

**NOM DU DOSSIER**

**Sous titre**

Date

Service

**RAPPORT DE PROJET**

**Antoine ALAVERDOV, Clémence LEMEILLEUR**

**Promo 56, Année 2021/2022 – 4IR-SI-B1**

*« TP IA»*

Mars 2022

Encadrant : A.BIT-MONNOT

**RAPPORT DE PROJET**

Antoine ALAVERDOV, Clémence LEMEILLEUR

Promo 56, Année 2021/2022 – 4IR-SI-B1

*“TP IA*”

Mars 2022

Encadrant: A.BIT-MONNOT

SOMMAIRE :

1. **Déroulé des TP et réponses aux questions1**
2. **Code source et tests unitaires1**
3. **Temps de réponse du programme pour différents problèmes de difficulté variée1**
4. **Limitations du programme, adaptabilité à d’autres problèmes1**
5. **Extensions réalisées ou entrevues1**

**Table des illustrations13**

**Table des annexes14**

**I-** **Déroulé des TP et réponses aux questions**

ALGORITHME A\* - APPLICATION AU TAQUIN

**1 Familiarisation avec le problème du Taquin 3**×**3**

**1.2 a) Quelle clause Prolog permettrait de représenter la situation finale du Taquin 4x4 ?**

etat\_final\_4x4([ [1, 2, 3, 4],

[5, 6, 7, 8],

[9, 10, 11, 12],

[13, 14, 15, vide] ]).

**b) A quelles questions permettent de répondre les requêtes suivantes :**

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

Récupérer la ligne et la colonne respectivement dans L et C de d dans l'état initial

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

Récupérer la pièce qui se trouve à la 3eme ligne et la 2eme colonne à l'état final

**c) 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,d), final\_state(Fin), nth1(L,Fin,LigneF), nth1(C,LigneF,d).

**On s’intéresse maintenant au prédicat rule/4. Comment l’utiliser pour répondre aux questions suivantes :  
d) quelle requête permet de trouver une situation suivante de l’état initial du Taquin 3×3 (3 sont possibles) ?**

initial\_state(Ini), rule(\_,\_,Ini,Next).

**e) quelle requête permet d'avoir ces 3 réponses regroupées dans une liste ? (*cf*. findall/3 en Annexe).**

findall(Next, (initial\_state(Ini), rule(\_,\_,Ini,Next)), R).

**f) 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 ?**

findall([A, Next], (initial\_state(Ini), rule(A,\_,Ini,Next)), R).

**2. Développement des 2 heuristiques**

2.1 L’heuristique du *nombre de pièces mal placées*.

On a choisi la première méthode qui consiste à rechercher la liste de toutes les *pièces mal placées* (cf. prédicat **findall**) et calculer sa taille (cf. prédicat **length**). Une pièce (attention ‘**vide**’ n’en est pas une) est *mal placée* en U si le terme de mêmes cordonnées dans l’état final F est un terme diffèrent. Le prédicat liant un terme à ses coordonnées dans une matrice doit donc être définie.

malplace(P, U, Final):-

nth1(L,U,Ligne), nth1(C,Ligne,P), nth1(L,Final,LigneF),

nth1(C,LigneF,M), P\=M, P\=vide.

heuristique1(U, H) :-

final\_state(Final), findall(P, malplace(P, U,Final), Res),

length(Res, H).

2.2 L’heuristique basée sur la *distance de Manhattan*.

**Coder les prédicats logiques heuristique1(U,H) et heuristique2(U,H) :**

distManhat(P, U, Final, Dist) :-

coordonnees([Lu, Cu], U, P),

coordonnees([Lf, Cf], Final, P), P\=vide,

Dist is (abs(Lu-Lf)+abs(Cu-Cf)).

heuristique2(U, H) :-

final\_state(Final),

findall( Dist, distManhat(\_, U, Final, Dist), Res),

sumlist(Res,H).

Nous avons ensuite vérifié nos prédicats en les testant sur les deux situations extrêmes (*U0* et *F*).

**3. Implémentation de A\***

**3.1 Implémentation de P et Q par des arbres AVL**

Nous avons lu et compris les consignes de cette partie afin de pouvoir aborder la partie suivante.

