

# JEU DE TAQUIN

*et le programme permettant de résoudre les jeux de taquin*

## Informations

### RAPPEL DU SUJET :

L'objectif de ce projet est de réaliser un programme permettant de résoudre les jeux de taquin.

Le taquin est un jeu qui consiste à remettre des cases dans le bon ordre : les cases sont numérotées de 1 à  $N^2 - 1$  et elles sont disposées dans un carré  $N \times N$ . La case vide permet de bouger les cases adjacentes par déplacement horizontal ou vertical. Dans la figure ci-dessous, on présente un exemple d'un jeu de taille  $N = 4$  :

10	14	5	4
3	6		15
9	1	11	12
13	2	8	7

Configuration initiale

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Configuration finale

Le programme à réaliser devra être capable de générer des configurations aléatoires de taquin solubles et de trouver une méthode pour les résoudre. On s'intéresse ici aux jeux de taquins de taille  $N \times N$ .

### LISTE DES MEMBRES DU PROJET

L2, Mathématique-Informatique, Groupe 1 : Mathieu AFONSO PECADO (21955324), Tara AGGOUN (21961069), Jasmin JOKSIMOVIC (21957855), Elody TANG (21953199).

**Ce qu'il faut lire :**

**PRÉSENTATION - 2**

**LISTE DES  
FONCTIONNALITÉS - 4**

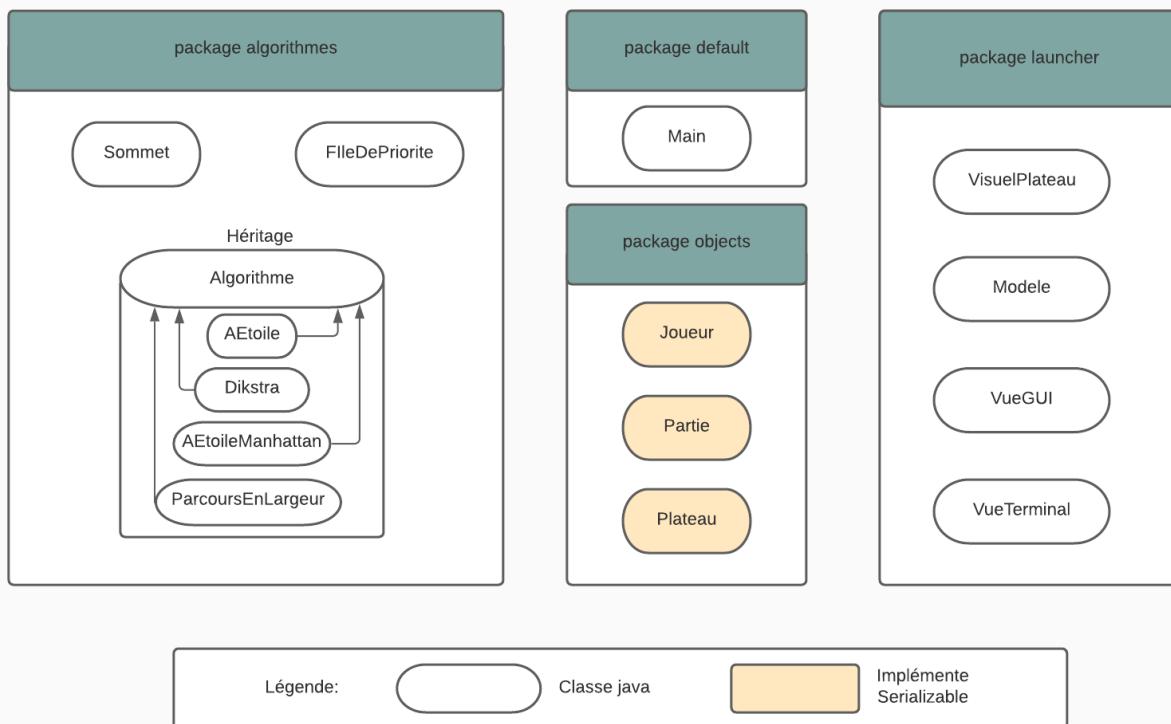
**STATISTIQUES ET  
COMPARAISON - 8**

**MÉTHODE DE TRAVAIL - 9**

**NOTRE PROJET EN IMAGE  
- 10**

# Présentation du projet

## SCHÉMA DES CLASSES JAVA ET DES DOSSIERS PRÉSENTS



## DESCRIPTION DES PACKAGES

### PACKAGE DE DÉFAUT (DEFAULT)

Pour lancer un programme, il faut déjà avoir une classe principale. Dans notre projet, la classe principale est la classe **Main**. Elle contient également d'autres fonctions qui aident au bon lancement du jeu.

Voici un petit rappel des commandes principales (que vous pouvez retrouver dans le README.md) :

**Pour compiler :** `javac Main.java`

**Pour exécuter sur l'interface graphique :** `java Main GUI`

**Pour exécuter sur le terminal :** `java Main terminal`

### Pour voir les statistiques et tester :

```
java Main --algo=[...] --taille=[...] --niveau=[...]
```

*o Arguments possibles dans "--algo=[...]" :*

