Documentation Développeur

Lucie Cubaud
Vittoria Vecchioli

1 Schéma UML

Nous avons partagé notre code en trois répertoires (Model, View, Controller) en essayant de reproduire le modèle MVC. Le répertoire Model contient toutes les fonctions qui définissent le jeu, le répertoire View contient les fonctions nécessaires à la visualisation et le répertoire Controller permet l'interaction entre le joueur et le plateau. Dans la figure suivante, nous présentons l'architecture du projet à l'aide d'un schéma UML. En bleu sont représentées les classes du répertoire Model, en vert celles de View et en rouge celle de Controller.

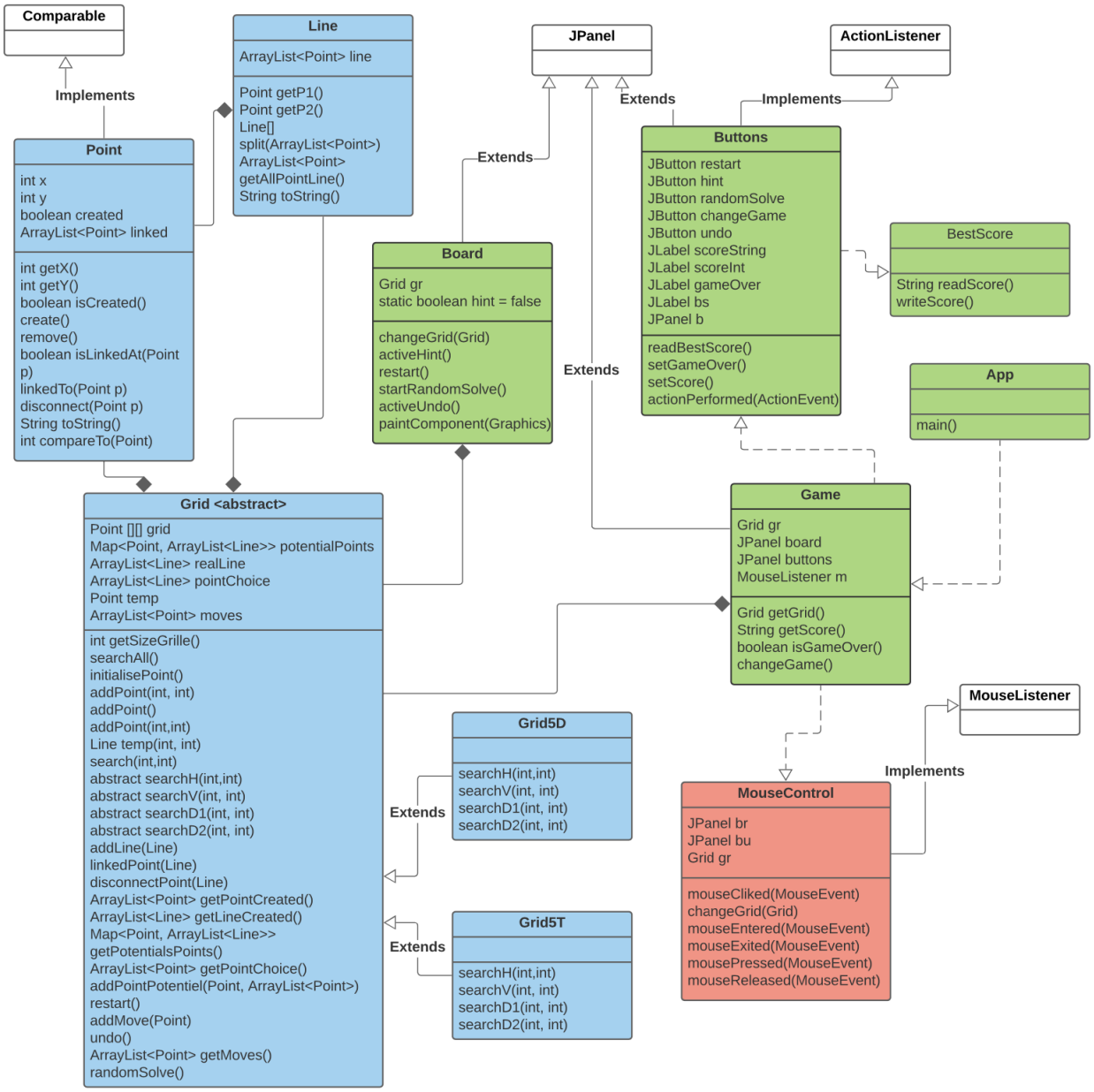

Figure 1: Schéma UML du projet.

