Ella GADELLE, Manon HINET, Mathis TEMPO, Lucas RIGOT

Ing1 GI Groupe 2

Rapport
Projet JAVA

Sujet : CY-Slide

Tuteur : Mohamed HADDACHE

CY TECH - Cergy Paris Université
Année 2022 - 2023

SOMMAIRE

I. Introduction

L'objectif de ce projet est d'implémenter le jeu du taquin en *Java*. Ce jeu doit posséder différents niveaux et la possibilité de mélanger les cases de deux manières :

- mélanger aléatoirement les cases du jeu en ayant la possibilité que le niveau n'aie plus de solution

- mélanger le niveau en déplaçant une à une des cases vers des emplacements disponibles aléatoirement.

L'utilisateur résout ensuite les niveaux dans l'ordre, sa progression étant sauvegardée avec son nombre de coups minimum à chaque succès. L'utilisateur doit aussi avoir la possibilité de voir la résolution automatique du niveau.

II. L’équipe

a. Organisation

Nous avons commencé par séparer l’implémentation du code principal, de l’interface graphique et de l'algorithme de résolution du jeu du Taquin. Lucas et Manon ont codé les fonctionnalités de base en *Java*, Mathis s’est occupé de l’interface graphique avec *JavaFX* et Ella a implémenté l'*algorithme A** adapté au jeu. Chacun a fait sa partie de code de son côté et nous avons fait des appels réguliers entre nous sur *Discord* pour voir l’avancement de chacun. Nous avons aussi eu des réunions tous les 2-3 jours avec notre tuteur Mohamed HADDACHE pour montrer notre avancement et poser des questions sur des points qui nous posaient problème. Une fois que nous avions assez avancé dans nos parties respectives, nous avons commencé au courant de la deuxième semaine de projet à fusionner les bouts de code et à voir ensemble comment les adapter pour arriver au jeu final. Nous avons commencé le rapport et la documentation à la fin de la deuxième semaine de projet.

Pour partager le code entre nous, nous utilisons *Discord, Teams* et *GitHub*. Et pour coder nous avons utilisé *IntelliJ IDEA*.

b. Planning

15/05	16/05	17/05	18/05	19/05	20/05	21/05
Réunion de l'équipe :	**Commencement :**	**Réunion avec notre tuteur :**	**Avancement :**	**Réunion avec notre tuteur :**		
Compréhension du sujet	Fonctionnalités de base du jeu	Partage en détails des tâches	Interface graphique élémentaire	Modification du diagramme de classes		
Partage des tâches	Apprentissage du fonctionnement de JavaFX	Partage du diagramme de classes	Fonctionnement des fichiers en Java	Mélange de la grille de jeu		
Début de conception	Diagramme de classes	Partage du code de base	Mélange de la grille de jeu			
			Affichage de la grille dans le terminal			
		Réunion de l'équipe :		**Réunion de l'équipe :**		
		Diagramme de classes		Début de l'implémentation de l'algorithme		
		Recherche d'algorithme				
		Fonctionnement des fichiers en Java				
		Compréhension GitHub				
22/05	**23/05**	**24/05**	**25/05**	**26/05**	**27/05**	**28/05**
Réunion avec notre tuteur :	**Réunion de l'équipe :**	**Réunion avec notre tuteur :**	**Réunion de l'équipe :**	**Réunion avec notre tuteur :**	**Avancement :**	**Finalisation :**
Partage de l'interface graphique	Fusion des fonctionalités de base avec l'interface graphique	Partage du code de l'algorithme	Essayer de réussir à déplacer les pièces sur l'interface graphique	Questions pour l'agorithme	Rapport	Commentaires et de la JavaDoc
Partage du début de l'algorithme	Problèmes avec l'algorithme	Partage de l'interface graphique reliée au code	Implémentation de la deuxième fonction pour mélanger la grille	Début du rapport	Commentaires	Rapport
				Déplacement des pièces du jeu (code sans visuel)		ReadMe
						Mise au propre du code
	Avancement :		**Avancement :**	**Réunion de l'équipe :**	**Finalisation :**	
	Algorithme		Diagramme de classes	Algorithme	Fonctions mélange du jeu	
	Interface graphique		Algorithme	Déplacement des pièces du jeu (visuel)	Diagramme de séquence	
			Interface graphique	Mise à jour du meilleur score	Diagramme des cas d'utilisation	

