INSA TOULOUSE

Rapport TP Intelligence Artificielle

UF : Systèmes Intelligents

Brandon ZHONG 4IR B1

Partie 1 : Algorithme A*- Taquin

1) Familiarisation avec le problème du Taquin 3×3

1.2.a) la clause final_state_4([[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).

?- final_state(Fin), nth1(3,Fin,Ligne), nth1(2,Ligne,P)
```

L'élement d dans la colonne C de la ligne L.
Vérifie le P de la colonne 2 dans la ligne 3.

c) initial_state(U0), coordonnees([L,C], U0, a),

final_state(F), coordonnees([L,C], F, a)).

On cherche les coordnonnées de a dans la situation U0, puis celui dans F, pour comparer.

d) initial_state(U0), rule(X,1,U0,U1). (On a trois rules seulement car on ne peut plus faire « down » depuis cette situation.)

```
?- initial_state(U0), rule(X, 1, U0, U1).
U0 = [[b, h, c], [a, f, d], [g, vide, e]]
X = up
U1 = [[b, h, c], [a, vide, d], [g, f, e]]
Yes (0.00s cpu, solution 1, maybe more)
U0 = [[b, h, c], [a, f, d], [g, vide, e]]
X = left
U1 = [[b, h, c], [a, f, d], [vide, g, e]]
Yes (0.00s cpu, solution 2, maybe more)
U0 = [[b, h, c], [a, f, d], [g, vide, e]]
X = right
U1 = [[b, h, c], [a, f, d], [g, e, vide]]
Yes (0.00s cpu, solution 3, maybe more)
No (0.00s cpu)
```

e) initial_state(U0), findall(X, rule(X,1,U0,U1),L).

f) initial_state(U0), findall([A,S], rule(A,1,U0,S),L).

```
?- initial_state(U0), findall([A, S], rule(A, 1, U0, S), L).
U0 = [[b, h, c], [a, f, d], [g, vide, e]]
A = A
S = S
L = [[up, [[b, h, c], [a, vide, d], [g, f, e]]], [left, [[b, h, c], [a, f, d], [vide, g, e]]], [right, [[b, h, c], [a, f, d], [g, e, vide]]]]
Yes (0.00s cpu)
```

2) Développement des 2 heuristiques

2.1.a) initial_state(U0),final_state(F),
findall(Elt, (mal_place(Elt,U0,F),Elt\=vide), L).

taille : initial_state(U0),final_state(F),
findall(Elt, (mal_place(Elt,U0,F),Elt\=vide), L), length(L,T).

2.2)

Aller à la fin du fichier **taquin.pl** et coder les prédicats logiques **heuristique1(U,H)** et **heuristique2(U,H)**.
*Ces prédicats ont été temporairement implémentés par un "bouchon" (**true**) pour passer la compilation.*

Tester les deux heuristiques sur les deux situations extrêmes (U_0 et F).

initial_state(U0), dm(c,U0,D).
initial_state(U0), heuristique1(U0, H). H = 4
initial_state(U0), heuristique2(U0, H). H = 5

```
?- final_state(F), heuristique1(F, H).
No (0.00s cpu)
?- final_state(F), heuristique2(F, H).
F = [[a, b, c], [h, vide, d], [g, f, e]]
H = 0
Yes (0.00s cpu)
```

3) Implémentation de A*

3.2) Situation S0: avec F0=H0+G0.

Tout est précisé dans le fichier aetoile avec des tests unitaires pour quelques prédicats. Pour tester A*, il suffit de taper « main. » puis more pour avoir toutes les réponses possibles (en fonction des situations initiales).
Heuristique2 a une meilleure performance après comparaison des résultats.

3.3) Analyse expérimentale

Pour le cas initial par défaut :

```
%Heuristique1a  : Execution took 5 ms.
%Heurisitique2 : Execution took 4 ms.
```

Pour le cas initial h2=2, f*=2

```
%Heuristique1a: Execution took 1 ms.
%Heuristique2: Execution took  1 ms
```

Pour le cas initial h2=10, f*=10

```
%Heuristique1a: Execution took 21 ms.
%Heuristique2: Execution took  6 ms
```

On peut donc observer que dans tous les cas, c'est l'heuristique 2 qui est le plus rapide, et plus représentative de la situation. En effet, l'heuristique 1 ne prends pas en compte de la complexité pour remettre une pièce mal placé mais seulement le nombre. Il est donc moins réaliste que le h2.

Partie 2 : Négamax – TicTacToe

1) Familiarisation avec le problème du TicTacToe 3×3

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


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

1.2) -Trouver le joueur initial pout la situation.
-Place à la ligne 3 colonne 2 le o.

2.2) J'ai complété les différentes alignements possibles dans le fichier tictactoe.pl pour retrouver les 8 alignements après la requête.
2.2) Quelques tests unitaire pour répondre aux différents cas :

```
?- 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
```

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.

A=[o,o,o], alignement_perdant(A,x) true
A=[o,o,o], alignement_perdant(A,o) false
A=[o,o,o], alignement_gagnant(A,o) true
A=[o,o,o], alignement_gagnant(A,x) false
A=[o,_,o], alignement_gagnant(A,o) false

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

2) Petites vérifications de la valeur de h après les implémentations.

situation_initiale(S), heuristique(J,S,H). h=0

situation(S_perdante), J=x, heuristique(J,S_perdante,H).

