Rapport - Projet de programmation - L2 Informatique S4
Taquin 4

Projet encadré par : François Laroussinie
Étudiants-membres du groupe: BA Ibrahima, CHHAY Davy, MANYIM Olivier, SILVESTRE Alaia, SIMONEAU Louis-Alexei

Pour ce quatrième semestre de licence, nous avons eu une UE consistant à la réalisation d'un projet. Grâce à nos bases acquises le semestre dernier sur le logiciel GitLab, et nos compétences en Java, nous avons pu implémenter différents algorithmes permettant de résoudre le célèbre jeu du taquin.

Les règles du jeu : Le jeu du Taquin consiste à avoir une grille carrée de chiffres pouvant avoir différentes tailles, comme un taquin ayant une taille 3x3, par exemple. On considère que dans cette configuration de taille n; n-1 cases sont remplies avec un chiffre allant de 1 à n-1 aléatoirement et ainsi la seule case restante est vide. Cette case vide va permettre à l'utilisateur de déplacer les cases numérotées afin de remettre les cases dans l'ordre croissant. Ce n'est qu'une fois toutes les cases bien rangées à leur place, avec la case vide en dernière position de la configuration, qu'on peut considérer le taquin comme résolu et terminé.

Sommaire :

1) Comment installer et utiliser le programme ?

2) Algorithme de Parcours en Largeur.

L'algorithme de parcours en largeur est le premier algorithme que nous avons dû implémenter. Cet algorithme consiste en un parcours de graphe, calculant la distance entre les nœuds depuis le nœud initial. Ce parcours procède de manière à explorer les sommets des graphes puis les successeurs qui n'ont pas encore été explorés.
Cet algorithme utilise une structure FIFO (first in, first out) comme une file, car nous allons pouvoir stocker les sommets rencontrés au fur et à mesure de l'exécution de l'algorithme. Dans notre cas du taquin, nous travaillons avec une variante de l'algorithme de parcours en largeur classique, qui est le parcours en largeur dit "à la volée".

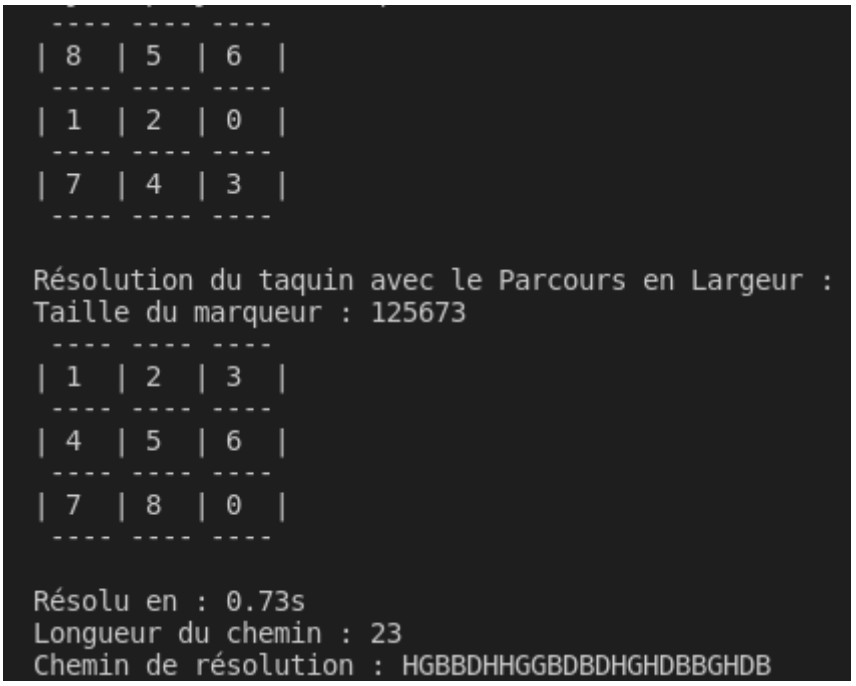
Durant l'implémentation de l'algorithme, nous avons eu quelques soucis par rapport au temps d'exécution, et de complexité. Nous avions, premièrement, fait usage d'une ArrayList ce qui rendait le temps d'exécution trop long.

Également dans l'implémentation, le temps d'exécution était long car nous faisions énormément de boucles.

Afin de résoudre ce problème de temps trop long et rendre l'algorithme beaucoup plus efficace, nous avons donc utilisé un HashSet. Nous avons également fait usage d'une Queue, qui est utilisée comme une file afin de pouvoir stocker les sommets du graphes. Cet algorithme est fondamental dans la résolution du taquin.

Voici quelques exemples de l'algorithme Parcours en Largeur sur différentes tailles de configurations du taquin :

- *Configuration taille 3x3 :*



- *Configuration taille 4x4 :*

Ainsi, nous pouvons passer au second algorithme étudié ce semestre qui est l'algorithme de Dijkstra.