*"pl"* : pour le parcours en largeur, *"dj"* : pour l'algorithme de Dijkstra, *"astar"* : pour l'algorithme A\* (avec le nombre de cases mal placées comme heuristique), *"astarManhattan"* : pour l'algorithme A\* (avec la distance de Manhattan comme heuristique), *"all"* pour tous les algorithmes

*o Arguments possibles dans `--niveau=[...]` :*

*"1"* : Niveau facile, *"2"* : Niveau moyen, *"3"* : Niveau difficile, *"4"* : Mode aléatoire

## PACKAGE OBJECTS

### "À quoi correspond ce package ?"

Les premières choses qu'on a programmé sont les classes **Partie** et **Plateau**, qui représentent respectivement une partie de jeu et le plateau du jeu. Un objet de type **Partie** est composé de plusieurs attributs : le nombre de mouvements, le plateau en cours de jeu (le plateau où les déplacements se font au fur et à mesure), le plateau initial. Un objet typé **Plateau** est composé d'un tableau à une dimension qui représente la permutation du jeu et d'une taille N. (Le tableau est d'une taille N<sup>2</sup>.)

10	14	5	4
3	6		15
9	1	11	12
13	2	8	7

#### *Exemple :*

#### *Configuration H*

*Taille = 4*

*Tableau = [10, 14, 5, 4, 3, 6, 0, 15, 9, 1, 11, 12, 13, 2, 8, 7]*  
*(le 0 représente le trou)*

La classe **Joueur** a été ajoutée bien après ! Elle représente les meilleures parties du joueur : à chaque fois qu'une partie est terminée, le programme va comparer le score de la partie terminée à celui qui est enregistré en tant que "meilleur score" et va enregistrer la meilleure partie.

## PACKAGE ALGORITHMES

Le package **Algorithmes** contient toutes les classes décrivant un algorithme précis et toutes les classes qui sont utilisées dans ces dernières comme les classes FiledePriorite et Sommet.

### "Qui hérite qui ?"

La classe **Algorithme** est la classe mère de toutes les autres classes qui représentent un algorithme précis (**ParcoursEnLargeur** ("pl"), **Dijkstra** ("dj"), **AEtoile** (astar) et **AEtoileManhattan** (astarM)). Ainsi les quatre classes citées précédemment héritent donc de **Algorithme** puisque ce sont des algorithmes.

La classe **FileDePriorite** désigne une file de priorité où les valeurs ayant la plus petite clé est extraite en première. La classe **Sommet** représente un sommet du graphe qu'on veut analyser, et représente la valeur (le plateau), son père et son nombre de mouvements.