**3.2 Algorithme A\* adapté aux structures AVL choisies**

**Développer ensuite chaque prédicat (affiche\_solution, expand, loop\_successors), spécifier des tests unitaires et vérifier le bon fonctionnement de chacun avant de tenter le test d'intégration final.**

Voici le développement des différents prédicats de A\* :

loop\_successors([], \_, \_, \_, \_, \_).

loop\_successors([[S]|Ls], Pf, Pu, Q, Newpf, Newpu) :-

/\*si S est dans Q, alors on oublie cet etat\*/

(S=[U, \_, \_, \_], belongs(U, Q), loop\_successors(Ls, Pf, Pu, Q, Newpf, Newpu)

);(

/\*si S est dans Pu, on garde la meilleure evaluation\*/

S=[U, [Fs, \_, \_], \_, \_], belongs([U,[Fpu, Hpu, Gpu],Perepu, Apu], Pu),

(Fs =< Fpu -> (

suppress([U,[Fpu, Hpu, Gpu],Perepu, Apu], Pu, Pu2),

suppress([U,[Fpu, Hpu, Gpu],Perepu, Apu], Pf, Pf2),

insert(S, Pu2, Pu3),

insert(S, Pf2, Pf3),

loop\_successors(Ls, Pf3, Pu3, Q, Newpf, Newpu)

);(

loop\_successors(Ls, Pf, Pu, Q, Newpf, Newpu)

)

)

);(

/\*S est une nouvelle situation, on l'insere dans Pu et Pf\*/

S =[\_,[F,G,H], Pere, Action],

insert(S, Pu, Pu2),

insert([[F,G,H], Pere, Action], Pf, Pf2),

loop\_successors(Ls, Pf2, Pu2, Q), Newpf, Newpu).

expand(U, G, Successeurs):-

findall( [S, [F, H, Ga], U, A],

(rule(A, \_, U, S), heuristique(S,H),Ga is G+1,F is G+H),

Successeurs).

affiche\_solution(\_, nil) :- write("Finito\n").

affiche\_solution(Q, U) :-

belongs([U, \_, Pere, \_], Q), suppress([U, \_, Pere, A], Q, NewQ),

write(A), write(" ; "),

write(U), write(\n),

affiche\_solution(NewQ, Pere).

aetoile(Pf, Pu, \_) :-

empty(Pf), empty(Pu),

print("PAS de SOLUTION : L’ETAT FINAL N’EST PAS ATTEIGNABLE !").

aetoile(Pf, \_, Q) :-

suppress\_min([[\_, \_, \_], Fmin], Pf, \_), final\_state(Fmin),

affiche\_solution(Q, Fmin).

aetoile(Pf, Pu, Q) :-

suppress\_min([[F,H,G], Umin], Pf, Pf2),

suppress([Umin, [F,H,G], Pere1, A], Pu, Pu2),

expand(Umin, G, Succ), loop\_successors(Succ, Pf2, Pu2, Q, Newpf, Newpu),

insert([Umin,[F,H,G],Pere1,A], Q, NewQ),

aetoile(Newpf, Newpu, NewQ).

Voici les tests unitaires et le test d’intégration final :

initial\_state1([ [ a, b, c],

[ g, h, d],

[vide,f, e] ]). % h2=2, f\*=2

initial\_state2([ [b, c, d],

[a,vide,g],

[f, h, e] ]). % h2=10 f\*=10

initial\_state3([ [f, g, a],

[h,vide,b],

[d, c, e] ]). % h2=16, f\*=20

initial\_state4([ [e, f, g],

[d,vide,h],

[c, b, a] ]). % h2=24, f\*=30

initial\_state5([ [a, b, c],

[g,vide,d],

[h, f, e]]). % etat non connexe avec l etat final (PAS DE SOLUTION)

**3.3 Analyse expérimentale**

**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.**

Nous avons utilisé la commande suivante afin d’obtenir le temps de calcul en fonction de l’heuristique :

time(main). %en ms

main :- initial\_state2(S0), heuristique2(S0, H0), G0 is 0, F0 is H0+G0,

empty(Pf), empty(Pu), empty(Q),

insert( [[F0,H0,G0], S0], Pf, Pf),

insert( [S0, [F0,H0,G0], nil, nil], Pu, Pu),

aetoile(Pf, Pu, Q).