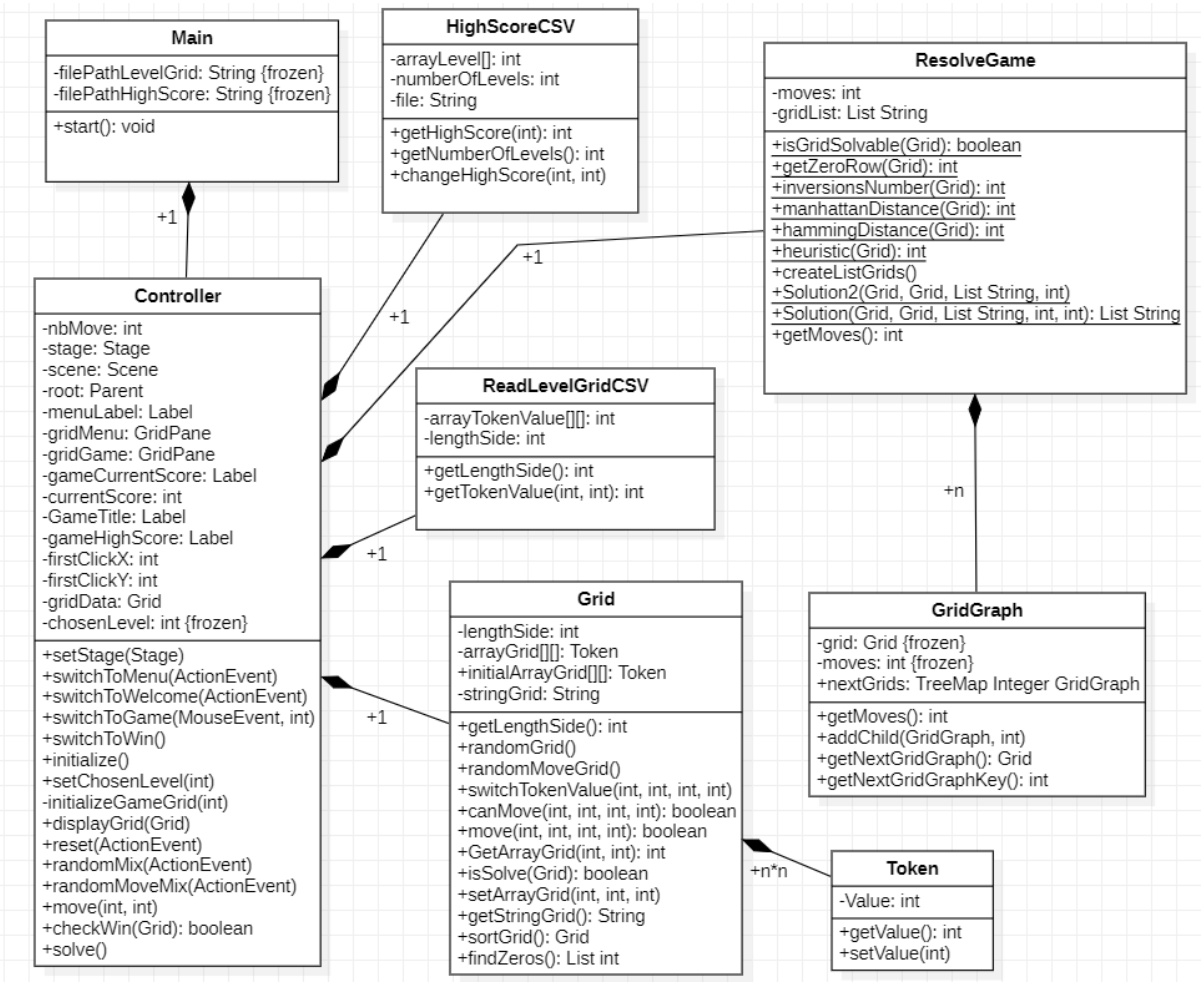
c. Problèmes rencontrés

Concernant les diagrammes que nous devions réaliser, notre tuteur nous a conseillé de faire le diagramme de classes dès le début mais il était compliqué pour nous de le faire avant d'avoir codé le jeu car nous avions du mal à visualiser la structure finale de notre projet. Nous avons donc modifié plusieurs fois au fur et à mesure de l'avancement du projet.

