

Rapport TPs IA

- Algorithme A* pour la recherche heuristique dans les graphes d'état
- Negamax - TicTacToe

Imad Boukezzata
MEJRI Hazem

Rapport TPs IA

- Algorithme A^* pour la recherche heuristique dans les graphes d'état
- Negamax - TicTacToe

Imad Boukezzata
MEJRI Hazem

Table des matières

1	Algorithme A*	1
1.1	Familiarisation avec le problème du Taquin 3×3	1
1.2	Développement des 2 heuristiques	3
1.2.1	L'heuristique du nombre de pièces mal placées	3
1.2.2	Heuristique 2 : Heuristique basé sur la distance de Man- hattan	3
1.3	Algo A*	4
1.3.1	Implémentation de P et Q par des arbres AVL	4
2	Negamax - TicTacToe	4
2.1	Familiarisation avec le problème du TicTacToe 3×3	4
2.1.1	Réponse	4
2.1.2	Test Prédicat	5
2.1.3	Algo Heuristique	6
2.2	Algorithme Negamax	6
2.2.1	Prédicats et Tests	6

1 Algorithme A*

1.1 Familiarisation avec le problème du Taquin 3×3

a)

La commande `final_state(S)` permet de visualiser l'état final du jeu de Taquin 4*4

A voir avec `final_state([[f, g, a],[h,vide,b], [d, c, e]])` où la matrice indiquée est un état initial

b)

- `initial_state(Ini)` : Permet de donner le où les états initiaux du jeu

- `nth1(L,Ini,Ligne)` : Retourne une ligne de la structure `Ini` et donc en faisant la query , ici on peut dire que `Ligne = Ini(L)`

query : `initial_state(Ini)` , `nth1(L,Ini,Ligne)`

-> Renvoie les ligne de la structure `Ini` qui ici est la matrice 3*3 du jeu de taquin

- `nth1(C,Ligne, Constante)` : renvoie les Constantes de la ligne en faisant `Constante = Ligne(C)`

-> donc `nth1(C,Ligne, d)` permet de localiser la constante "d" en mettant la query adapté , ici cette query serait

query = `initial_state(Ini),nth1(L,Ini,Ligne), nth1(C,Ligne, d)`

renvoi

`Ini = [[b, h, c], [a, f, d], [g, vide, e]]`

`L = 2`

`Ligne = [a, f, d]`

`C = 3`

Yes (0.00s cpu, solution 1, maybe more)

"d" est à la 3eme place de la ligne 2

c)

-`Final_state(Fin)` : renvoie la matrice 3*3 qui est la solution au jeu de taquin

renvoi

`final_state(Fin).`

`Fin = [[a, b, c], [h, vide, d], [g, f, e]]`

- `nth1(3,Fin,Ligne)` : Renvoie la 3ème ligne de la structure "Fin"

query : `final_state(Fin),nth1(3,Fin,Ligne)`

renvoi

`Fin = [[a, b, c], [h, vide, d], [g, f, e]]`

`Ligne = [g, f, e]`

- `nth1(2,Ligne,P)` : va renvoyer la constante à la position 2, de la structure `Ligne` , on va donc utiliser la query

query : `final_state(Fin),nth1(3,Fin,Ligne),nth1(2,Ligne,P)`

renvoi

`Fin = [[a, b, c], [h, vide, d], [g, f, e]]`

`Ligne = [g, f, e]`

`P = f`

d)

Pour trouver les actions possible à la situation initiale du jeu de taquin, on va utiliser la *query*

query : initial_state(Ini), rule(A, Cost, Ini, _Suivant)

où

A : représente les action réaliser (ce qqui nous intéresse) Cost : le coût de chaque action (1 à chaque fois ici) Ini : la structure où l'on recherche _Suivant : l'état suivant l'action réalisé mais ici ne nous intéresse pas donc on va mettre " _ " devant

renvoi

Ini = [[b, h, c], [a, f, d], [g, vide, e]]
A = up
Cost = 1
_Suivant = [[b, h, c], [a, vide, d], [g, f, e]]
Yes (0.00s cpu, solution 1, maybe more)
Ini = [[b, h, c], [a, f, d], [g, vide, e]]
A = left
Cost = 1
_Suivant = [[b, h, c], [a, f, d], [vide, g, e]]
Yes (0.00s cpu, solution 2, maybe more)
Ini = [[b, h, c], [a, f, d], [g, vide, e]]
A = right
Cost = 1
_Suivant = [[b, h, c], [a, f, d], [g, e, vide]]
Yes (0.00s cpu, solution 3, maybe more)
No (0.00s cpu)

e)

