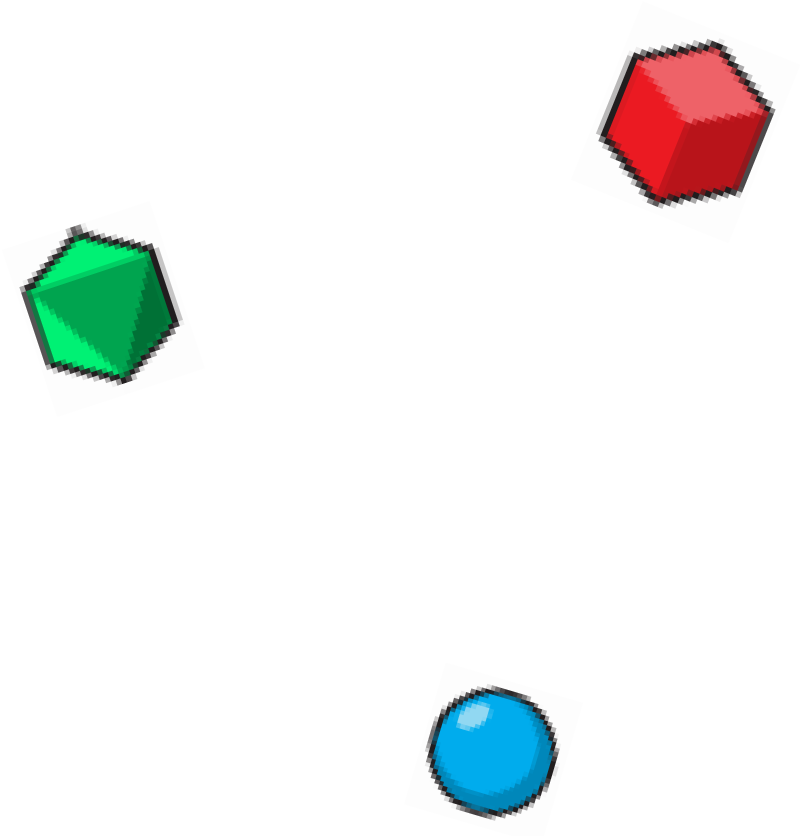
PT 2.1 APL

Projet « Same Game »

Étudiants

Dylan Girault

Guillaume Toutain



Sommaire

Description du projet

Le but de ce projet est de réaliser un jeu 2D écrit en Java, sans emprunt extérieur.

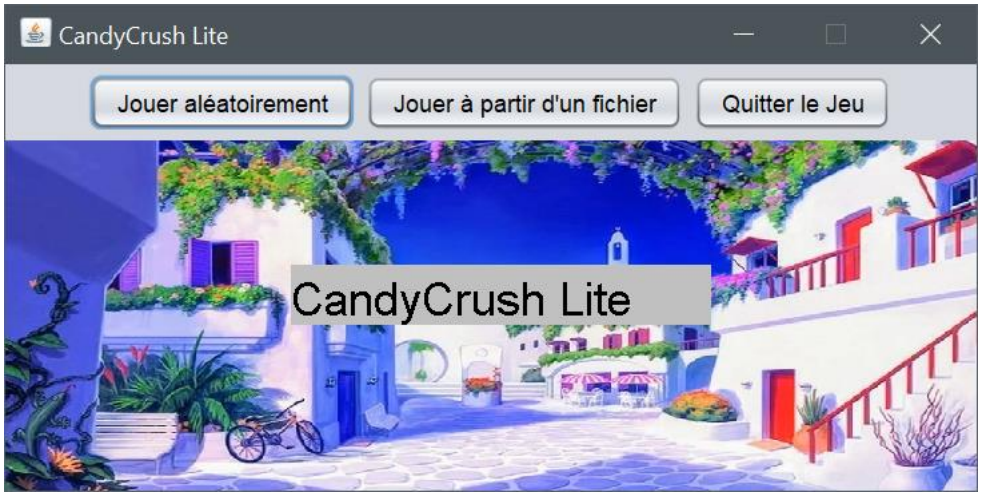
Les règles du jeu sont simples, une grille de 15 par 10 est remplie d'éléments. Chaque élément est soit bleu, rouge ou vert, l'utilisateur peut sélectionner et supprimer un groupe d'élément de la même couleur. Un groupe est constitué d'au moins 2 éléments directement côte à côte. Le but est de vider la grille avec le meilleur score. Pour finir le gain est calculé par rapport au nombre d'éléments supprimés en un seul clic.

La partie est terminée dès que le plateau ne contient plus de groupes d'éléments.

