



Licence 2 Informatique

Rapport du projet IF06

## Jeu de Taquin

Réalisé par :

**ARNOULT Simon, MEKHILEF Wissame, OUSSAD Jihad,  
RETY Martin**

15 mai 2015

## Résumé

Nous avons le plaisir de vous présenter notre travail sur ce projet. Durant la deuxième année de la l2 Informatique à l'Université d'Orléans, nous avons travaillé en groupe de quatre sur un projet de résolution de Taquin.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Etude</b>	<b>6</b>
2.1	Analyse de faisabilité . . . . .	6
2.2	Conception UML . . . . .	7
<b>3</b>	<b>La gestion de projet</b>	<b>9</b>
3.1	Le travail de groupe . . . . .	9
<b>4</b>	<b>Phase de développement</b>	<b>11</b>
4.1	L'architecture . . . . .	11
4.2	Les fichiers test . . . . .	12
4.3	Fonction non implémenté . . . . .	12
<b>5</b>	<b>Analyse et Conclusion</b>	<b>13</b>
5.1	Analyse des Benchmark . . . . .	13

---

5.1.1	Efficacité des fonctions du Taquin . . . . .	13
5.1.2	Efficacité des algorithmes . . . . .	14
5.1.3	Etude sur l'ensemble incomplet . . . . .	15
5.2	Conclusion . . . . .	16
<b>6</b>	<b>Resources utilisées</b>	<b>17</b>

# Table des figures

2.1	Représentation UML du package jeu . . . . .	7
2.2	Représentation UML du package algo . . . . .	8
2.3	Représentation UML du package automate . . . . .	8
3.1	Dépôt sur GitHub . . . . .	10
5.1	Efficacité des fonctions du Taquin . . . . .	14
5.2	Diagramme des temps d'exécution . . . . .	15
5.3	Exemple . . . . .	15

# Chapitre 1

## Introduction

Nous allons ensemble, aborder quatre points principaux dans ce rapport :

- L'étude du projet
- L'organisation de travail au sein du groupe
- Le développement du code
- L'analyse des algorithmes.

# Chapitre 2

## Etude

L'étude du projet fut une première étape importante pour se mettre dans une bonne dynamique de groupe. Dès la première semaine nous avons réalisé une version jouable du Taquin, puis nous avons travaillé sur cette version durant toute la première phase d'essais.

Rapidement il nous a fallu faire des essais sur ce jeu pour qu'il puisse se résoudre algorithmiquement. Nous avons chacun travaillé de notre côté sur nos idées pour pouvoir en tester un plus grand nombre, mais tout en restant en contact régulièrement pour avancer ensemble.

Cette manière de répartir les tâches nous a permis de se rendre compte des difficultés que l'on allait rencontrer plus rapidement.

### 2.1 Analyse de faisabilité

L'analyse de faisabilité fut un moment qui a duré du début du projet jusqu'aux vacances de février, durant cette phase chacun travaillant sur des fonctionnalités différentes, nous avons pu voir ou étaient les problèmes dans notre architecture de départ.

Ces tests ont été très divers. Nous avons travaillé sur les fonctionnalités VT100 du terminal et la récupération des touches tapées par l'utilisateur. Mais nous avons aussi créé des algorithmes pour essayer de résoudre le Taquin.

Tous ces tests nous ont permis au mois de février d'avoir une idée claire de l'architecture du projet.

## 2.2 Conception UML

Cette analyse fin février a permis d'aboutir au diagramme UML suivant, diagramme qui n'a pas beaucoup évolué jusqu'à la version finale. Pour faciliter la lecture nous avons divisé l'architecture en packages, seulement 3 des packages sont présentés.