Pour mettre toutes les actions réalisable dans une liste on va mettre ca dans une liste en utilisant le prédicat findall

query : initial_state(Ini),findall(A , rule(A, Cost, Ini, _Suivant),L)

renvoi

Ini = [[b, h, c], [a, f, d], [g, vide, e]]
A = A
Cost = Cost
_Suivant = _Suivant
L = [up, left, right]
Yes (0.00s cpu)

f)

On va aussi utiliser un findall mais en modifiant un peu la requête pour avoir des couples

query : initial_state(Ini),findall([A,Suivant] , rule(A, Cost, Ini, Suivant),L)

renvoi

Ini = [[b, h, c], [a, f, d], [g, vide, e]]
A = A
Suivant = Suivant
Cost = Cost
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]]]]

1.2 Développement des 2 heuristiques

1.2.1 L'heuristique du nombre de pièces mal placées

Dans cet algorithme on calcule juste le nombre de pièce mal placée :

-heuristique1(U,H) :-

Notre première fonction `Heuristique1` récupère le résultat final en faisant appel à la fonction `heuristique_aux1`

-heuristique1_aux([H1|T1],[H2|T2],N) :-

Cette fonction-aux fait l'appel récursif pour que à chaque élément du jeu on détermine si il est bien placé ou pas et au final récupère la somme des pièces mal placées

-heuristique1_aux2([L|T], [L|T2],N) :-

Finalement cette fonction vérifie par premier élément passe dans l'état `Ini` et l'état `Fin`, si les premières pièces sont identiques on passe au second élément sinon on incrémente un compteur (qui est ici le coût) et on traite les autres éléments

Test :

soit avec un jeu taquin 4x4 :

Soit :

`initial_state([[5, 1, 2, 3], [9, 6, 7, 4], [13, 10, 11, 8], [14, 15, 12, vide]]).`

`final_state([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, vide]]).`

query : `initial_state(Ini), heuristique1(Ini,H).`

renvoi

`Ini = [[5, 1, 2, 3], [9, 6, 7, 4], [13, 10, 11, 8], [14, 15, 12, vide]],`

`H = 11 .`

1.2.2 Heuristique 2 : Heuristique basé sur la distance de Manhattan

-heuristique2(U,H) :-

`Findall` récupère la liste des distances de manhattan pour chaque pièce, et `sumlist` fait la somme de cette liste pour obtenir le coût heuristique total

-heuristique2_choice(Lettre,P,Ini,Fin) :-

Pour chaque lettre de l'alphabet (appeler par `Findall`), cette fonction `_aux` calcule la distance entre sa place dans `Ini` et celle dans `Fin` et le renvoie en tant que élément de la liste retournée par `Findall` dans `heuristique2`

Test :

`initial_state([[14, 13, 9, 5], [15, 6, 7, 1], [12, 10, 11, 2], [8, 4, 3, vide]]).`

query : `initial_state(Ini), heuristique1(Ini,H).`

renvoi

`Ini = [[14, 13, 9, 5], [15, 6, 7, 1], [12, 10, 11, 2], [8, 4, 3, vide]],`

`H = 44.`

1.3 Algo A*

1.3.1 Implémentation de P et Q par des arbres AVL

- Le main sert à initialiser les éléments de base de A* , cad Q, Pu et Pf. L'Heuristique de l'état initiale H0 est initialisé et permet d'avoir F0. G0 est égale à 0 au début. On insert ensuite les états initiaux dans Pf et Pu.

prédicat :

main :-

```
    initial_state(Ini),
    heuristique2(Ini,H0),
    G0 is 0,
    F0 is H0+G0,
    Data=[F0,H0,G0],
    empty(Pf),
    empty(Pu),
    empty(Q),
    insert([Data,Ini],Pf,Pf),
    insert([Ini,Data,nil,nil],Pu,Pu),
    write('\n'),
    write(Data),
    aetoile(Pf,Pu,Q).
```

aetoile(Pf,Pu,Q) :-

— **expand/2**

Ce prédicat a pour but de recuperer la liste de tous les successeurs à partir d'un etat U ainsi que leurs etats , le tripple [F,H,G],

Appel du prédicat loopsuccessors

— **loopsuccessors/6**

Traitement de Cas suivant la fonctionnalité passé en sujet de TP :

— Cas 1 :

Appartenance à Q

— Cas 2 :

Appartenance à Pu :

si il ne l'est pas , ajout à Pu avec priorite calculé suivant le cout total
sinon mise a jour du cout total s'il est meilleur que l'ancien

— Cas 3 :

Ajout des Successeurs à Pu et Pf.