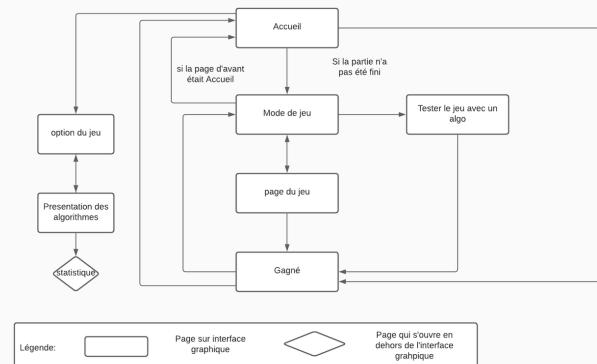
## PACKAGE LAUNCHER

Le package **Launcher** est le dernier package qu'on a ajouté au projet. L'interface graphique et l'interface terminal sont programmés dans cet package.

*Classes utilisées pour l'interface graphique : VueGUI, VisuelPlateau*

La classe **VueGUI** hérite de la classe **JFrame**, et donc désigne la fenêtre de l'interface graphique. Elle est composée d'une CardLayout qui change de vue à chaque fois qu'on change de fenêtre. La classe **VisuelPlateau** désigne le visuel du plateau : c'est dans cette classe où on programme les animations du robot et les déplacements des cases lorsque le joueur joue.

Voici le cheminement des différentes pages :



*Classe utilisée pour l'interface terminal : VueTerminal*

*Classe utilisée pour les deux deux interfaces : Modele*

La classe **Modele** est la classe qui "gère" les données de la partie. Toutes les sauvegardes passent par cette classe : c'est cette classe qui gère le bon fonctionnement de la vue graphique.

# Liste des fonctionnalités

## FONCTIONNALITÉS DEMANDÉES ET RÉALISÉES

Comme demandé, notre programme est capable de générer des configurations aléatoires de taquin et de les résoudre à l'aide de 3 algorithmes différents : le parcours en largeur, Dijkstra et A\* avec deux heuristiques (la distance de Manhattan et le nombre de cases mal placées).

De plus, l'utilisateur peut choisir la taille du taquin qu'il veut jouer.

### **Exemple avec un jeu de taille 2 (voir image sur la gauche)**

Notre programme nous donne aussi le nombre d'éléments dans la file, le nombre d'éléments dans l'ensemble, le nombre de mouvements et le temps d'exécution en millisecondes.

Il donne aussi le cheminement à suivre pour arriver à la solution.

```
EXEMPLE D'APPLICATION :
Nombre d'elements dans la file : 6
Nombre d'elements dans l'ensemble : 6
Nombre de mouvements : 2
Temps en millisecondes : 2

Mouvement 0 :
0 1
3 2
Il faut bouger la case 1.

Mouvement 1 :
1 0
3 2
Il faut bouger la case 2.

Mouvement 2 :
1 2
3 0
```

## FONCTIONNALITÉS RÉALISÉES MAIS PAS DEMANDÉES (EXTENSION)



Durant le semestre, on a eu le temps d'implémenter différentes fonctionnalités qui ne sont pas obligatoires. Tout d'abord, on a implémenté **un jeu praticable par l'utilisateur** et une nouvelle façon de créer un taquin grâce à des permutations, ce qui nous a permis de **créer 4 différents niveaux** :

- les niveaux 1, 2 et 3 : le jeu est généré avec la méthode suivante, on prend la configuration initiale, on déplace une case jouable M fois (M choisi en fonction de la difficulté choisie, plus la difficulté est grande, plus le nombre de permutations est grand) et on retourne le jeu.
- le niveau 4 : le jeu est généré aléatoirement et ensuite, le programme teste si le jeu généré est soluble.



On a aussi implémenté **une interface graphique sur laquelle le joueur peut interagir**. Toutes les pages de cette interface sont **ajustables à la taille que l'utilisateur souhaite**.

