Projet d'Intelligence Artificielle

Algorithme de résolution du jeu de taquin

Jean Hastoy et Antoine Conqui

IA 1er semestre

Table des matières

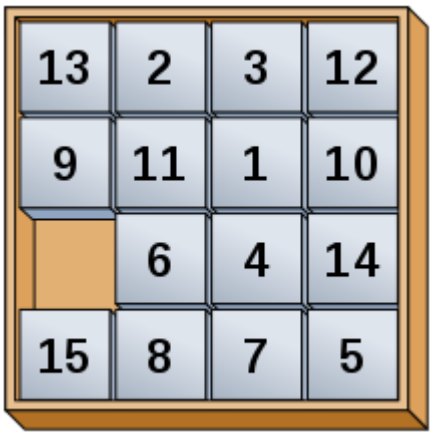
Contexte du projet

Objectif

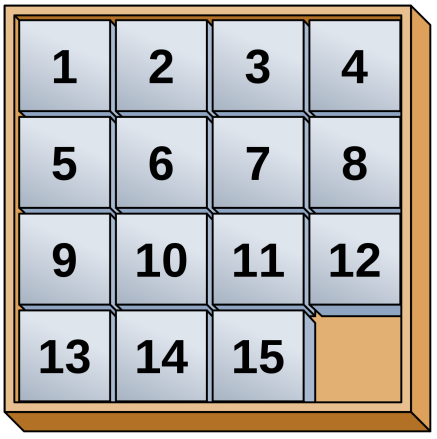
L'objectif du projet est de concevoir un algorithme d'intelligence artificielle accompagné d'une interface graphique permettant de résoudre le jeu du "Taquin". Cet algorithme, développé en C#, doit être capable de résoudre toute sorte de taquin de taille 3x3 et 5x5 dans un temps raisonnable (moins de 60 secondes). L'interface graphique, développée en WinForm, doit permettre de lancer la résolution d'un taquin définit, et être claire et ergonomique. Ce projet est réalisé en binôme.

Jeu du Taquin

Le jeu du Taquin a été inventé par Sam Loyd en 1870. Le jeu original est composé d'une grille 4x4. Sur chaque case, on trouve un carreau numéroté de 1 à 15, une case étant vide. L'objectif est de déplacer les carreaux grâce à la case vide dans le but de les ordonner de 1 à 15.



Exemple d'un état initial



Etat final

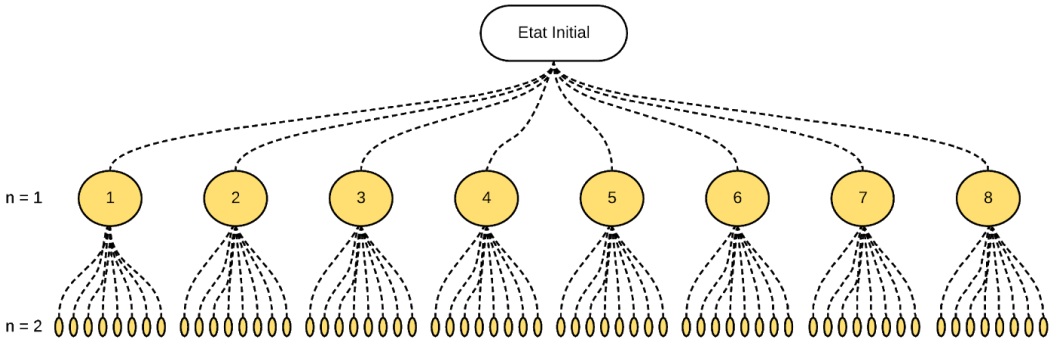
Le jeu du taquin est particulier car, pour un taquin de taille $n \times n$, le nombre d'états possibles est : $(n^2)!$. Ainsi, à partir d'une certaine taille, il existe un nombre astronomiques d'états possibles qui ne peuvent pas être stockées dans la mémoire d'un ordinateur. Nous devons donc avoir recours à des algorithmes d'intelligence artificielle qui n'explorent pas tous les états afin de trouver une solution.