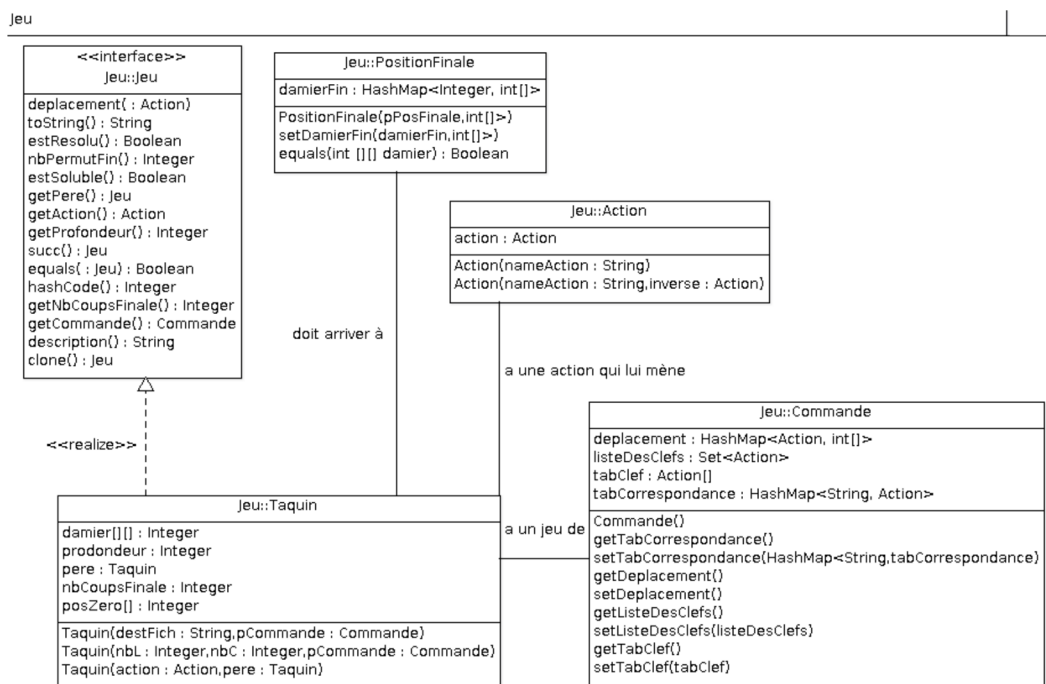


FIGURE 2.1 – Représentation UML du package jeu



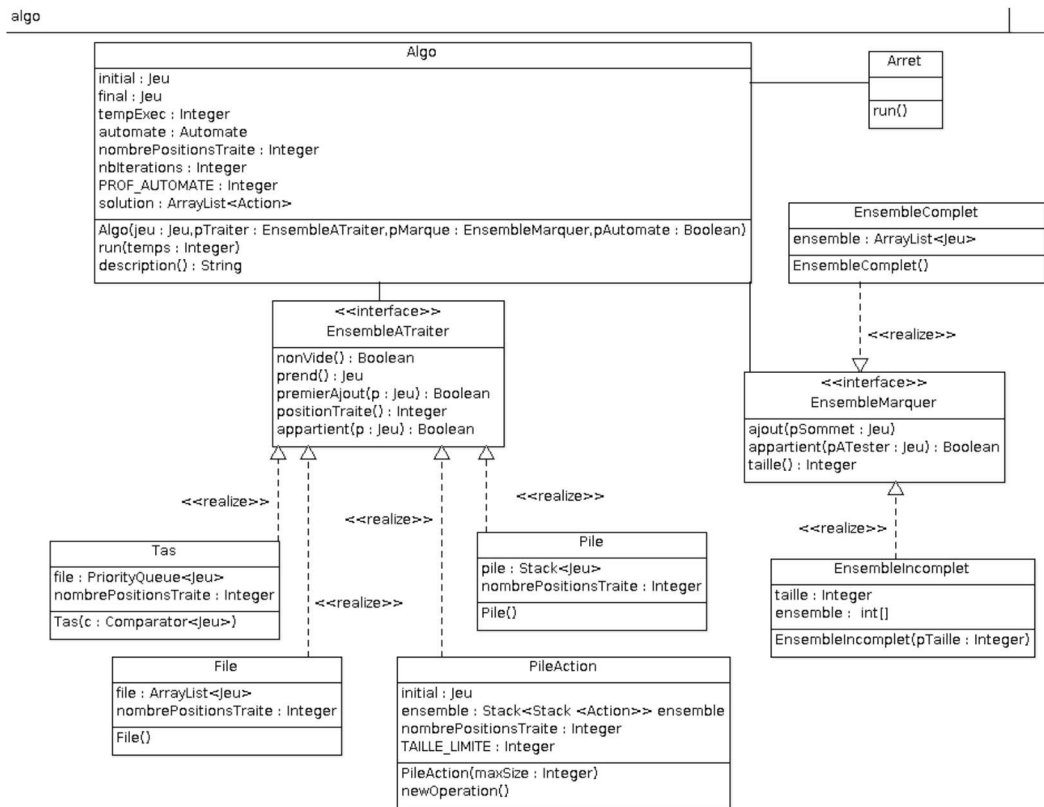


FIGURE 2.2 – Représentation UML du package algo

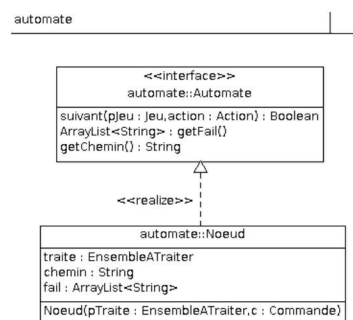


FIGURE 2.3 – Représentation UML du package automate

# Chapitre 3

## La gestion de projet

La gestion de projet est à la base de tout projet, et encore plus quand il se fait avec une équipe de quatre personnes.

### 3.1 Le travail de groupe

La gestion de projet fut au coeur de nos préoccupations avant même que le projet ne démarre, nous avons cherché à créer une équipe dynamique. Nous nous sommes donc vus régulièrement dans les salles de l'Université. Malgré cela il nous a