Si le joueur le souhaite, pendant une partie en cours, il peut faire **un appel au robot qui va utiliser l'algorithme choisi sur la page des paramètres**. Le robot, lui, bouge la case qu'il pense être le meilleur coup. Vous pouvez aussi remarquer qu'il y a des animations pour faire bouger les cases. Le joueur **peut aussi tester l'algorithme** qu'il veut avec le bouton "Tester l'algo".

## LISTE DES FONCTIONNALITÉS

Sur notre projet, on a aussi la possibilité d'**importer un plateau et d'y jouer sur l'interface graphique**.

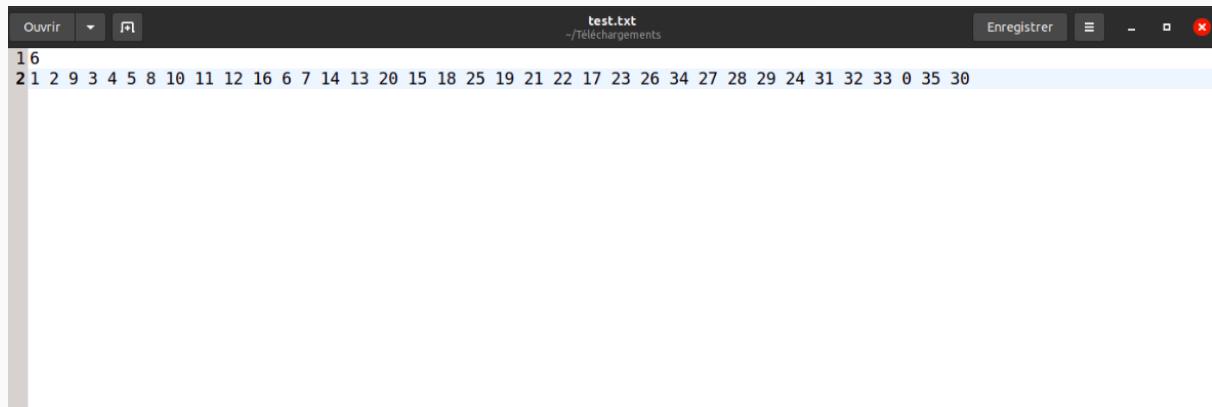
Pour respecter correctement l'écriture et la syntaxe de nos fichiers, nous vous proposons un petit tutoriel.

- Ouvrez un fichier de type .txt.
- Écrivez sur la première ligne la taille de votre plateau.
- Écrivez sur la deuxième ligne les chiffres dans l'ordre espacé avec des espaces.

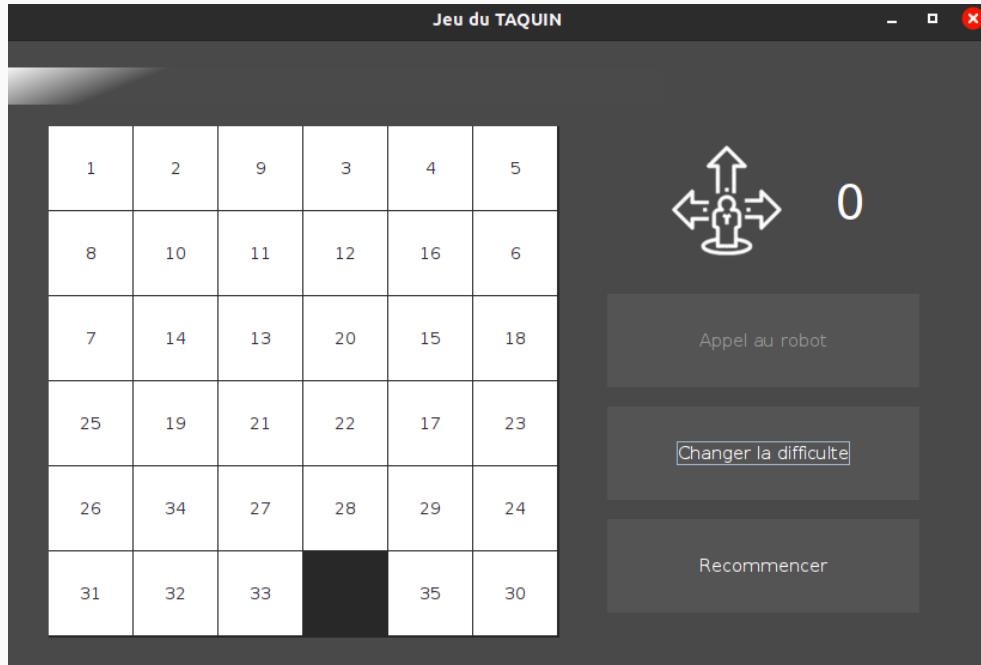
**Attention :** le nombre de chiffres écrits sur la deuxième ligne doit être égale à la taille \* la taille du chiffre écrit sur la première ligne.