Fonctionnalités

Lancement du jeu

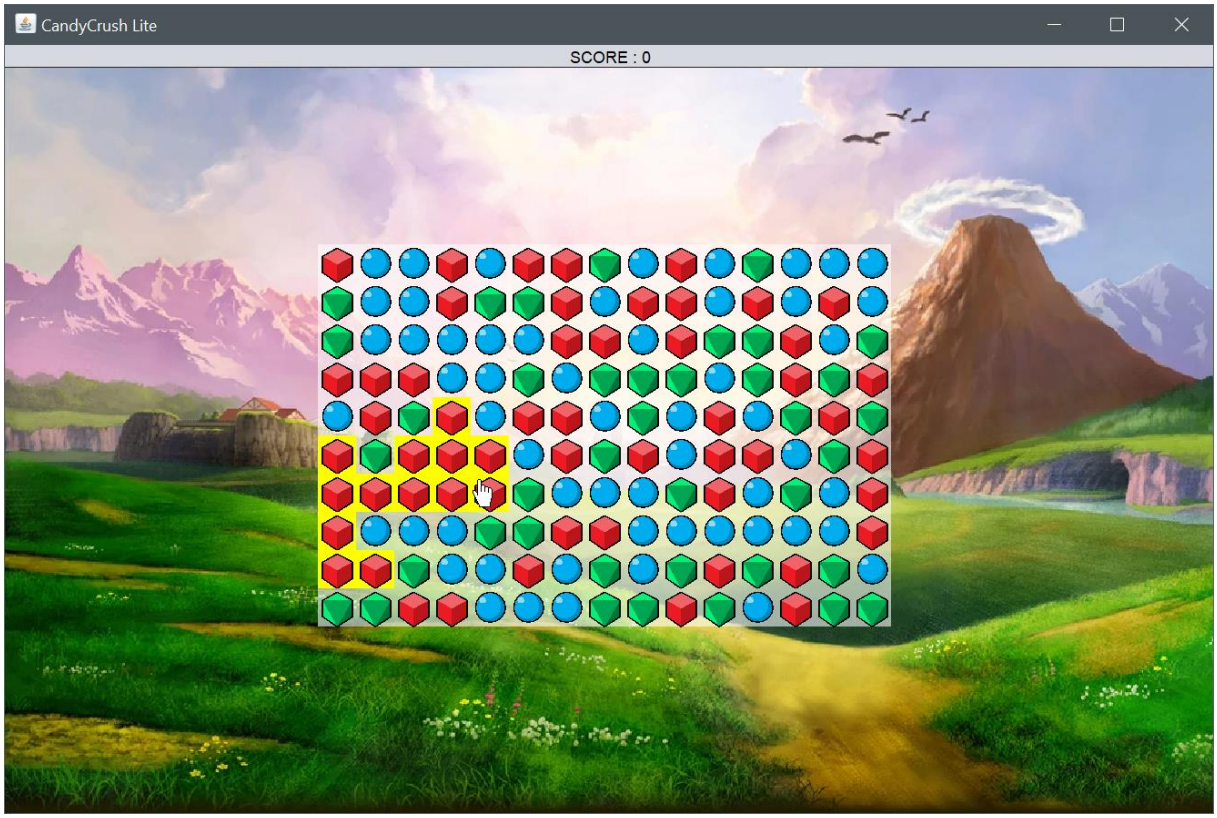
Au lancement du jeu, l'utilisateur peut choisir d'obtenir une grille aléatoire soit une grille sauvegardée dans un fichier (il est à noter que le programme ne permet pas l'enregistrement d'un plateau)



Écran de lancement

Affichage des groupes

Lors du survol d'un groupe valide, le jeu surlignera tous les éléments du groupe.

