*CASTELAO Romain*

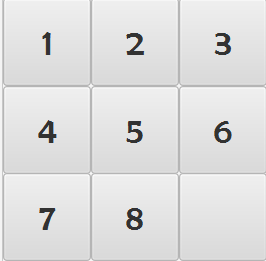
*HASSANI Abdellah*

*BAHHAR Sarah*

*REN Simon*



*ZUO Franck*

**

Notre projet se reposait essentiellement sur le jeu du taquin. Le jeu du taquin est un jeu inventé au XIXe siècle et à l’origine le jeu est composé de 15 petits carreaux numérotés de 1 à 15 qui glissent dans un cadre prévu pour 16. Il consiste à remettre dans l'ordre les 15 carreaux à partir d'une configuration initiale quelconque. Avec certaines dérivées on peut remplacer les numéros par des images, changer les formes ou ajouter des cases vides. Notre projet consistait donc à implémenter 10 niveaux différents en variant un maximum les types de taquin grâces aux différents critères cités ci-dessus. Nous devions être capable de mélanger aléatoirement ou étape par étape la grille et lui donner son score (son score étant le nombre de coup effectué après un mélange). Nous devions également créer un algorithme pouvant résoudre notre jeu du taquin lorsque l’utilisateur le souhaitait.

1

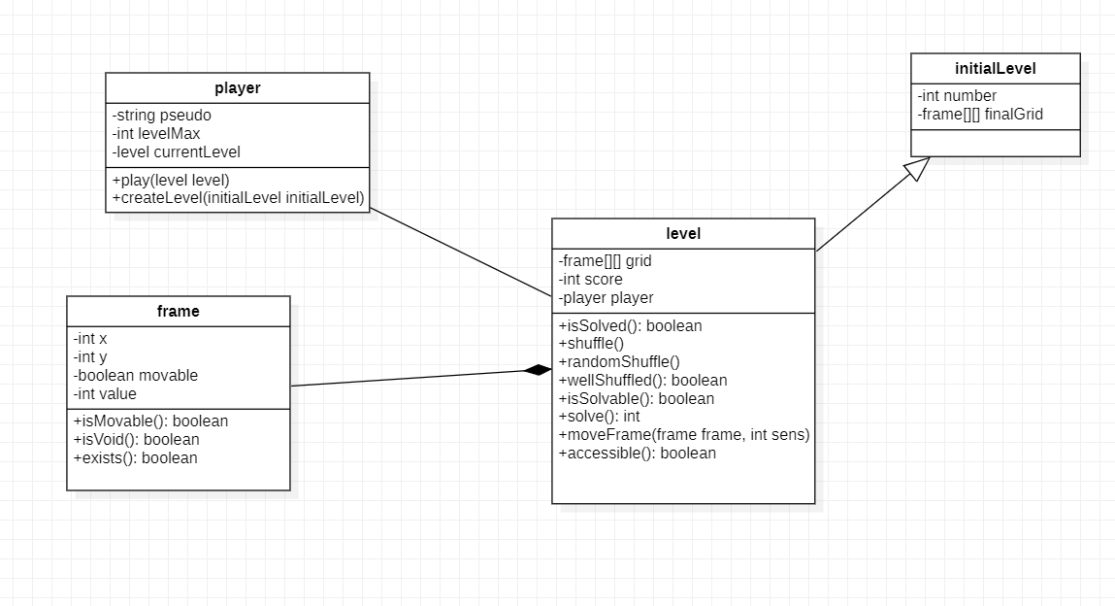
****Tables des Matières :**

1. Organisation
2. Première semaine 3
3. Deuxième semaine 4
4. Problème et solution
5. Algorithme et Implémentation 5
6. Interface Graphique 6