**Attention 2 :** Le 0 signifie le trou, n'oubliez pas de le placer.

*Exemple (pour un plateau de taille 6) : (voir les images ci-dessous)*



*Présentation du fichier .txt*



*Résultat sur l'interface graphique*

Pour **tester cette fonctionnalité**, nous avons importé un fichier 'TEST - IMPORTATION.txt' dans le dossier principal. Vous pouvez l'utiliser si vous le souhaitez.

## LISTE DES FONCTIONNALITÉS

Sur notre projet, vous pouvez aussi retrouver des **fichiers PDF qui récapitulent les statistiques de chaque algorithme**. Vous pouvez les aussi retrouver sur l'interface graphique sur la page "Présentation des algorithmes" en appuyant sur le "+".

Parcours en Largeur (sur un jeu de taille 3 sur 20 essais)				
Mode Aléatoire				
Nom du test	Nb de configurations mises dans la file	Nb de configurations mises dans l'ensemble	Nb de mouvements trouvés	Temps en millisecondes
1	85293	85293	21	1377
2	109493	109493	21	1193
3	134933	134933	23	1702
4	181438	181438	29	2743
5	200289	200289	24	1782
6	151454	151454	24	1991
7	138394	138394	23	1750
8	118724	118724	22	1433
9	58795	58795	20	435
10	176453	176453	26	2454
11	177497	177497	20	2447
12	124824	124824	22	1558
13	53287	53287	23	1731
14	132233	132233	23	1731
15	140190	140190	24	1640
16	100171	100171	22	1118
17	8072	8072	24	731
18	56409	56409	19	592
19	99864	99864	22	1116
20	9320	9320	21	1053
MOYENNE	107227.45	107227.45	23.15	1413.75

Difficulté 1 (20 permutations)				
Nom du test	Nb de configurations mises dans la file	Nb de configurations mises dans l'ensemble	Nb de mouvements trouvés	Temps en millisecondes
2	490	131	4	46
3	490	121	6	99
4	31450	121	4	12
5	20453	1145	12	1183
6	16299	16299	12	455
7	417	417	6	5
8	522	522	6	7
9	5545	5545	10	64
10	5493	5493	10	104
11	4652	4652	10	67
12	24	24	2	1
13	105366	105366	14	14
14	582	582	6	63
15	13119	13119	8	21
16	138	138	4	3
17	24	24	2	0
18	368511	368511	16	5961
19	365	365	6	4
20	136	136	4	61
MOYENNE	28861.2	28861.2	7.7	426.9

*Statistiques pour l'algorithme Parcours en Largeur pour les tailles 3 et 4 au niveau 1*

Nous avons aussi implémenté deux types de sauvegarde. La première est **la sauvegarde de partie** si vous quittez la partie en cours sans l'avoir terminé. Quand vous reviendrez, vous pourrez la récupérer en cliquant sur le bouton "Restaurer la partie en cours" sur la page d'accueil. La deuxième est visible sur la page "Option du jeu", **le meilleur nombre de coup pour chaque taille de Taquin est stocké**.