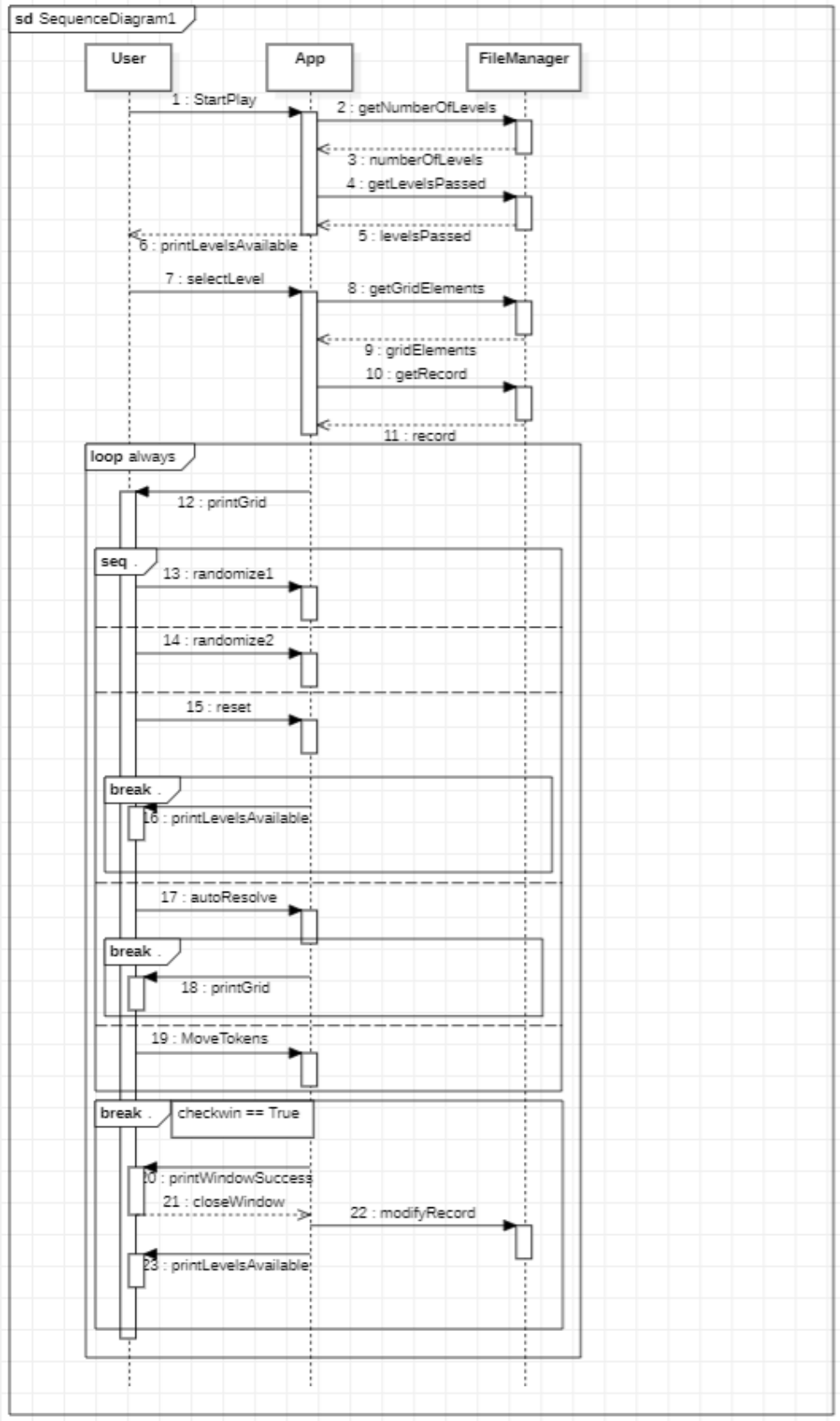
Il est aussi compliqué pour qu'après le mélange des cases, toutes les cases ne soient pas à leur position initiale et que la grille aie une solution (algorithme non terminal).