2

**

1. Organisation
2. Première semaine



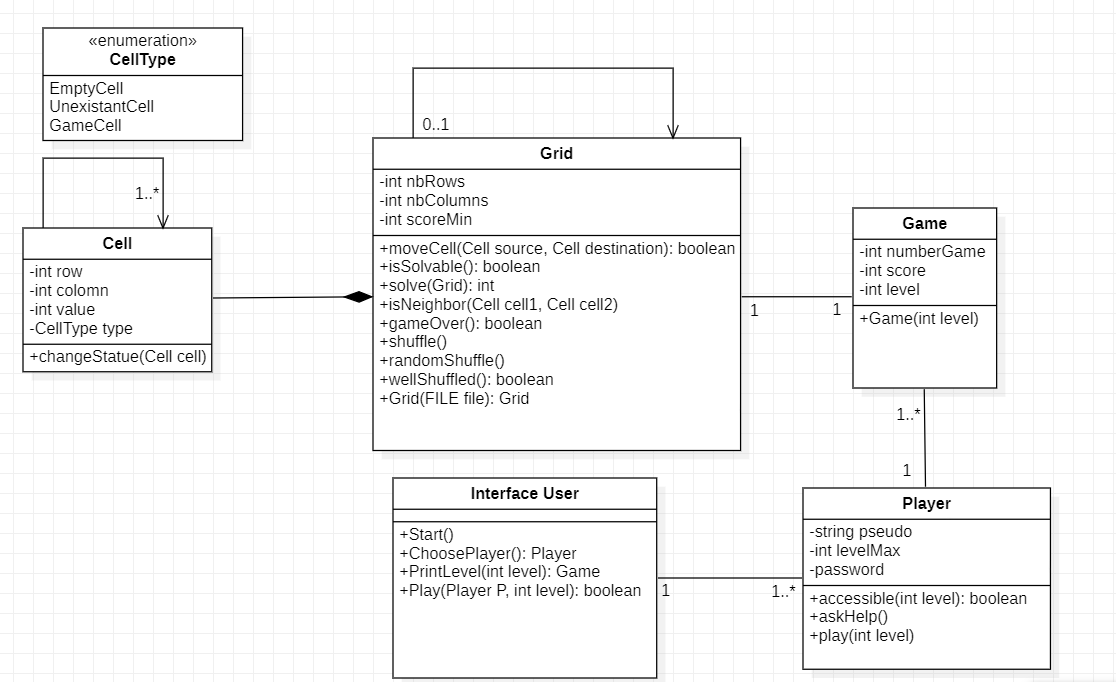
Notre idée principale était de nous occuper de la partie algorithmique la première semaine et la partie visuelle était réservée pour la seconde. Voici ci-dessus une première version de notre diagramme de classe. Nous avons tous participé à sa création puis nous avons distribué nos rôles. Sarah, Romain devaient implémenté les classes et améliorer si besoin le diagramme de classe, Simon s’occupait de la fonction moveFrame(), Abdellah faisait la résolution du taquin et son algorithme et Franck implémentait toutes les fonctions en rapport avec le shuffle (shuffle(), randomshuffle(), wellshuffled()).

3

1. **Deuxième semaine

Lors de la fin de la première semaine, nous avons eu un échange avec notre professeure référente, Mme Djaouida, nous avons tout refait. Effectivement, elle nous a fait remarquer que la structure de notre diagramme de classe était fausse et peu claire. Ayans moitié de temps prévu et tout à refaire, nous avons dispersé notre équipe. Sarah et Simon s’occupe de l’interface graphique, Abdellah et Romain s’occupait de l’implémentation des algorithmes et Franck devait faire l’algorithme solved(). Une fois que la plupart des algorithmes finis, Romain a rejoint l’équipe de Sarah et Simon et Abdellah a aidé Franck avec l’algorithme solved().

Voici la nouvelle version de notre diagramme de classe :



4

1. **Problèmes et résolutions
2. Partie algorithmique et implémentation

Comme vous pouvez le constater, le nouvel diagramme est bien plus concis et épuré, cela nous a beaucoup aidé dans la répartition des tâches. Le principal défaut de notre ancien diagramme de classe est la façon dont on l’a construit. Par exemple, nous avions mis des tableaux dans certaines classes, alors que l’on ne peut pas. De plus, certaines de nos relations n’étaient pas assez explicites et cela rendait l’implémentation de nos algorithmes bien plus compliquée. Grâce à Mme Djaouida, on a reconstruit étape par étape notre diagramme de classe. Le processus fut plus long mais le résultat est bien meilleur.

