TP 1 : Algorithme A*

Contexte :

Le but de ce TP est de programmer une intelligence artificielle permettant de résoudre le Taquin en utilisant l'algorithme `A*`.

Le taquin est un jeu solitaire en forme de damier. Il est composé de 15 petits carreaux numérotés de 1 à 15 qui glissent dans un cadre prévu pour 16. Il consiste à remettre dans l'ordre les 15 carreaux à partir d'une configuration initiale quelconque.

Dans la figure 1 nous illustrons une version du jeu avec 8 cases. A gauche une situation de départ quelconque, et à droite la situation but désirée.

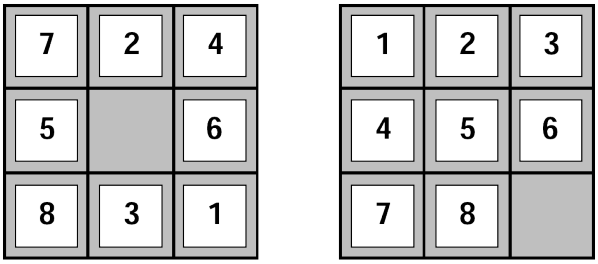

FIGURE 1 – Exemple du jeu du taquin en version 8.

Un mouvement du taquin consiste à faire glisser une case numérotée vers la case vide ; La case vide prend alors la place de la case déplacée. Pour rappel, l'évaluation $f(n)$ d'un noeud n est égale à $f(n) = g(n) + h(n)$ avec $g(n)$ le coût (nombre de mouvements ayant mené à l'état courant) et $h(n)$ l'estimation du coût restant au but.

Dans ce TP on va tester deux heuristiques différentes :

— $h1(n)$ = nombre de cases mal placées,
— $h2(n)$ = la somme des distances des cases par rapport à leurs positions cibles.

Par exemple, pour la grille initiale de la figure 1, $h1 = 1+0+1+1+1+0+1+1 = 6$ et $h2 = 4+0+3+3+1+0+2+1 = 14$

Implémentation

Votre projet sera constitué de 2 principales classes :

La classe Noeud

Cette classe définit un état du jeu qui est également un noeud de l'arbre de recherche. Une instance `Noeud`. est définie par une grille du jeu, une référence vers le noeud père et le nombre de mouvements `g` effectués depuis l'état initial (le noeud racine).

Elle est composée d'attributs et de méthodes suivantes :

- `Attributs :`
  - `grille` : correspond à la grille du puzzle. Elle est implémentée avec une matrice carrée d'entiers (nombre de lignes = nombre de colonnes).
  - `pere` : est la référence vers le noeud père dans l'arbre de recherche.
  - `g` : est un champs qui sauvegarde le nombre de mouvements effectués depuis la racine de l'arbre.
- `Méthodes :`
  - Un constructeur de classe qui initialise l'ensemble des attributs avec des valeurs passées en paramètres.
  - Un *getter* pour chaque attribut.
  - Une méthode pour afficher un noeud.
  - Une méthode qui renvoie $True$ si deux noeuds ont des grilles identiques.
  - Une méthode qui calcule la valeur h du noeud.
  - Une méthode qui calcule et retourne l'évaluation du noeud $f = g + h$.
  - La méthode `estUnEtatFinal` retourne $True$ si le noeud contient la configuration finale du jeu.
  - La méthode `successeurs` retourne la liste de tous les noeuds successeurs obtenus en déplaçant la case vide vers le haut, le bas, la gauche et la droite. Le nombre de successeurs varie entre 2 et 4 selon la position de la case vide.

Testez votre classe en créant un noeud avec l'état initial de la Figure 1.

1. Affichez le noeud.
2. Affichez la valeur h de ce noeud en utilisant l'heuristique h_1 puis h_2.
3. Générer et afficher les successeurs de ce noeud.

La classe Astar

Cette classe implémente l'algorithme A^* pour résoudre une grille quelconque du Taquin. L'algorithme A^* utilise deux listes pour explorer les noeuds de l'arbre de recherche : $liste\_ouverte$ et $liste\_fermee$

- **Méthodes :**
  - La méthode $algoAstar(n : Noeud)$ implémente l'algorithme A^* pour résoudre la grille passée en paramètre et retourne le noeud correspondant à la grille résolue sinon elle retourne *null* (certaines grilles n'ont pas de solution).
  - La méthode $cheminComplet(n : Noeud)$ affiche toutes les étapes (noeuds intermédiaires) qui ont permis d'atteindre le noeud final passé en paramètre ainsi que le nombre de mouvements.

Tester votre algorithme sur différentes grilles.

Description détaillée des étapes d'implémentation de l'algorithme `A*` :

L'algorithme `A*` est implémentée avec deux listes. La première liste appelée `liste_ouverte` contient tous les noeuds étudiés. Dès que l'algorithme va se pencher sur un noeud du graphe, il passera dans la liste ouverte (sauf s'il y est déjà). La seconde liste `liste_fermee`, contiendra tous les noeuds qui ont été considérés comme faisant partie du chemin de la solution. Avant de passer dans la liste fermée, un noeud doit d'abord passer dans la liste ouverte. En effet, il doit d'abord être étudié avant d'être jugé comme bon. Ci dessous les étapes pour implémenter l'algorithme `A*` .

1. Le noeud initial est ajouté à dans la liste ouverte $liste\_ouverte$.
2. On cherche le meilleur noeud de la liste ouverte. Il s'agit du noeud qui a la valeur f minimale. On le note $noeudCourant$.
3. On ajoute $noeudCourant$ à la liste fermée et on le retire de la liste ouverte.
4. On génère tous les noeuds successeurs du $noeudCourant$ en déplaçant la case vide.
5. Pour chaque noeud successeur s **non présent dans la liste fermée.**
   - Si s n'est pas dans la liste ouverte, on le rajoute à cette liste.
   - Sinon, s'il existe dans la liste ouverte un noeud n avec une grille identique à celle de s, alors on remplace n par s dans la liste ouverte si et seulement si l'évaluation du noeud s est meilleure que celle de n : $f(s) < f(n)$.
6. On réitère à partir de l'étape 2 jusqu'à ce qu'on trouve le noeud but (la grille finale) ou bien la liste ouverte est vide (pas de solution).