III. Code principal

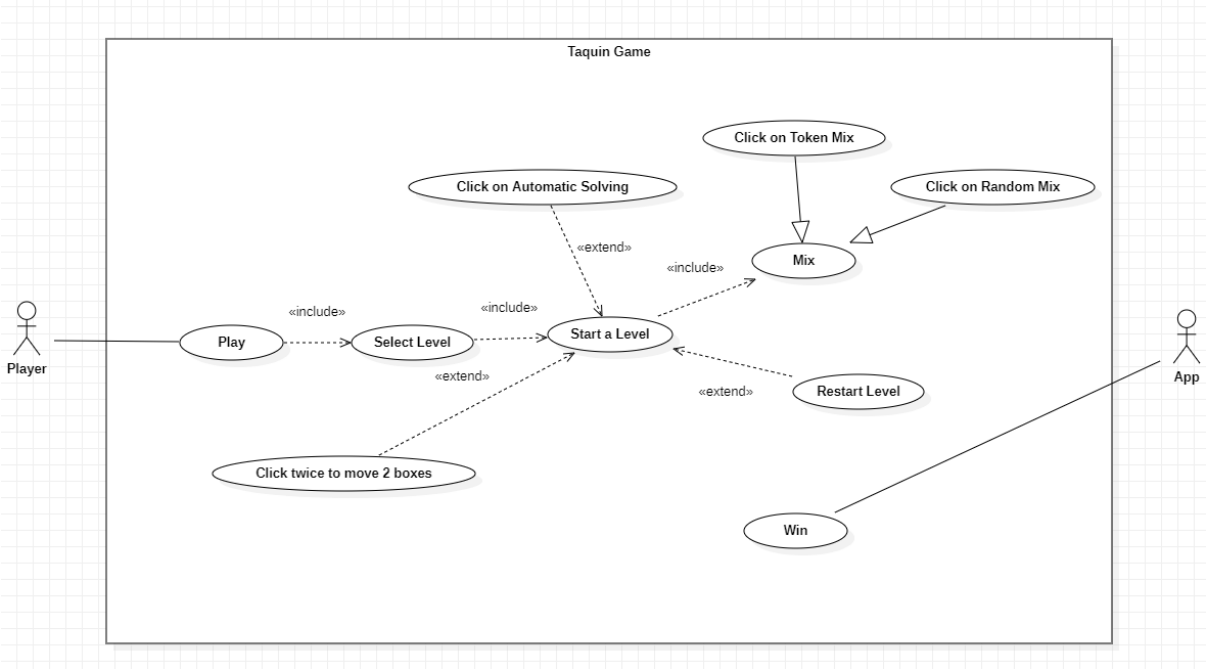
a. Diagramme de classes



b. Diagramme de séquence



c. Diagramme de cas d'utilisation



d. Limites du projet

Pour afficher les changements dans la grille de jeu, il est nécessaire de passer à une nouvelle scène. Cela signifie que le changement de la grille est instantané lorsqu'elle est affichée. Toutefois, nous avons tenté de mettre en place un affichage dynamique qui permettrait de voir le déplacement des cases. Une des solutions que nous avons envisagées était de créer une *TranslateTransition* dans le contrôleur au moment de l'affichage de la grille. Malheureusement, cette approche s'est révélée difficile à mettre en œuvre, car la taille de chaque niveau de la grille varie en fonction de la configuration sur notre scène, tandis que notre transition est réglée sur une valeur précise, à savoir la taille du côté de chaque case. Cette incompatibilité a rendu difficile la réalisation de l'affichage dynamique que nous souhaitions.

L'autre difficulté est liée à la résolution automatique de la grille. En effet, selon le nombre de cases et la composition de ces dernières, la complexité d'un algorithme qui trouve le chemin optimal est grande. Nous pouvions améliorer la méthode grâce à un heuristique mais la résolution restait malgré tout très longue. De plus, il était difficile de trouver un algorithme qui fonctionnait dans le cas d'une grille avec plusieurs cases blanches et/ou des cases indestructibles.

IV. Interface graphique

---

a. Implémentation du fxml

Dans notre approche pour l'affichage, nous avons opté pour l'utilisation de fichiers *FXML*. Ces fichiers contiennent les données nécessaires à l'affichage de chaque scène de notre application. Ainsi, il est nécessaire d'avoir un fichier *FXML* dédié à chaque scène que nous souhaitons afficher. Chaque fichier *FXML* est lié à un fichier *CSS* qui définit le style des composants tels que les boutons, les labels, etc.

Pour faciliter le positionnement des composants dans nos interfaces graphiques, nous avons utilisé un outil appelé *Scene Builder*. Il nous a permis de disposer les éléments visuels de manière intuitive et de visualiser l'apparence finale de nos scènes.

Afin d'établir le lien entre notre interface graphique et le code fonctionnel en Java, nous avons dû créer des contrôleurs. Le contrôleur est une classe *Java* que nous avons mise en place pour implémenter les méthodes nécessaires à l'affichage graphique et à la gestion des interactions avec les utilisateurs. Il faut ensuite relier le contrôleur aux différents fichiers *FXML*.