Cela nous permet d’avoir le temps d’exécution de la commande main en changeant les paramètres de séquence et d’heuristique dans ce dernier.

Les résultats obtenus sont résumés sous la forme d’un tableau.

|  |  |  |
| --- | --- | --- |
| **Temps de calcul pour une séquence** | **Heuristique 1** | **Heuristique 2** |
| Initial\_state1 |  |  |
| Initial\_state2 |  | 0.010 seconds |
| Initial\_state3 |  | 0.011 seconds |
| Initial\_state4 |  |  |
| Initial\_state5 | \_\_\_\_\_ | \_\_\_\_\_\_\_\_\_ |

On remarque donc que la séquence optimale pour heuristique 1 est de … actions et qu’elle est de … pour heuristique 2.

A FAIRE

**Quelle longueur de séquence peut-on envisager de résoudre pour le Taquin 4x4 ?**

La distance de Manhattan sera de maximum 6 pour un taquin 4\*4. Le nombre de pièces sera de 15. Nous aurons donc une séquence de 6\*15=90.

**A\* trouve-t-il la solution pour la situation initiale suivante ?**

Une image contenant texte, équipement électronique, clavier

Description générée automatiquementCette situation correspond à initial\_state5.

L’algorithme A\* ne trouve pas la solution pour cette situation initiale. En effet nous obtenons le message d’erreur suivant : OUT\_OF\_LOCAL\_STACK.

**Quelle représentation de l’état du Rubik’s Cube et quel type d’action proposeriez-vous si vous vouliez appliquer A\*?**

Un Rubik’s Cube est un jeu de taquin en 3D en quelques sortes avec 3\*3 carrés de couleurs et on ne peut modifier qu’une ligne ou une colonne à la fois en la « tournant ». Il est composé de 6 faces de chacune 1 couleur différente.

ALGO MINMAX - APPLICATION AU TICTACTOE

**1 Familiarisation avec le problème du TicTacToe 3**×**3**

**1.2 Compiler ce programme et répondre aux questions suivantes :  
Quelle interprétation donnez-vous aux requêtes suivantes :**

**?- situation\_initiale(S), joueur\_initial(J).**

Les numéros correspondent aux cases non instanciées et J est le joueur à qui c'est le tour

**?- situation\_initiale(S), nth1(3,S,Lig), nth1(2,Lig,o)**

Met un 'o' a la deuxième case de la 3eme ligne.

**2.2 Comment accéder à une ligne, une colonne, une diagonale d’une matrice carrée NxN ?**

Voici comment accéder à une ligne, colonne, diagonale, d’une matrice carrée NXN M :

ligne(L, M) :- nth1(\_,M,L).

colonne(C,M) :- maplist(nth1(\_), M, C).

diagonale(D, M) :-

premiere\_diag(1,D,M).

diagonale(D, M) :-

seconde\_diag(3,D,M).

**Compléter le programme pour définir les différentes formes d’alignement retournées par le prédicat alignement(Ali, Matrice) ?**

alignement(L, Matrix) :- ligne(L,Matrix).

alignement(C, Matrix) :- colonne(C,Matrix).

alignement(D, Matrix) :- diagonale(D,Matrix).

**Définir le prédicat possible(Ali, Joueur)**

possible( [], \_).

possible([X|L], J) :- unifiable(X,J), possible(L,J).

unifiable(X,J) :- var(X); X=J.

**Définir 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).**

alignement\_gagnant(Ali, J) :- possible(Ali, J), ground(Ali).

alignement\_perdant(Ali, J) :- J=x, possible(Ali, o), ground(Ali);

J=o, possible(Ali, x), ground(Ali).

**Proposer des requêtes de tests unitaires pour chaque prédicat.**

A FAIRE

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

**Développer le prédicat heuristique(Joueur,Sit,H) qui retourne la valeur de l’heuristique pour le joueur J dans une situation donnée.**

heuristique(J,Situation,H) :-

H = 10000,

alignement(Alig,Situation),

alignement\_gagnant(Alig,J), !.

heuristique(J,Situation,H) :- % cas 2

H = -10000, % grand nombre approximant -infini

alignement(Alig,Situation),

alignement\_perdant(Alig,J), !.