3) Algorithme de Dijkstra.

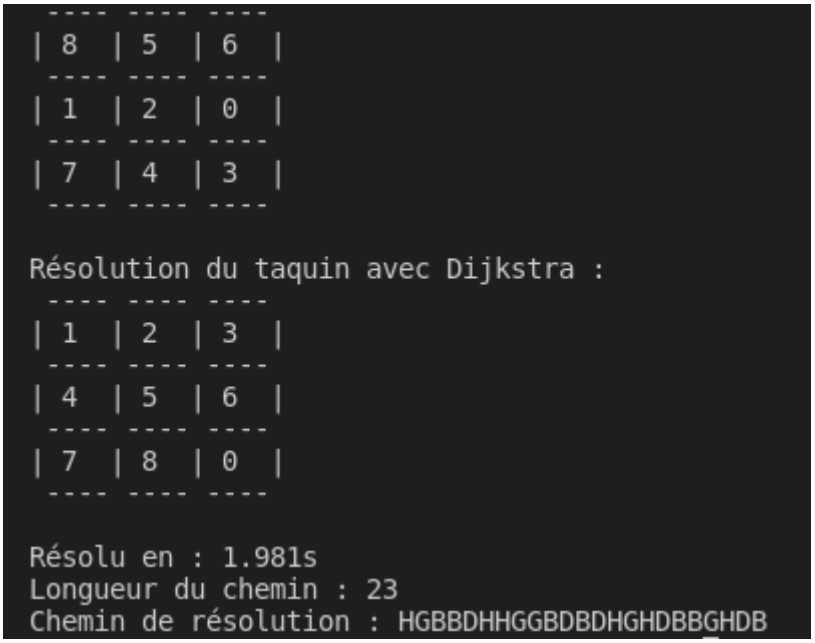
Ce célèbre algorithme appliqué à la théorie des graphes, a pour objectif de calculer le plus court chemin dans un graphe. L'algorithme utilise une file de priorité, c'est-à-dire une structure de données où l'on va pouvoir stocker, dans notre cas, des configurations du taquin.
La configuration renvoyée par l'algorithme est issue de la configuration créée par la fonction ExtraireMin() dans la classe FdPg correspondant à l'implémentation de la File de Priorité.

Pour l'implémentation de cet algorithme, nous n'arrivions pas à implémenter la classe FdPg.java qui est essentielle pour ses nombreuses fonctions telles que celle de l'extraction du minimum, de la mise à jour et de l'ajout, par exemple.
Nous avons également connu un problème par rapport au calcul de la distance entre les sommets du graphe.

Pour résoudre le problème d'implémentation de la classe FdPg, notre professeur-encadrant nous a donné le code, avec les fonctions implémentées car nous commencions à perdre beaucoup de temps.
Nous avons de nouveau fait usage d'un HashSet pour stocker les configurations déjà visitées.

Voici quelques exemples de l'algorithme de Dijkstra sur différentes tailles de configurations du taquin :

- *Configuration 3x3 :*



- *Configuration 4x4 :*

4) Algorithme A*.

L'algorithme A* est un algorithme utilisé pour trouver des chemins entre un sommet et un autre sur des graphes très grands. Il réutilise l'algorithme de Dijkstra. Pour cet algorithme, nous utilisons l'heuristique de Manhattan pour calculer la distance restante.

Il y a eu une confusion entre la distance d'un sommet depuis la Configuration de départ (notée d[u]) et la clé d'un sommet u dans la file (calculée à partir de d[u] et de la distance de Manhattan entre u et la configuration finale) au moment du codage de A*. Notre problème venait au niveau des tests sur les successeurs.

Comme solution à ce problème, nous avons dû revoir toute l'implémentation de l'algorithme en utilisant cette fois-ci un HashMap au lieu d'un HashSet, cependant nous n'avons pas réussi à faire fonctionner l'algorithme sur des configurations aléatoires.

Voici quelques exemples de l'algorithme A* sur différentes tailles de configurations du taquin :

- *Configuration 3x3 :*

```
 ---- ---- ----
| 8  | 5  | 6  |
 ---- ---- ----
| 1  | 2  | 0  |
 ---- ---- ----
| 7  | 4  | 3  |
 ---- ---- ----

Résolution du taquin avec A* :
 ---- ---- ----
| 1  | 2  | 3  |
 ---- ---- ----
| 4  | 5  | 6  |
 ---- ---- ----
| 7  | 8  | 0  |
 ---- ---- ----

Résolu en : 0.126s
Longueur du chemin : 23
Chemin de résolution : GBDHHGGBDDBGHHDBBGHHDBB
```