2 Model

2.1 Point

La classe Point permet de modéliser un point sur le jeu. Initialement, nous avions symbolisé un point uniquement par ses coordonnées x et y, en plaçant arbitrairement le point (0,0) au milieu du plateau. Dans notre implémentation finale, nous avons ajouté un Boolean created qui indique si le point est déjà tracé sur le jeu ainsi qu'une ArrayList de Point qui permet de savoir avec quels autres points celui-ci est lié pour former une ligne. Ces deux ajouts sont le résultat d'un changement d'approche dans la modélisation de notre jeu, nous détaillons ces changements dans la section Grid.

La classe Point implémente l'interface Comparable afin de pouvoir ordonner les points dans une liste. En effet, au cours de la recherche, les points potentiels sont ajoutés au fur et à mesure qu'ils sont identifiés. Pour ensuite vérifier si ces points forment des lignes, il est plus simple de les ranger en fonction de leur abscisse, par ordre croissant (de gauche à droite). La classe contient donc la fonction compareTo() qui permet de comparer deux points entre eux. Il y a aussi une fonction disconnect() qui permet de déconnecter un point d'une ligne et donc "d'effacer" une ligne tracée. Cette fonction est utilisée notamment pour la fonctionnalité Undo.

2.2 Line

Les lignes sont représentées dans la classe Line. Dans notre première implémentation, une ligne était représentée par seulement deux points, les extrémités, et une direction (horizontale, verticale, diagonale vers le haut ou vers le bas). Cette représentation n'était par optimale car nous devions recalculer les points de la ligne en fonction de sa direction à chaque nouvelle recherche de points potentiels. Nous avons donc décidé de représenter une ligne par une ArrayList de 5 points. La classe Line contient les fonctions pour accéder aux points de la ligne ainsi qu'une fonction qui permet de former des lignes en passant un ArrayList de points en argument. Cette fonction est utile dans la classe Grid pour générer toutes les lignes potentielles identifiées.

2.3 Grid

Notre première approche pour modéliser la grille du jeu a été de créer une classe Grille avec un nombre de lignes et de colonnes fixe, une liste de points initiaux représentant le dessin de base, une liste contenant les points tracés et une liste contenant les lignes tracées. La grille de départ était initialisée avec les coordonnées des points initiaux. Cette implémentation n'a pas aboutie car elle était trop complexe à maintenir. Notamment la représentation de la grille par une liste de points tracés nécessitait des fonctions de recherches fastidieuses car il fallait faire plusieurs "for loop" pour vérifier si un point était contenu dans une ligne, et si oui dans quelle direction. Une autre difficulté liée à cette implémentation était que les coordonnées choisies pour les points ne correspondaient pas aux coordonnées du Frame (le (0,0) était en pleins milieu du plateau et non pas en haut à gauche). Il fallait donc faire une formule pour recentrer les points au milieu de la fenêtre de jeu.

Cette implémentation a donc été abandonnée au profit d'une autre, plus simple à modéliser où la grille est représentée par une liste de liste de Boolean. Une liste représente une ligne du plateau et un Boolean représente une intersection entre une ligne et une colonne et a pour valeur True si le point est tracé et False autrement. Cette représentation est plus visuelle et surtout plus facile à mobiliser dans nos fonctions de recherche. La liste est actualisée à chaque point ajouté ou retiré. Nous avons gardé les listes de points potentiels et des lignes tracées.

Nous avons décidé qu'il était plus cohérent de créer une classe abstraite Grid et deux classes Grid5D et Grid5T héritant de Grid afin d'éviter une redondance. En effet nous avons d'abord commencé par implémenter la version 5D. Lorsque nous avons implémenté la version 5T nous avons remarqué qu'à l'exception des fonctions de recherche de points potentiels, les deux versions étaient semblables. Cet héritage nous a permis de faire passer la grille en paramètre facilement, puisqu'il n'y a pas besoin de préciser quel type de grille on veut afficher.

Grid5D

L'algorithme de recherche de points potentiels est celui qui nous a donné le plus de fil à retordre. Dans un premier temps, nous pensions faire une recherche circulaire sur chaque point mais notre implémentation n'était pas assez structurée et nous avons donc abandonnée cette idée. Nous avons donc divisé la fonction de recherche en quatre sous parties pour nous concentrer sur un type de ligne (horizontale, verticale, diagonale) à la fois. Nous voulions ajouter chaque ligne trouvée au fur et à mesure de la recherche mais l'implémentation d'une telle méthode était trop compliquée. Alors nous avons pensé que nous pouvions

créer un seule liste pour chaque direction contenant tous les points potentiels. Une fois seulement cette liste trouvée nous créons différentes lignes potentielles. Le choix d'une Map est alors cohérent puisque nous voulions associer plusieurs lignes à un même point.

