Jeu du taquin

L3 CILS, L3 BI, DL3

Le projet est à réaliser en binôme. Le travail sera évalué par 1) un rapport et 2) une soutenance. Pour les préparer, il est judicieux de prendre des notes pendant la conception, l'implémentation, les tests, et les expérimentations de vos programmes.

Problème

Un taquin 3*3 est un puzzle carré, composé de 8 nombres et d'un trou. Le jeu consiste à déplacer le trou au nord, au sud, à l'est ou à l'ouest à partir d'un état initial jusqu'à un état final donné, en limitant le nombre de coups.

1	3	X
5	7	6
4	2	0

Etat E

0	1	2
3	4	5
6	7	X

Etat final

Algorithme A*

On part sur l'idée d'utiliser une heuristique. Soit E un état.

- On note $g(E)$ la longueur du plus court chemin menant de l'état initial à E. C'est le plus petit nombre de déplacements du trou permettant de passer de l'état initial à E.
- L'heuristique $h(E)$ est une estimation du nombre de déplacements nécessaires pour passer de E à l'état final. $h()$ est en fait une distance entre états, et plusieurs sont proposées ci-après.
- La fonction d'évaluation est définie par $f(E) = g(E) + h(E)$.

Plusieurs choix d'implémentation doivent encore être faits.

Concernant les **états**, il faut une structure de données permettant de stocker la configuration du plateau, mais aussi le plus court chemin qui a permis de le découvrir et sa longueur. Une façon de faire est d'attribuer une adresse à chaque état, de conserver pour chaque état l'adresse de son père ainsi que la longueur du chemin l'attachant à l'état initial (voire les valeurs de $h(\cdot)$ et de $f(\cdot)$ pour cet état) ; on rappelle que tous les états rencontrés sont soit dans l'ensemble exploré, soit dans l'ensemble frontière, ce qui permet de les retrouver facilement à partir de leurs adresses. Une autre façon de faire est de conserver dans chaque état le chemin sous la forme d'une chaîne de caractère, qui code les mouvements du trou pour obtenir l'état à partir de l'état initial (par exemple `"ONN"` pour `ouest`, puis `nord`, puis `nord`) ; dans ce cas, on a directement la longueur du grâce à la longueur de la chaîne.

Concernant l'**ensemble exploré**, il faut utiliser un dictionnaire d'états pour les stocker. Logiquement, il ne faut pas utiliser une simple liste, mais un *arbre binaire de recherche* par

exemple, pour optimiser les temps de calcul. On peut également utiliser une table de hachage. En tout cas, il faut utiliser une structure de données adaptée.
Enfin concernant l'**ensemble frontière**, il faut une *file de priorité* d'états, ordonnée par valeur croissante de $f(\cdot)$. La question des états redondants au sein de cet ensemble peut se poser, être gérée ou non, au moment de l'entrée ou de la sortie d'un état de la file.

Les heuristiques

On définit six heuristiques $h_1(\cdot), \ldots h_6(\cdot)$ en calculant des distances de Manhattan pondérées réduites. D'autres sont possibles. L'heuristique $h_6(\cdot)$ est la distance de Manhattan standard :

1. La distance élémentaire $\varepsilon_E(i)$ d'un élément i d'un état E est le nombre de déplacements élémentaires qu'il faut faire pour le mettre à sa position dans l'état final. Ainsi, avec l'exemple de l'état E et l'état final dessinés en page 1, $\varepsilon_E(1) = 1$, $\varepsilon_E(3) = 2$, $\varepsilon_E(6) = 3$, $\varepsilon_E(0) = 4$, ...
2. On obtient les heuristiques $h_1(\cdot), \ldots h_6(\cdot)$ en faisant une somme pondérée des valeurs de la distance élémentaire ε_E, selon les jeux de poids suivants :

	0	1	2	3	4	5	6	7	8
π_1	36	12	12	4	1	1	4	1	0
$\pi_2 = \pi_3$	8	7	6	5	4	3	2	1	0
$\pi_4 = \pi_5$	8	7	6	5	3	2	4	1	0
π_6	1	1	1	1	1	1	1	1	0

3. Enfin, on propose de réduire les sommes précédentes à l'aide de coefficients de normalisation ; on pose $\rho_1 = \rho_3 = \rho_5 = 4$ et $\rho_2 = \rho_4 = \rho_6 = 1$.
4. Au final, on définit pour $1 \leq k \leq 6$:

$$h_k(E) = \left(\sum_{i=0}^{8} \pi_k(i) \times \varepsilon_E(i) \right) \text{ div } \rho_k$$

où div dénote le quotient de la division entière.

Travaux d'expérimentation

Il est intéressant de comparer les performances de vos implantations en fonction de l'état initial choisi et de la distance utilisée. De bons critères de comparaison sont le nombre d'états sortis de la frontière, le nombre d'états explorés, la longueur de la solution, le temps CPU du calcul, *etc.*
Attention : comme pour le Rubik's Cube, seulement la moitié des états initiaux permettent d'atteindre l'état final. Il parait intéressant de creuser la question avant de se lancer.
Il peut aussi être intéressant de construire un taquin 4*4, ou même N*N, en adaptant les heuristiques précédentes, ou bien de faire une petite interface graphique ...vos idées sont les bienvenues !
Pour toute extension, optimisation, références bibliographiques, regarder :
`http://www.enseignement.polytechnique.fr/informatique/ARCHIVES/IF/projets/pottier/sujet.html`
Mais attention, mieux vaut un programme "simpliste" qui fonctionne que pas de programme.