Dans la classe du contrôleur, nous avons défini différentes méthodes en fonction des besoins de notre application. Par exemple, nous avons une méthode appelée "*SwitchToMenu*" qui permet de changer la scène affichée dans la fenêtre principale. Nous avons également une méthode "*Initialize*" qui s'exécute automatiquement chaque fois qu'une nouvelle scène est chargée, ce qui nous permet de réaliser des initialisations spécifiques à cette scène.

b. Affichage dynamique

Pour afficher dynamiquement les niveaux disponibles dans notre application, un seul fichier *FXML* ne suffit pas. Nous devons d'abord vérifier dans un fichier *CSV* le nombre de niveaux disponibles avant de les afficher. Cette tâche nécessite l'intervention du contrôleur pour lire le fichier *CSV* correspondant et ensuite afficher les niveaux correspondants en utilisant *JavaFX*.

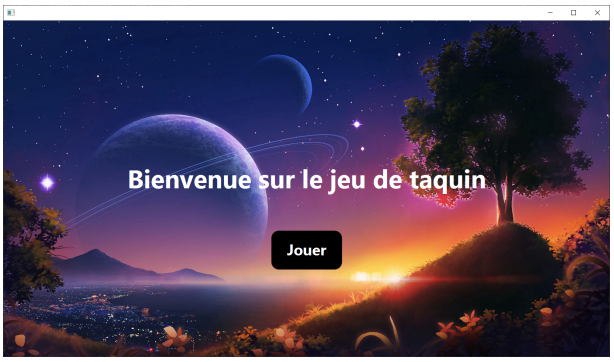
Nous avons utilisé cette approche d'affichage dynamique à plusieurs reprises, notamment lors de l'affichage de la grille de jeu. Pour afficher la grille, nous avons une méthode "*DisplayGrid*" qui est appelée autant de fois que nécessaire. Cette méthode peut être déclenchée lorsqu'un utilisateur clique sur un label ou un bouton spécifique. Il est important de noter que ce code d'affichage n'est pas présent dans le fichier *FXML* lui-même. Il est géré dans le contrôleur de l'application.

Dans le texte mentionné précédemment, nous avons rencontré un problème lors de la gestion de la méthode "*initialize*" dans le contrôleur. Cette méthode s'exécute chaque fois qu'une nouvelle scène est chargée, ce qui signifie qu'elle peut être déclenchée lors de tout changement de scène. Cependant, nous ne voulions pas exécuter certaines opérations spécifiques à une scène donnée si la scène actuelle n'était pas celle que nous recherchions.

Pour résoudre ce problème, nous avons ajouté des conditions dans la méthode "*initialize*" qui vérifient l'existence de certains labels ou boutons spécifiques à la scène que nous souhaitons traiter. Ainsi, si ces éléments sont présents dans la scène actuelle, les opérations correspondantes sont exécutées. Sinon, la méthode "initialize" se termine sans effectuer ces opérations spécifiques.

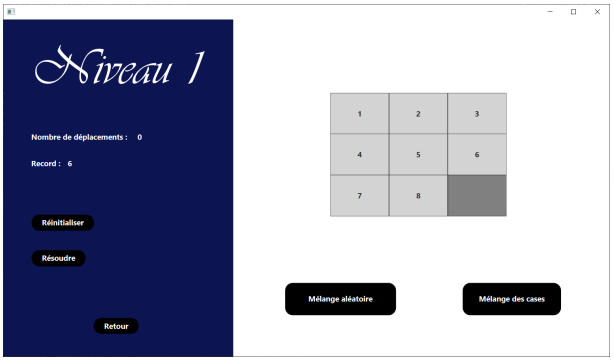
En résumé, en utilisant un contrôleur et en gérant l'affichage dynamique à partir de celui-ci, nous avons pu lire les données à partir d'un fichier *CSV*, afficher les niveaux disponibles et contrôler l'exécution de certaines opérations en fonction de la scène actuelle. Cette approche nous a permis d'obtenir un affichage flexible et adaptatif dans notre application.