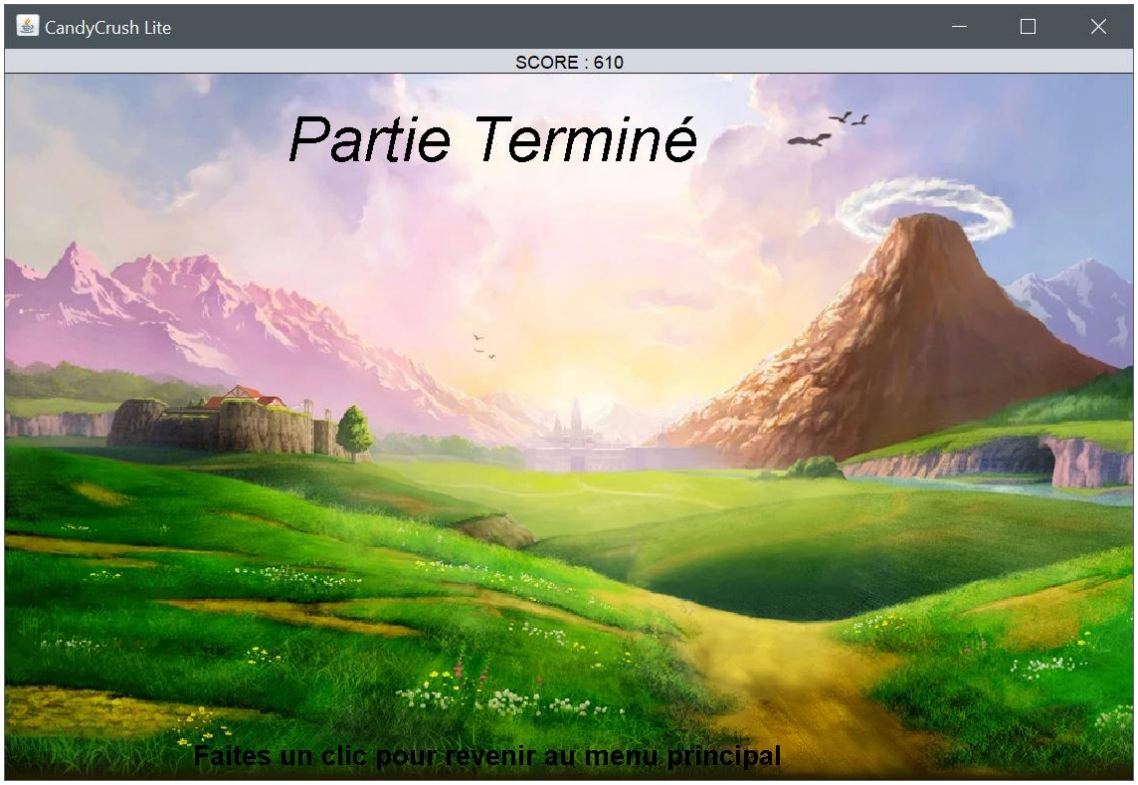


Redimensionnement

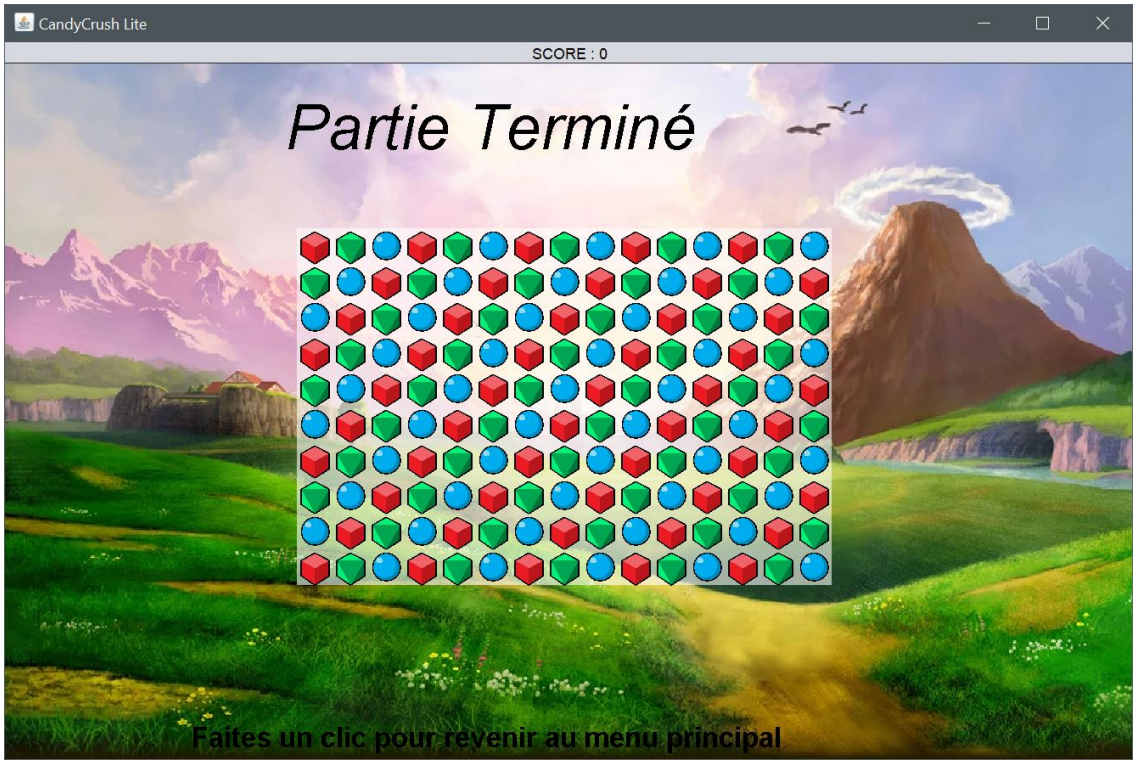
L'affichage peut s'adapter au redimensionnement de la fenêtre de jeu