```
?- situation(S_perdante), J = x, heuristique(J, S_perdante, H).
S_perdante = [[o, _397, a], [o, b, _407], [o, _413, _415]]
J = x
H = -10000
Yes (0.00s cpu)
```

Situation(S_gag), J=o , heuristique(J,S_gag,H).

```
?- situation(S_gag), J = o, heuristique(J, S_gag, H).
S_gag = [[o, _396, a], [o, b, _406], [o, _412, _414]]
J = o
H = 10000
Yes (0.00s cpu)
```

3) Développement de l'algorithme Negamax

3.2)

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.

```
?- situation_initiale(S), joueur_initial(J), successeurs(J, S, Succ).
S = [[_528, _530, _532], [_536, _538, _540], [_544, _546, _548]]
J = x
Succ = [[[1, 1], [[x, _601, _603], [_607, _609, _611], [_615, _617, _619]]], [[1,
Yes (0.00s cpu)
```

3.3) Commentaire sur les paramètres de loop_negamax/5 :

```
loop_negamax(J,P,Pmax,[[Coup,Suiv]|Succ],[[Coup,Vsuiv]|Reste_Couples]) :-
```

J : Joueur concerné par l'exécution de negamax

P : La profondeur actuelle du jeu (1..9)

Pmax : Profondeur max (9)

[[Coup,Suiv]|Succ] : Liste de Succ avec les éléments [Coup,Suiv], le coup joué et l'état du tableau après l'action.

[[Coup,Vsuiv]| Reste_Couples] : Liste de [coup, heuristique associé au coup]. Vsuiv c'est la valeur de l'heuristique associé au coup joué.

4) Expérimentation et Extensions

4.1) Meilleur coup à jouer c'est au centre [2,2], qui permet un V = 4.

Pour un profondeur de 9, seul le choix de match nul ou gagnant est possible, donc on est à nouveau dans le résultat [2,2].

```
?- main(B, V, 0).
B = rien
V = 0
Yes (0.00s cpu)
?- main(B, V, 1).
B = [2, 2]
V = 4
Yes (0.00s cpu, solution 1, maybe more)
?- main(B, V, 2).
B = [2, 2]
V = 1
Yes (0.00s cpu, solution 1, maybe more)
?- main(B, V, 3).
B = [2, 2]
V = 3
Yes (0.02s cpu, solution 1, maybe more)
```

```
?- main(B, V, 4).
B = [2, 2]
V = 1
Yes (0.05s cpu, solution 1, maybe more)
?- main(B, V, 5).
B = [2, 2]
V = 3
Yes (0.25s cpu, solution 1, maybe more)
No (0.25s cpu)
?- main(B, V, 6).
B = [2, 2]
V = 1
Yes (0.90s cpu, solution 1, maybe more)
No (0.92s cpu)
```

```
?- main(B, V, 7).
B = [2, 2]
V = 2
Yes (2.32s cpu, solution 1, maybe more)
?- main(B, V, 8).
Abort
```

Après avoir implémenté les différentes, nous procédons au test pour vérifier le bon fonctionnement et le temps d'éxécution qui s'amplifie au fur et à mesure que le jeu avance (1..9). De plus, on voit bien que avec cette implémentation, mon code ne permet pas d'arriver jusqu'à la fin du jeu (8 et 9), car le temps d'éxécution est trop grande.

Sur les deux TP, il est possible d'améliorer la performance des prédicats. Cependant, le temps d'exécution est correct et tous les prédicats fonctionnent correctement.

4.3) Puissance 4 : un tableau de différente taille (je n'ai pas les dimensions exactes). De plus les règles sont différentes :

- remplir de bas en haut par colonne (on peut passer à la ligne 2 ssi ligne 1 est placé).

- Victoire sur alignement de 4 jetons (même alignement)

4.4) En utilisant alpha-beta, on peut conserver les valeurs de J et pour l'adversaire. (alpha et beta)

Si pendant la recherche, la valeur est pas compris entre alpha et beta, nous pouvons élagué la branche concernée.