## INFORMATIONS SUPPLÉMENTAIRES

*"Avez-vous fait tout ce que vous aviez prévu de faire au début du projet ?"*

Initialement, on avait prévu de faire **un autre algorithme en tant qu'extension, basé sur les combinatoires**. Par manque de temps, on a préféré passer plus de temps sur le code déjà programmé pour perfectionner ce qu'on avait déjà implémenté à la place de faire un nouvel algorithme.

Ensuite, nous avions prévu de faire une interface graphique différente. Il peut être intéressant de voir ce que nous avions prévu de faire et ce que nous avons fait :



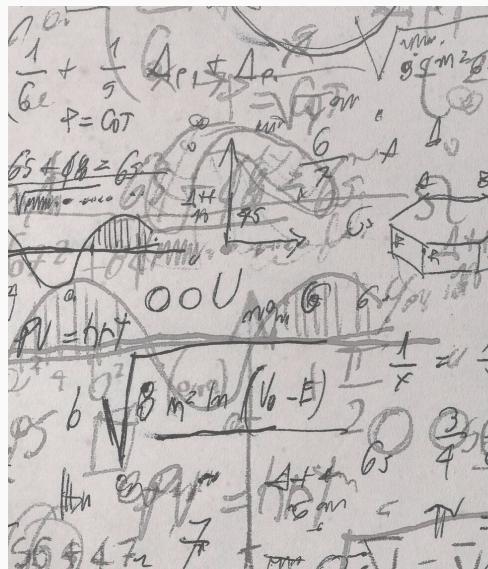
*idées initiales pour le design de l'interface graphique (AVANT)*



*Interface graphique finale (APRÈS)*

# Statistiques et comparaison

## CHIFFRES ET ALGORITHMES



Les statistiques d'un algorithme sont mesurées par la durée que prend ce dernier pour résoudre un jeu.

*"Comment avez-vous fait pour estimer le temps d'exécution de vos algorithmes ?"*

Pour déterminer le temps d'exécution de nos algorithmes, nous avons réalisé 20 générations de jeux pour les tailles 3, 4 et 5 sur tous les niveaux. Puis ensuite, rajouter cette ligne :

```
long debut = System.currentTimeMillis();
(Exécution du programme)
System.out.println(System.currentTimeMillis()-debut);
```

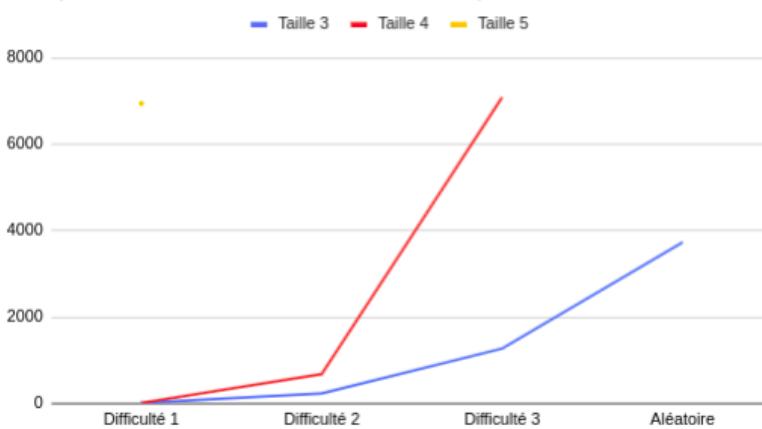
Et on récupère les données !

*"Quel est l'algorithme le plus rapide ?"*

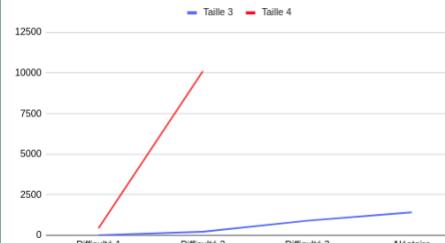
Pour répondre à cette problématique, nous avons récupérer les moyennes des durées de résolution pour chaque taille et chaque niveau de jeux pour chaque algorithme codé.