fallu mettre en place des moyens dédiés pour faciliter le travail et éviter une dégradation de l'entente.

Dès la première semaine, nous avons mis en place un dépôt git sur le site de l'hébergeur GitHub, ce dépôt nous a permis d'avoir le réflexe de l'utiliser même si nous avons rencontré quelques problèmes, il nous a permis à chacun de voir l'avancement du projet. Le dépôt est disponible à cette adresse : <https://github.com/wissame95/IF06-Projet.git>.

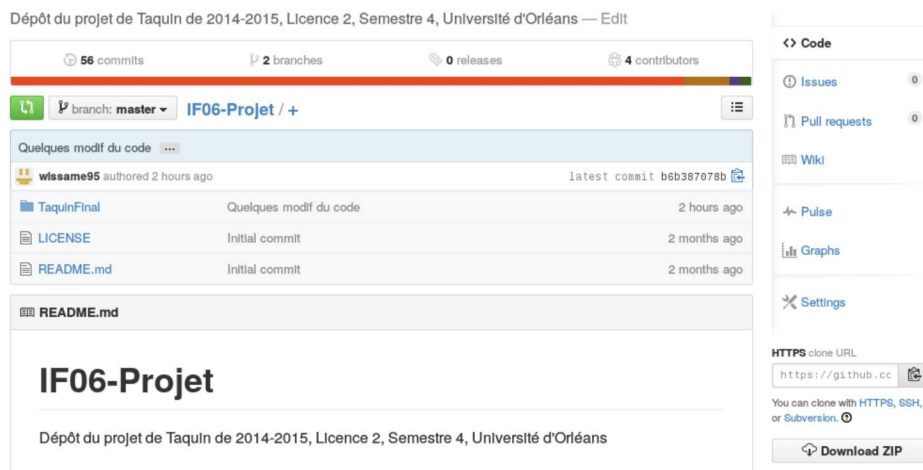


FIGURE 3.1 – Dépôt sur GitHub

Cependant un tel dépôt ne répond pas à la question de la communication, chacun habitant une ville différente, nous avons donc communiqué via Skype pour remédier à ce problème.