2 Negamax - TicTacToe

2.1 Familiarisation avec le problème du TicTacToe 3×3

2.1.1 Réponse

1.2)

query : situation_initiale(S), joueur_initial(J).

renvoi

S = [[_, _, _], [_, _, _], [_, _, _]],
J = x.

Ce predicat renvoie la situation Initial du jeu et le joueur qui commence

query : situation_initiale(S), nth1(3,S,Lig), nth1(2,Lig,o).

renvoi

S = [[_, _, _], [_, _, _], [_A, o, _B]],
 Lig = [_A, o, _B].

- situation_initiale(S) : Recherche la situation inial du jeu
 - nth1(3,S,Lig) : Renvoi la 3ème ligne de la structure S
query : situation_initiale(S) ,nth1(3,S,Lig)
 renvoi

S = [[_, _, _], [_, _, _], [_A, _B, _C]],
 Lig = [_A, _B, _C].

- nth1(2,Ligne,P) : verifie si le deuxieme element de la ligne Lig est unifié avec "o"
query : situation_initiale(S), nth1(3,S,Lig),nth1(2,Lig,o)
 renvoi

S = [[_, _, _], [_, _, _], [_A, o, _B]],
 Lig = [_A, o, _B].

2.1.2 Test Prédicat

2.2)

Test pour les autres prédicats :

— unifiable/2 :

query : unifiable(X, 5).
 renvoi

true.

query : unifiable(X, o).

true.

query : unifiable(x, o).

false.

— alignement_gagnant/2 et alignement_perdant/2 :

query : alignement_gagnant([x, x, x], x).
 renvoie

true.

query : alignement_perdant([x, x, o], x).
 renvoi

false.

2.1.3 Algo Heuristique

Evaluation de Tictactoe :

Traitement de cas :

— Cas 1 : Situation gagnante pour J :

Affiche une valeur 10000 indiquant une forte preference a cette situation

— Cas 2 : Situation Perdante pour J :

Affiche une valeur -10000 indiquant une forte aversion a cette situation

— Cas 3 : Situation Neutre pour les 2 joueurs :

L'heuristique calcule la difference entre le nombre d'alignement possible pour le joueur J , et celle possible au joueur adversaire. Cette difference est l'heuristique recherche qui indique au joueur J la situation , plus la difference est grande plus la situation est favorable pour J

— Test

query : situation_initiale(S), successeur(x, S, [1, 1]), successeur(x, S, [1, 2]), successeur(x, S, [1, 3]), heuristique(x, S, H).

renvoi

S = [[x, x, x], [_ , _ , _], [_ , _ , _]],
H = 10000.

situation_initiale([[o,_,x], [_,_,_], [_,x,_]]).
query : situation_initiale(S), heuristique(o, S, H).
renvoi

S = [[o, _ , x], [_ , _ , _], [_ , x, _]],
H = -10000.

2.2 Algorithme Negamax

2.2.1 Prédicats et Tests

— meilleur/2

Choix du meilleur couple A=[C,V] selon la plus petite valeur de V

test :

query : meilleur([1, 10], [2, 5], [3, 8], A).

A = [2,5] .

— negamax/5

Traitement de cas :

— Cas 1 :

Ce cas est lorsque la profondeur max Pmax est atteinte , on évalue alors l'heuristique.

negamax (_J,Etat,P,Pmax,[_,V]) :-
 P==Pmax,
 heuristique(_ ,Etat,V).

— Cas 2 :

l'etat est instancié et le joueur ne peux pas jouer

negamax(J,Etat, _P, _Pmax,[_,V]) :-
 ground(Etat),
 successeurs(J,Etat,_),
 heuristique(_ ,Etat,V).

— Cas 3 :

Idée : (**Partie non finie**)

Profondeur max Pmax non atteinte, on genere les successeurs et on evalue l'heuristique.

negamax(J,Etat,P,Pmax,[C,V]) :-

P < Pmax,

successeurs(J,Etat,Succ),

loop_negamax(J,P,Pmax,Succ,Liste_Couples),

meilleur(Liste_Couples,[C,Valeur]),

V is -Valeur.