Écran de fin

Si le plateau est vide, la partie se termine

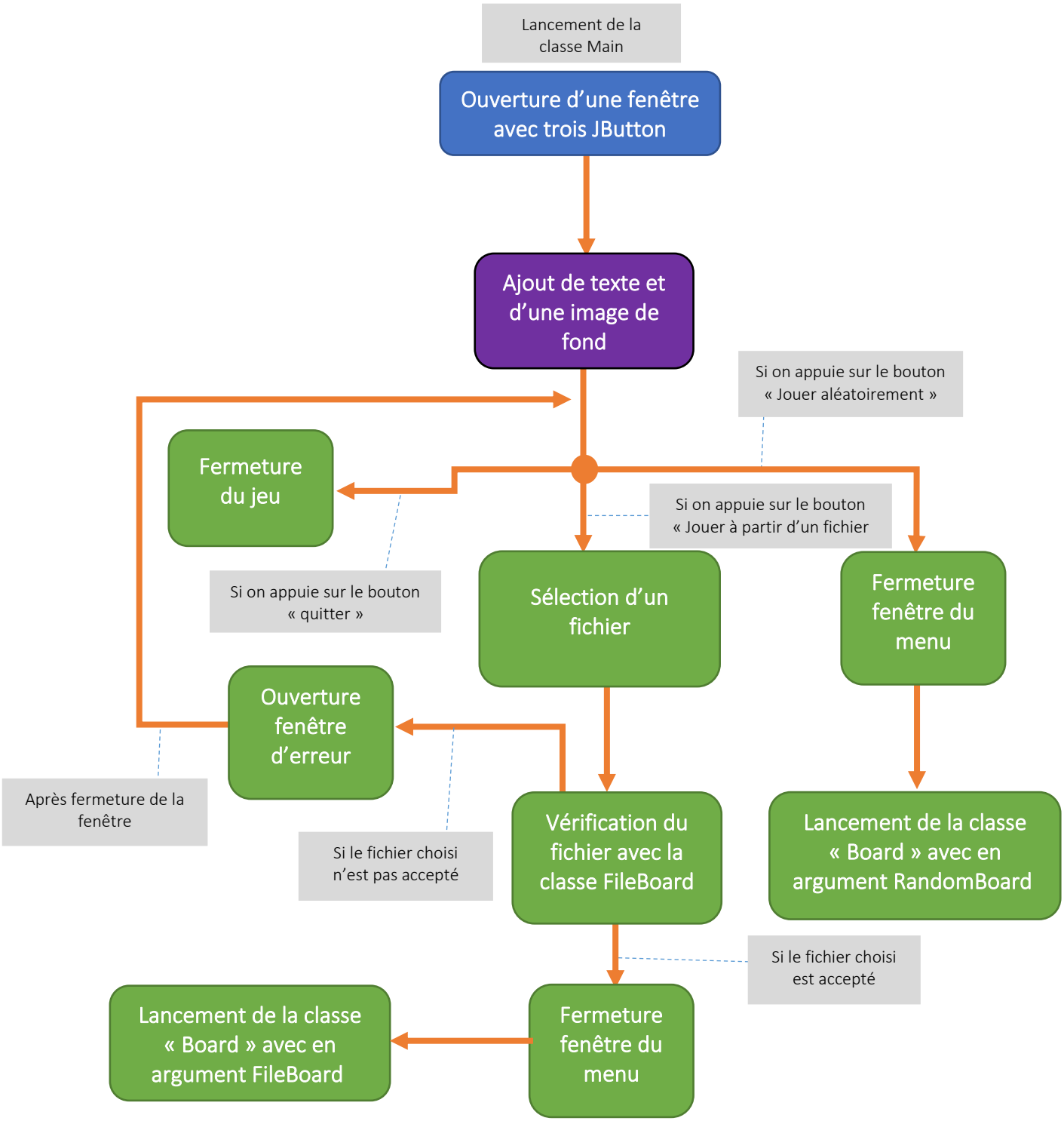


S'il ne reste aucun groupe d'élément, la partie se termine



Structure du programme

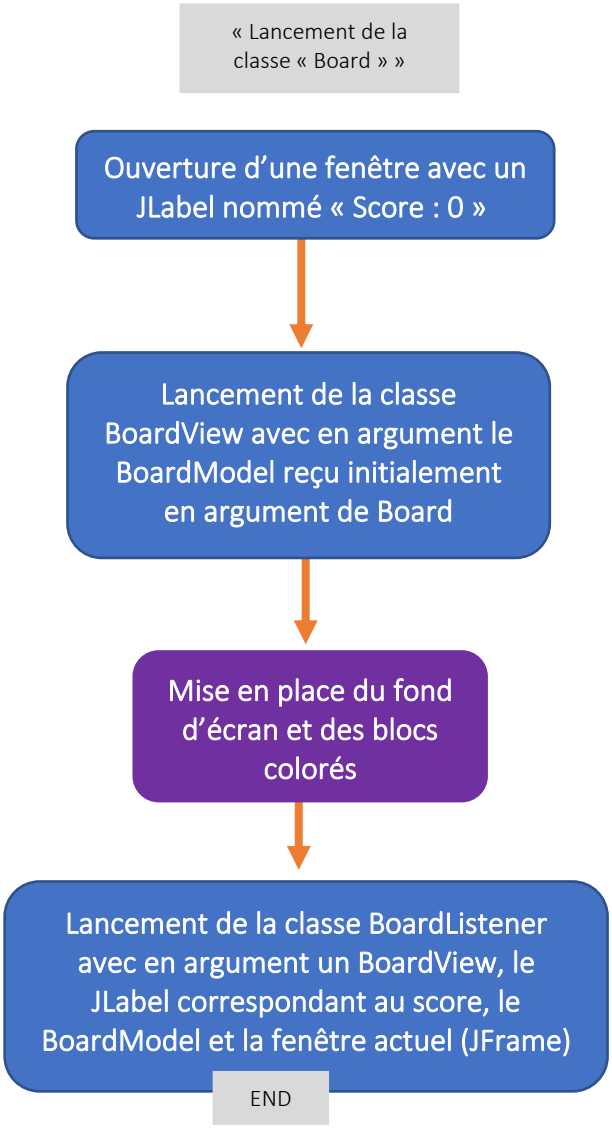
Commencement



Classe responsable de l'action :