La première version de notre algorithme permettant de résoudre le jeu du taquin proposé par Abdellah était une sorte de parcours en profondeur. Mais grâce à une remarque de notre professeure, on a remarqué que notre fonction solved() était loin d’être optimal. En effet, le jeu du taquin est un problème np complet. Nous nous sommes donc tournés vers l’algorithme A\*, un algorithme qui utilise une méthode heuristique pour résoudre notre problème. De par sa construction, l’algorithme A\* renvoie dès sa première solution, une des meilleures solutions. Cet algorithme applique à chaque nœud une fonction pour calculer son coût Heuristique pour estimer le meilleur chemin le passant. Pour l’algorithme A\*, on peut utiliser plusieurs caractéristiques pour le coût Heuristique comme la distance de Manhattan ou encore le nombre de case mal placé. Ici on appliquera la distance de Manhattan comme méthode heuristique. La distance de Manhattan est la somme des distances qui sépare chaque case avec sa position « goal » (position qu’elle doit se trouver à la fin de l’algorithme). Cet algorithme demande une grande rigueur, chaque variable mal utilisée fait grandir exponentiellement la complexité temporelle et spatiale de la fonction. C’était le plus gros problème rencontré dans notre projet.

L’autre gros problème dans notre implémentation était la fonction isSolvable(). En effet, étant donné qu’il existe plusieurs types de mélange dont un mélange aléatoire qui ne garantit pas sa faisabilité, il nous fallait être certain lorsqu’on applique la fonction solved() que la grille soit solvable au risque de faire des calculs inutiles. Pour cela, on s’est inspiré de la vidéo de Mickaël Launay *Le taquin impossible- Micmaths ,* cette vidéo fut très instructif, et on a donc décidé d’utiliser sa condition et sa technique. Pour ce faire, on va compter le nombre de pair dans le mauvais ordre, c’est-à-dire, pour chaque pair de chiffre dans le taquin si le nombre le plus petit dans la paire apparaît effectivement avant l’autre nombre, alors la paire est dîtes dans le bon ordre. Dans le cas contraire, la paire est dans le mauvais ordre et on incrémente d’un le compteur. Lorsqu’on déplace une case, le compteur peut seulement soit incrémenté de 2, de 0 ou décrémente de 2. Il est donc évident que si le nombre de pair dans le mauvais sens est impair, alors il est impossible de refaire le taquin. On va se baser sur cette condition. Cependant, n’étant pas sûr que cela fonctionne avec chaque forme, et ayant aucune façon de le démontrer rigoureusement, on a décidé que pour chaque forme, on prendra simplement le plus petit carré qui englobe la grille et on comptabilisera chacune des cellules même si elles ne sont pas déplaçables.

5

1. **Partie interface graphique

Le plus grand défi dans cette partie était notre inexpérience. En effet, la majorité des personnes de ce groupe viennent de prépa et sont très peu familiers avec JavaFX ou n’importe quel type d’application d’interface graphique. De plus, nous avions eu très peu de cours sur JavaFX et très peu de méthode ou de technique ont été vu en cours, la plupart de nos connaissances viennent de recherche internet. En plus de cela, comme dit précédemment, nous avions eu semaine avec seulement 2-3 personnes sur cette partie alors qu’on aurait idéalement voulu être 5 pour nous concentrer pleinement dessus. Puis, l’avancée de l’interface graphique est grandement influée par l’évolution de la partie algorithmique qui devait tout refaire. Chaque modification du code nous a pousser à réadapter notre interface graphique, détruisant parfois complètement certains de nos travaux. On a donc été très limité par le temps. Finalement, notre plus grand défaut fût notre gourmandise, étant tous fan de jeux-vidéos et ayant grandi avec ce genre de jeu, nos attentes été très élevées. En effet, nous savions quel type de jeu nous voulions et à quoi il va ressembler, et cela nous a plus porté préjudice qu’autre chose.

Le concept du jeu étant ancien, on voulait garder ce côté tout en l’accouplant avec une touche de modernité. Le style fût vite trouvé mais nous avions eu beaucoup de débat sur comment on devait se connecter, quels types de données nous allions affichés, la police. L’équilibre de cette dualité fût compliqué à trouver mais nous pensons que nos choix devraient correspondre aux goûts des plus vieux et des plus jeunes. On voulait également laisser le côté jeu d’arcade que le taquin a, comme Tetris, et nous avons donc décidé de laisser chaque utilisateur se connecter uniquement avec son nom.

Finalement, comme dit ci-dessus, étant très pointilleux, nous avions ajouter quelques fonctionnalités pour rendre notre jeu plus vivant comme le fait que lorsque deux cases ne puissent pas être bougées, alors elles changent momentanément de couleur, et ces quelques changements fût très coûteux en temps.

6