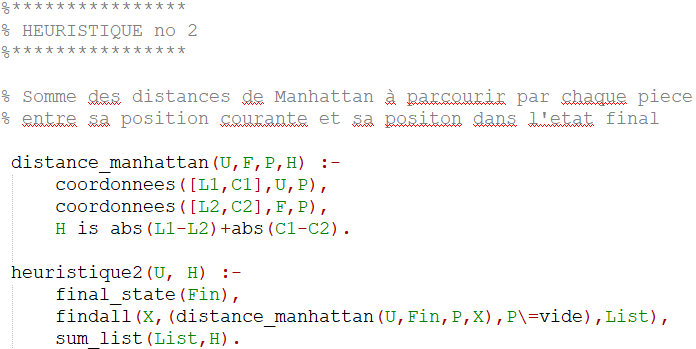


Figure 5 : Heuristique 2

|  |  |
| --- | --- |
|  | La requête suivante :  distance\_manhattan([  [d,vide,vide],  [vide,vide,vide],  [vide,vide,vide]],  [[vide,vide,vide],  [vide,vide,d],  [vide,vide,vide]],d,H).  Nous donne bien H=3. (Nous plaçons arbitrairement « vide » dans les cases restantes pour que ce ne soit pas des « d ») |

## Implémentation de A\*

#### Prédicat main/0

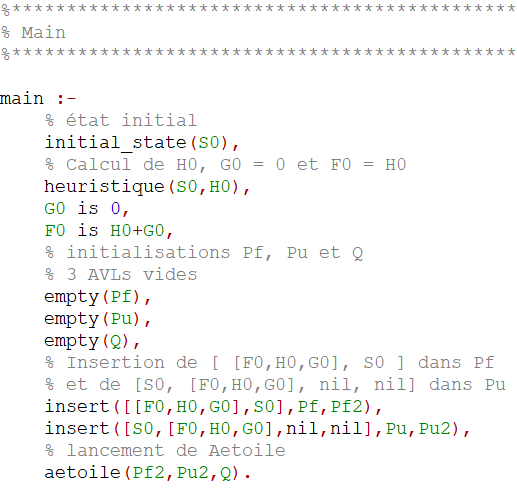


Figure 6 : Prédicat main/0

#### Prédicat aetoile/3

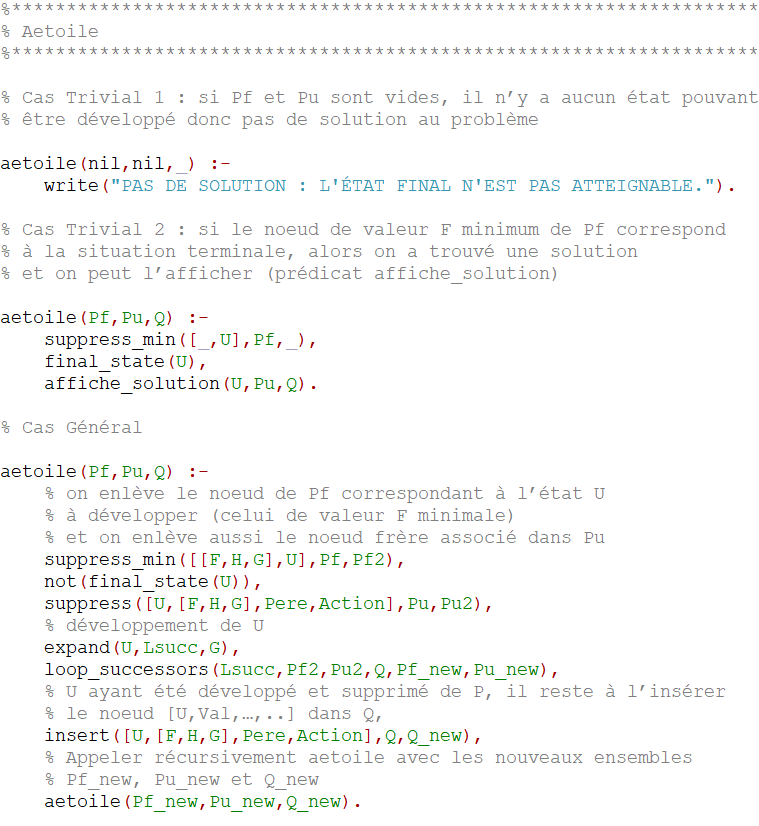


Figure 7 : Prédicat aetoile/3

#### Prédicat affiche\_solution/3

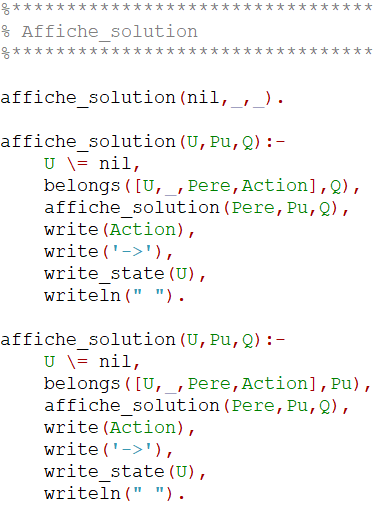


Figure 8 : Prédicat affiche\_solution/3

#### Prédicat expand/3

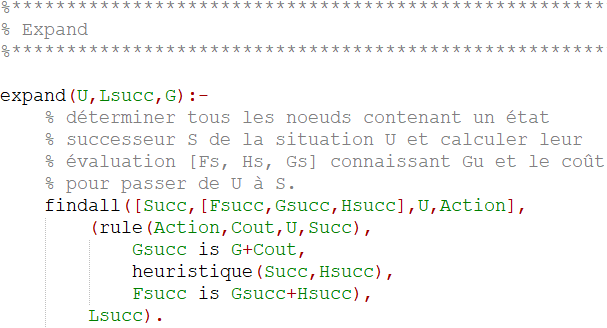


Figure 9 : Prédicat expand/3

#### Prédicat loop\_successors/6

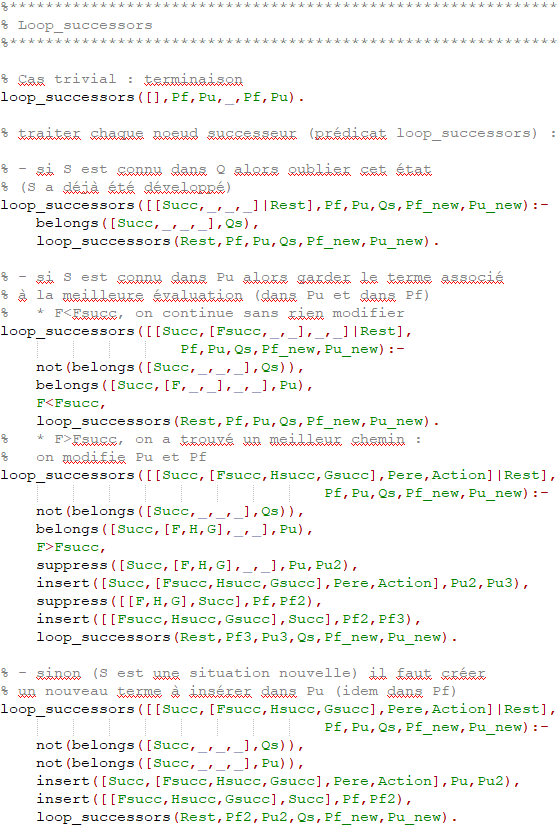


Figure 10 : Prédicat loop\_successors/6

## Analyse Expérimentale

### Résultat de l’algorithme, taquin 3x3

|  |  |
| --- | --- |
| Figure 11 : Résultats A\* | Le programme retourne le résultat ci-contre avec pour état initial U0 et état final F définis dans [l’introduction](#_Introduction).    Figure 12 : A\* autre situation initiale  Dans la situation ci-dessus, l’algorithme A\* retourne « false ». En effet, il n’existe pas de solution à partir de cet état (non connexe à l’état final). |

### Temps de calcul de A\* et influence du choix de l’heuristique

|  |  |  |
| --- | --- | --- |
| Heuristique | Taille de séquences optimales | Temps de calcul (ms) |
| N°1 | 6 | 68 |
| N°2 | 6 | 52 |

### Taquin 4x4 et Rubik’s Cube

Nous pouvons envisager une longueur de séquence de ? actions pour résoudre le Taquin 4x4. Pour le Rubik’s Cube, nous aurions une liste avec les 6 faces du cubes (chacune représentée comme dans le Taquin 3x3) et 12 actions possibles : 2 sens de rotation pour chaque colonne et 2 sens pour chaque ligne.

# TP2 – Algo minmax – Application au TicTacToe