Menu.java → ListenerMenu.java → MenuView.java →

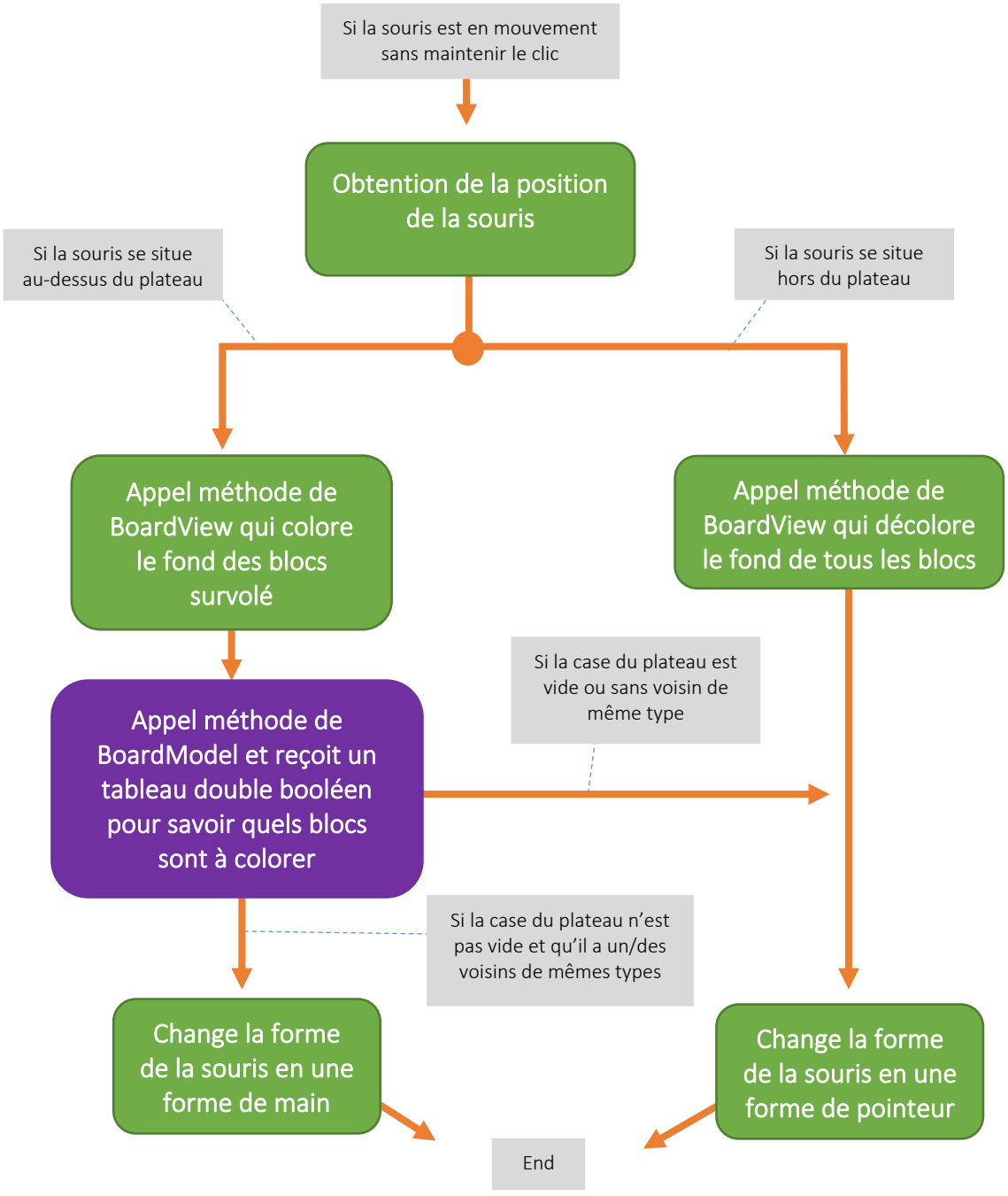
Affichage du jeu



Board.java →

BoardView.java →

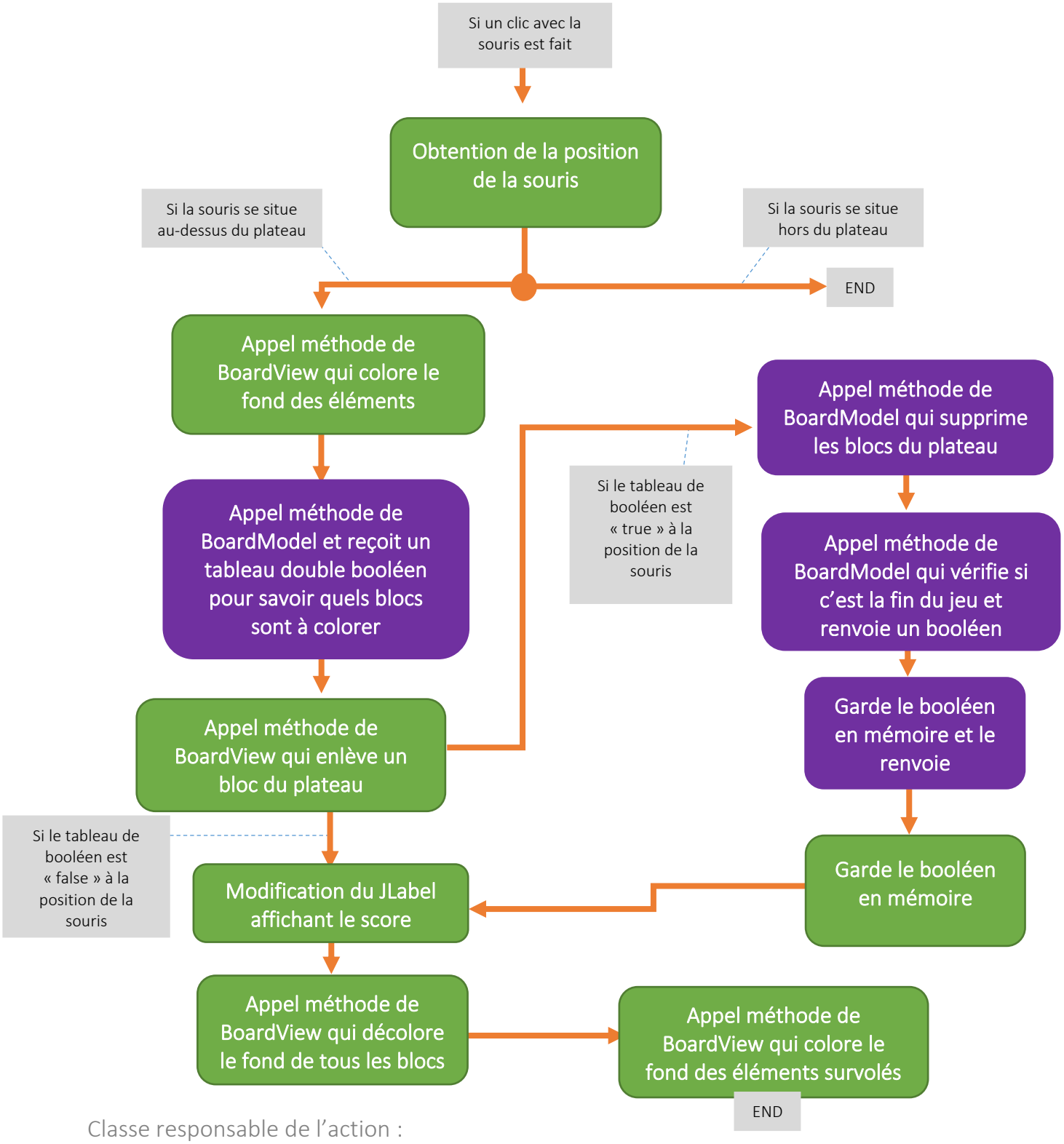
Gestion utilisateur



Classe responsable de l'action :

BoardListener.java →

BoardView.java →



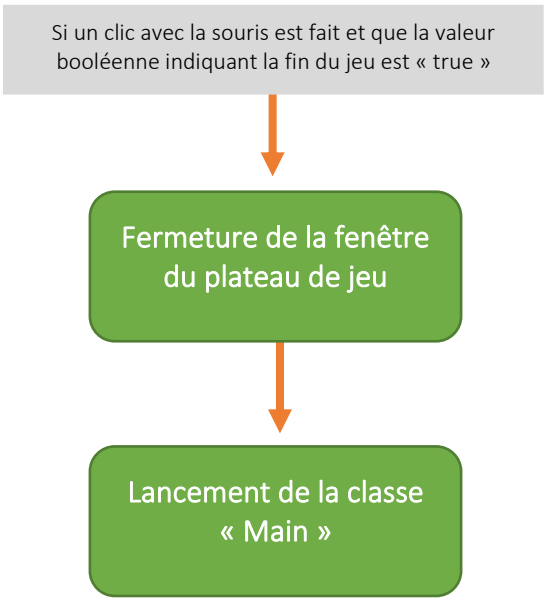
Classe responsable de l'action :

BoardListener.java →

BoardView.java →

Lorsqu'un clic est effectué et que des blocs ont été enlevés, il y a une vérification pour savoir si la partie est terminé ou pas à travers le renvoi d'un booléen qui provient d'une méthode de BoardModel.

Si le booléen envoyé est « false », la partie continue et les deux schémas représentés au-dessus restent valables. En revanche si elle renvoie « true », les actions après un clic changent.



BoardListener.java → 

Gestion Back-End

Principe

La classe « BoardModel » s'occupe des règles du jeu tels que la gravité ou le déplacement sur la gauche des colonnes vides. Elle gère aussi la fin du jeu, la suppression des groupes et le score. Cette classe est abstraite car elle ne fait que gérer un plateau déjà initialisé.

En effet le jeu possède 2 modes d'initialisation, par l'aléatoire ou par lecture d'un fichier. On a donc 2 classes « RandomBoard » et « FileBoard » qui héritent tous deux de « BoardModel ».