- *Configuration 4x4 :*

```
 ---- ---- ---- ----
| 5  | 1  | 3  | 4  |
 ---- ---- ---- ----
| 2  | 12 | 7  | 11 |
 ---- ---- ---- ----
| 13 | 15 | 6  | 10 |
 ---- ---- ---- ----
| 8  | 9  | 0  | 14 |
 ---- ---- ---- ----

Résolution du taquin avec A* :
 ---- ---- ---- ----
| 1  | 2  | 3  | 4  |
 ---- ---- ---- ----
| 5  | 6  | 7  | 8  |
 ---- ---- ---- ----
| 9  | 10 | 11 | 12 |
 ---- ---- ---- ----
| 13 | 14 | 15 | 0  |
 ---- ---- ---- ----

Résolu en : 10.35s
Longueur du chemin : 35
Chemin de résolution : HGBGHDBDDHGGHDBGBDDHGBDHHGGGHDBBDDB
```

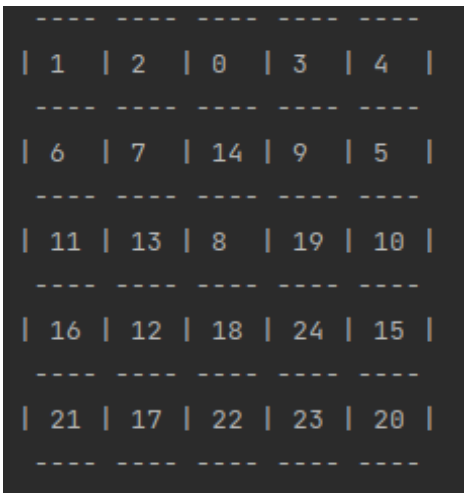
5) Comparaisons entre les différents algorithmes, lequel est le plus efficace ?

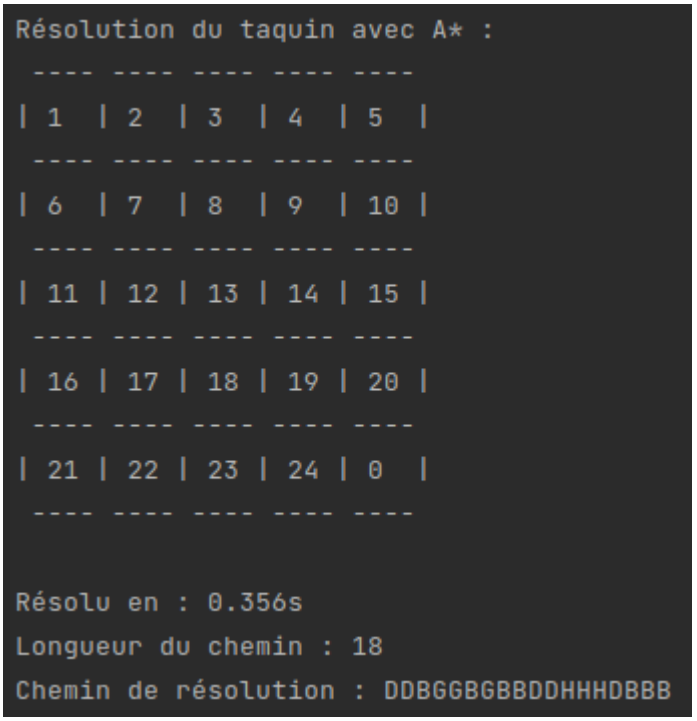
A travers l'implémentation de ces trois algorithmes, nous avons constaté que le plus efficace d'entre-eux est l'algorithme A*. Bien que celui du parcours en largeur est efficace, on a remarqué que l'algorithme de Dijkstra prenait beaucoup plus de temps au moment de

l'exécution que le parcours en largeur alors que ce sont deux algorithmes qui sont censés afficher les mêmes résultats.

A travers, les différents exemples vus pour chaque algorithme, on s'aperçoit que l'algorithme A* est plus efficace en temps par rapport aux autres algorithmes. Nous pouvons résoudre des taquins plus grands et plus compliqués avec A* contrairement à Parcours en Largeur et Dijkstra qui sont incapables de résoudre de grands taquins.