# Chapitre 4

## Phase de développement

Pour la phase de développement nous avons chacun de notre côté développé une partie de l'application. Nous nous sommes partagés les différents algorithmes, mais aussi les autres fonctionnalités comme les tests JUnit.

### 4.1 L'architecture

Nous avons fait le choix de répartir dans différents packages les classes et interfaces. Nous avons eu des choix à faire à plusieurs niveaux. Par exemple un choix simple pour la grille de jeu, nous avons choisi une matrice d'entier en remplacement d'une ArrayList d'entiers. Mais aussi des choix plus compliqués, comme la représentation d'un sommet qui dans notre cas est la classe Taquin.

L'architecture se décompose en 7 packages :

**algo** Contient les interfaces EnsembleATraiter et EnsembleMarque, toutes les classes implémentant ces interfaces et une classe Algo.

**jeu** Contient une interface Jeu, la classe Taquin. Mais aussi les commandes, les actions, et les positions finales.

**comparateur** Contient deux comparateurs Manhattan et DepthManhattan.

**exceptions** Contient toutes les exceptions que nous avons dû créer.

**automate** Contient les classes relatives à l'automate.

**junit** Contient les différents tests JUnit.



**main** Contient une seule classe, Main. Elle gère la lecture des paramètres et l'exécution des méthodes dans les autres classes, c'est le lien entre l'utilisateur et l'application.

## 4.2 Les fichiers test

Rapidement pour vérifier le fonctionnement du programme nous avons dû créer des fichiers tests de taquins témoins. Ces fichiers se trouvent dans le dossier taquin à la racine du projet. Nous en avons utilisés un petit nombre, avec des tailles différentes et "bien mélangés".

## 4.3 Fonction non implémenté

Nous avons rencontré des soucis sur l'implémentation de la pile d'action, le problème intervient au moment de remonter les parents d'un Taquin, il vient certainement d'un null. Ceci a donc engendré le fait de ne pas pouvoir tester nos méthodes sur le parcours progressif. Enfin, nous avons aussi rencontré des soucis de lecture des Junit et benchmarks dans une classe interne.

# Chapitre 5

## Analyse et Conclusion

### 5.1 Analyse des Benchmark

Les benchmarks fournis par le module h2, nous ont permis d'améliorer l'efficacité de l'application, mais aussi de montrer le bon fonctionnement des méthodes. Nous avons donc pour Algo et pour Taquin créés deux classes, une prouvant que le programme tourne correctement et l'autre pour montrer la rapidité d'exécution avec un problème de plus en plus grand. Nous avons aussi écrit un test pour l'ensemble incomplet pour observer la résolution et le temps nécessaire à celle-ci, en fonction de la taille de l'ensemble utilisé.

#### 5.1.1 Efficacité des fonctions du Taquin

Ci-dessous vous pouvez voir un récapitulatif des fonctions coûteuses de la classe Taquin.

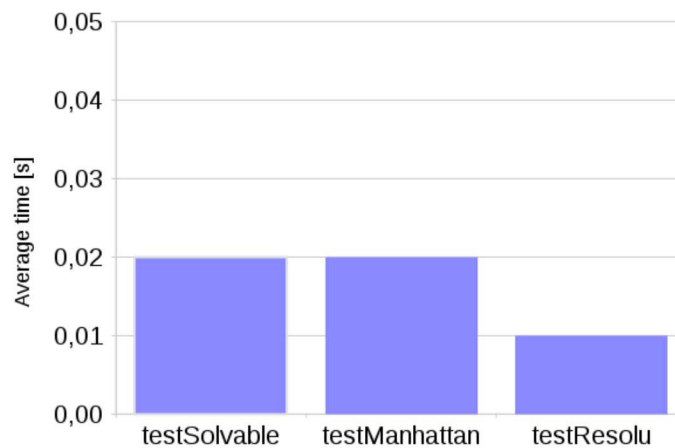


FIGURE 5.1 – Efficacité des fonctions du Taquin

### 5.1.2 Efficacité des algorithmes

Les tests suivants ont été réalisés à l'aide de `taq1.taq` qui est un taquin de taille 3x3. On peut observer les différentes vitesses de résolution pour le même Taquin en fonction de l'algorithme.