Grid5T

Après avoir implémenté la classe Grid5D, nous avons implémenté la classe Grid5T qui, de part ses règles plus souples, rend l'algorithme de recherche plus compliqué. Nous avons testé une condition sur la taille de la ligne (maximum 9 points) mais ce n'était pas concluant. Après avoir énuméré les conditions de recherche nous avons conclus qu'à la différence d'une recherche 5D qui s'arrête lorsque le point essaye de se lier avec un point déjà lié dans la même direction, la recherche 5T inclue ce point dans la ligne. Il nous a fallu de reprendre l'algorithme de recherche de 5D et ajouter cette condition.

3 View

3.1 Game

La classe Game est un contient les classe Board et Button. Elle permet de lancer la grille de jeu. Au début, nous avions créer une frame et avons lancer Board et Buttons dedans, cela nous a posé problème par la suite lorsque nous avons voulu avoir la possibilité de changer de plateau de jeu (5T et 5D). La solution que nous avons trouvé pour régler ce problème est de créer une classe App qui permet d'indiquer que la fenêtre doit contenir Game comme conteneur. Bien que cela nous a permis de trouvé une solution pour le changement de grille, la disposition de nos panels a été modifiée. En effet, au lieu d'avoir les boutons au dessus de notre grille de jeu, nous avions les boutons à gauche et la grille à droite. Après des recherches nous avons fini par trouvé une méthode (setLayout) qui nous permet d'obtenir la disposition souhaité.

3.2 Board

Board est une classe permettant l'interaction avec le joueur. L'instance de Board est créée dans la classe Game. Initialement, nous voulions déclarer la variable de Grille dans cette classe. Il était finalement plus simple de la passer en argument du constructeur de Board afin que la grille soit donnée en argument à d'autre classe comme MouseControl, pour faciliter l'interaction entre les instances.

Notre implémentation de la classe Grille nous permet assez facilement de construire l'interface graphique, puisqu'un des éléments qui compose notre grille est une matrice de point. Ainsi, les coordonnées de la matrice peuvent être réutilisées dans l'interface pour dessiner nos points.

La classe est composé de plusieurs fonctions, des fonctions de modification du Board (activHint, restart, startRandomSolve, activeUndo) et la fonction paintComponent qui permet d'afficher notre grille. Nous avons implémenté les fonctions de modification de sorte qu'elle appellent des fonctions de Grid qui vont modifier, ou non, les listes de Grid, puis de repeindre notre Board en fonction des transformations apportées. Ainsi notre fonction paintComponent() fait des parcours sur les différentes List représentant différentes facettes de notre grille. Nous affichons les points et les lignes en fonction du contenu de ces listes. Cette implémentation nous permet d'ajouter, assez facilement, de nouvelles fonctions d'affichage en ajoutant de nouveaux getters dans la classe Grid.

Un problème qui nous est apparu dans cette classe est un problème d'interaction entre le joueur et le jeu. Lorsque l'on créait un point, la ligne qui apparaissait se confondait avec le quadrillage de la grille. Nous ne voulions pas que les points et lignes tracées aient des couleurs différentes et voulions garder une interface sobre. La seule solution était d'épaissir le trait. Nous avons dû faire des recherches sur Internet pour trouver la solution.

3.3 Buttons

La classe Button est un JFrame contenant différents boutons, le score courant de la partie et le meilleur score toutes parties confondus. Elle indique également si la partie est perdue ou non. L'implémentation de la classe Buttons a été de prime abord assez simple. Nous avions toujours le même raisonnement : créer un bouton, le relier avec la fonction correspondante du Board. C'est ainsi qu'on été créés les boutons RandomSolve, Undo et Hint.

L'implémentation du changement de partie a été plus compliqué. En effet la classe Button n'a aucune référence à la grille et passe toujours par Board. Hors, la classe MouseControl avait aussi besoin de changer de grille. Nous avons alors pensé qu'il y avait surement une possibilité d'avoir accès à la classe conteneur Game qui contient la panel Button. Apres avoir fait des recherches sur Internet nous avons

trouvé la fonction getParent(). Nous avons donc choisi de créer une fonction changeGame dans la classe Game. Ainsi Game peut créer une nouvelle instance de Grid et la donner aux autres classes.

