MVC - partie 2

1 Apprentissage

Lorsqu'une application devient complexe, son contrôleur prend de l'ampleur et devient difficile à maintenir.
On a alors intérêt à utiliser plusieurs contrôleurs, soit simultanément, soit en alternance.

1.1 GIT

Reprenez votre dépôt git et créez une branche "tp4" depuis la branche `master`.
Ajoutez-y le projet que vous trouvez sur eCampus. Toute ressemblance avec le TP de la semaine dernière n'est pas fortuite...

1.2 Les scènes multiples

Ce projet contient deux fichiers FXML, deux contrôleurs et deux modèles.

1. le plateau de jeu, son contrôleur et le modèle d'une partie.
2. la table des scores, son contrôleur et le modèle associé.

— Créez une instance de **Scores** dans la fonction `start`.
— Passez-la au constructeur de **GrilleController** (il faut bien sûr le modifier).
— Dans la méthode `onMenuTable`, chargez le fichier FXML "table.fxml".
— Récupérez le contrôleur **TableController** que **FXMLLoader** a créé automatiquement.
— Passez-lui la table des scores (méthode `setScores`).
— Modifiez la racine de la scène courante.
Pour cela, sachez que tout composant garde un lien vers sa scène, disponible avec `getScene()`.
Le contenu de la scène peut alors être modifié avec la méthode `setRoot()`.

Testez l'application et committez vos fichiers.

Lorsque la partie se termine, le contrôleur de jeu appelle sa fonction `onGagne` avec la pièce du joueur gagnant ou `null`. S'il y a un gagnant, ouvrez un **TextInputDialog** pour demander le nom du joueur victorieux. Sinon, c'est que la partie est nulle.
Mettez à jour la table des scores avec les méthodes `ajouteVictoire` ou `ajouteNulle`, et affichez la table des score.

Pour terminer, il serait bon de pouvoir revenir au jeu... Modifiez donc la méthode `onFermer` du **TableController**.
Vous noterez bien sûr que le contrôleur reçoit la table des scores dans les deux cas... mais pas de la même manière!
On aurait aussi pu utiliser un patron Singleton, que nous verrons l'an prochain...

Testez l'application et committez vos fichiers.

1.3 Les contrôleurs *nested*

Lorsqu'un fichier FXML est trop gros, on peut le casser en petits fichiers individuels, reliés par des `fx:include`. Chacun de ces fichiers peut avoir son propre contrôleur !

- Créez un nouveau fichier FXML "menus.fxml".
- Déplacez dedans la barre de menus, avec ses menus et sous-menus.
- Remplacez la barre de menus par un `<fx:include source="menus.fxml" fx:id="menus"/>` dans `grille.fxml`.
- Réparez l'entête du fichier FXML.
  On rappelle qu'un fichier XML n'a le droit qu'à une seule balise racine.
  Cette balise racine doit contenir les informations de *namespace* (attributs `xmlns`).
- Ajoutez-lui un contrôleur **MenusController**.
- Déplacez dans cette nouvelle classe les trois méthodes `onMenu*` du **GrilleController**.

Ceci va évidemment poser un petit problème, car le code a besoin du **GrilleModel** (pour faire une nouvelle partie) et du **Scores** (pour afficher la table des scores).

Heureusement, il est possible de référencer le contrôleur "interne" (*nested*) en ajoutant au **GrilleController** une variable d'instance `private @FXML MenusController menusController`.
Le nom de cette variable est constitué d'une part du `fx:id` du `fx:include` et d'autre part du mot "`Controller`".

- Ajoutez au **GrilleController** une référence au **MenusController**.
- Ajoutez dans le **MenusController** une méthode `setParams` qui lui permettra de recevoir le **GrilleModel** et le **Scores**.
- Appelez cette méthode dans le `initialize` du **GrilleController**.

Testez votre application, et commitez vos fichiers.

Une autre solution aurait été de référencer **GrilleController** depuis **MenusController**.
Dans un cas complexe, on préférera souvent avoir un contrôleur central qui référence tous les "petits" contrôleurs. Ainsi, les "petits" contrôleurs n'ont que le contrôleur central à connaître pour accéder à toute l'application.

2 Réutilisation

Revenez à la branche "gribouille_stable" et créez une nouvelle branche "gribouille_tp4".
Vous allez mettre en place dans ce TP les contrôleurs de Gribouille.