Ainsi, nous avons pu en déduire que l'algorithme le plus performant de notre projet est l'algorithme A\*, avec une heuristique choisie : la distance de Manhattan et le nombre de déplacements. (C'est aussi le seul algorithme qui a réussi à résoudre des jeux de taille 5.)

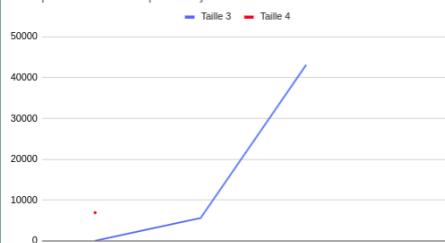
Temps de l'exécution en millisecondes pour A\* Manhattan



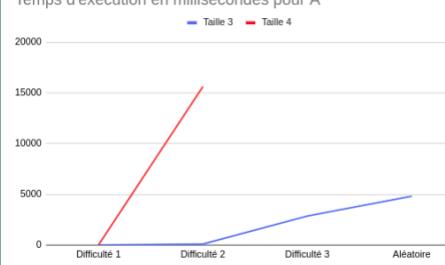
Temps d'exécution en millisecondes du ParcoursEnLargeur



Temps d'exécution pour Dijkstra en millisecondes



Temps d'exécution en millisecondes pour A\*



# Méthode de Travail

## ORGANISATION

Pour finaliser le rapport de notre projet, il nous a semblé important de montrer comment notre groupe s'est organisé et s'est réparti le travail pendant toute la durée impartie.

Le vendredi, nous avions le rendez-vous hebdomadaire avec notre professeur, François Laroussinie à 8h45. Après les instructions et les conseils que nous donnait notre professeur, nous nous appelions sur Discord pour se répartir les tâches à faire pour la semaine d'après.

Nous découpons notre groupe en deux sous-groupes, composés de deux personnes. Et chaque binôme pouvait choisir la tâche qu'il voulait.

Nous essayons de terminer les tâches pour le lundi, et nous rapportions les avancées par mail à notre professeur avant le mercredi soir pour qu'il puisse se préparer à notre rendez-vous hebdomadaire.



## RÉSOLUTION DES PROBLÈMES RENCONTRÉS

### "Avez-vous rencontré des problèmes ?"

Oui, nous avons eu des problèmes ! La première semaine, nous ne savions absolument pas vers où nous diriger. Nous ne savions pas ce qu'il fallait faire pour bien débuter, nous ne savions ni ce qu'il fallait chercher, ni à quoi notre rendu allait ressembler.

Alors, pour remédier à ça, pendant toute la première semaine, nous avons fait de longues recherches sur les différents algorithmes et les différentes façons de coder un jeu de taquin. Nos premiers recherches nous ont permis de comprendre ce qu'on attendait à la fin de notre projet et nous a lancé dans la réalisation des premières lignes de code.