Une autre difficulté fut l'implémentation du score. Contrairement aux boutons, le score est le résultat d'une action du joueur sur le Board. A l'instar du bouton changeGame, nous avons finalement compris qu'il fallait réutiliser la fonction getParent() pour avoir accès à la taille du tableau moves de Grid, et ainsi connaître le nombre de coup joués. On demande cette taille à trois occasions. Au moment où le joueur click sur le board pour ajouter un point (au niveau de MouseControl) de cette façon le score s'incrémente. Une deuxième occasion est quand le joueur veut faire un pas en arrière, ainsi le score diminue. Enfin, lorsque le joueur demande à résoudre la partie avec un RandomSolve, le score de l'algorithme est donné.

3.4 BestScore

Nous avons créé une classe BestScore qui permet de lire un fichier de score. Cette classe implémente deux fonctions, la lecture du fichier BestScore et l'écriture de ce fichier. Nous avons utilisé les classes BufferedWriter et BufferedReader afin d'effectuer les transformations sur le fichier. Nous aurions pu écrire ses fonctions dans la classe Buttons puisque seulement cette classe utilise le BestScore. Cependant, nous avons préféré créer une classe qui s'occupe de gérer le fichier dans le cas ou notre implémentation du projet évolue et que plusieurs classes veuillent avoir accès à ce score.

4 Controller

4.1 MouseControl

La classe MouseControl permet de détecter les actions du joueur sur la grille de jeu. La classe ne fait aucun test à part de vérifier que le curseur de la souris est bien dans la zone de notre grille. Ainsi il faut convertir la position de la souris en des coordonnées de notre grille (point). Nous avons voulu diminuer la précision d'ajout d'un point, avec la méthode round, pour améliorer l'expérience joueur.

5 Fonctionnalités

5.1 Hint

Nous avons voulu créer une fonction qui génère des indices pour le joueur. Cette fonction utilise la Map pointPotentiel et fait un parcours sur ses clés pour trouver les points valables.

5.2 Undo

Cette fonctionnalité permet de revenir en arrière autant de fois qu'on le souhaite. Lorsque l'on clique sur le bouton, le dernier coup joué est annulé et la ligne correspondante est effacée. On peut alors rejouer le même coup ou en faire un autre. Pour réaliser cette fonctionnalité, nous avons adapté l'implémentation de la Grid pour garder en mémoire tous les points tracés dans l'ordre (dans l'ArrayList de points move). A chaque Undo, le dernier point de la liste est enlevé ainsi que la dernière ligne, on déconnecte tous les points de la ligne pour qu'ils ne soient plus associés les uns aux autres. Enfin, on réinitialise les points potentiels et on relance la recherche pour trouver tous les coups potentiels dans cette nouvelle configuration.

5.3 Scores

Le score indique le nombre de lignes tracées sur le jeux. Il est incrémenté à chaque nouveaux point ajouté. Pour réaliser cette fonctionnalité, nous avons créé un label score dans la classe Button ainsi qu'une fonction setScore() qui récupère le nombre de coups joués jusqu'à présent et actualise le label score. Cette fonction est appelé dans le MouseControl à chaque fois qu'un point est ajouté. Une autre fonction getScore(), définie dans la classe Game, fait intervenir getParent() pour connaître le nombres de coups joué sur le jeux en cours. C'est cette fonction qui permet à setScore() d'obtenir le score. Le score est réinitialisé à chaque nouvelle partie.

5.4 Best Score

Le meilleur score est comme nous l'avons dit plus tôt, affiché au niveau de la classe Button. Nous avons choisi de ne garder qu'un meilleur score pour ne pas avoir trop d'élément sur notre fenêtre de jeu.

Le meilleur score est mis à jour lorsque le joueur arrive jusqu'au gameOver ou lorsqu'il recommence une partie de même type (5D ou 5T).

5.5 Switch 5D, 5T

Le switch n'est pas vraiment une extension comme décrite dans le sujet car il était demandé d'avoir la possibilité de jouer avec les deux grilles, seulement nous voulions vraiment avoir la possibilité de changer de partie à n'importe quel moment du jeu. On a voulu que ce changement soit représenté à travers un bouton, qui change son texte en fonction du type de partie jouée.

5.6 Fonctionnalités non explorées

Nous n'avons pas rajouté d'autres fonctionnalités par manque de temps. Cependant, nous pensions rajouter d'autres grilles de jeu. Pour cela, nous il aurait fallu revoir notre architecture et créer une classe Canevas de type Enumération. Cette classe contiendrai autant de types énumérés que de plateaux de jeu. Les types auraient un tableau de booleans. On devra ajouter une fonction à Grid pour pouvoir changer de grille initiale. Il faudra également changer le bouton Switch de la classe Buttons. On pourrait par exemple utiliser un Menu afin de voir les choix de grille.