Dans notre projet, nous nous intéressons aux taquin de taille 3x3 et 5x5 dans lesquels deux cases sont vides. Dans le premier cas, 7 carreaux doivent être ordonnés, dans le second cas, 23.

Algorithmes d'IA privilégiés

Nous pouvons représenter les états successifs de notre taquin par un arbre de possibilité. Chaque état représente l'état du plateau au n-ième coup.

Ainsi, pour un taquin 5x5 avec les deux cases vide au milieu du plateau, nous pouvons représenter l'arbre de possibilité suivant :



...

L'objectif est de parcourir cet arbre jusqu'à trouver un état représentant la solution au Taquin. Cependant, le nombre d'état augmentant exponentiellement suivant le nombre de coups, il est impossible de parcourir cet arbre séquentiellement à la recherche d'une solution.

Ainsi, nous allons utilisé (comme recommandé dans le sujet) un algorithme de pathfinding ou "recherche de chemin". L'objectif de ces algorithmes est de trouver un chemin permettant d'aller d'un point A à un point B suivant certaines contraintes.

Il existe deux types d'algorithme de path finding :

Algorithme de Dijkstra (ou algorithme du plus court chemin)

Cet algorithme permet de trouver le plus court chemin entre deux points. Pour cela, on définit des coûts de passages entre chaque état et l'algorithme s'occupe de rechercher le chemin menant à la solution avec le coût le plus faible. Cet algorithme est cependant lent et coûteux en mémoire car il explore l'arbre de possibilité en largeur.

Algorithme A*

Plus rapide, l'algorithme A* permet de trouver un chemin en utilisant une heuristique. En connaissant le point d'arrivée, l'heuristique permet de "guider" les choix des états à explorer. Suivant l'heuristique utilisée et le problème à résoudre, l'algorithme A* n'est pas assuré de trouver le plus court chemin. On dit qu'un algorithme de path finding est "admissible" lorsqu'il trouve toujours le plus court chemin. Pour que A* soit admissible il est nécessaire (mais pas suffisant) que l'heuristique utilisée soit aussi admissible, c'est à dire que celle-ci sous-estime en permanence le coût jusqu'à l'état final par rapport au réel coût du chemin le plus court.

Ainsi pour chaque état, est calculé le coût depuis l'état initial, auquel s'ajoute le coût estimé par l'heuristique jusqu'à l'état final. A* cherche donc toujours à minimiser ce coût total par le choix des états suivant à explorer.

Dans notre cas, en prenant l'état final de plateau et ne cherchant pas forcément le plus court chemin mais plutôt la rapidité de recherche, il est plus judicieux pour nous d'utiliser l'algorithme A*.

Travail réalisé

Modélisation

Nous avons optimisé les classes SearchTree et GenericNode pour pouvoir traiter de la même manière les taquins de taille 3x3 et 5x5.

Heuristiques

Au cours de notre réflexion pour résoudre les différents taquins, nous avons pu mettre au point plusieurs heuristiques successives afin de tester ces performances.

Heuristique de "la pièce mal placée"

Cette heuristique très simple ajoute au coût heuristique une constante pour chaque pièce mal placée sur le plateau.

Elle est très peu efficace car elle n'apporte pas l'information sur le fait que la pièce soit loin ou non de son bon placement et ne permet ainsi pas de "guider" le déplacement des pièces.

```
public double getHeuristicCost()
{
```

```
        double cost = 0;
        for (int i = 0; i < size; i++)
            for (int j = 0; j < size; j++)
            {
                if(successeurTab[i,j] != finalTab[i,j])
                    cost += 1;
            }
        return cost;
    }
```

Heuristique de Manhattan

Cette heuristique prend en compte le fait qu'une pièce soit mal placée, mais aussi sa distance par rapport à son bon emplacement. Ainsi, pour chaque pièce mal placée, on ajoute au coût heuristique sa distance vectorielle par rapport à son emplacement final.Cette heuristique permet donc de "guider" le déplacement des pièces vers son bon emplacement en minimisant leurs distances.

```
public double getHeuristicCost()
{
    double cost = 0;
    for (int i = 0; i < size; i++)
        for (int j = 0; j < size; j++)
        {
            int[] indexFinal = getFinalIndex(value); // retourne un tableau contenant les coordonnées du bon emplacement de la valeur donnée en paramètre
            cost += Math.Abs(i - indexFinal[0]) + Math.Abs(j - indexFinal[1]);
        }
    return cost;
}
```