L'utilisation de l'héritage offert par Java nous a permis de travailler en binôme facilement. En effet l'héritage permet d'utiliser la substitution de Liskov. En effet quel que soit le choix de l'utilisateur, le programme se contente des méthodes offertes par héritage de « BoardModel »

Gestion de la gravité

On recherche colonne par colonne (de gauche à droite, donc du plus petit index au plus grand), puis pour chaque colonne on regarde un par un ses éléments (en commençant par le bas soit le plus grand index). Si on trouve un caractère espace alors le plateau contient un espace, il faut donc appliquer la gravité. Pour cela on copie les éléments ayant un index inferieur a cueilli de l'espace, vers l'index suivant. (On « décale » vers l'index supérieur les valeurs). Et on écrase la valeur du plus petit index par un espace.

Gestion du déplacement vers la gauche

D'après les règles s'il y a une colonne vide on doit déplacer les colonnes « orphelines » vers la gauche pour compléter le plateau.

Pour cela on boucle sur chaque colonne du plateau (on part du plus grand index, pour éviter de se « marcher dessus »). Si la colonne est vide, on copie toutes les lignes à gauche (d'index inferieur) vers le nouveau tableau, puis on copie toutes les lignes à droite (d'index supérieur) vers le nouveau tableau mais en diminuant leur index. Ainsi le vide est comblé par les éléments de droite.

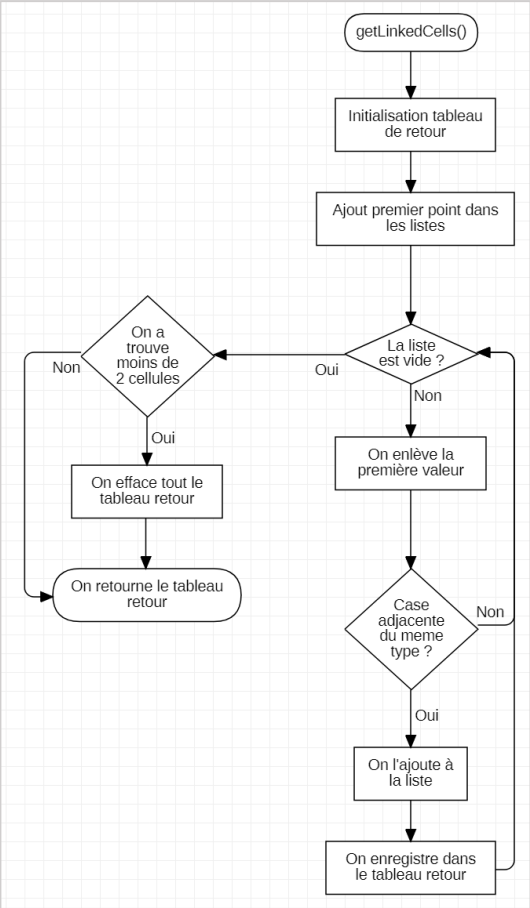
Gestion de la fin

Pour détecter la fin on cherche simplement s'il existe un groupe, pour cela on boucle sur chaque coordonnées (dans l'ordre d'index croissant) et on utilise l'algorithme d'indentification pour savoir si on est sur un groupe, si oui on retourne immédiatement la valeur pour éviter de perdre du temps à chercher plus loin.

Algorithme d'identification

Principe

- L'algorithme d'identification est situé dans la classe BoardModel.java



Voici un schéma simplifié de la logique de la méthode.

La méthode repose sur l'utilisation de deux listes sous la forme **L**ast **I**n **F**irst **O**ut. Ces deux listes stockent les coordonnées x et y des cases à vérifier.

Au début on initialise un tableau de boolean à false. Ce tableau indiquera par une valeur true, l'emplacement du groupe s'il y en a un.

Puis on ajoute les coordonnées passées en paramètre de la méthode aux listes.

Ensuite tant que les listes sont remplies, on retire la première valeur. On marque ces coordonnées dans le tableau retour. On vérifie ensuite si les cases adjacentes sont du même type. Si oui on ajoute les coordonnées de la case adjacente à la liste

A la fin on vérifie si le groupe est valide, s'il ne l'est pas on efface le tableau retour

Optimisations

- Une « liste chainée » a été préférée, car parcourir tout le tableau aurait été plus gourmand.
- L'algorithme n'ajoute jamais aux listes une cellule marqué par le tableau retour
- La liste utilise un tableau de taille fixe, car on connait la taille maximale du nombre de points pouvant être stockées.

Conclusions

Dylan Girault

Par rapport au premier projet, la gestion d'équipe était bien plus agréable car la création de classe permet de définir de façon simple les rôles de chacun sans la nécessité de comprendre la formation de la classe. Dès qu'un de nous avait besoin de quelque chose, l'autre n'avait qu'à faire une méthode pour l'obtenir ce qui rendit le travail bien plus pratique et limitant les incohérences ou les risques de bug.

Cependant, mettre des commentaires l'obligation de mettre des commentaires dans une certaine forme était un peu contraignant même s'il s'est avéré utile très rapidement lorsque j'avais besoin de chercher rapidement certains bouts de code. L'une des difficultés que j'ai rencontrées est pour le centrage des éléments que j'ai absolument voulu mettre pour rendre le jeu plus propre.