c. Photos du jeu

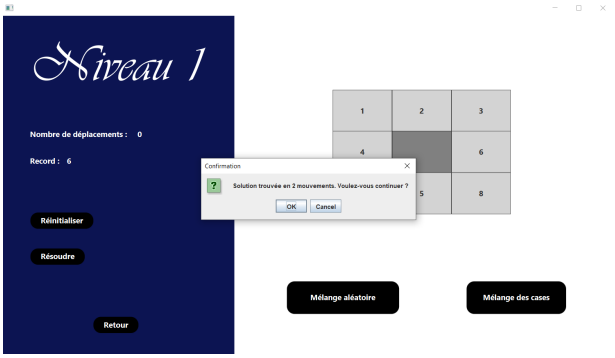


Page d'accueil du jeu

Page avec le choix des niveaux



Niveau 1 du jeu



Niveau 1 du jeu avec la fenêtre du bouton "résoudre" ouverte

V. Algorithme de résolution

a. Jouabilité du niveau

Pour la résolution de la grille, nous avons tout d'abord dû vérifier si la grille pouvait atteindre sa solution. Pour cela, il fallait considérer deux cas :

- Dans le premier cas, la taille de la grille (taille du côté) était paire. Il faut alors calculer le nombre d'inversion, c'est-à-dire le nombre de couple de valeurs inversées, ainsi que la ligne où se trouve le zéro. Si la somme de ses deux nombres est impaire, alors il est possible de finir la partie.
- Dans le second cas, la taille de la grille est impaire. Dans ce cas-là, seulement calculer le nombre d'inversion. S'il est pair, alors la grille à une solution.

Malheureusement, ces cas ne sont valables que pour les grilles simples (avec une unique case vide et sans case blanche). Nous avons donc essayé de l'adapter au mieux pour les grilles complexes mais l'algorithme ne renvoie pas toujours le bon résultat.

b. Algorithme de résolution

Après quelques recherches, l'algorithme qui nous a semblé le plus adapté était l'*algorithme A**. Il permettait, grâce à un heuristique, de trouver la solution la plus courte pour terminer le jeu. Nous avons utilisé deux heuristiques : la distance de *Manhattan*, qui calcule la distance entre la position d'une valeur dans la grille de base et dans la grille résolue ; la distance de *Hamming*, qui calcule le nombre de cases inversées. Le choix de la fonction heuristique à ensuite été fait par test pour trouver la combinaison qui rend l'algorithme le plus optimal.

Cette fois le problème à été la complexité de l'algorithme : plus la grille est grande, plus les distances de *Manhattan* et de *Hamming* sont élevées, plus sa complexité sera grande.

Nous avons donc créé un second algorithme, cette fois bien plus rapide car il ne visite pas toutes les possibilités de mouvement. Il ne trouve donc pas toujours le chemin le plus optimal.

VI. Conclusion

Le projet *CY-Slide*, qui consistait à implémenter le jeu du taquin en Java, a été une expérience enrichissante. Nous avons travaillé en collaboration pour développer différentes parties du projet : le code principal, l'interface graphique et l'algorithme de résolution. L'organisation au sein de l'équipe a été cruciale pour coordonner nos efforts. Nous avons réparti les tâches en fonction de nos compétences et nous avons régulièrement communiqué pour partager nos avancées et résoudre les problèmes rencontrés. Les outils tels que *Discord*, *Teams* et *GitHub* nous ont permis de collaborer efficacement et de partager notre code.

Nous avons fait face à plusieurs défis techniques tout au long du projet. L'élaboration des diagrammes, en particulier celui de classes, a demandé plusieurs ajustements au fur et à mesure que nous progressions dans le développement du jeu. L'interface graphique a également été un aspect important, avec l'utilisation des fichiers *FXML* et du *Scene Builder* pour créer des scènes dynamiques. Cependant, nous avons rencontré des difficultés pour réaliser l'affichage dynamique des mouvements des cases, en raison de la complexité de la gestion des transitions dans *JavaFX*. En ce qui concerne l'algorithme de résolution, nous avons exploré différentes approches et avons finalement opté pour l'*algorithme A** avec deux heuristiques : la distance de *Manhattan* et la distance de *Hamming*. Cependant, nous avons dû prendre en compte la complexité de l'algorithme, en particulier pour les grilles de grande taille, ce qui nous a amenés à développer un deuxième algorithme plus rapide mais moins optimal.

En conclusion, le projet *CY-Slide* nous a permis de mettre en pratique nos compétences en programmation *Java* et en conception logicielle. Malgré les défis rencontrés, nous avons réussi à développer un jeu fonctionnel avec différents niveaux et la possibilité de résoudre les grilles. Nous avons acquis une expérience précieuse en travaillant en équipe, en gérant les difficultés techniques et en prenant des décisions pour optimiser le projet.