Optimisation de l'Heuristique de Manhattan

Ce calcul de coût affecte des poids différents à l'heuristique et le coût initial. Une pondération optimisée réduit d'environ 400% le nombre de noeuds ouverts pour le taquin de taille 3.
Cette pondération pour le 3x3 est : coutTotal = 1 * coutInitial + 8 * heuristicManhattan.

Résolution Humaine

Afin de résoudre en temps raisonnable le taquin de taille 5, nous avons dû mettre en place une méthode de résolution basée sur l'intuition humaine.

Notre méthode est la suivante :

- L'algorithme place les pièces une par une, dans l'ordre pour les 3 premières lignes; puis par colonne pour les 2 dernières lignes.
- Pour chaque pièce à placer, on calcule le coût heuristique de chaque descendant avec la formule suivante → heuristicCost = somme de {
    - 7 * distance de la pièce à sa position finale
    - 4 * distance de la pièce au trou le plus proche
    - 2 * distance de la pièce au trou le plus éloigné
    - 8 * distance de la position finale au trou le plus proche }
- Une fois une pièce placée, on récupère le chemin partiel et on l'ajoute au chemin total. On passe ensuite à la pièce suivante.

Cette méthode garantie la résolution de n'importe quel taquin de taille 5x5 en moins d'une seconde.

Interface

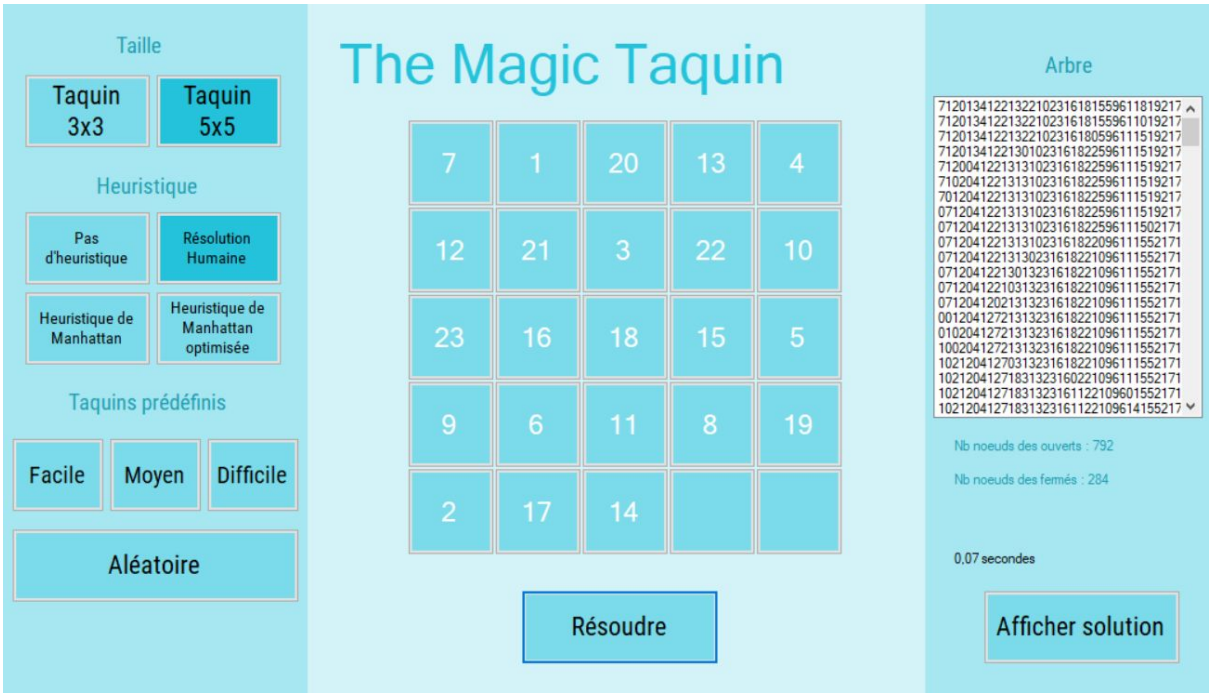
Nous avons choisi de rassembler toutes les options du programme dans une seule et même fenêtre. Cette fenêtre est ainsi divisée en 3 parties.