**RAPPEL : 0 signifie le "trou".**

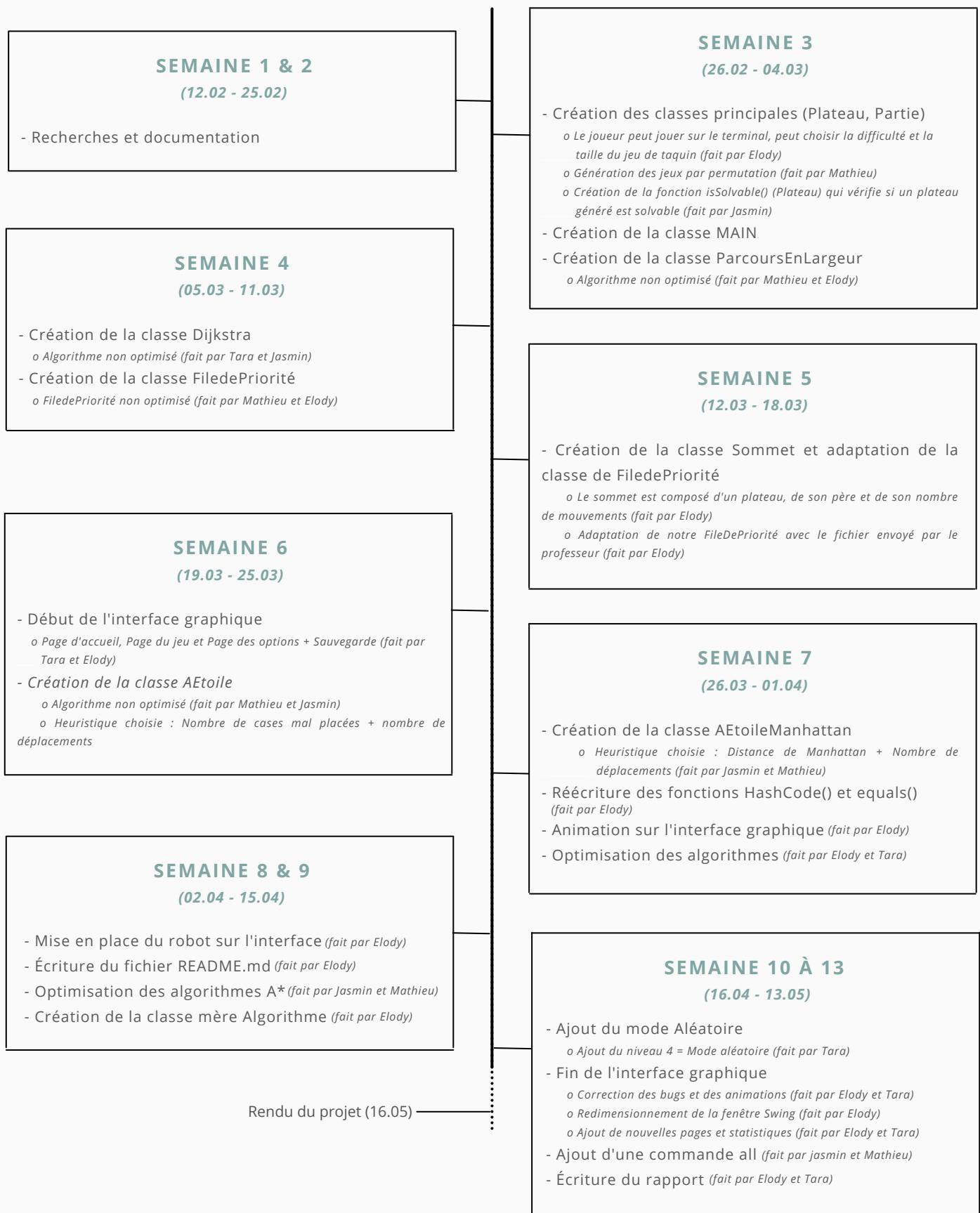
4	1	6	3
7	11	10	8
2	5	0	12
13	14	9	15

*Affichage du jeu sur le terminal (27.02.2021)*

Pendant la réalisation des algorithmes, nous avons rencontré un deuxième problème : nous n'avions pas l'habitude de lire du pseudo code. Il nous a fallu plusieurs jours de compréhension et de réflexion pour pouvoir écrire correctement les algorithmes voulus.

## MÉTHODE DE TRAVAIL

### ORGANISATION ET PLANIFICATION DES TÂCHES



# Notre projet en image

## CAPTURES D'ÉCRAN



"Après tout, c'est un jeu de taquin, j'ai le droit de vous taquiner, haha !"

- François Laroussinie  
(16.04.2021) -