% cas 3

heuristique(J,Situation,H) :-

findall(AliJoueurJ, (alignement(AliJoueurJ, Situation), possible(AliJoueurJ, J)), ListeAliJoueurJ),

length(ListeAliJoueurJ, NBJJ), %nb de cas gagnant pour le joueur X

adversaire(J, A),

findall(AliJoueurO, (alignement(AliJoueurO, Situation), possible(AliJoueurO, A)), ListeAliJoueurO),

length(ListeAliJoueurO, NBJO), %nb de cas gagnant pour le joueur O

H is NBJJ - NBJO.

**Proposer une requête permettant de tester votre heuristique dans la situation initiale (elle doit retourner 0 quel que soit le joueur).**

A FAIRE

**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’aucun joueur n’ait gagné).**

test\_heur\_gagnante(H) :- test\_gagnant\_J(S), joueur\_initial(J), heuristique(J,S,H).

test\_heur\_perdante(H) :- test\_perdant\_J(S), joueur\_initial(J), heuristique(J,S,H).

test\_heur\_egalite(H) :- test\_egalite(S), joueur\_initial(J), heuristique(J,S,H).

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

**3.2 Quel prédicat permet de connaitre 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.**

A FAIRE

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

A FAIRE

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

A FAIRE

Voici les tests unitaires :

test\_negmax1([ [x,o,\_],

[o,\_,o],

[x,o,\_] ]).

test\_negmax2([ [x,\_,\_],

[\_,\_,o],

[x,o,\_] ]).

:- joueur\_initial(J), test\_negmax1(S), negamax(J,S,0,3,A).

**4. Expérimentation et extensions**

**4.1 Quel est le meilleur coup à jouer et le gain espéré́ 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).**

A FAIRE

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

A FAIRE

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

A FAIRE

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

A FAIRE

**II-** **Code source et tests unitaires**

Afin de compléter les extraits de code présentés dans les différentes réponses aux questions vous trouverez l’intégralité de notre code source commenté et nos test unitaires sur le dépôt Git suivant si besoin :

https://github.com/Clemence-Lemeilleur/TP\_IA

**III-** **Temps de réponse du programme pour différents problèmes de difficulté variée**

Afin d’obtenir le temps de réponse des programmes nous avons utilisé la commande suivante :

time(main). %en ms

Nous avons obtenu les résultats suivants pour les différents programmes avec des difficultés variables :

* A\* : voir plus haut
* Negamax :

A mettre

**IV-** **Limitations du programme, adaptabilité à d’autres problèmes**

Comme tous programmes, ceux que nous avons codés comportent des limites. Dans notre cas elles nous semblent être les suivantes :

* A\* : Nous avons travaillé sur des
* Negamax :

A mettre

Ces programmes pourraient s’adapter à d’autres problèmes comme par exemple un jeu de puissance 4 comme l’ouverture a été abordée dans le TP ou d’autres jeux opposant 2 jours, comme une bataille navale par exemple.

**V-** **Extensions réalisées ou entrevues**

Dans les programmes que nous avons codés nous pouvons rajouter certaines extensions afin de toucher plus de cas possible ou résoudre une variété plus grande de problèmes.

Pour reprendre ce qui a été abordé dans la partie précédente il serait envisageable de reprendre les codes écrits pour le jeu de taquin et tictactoe dans divers autres jeux de plateaux comme puissance 4, la bataille navale ou encore les échecs ou les dames.

***Conclusion :***

Tout au long de ces TP, nous avons réussi à développer répondre aux problèmes posés en codant au fur et à mesure les prédicats en Prolog. Nous avons réussi à gérer l’approche des différents problèmes de jeu de taquin, morpion par exemple, puis le codage et enfin les tests unitaires. Nous avons pu expérimenter et mieux comprendre les principes de base de l’intelligence artificielle. Cela nous permet donc de nous mettre une fois de plus dans le rôle de l’ingénieur qui est d’utiliser ses connaissances et de les appliquer à des cas réels.

Table des illustrations

**Figure 1.** – Screenshot personnel [25/03/2022]. *Code*.

Table des annexes

1. Annexe 1 : Lien GitHubA

**Annexe 1 :** Lien GitHub : https://github.com/Clemence-Lemeilleur/TP\_IA