Dans le menu gauche nous faisons apparaître différents boutons permettant de sélectionner des options avant de pouvoir résoudre le Taquin :

- Taille : choix entre le taquin de taille 3x3 et 5x5.
- Heuristique : choix entre les 4 heuristiques que nous avons développées.
- Taquins prédéfinis : choix de taquins pré-enregistré suivant leur difficulté de résolution (proposés dans le sujet du projet) et bouton de création d'un taquin aléatoire.

Au centre, nous retrouvons l'affichage du taquin se mettant à jour selon la taille et le taquin prédéfinis choisis, ainsi que le bouton "Résoudre" permettant de lancer la résolution du taquin. A noter que ce bouton est actif uniquement si les conditions suivantes sont remplies :

- une taille de taquin est sélectionnée
- une heuristique de résolution est sélectionnée
- le taquin défini est résolvable



Enfin, à droite, nous retrouvons le menu de résolution, avec l'affichage du chemin de résolution, le nombre de noeuds ouverts et fermés, le temps de résolution, et un bouton permettant d'afficher visuellement pas à pas la résolution.

Résultats obtenus et tests

			Heuristique			
			Aucune	**Manhattan**	**Manhattan optimisée**	**Résolution humaine**
3x3	Facile	Noeuds ouverts :	936	47	25	19
		Noeuds fermés :	1190	27	10	9
		Temps de calcul (en sec) :	0,49	0	0	0
	Moyen	Noeuds ouverts :	7794	128	94	25
		Noeuds fermés :	11203	84	49	12
		Temps de calcul (en sec) :	48,27	0,01	0	0
	Difficile	Noeuds ouverts :	?	293	79	57
		Noeuds fermés :	?	207	41	24
		Temps de calcul (en sec) :	?	0,06	0	0
	Moyenne sur 20 taquins aléatoires :	Noeuds ouverts :	?	887,55	203,15	55,65
		Noeuds fermés :	?	693,4	118,95	26,1
		Temps de calcul (en sec) :	?	0,491	0,028	0,006
5x5	Facile	Noeuds ouverts :	?	122	122	49
		Noeuds fermés :	?	55	55	12
		Temps de calcul (en sec) :	?	0,01	0,01	0
	Moyen	Noeuds ouverts :	?	?	?	755
		Noeuds fermés :	?	?	?	280
		Temps de calcul (en sec) :	?	?	?	0,07
	Difficile	Noeuds ouverts :	?	?	?	792
		Noeuds fermés :	?	?	?	284
		Temps de calcul (en sec) :	?	?	?	0,07
	Moyenne sur 20 taquins aléatoires :	Noeuds ouverts :	?	?	?	760,7
		Noeuds fermés :	?	?	?	267,5
		Temps de calcul (en sec) :	?	?	?	0,064

On peut donc constater que :

- l'heuristique est primordiale, sans elle, nous sommes uniquement capable de résoudre les taquins 3x3 de difficultés faciles et moyennes dans un temps acceptable.
- les heuristiques de Manhattan sont très efficaces sur les taquins 3x3 en général, mais vites limitées sur le taquin 5x5 puisqu'elle n'arrivent à résoudre que ceux de difficultés faciles dans un temps raisonnable.
- l'heuristique de résolution humaine est la plus efficace, notamment sur les taquins 5x5 de difficultés moyennes et difficile où la différence de temps de résolution avec Manhattan est au moins de facteur 100.

Gestion d'équipe

Grâce à GitHub et LiveShare (de Visual Studio) nous avons pu travaillé en collaboration sur le projet. Chacun a pu autant s'investir sur la réalisation de l'interface que sur le développement de l'algorithme de résolution.