En résumé, j'ai trouvé ce projet bien plus enrichissant que le premier et agréable à créer.

Guillaume Toutain

Personnellement j'ai préféré ce projet, à celui du premier semestre car la programmation orienté objet permet une très grande flexibilité. Cela nous a permis de répartir le travail en Front et Back End. Pour ma part le Back End a été facilité par le mode « orienté objet » de Java. Il m'a suffi de faire mes classes qui gèrent le plateau, sans trop me soucier de comment les données sont affichées. Également si mon binôme à besoin d'une donnée supplémentaire (par exemple savoir si la partie est finie), il n'a pas à se soucier du fonctionnement de ma méthode, il lui suffit juste de l'appeler.

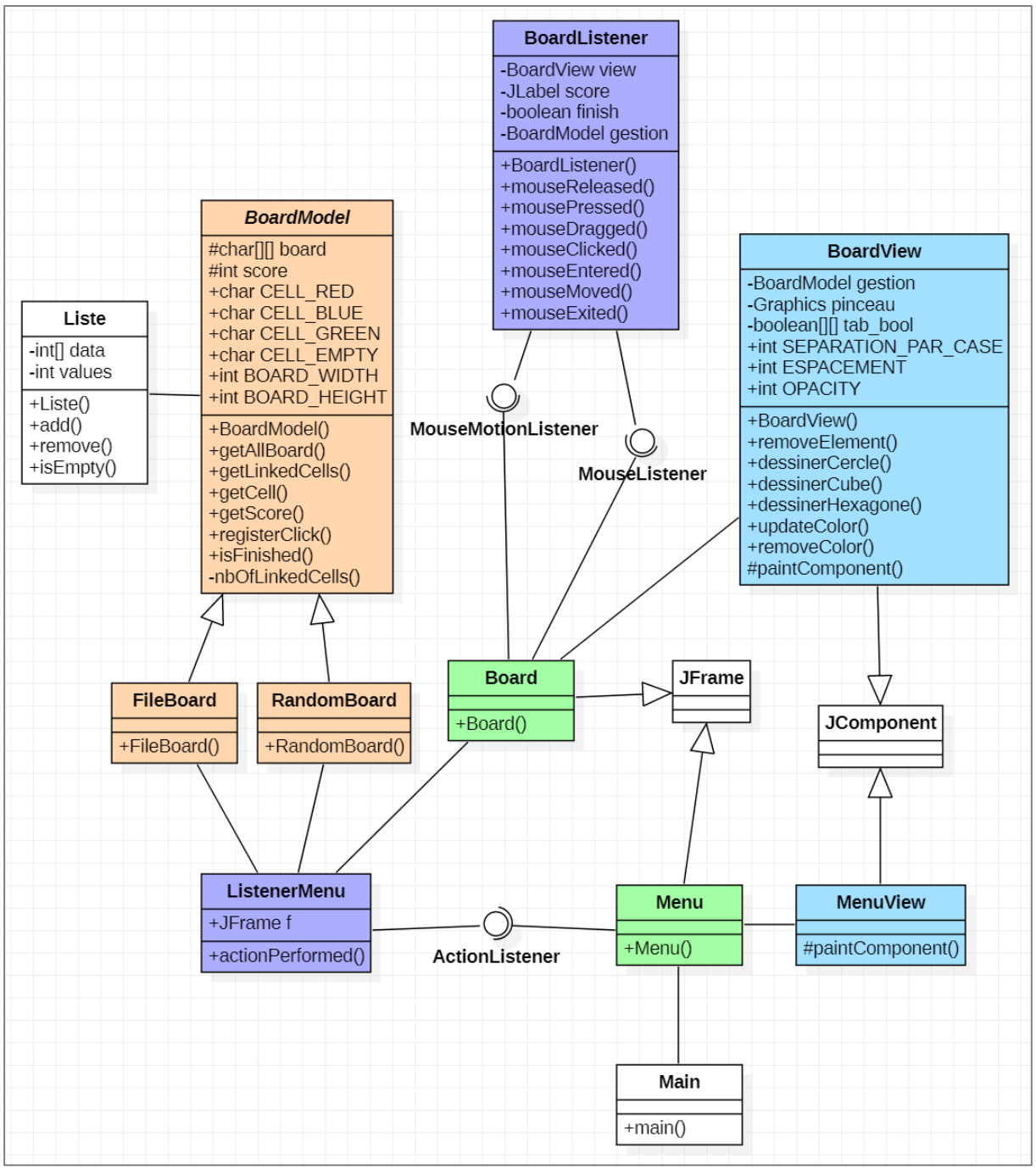
J'ai particulièrement apprécié l'utilisation d'une classe abstraite qui m'a permis de décliner mon model en 2 versions, une pour l'aléatoire et une autre pour le fichier.

Ma plus grosse difficulté sur ce projet à été la détection des groupes, même si je trouve ma solution assez efficace, elle pourrait être améliorée avec l'utilisation d'une vraie liste chainée.

Pour finir je rejoins l'avis de mon camarade sur le fait que j'ai également préféré ce projet.

Annexe

Diagramme général du programme



Légende :

Bleu	View
Violet	Controller
Orange	Model
Vert	Fenêtre