Les algorithmes utilisant un ensemble complet sont plus lent que les algorithmes utilisant un ensemble incomplet. De même on observe que les tas sont plus rapides que les files et piles, qui demandent le plus de temps de calcul.

Les algorithmes basés sur un automate sont les plus efficaces, cependant l'efficacité dépend de la profondeur de l'apprentissage de l'automate et de la taille du taquin.

En effet, le nombre de coups redondants grandit en même temps que la taille du taquin, il est donc préférable d'utiliser cet automate sur un taquin de grande taille.

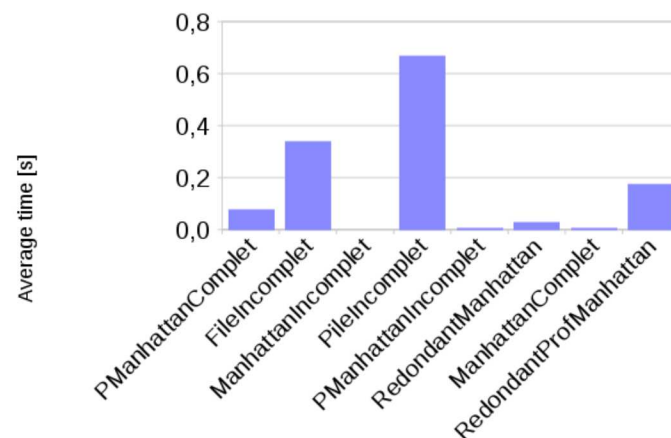


FIGURE 5.2 – Diagramme des temps d'exécution

### 5.1.3 Etude sur l'ensemble incomplet

Ci-dessous vous pouvez voir un diagramme montrant le temps nécessaire à la résolution d'un taquin avec un ensemble incomplet sur 5 tailles différentes.

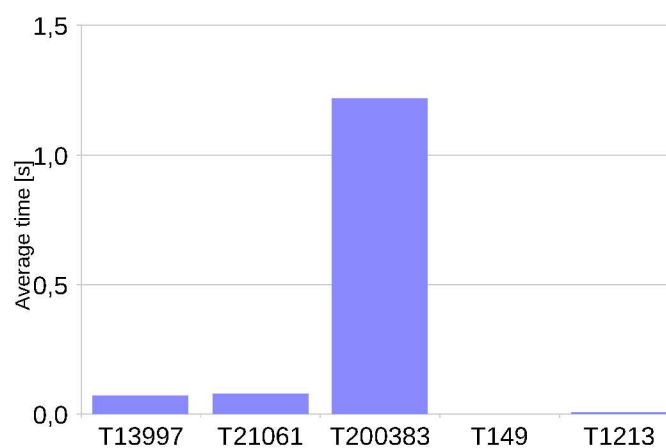


FIGURE 5.3 – Exemple





## 5.2 Conclusion

En définitive, ce projet nous a permis d'acquérir une certaine autonomie, puisque nos simples connaissances ne suffisaient pas à implémenter efficacement tous les aspects du projet. Nous avons également dû faire preuve de rigueur et d'inventivité afin de contourner certaines difficultés inhérentes à la complexité algorithmique du projet.

# Chapitre 6

## Resources utilisées

- JUNIT
- JAVA
- Dia
- Eclipse
- Argouml
- L<sup>A</sup>T<sub>E</sub>X(sous kile)
- Google Drive
- Kate
- Git, GitHub