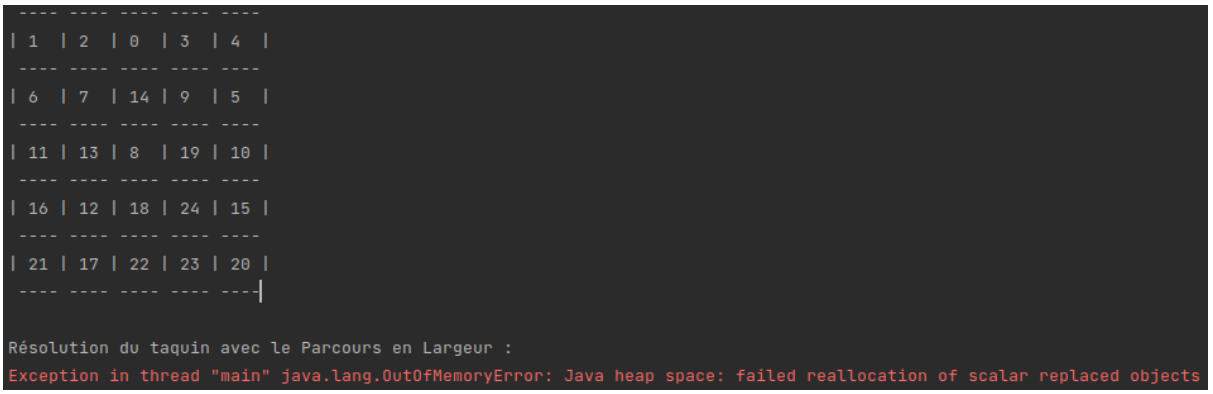
Voici un exemple de résolution de configuration de taille 5x5 avec l'algorithme A* :





Nous constatons que l'algorithme résout bien cette configuration ayant une taille supérieure aux configurations étudiées jusqu'à maintenant.

Voici un exemple de la même configuration, pour une résolution avec l'algorithme de Parcours en Largeur :



On a bien montré que A* est plus efficace et meilleur que le Parcours en Largeur.

6) Interface utilisateur.

Cette interface utilisateur sert à permettre au joueur de pouvoir jouer à un taquin classique en version terminal en déplaçant la case vide dans ce cas. Cette interface propose à l'utilisateur de résoudre lui-même le taquin ou d'exécuter l'algorithme de son choix.

7) Interface graphique.

A propos de l'interface graphique, c'est une partie en plus que nous avons décidé d'implémenter dans notre projet. Le but de cette interface est de pouvoir donner une forme esthétique au jeu du Taquin et rendre son utilisation plus simple que sur un terminal.
Ainsi l'interface graphique se compose d'une page d'accueil, d'une page expliquant les règles du jeu, un bouton permettant de charger une partie en cours non terminée, ainsi qu'un bouton menant à la page principale où apparaît une configuration aléatoire du taquin.
Il y a également un système de sauvegarde dans un fichier pour le pseudonyme qui est entré en paramètre de la zone de texte et pour les niveaux. Le joueur a la possibilité de revoir les niveaux, et il a le choix entre une cinquantaine de niveaux de jeu.
Sur cette page principale, on a le choix entre les différents algorithmes avec lesquels nous voulons résoudre le taquin automatiquement ou alors, on peut jouer à sa résolution.
A la fin de la résolution qu'elle soit automatique ou manuelle, un message s'affiche pour nous informer que nous avons gagné et résolu la configuration.

8) La contribution de chacun des acteurs du projet.

Concernant la participation de chacun des membres acteurs du projet : Alaia a travaillé sur les différents algorithmes qui devaient être implémentés tels que l'algorithme de Dijkstra, et A*. Et elle a également participé à l'élaboration de l'interface utilisateur avec Louis.

Olivier, a quant à lui, travaillé sur les algorithmes aussi, en étudiant leur pseudo codes et les manières de les implémenter. Il a contribué à Dijkstra, l'implémentation de l'heuristique de Manhattan et A*. Il a également participé à la partie de l'interface graphique en faisant la page où le taquin apparaît pour pouvoir y jouer, et y sélectionner les algorithmes que l'on veut appliquer.

Louis a également travaillé sur tous les algorithmes en plus de l'interface utilisateur et graphique. Il a vraiment tenté de rendre les algorithmes de plus en plus efficaces par rapport à leur complexité et pour gagner du temps d'exécution.

Ibrahima, a aussi participé à la réflexion et l'implémentation des algorithmes du parcours en largeur et A*. Il a également fait la page d'accueil de l'interface graphique du jeu. C'est également lui qui s'est chargé de prendre des notes à chaque réunion du groupe avec ou sans le professeur encadrant du projet et de la rédaction du rapport du projet.

Enfin, Davy a implémenté une partie du parcours en largeur, l'action de la souris sur la page d'accueil et de documentation Java pour les fonctions.

Nous avons tous contribué à la classe Configuration qui est la base du projet avec l'implémentation des différentes fonctions.
Nous nous sommes répartis les tâches de telle sorte que nous nous sommes impliqués dans chaque partie et chaque algorithme qu'il a fallu travailler dans ce semestre, en nous organisant avec des réunions régulières, en petits binômes, trinômes ou même avec tout le

groupe afin de discuter de nos difficultés, ce qui a été terminé, ou encore fixer les objectifs à atteindre pour la prochaine réunion.