2.1 Contrôleur global

Ce contrôleur va centraliser les données de l'application.

- Créez un *package* "`controleurs`".
- Créez une classe Controleur et ajoutez-lui le **Dessin**, la figure courante et les coordonnées `precX` et `precY`.
- Ajoutez-lui une propriété `couleur` (`new SimpleObjectProperty<Color>(Color.BLACK)`).
- Ajoutez-lui une propriété `epaisseur` (`new SimpleIntegerProperty(1)`).

Pour des raisons de facilité, tous les attributs seront `public final`, sauf la figure courante qui ne peut pas être `final`.

2.2 Le fichier FXML

Subdivisez le fichier **Gribouille.fxml** en 4 sous-parties :

menus.fxml la barre de menus (en haut de fenêtre)

statut.fxml la barre de statut (en bas de fenêtre)

couleurs.fxml le panneau de gestion des couleurs (à droite de la fenêtre)

dessin.fxml le panneau central (au milieu de la fenêtre)

Créez bien sûr un contrôleur pour chacun de ces fichiers.
Associez le contrôleur global au fichier `Gribouille.fxml`, et ajoutez-lui les 4 contrôleurs internes.

Répartissez les différentes variables `@FXML` dans leurs contrôleurs respectifs. On les laissera publiques pour des raisons de facilité.

Pour finir cette partie, ajoutez une variable d'instance **Controller** à chaque contrôleur interne, avec son *setter*, et appelez les depuis le `initialize` de **Controleur**.

2.3 Restaurer les fonctionnalités

Il va maintenant falloir remettre l'application en état... Cette partie est assez longue, n'hésitez pas à utiliser une branche temporaire pour *commiter* vos modifications au fur et à mesure!

2.3.1 le menu

Déplacez les méthodes associées aux menus dans son contrôleur.
Vous ajouterez dans le FXML et dans son contrôleur une méthode `onQuitte` qui appellera une nouvelle méthode `onQuitter` de **Controleur**.
Cette méthode recevra la demande de confirmation qui se trouvait dans la méthode `start` et retournera vrai si la fenêtre doit se fermer, ce qui autorisera l'appel à `Platform.exit()` depuis le contrôleur de menus.

Déléguez le `onCloseRequest` de la méthode `start` au contrôleur, en gardant le `evt.consume()` sur place.

2.3.2 le dessin

- Déplacez la liaison entre le **Canvas** et son **Pane** dans **DessinController**.
- Définissez une méthode `efface()` et une méthode `trace(x1, y1, x2, y2)` qui appellent respectivement les méthodes `clearRect` et `strokeLine` du contexte graphique du **Canvas**.
- Créez les méthodes `onMousePress`, `onMouseMove` et `onMouseDrag`, et appelez une méthode du même nom dans **Controleur**.
- Déplacez les 3 méthodes en question ainsi que `dessine()` de **GribouilleController** vers **Controleur**.
- Remettez en place le dessin du **Canvas** lorsqu'il est redimensionné.

2.3.3 la barre d'état

Dans la méthode `initialize` de **Controleur**, déplacez la liaison des propriétés `precX` et `precY` avec les labels de la barre d'état.
Ajoutez-y une liaison similaire vers les propriétés `epaisseur` et `couleur`.

2.3.4 le test

Supprimez la classe vide **GribouilleController** et testez votre application.
-> N'oubliez pas d'ouvrir le *package* `controleurs` dans le **module-info**!

Committez vos fichiers! (félicitations, c'était un gros morceau!)

2.4 Les outils

Comme vous avez pu le voir dans les menus et dans la barre d'état, deux outils "crayon" et "étoile" sont prévus. Il y a d'ailleurs deux types de **Figure** dans le modèle... Une fois encore, les modifications que vous allez effectuer vont casser l'application avant de la réparer. Une branche temporaire est donc toute indiquée.

- Créez une classe abstraite **Outil** dans laquelle vous déclarerez deux méthodes `onMousePress` et `onMouseDrag`.
- Donnez lui une variable d'instance `protected` **Controleur** qui sera reçue par son constructeur.
- Créez ses deux classes filles **OutilCrayon** et **OutilEtoile**.
- Dans **OutilCrayon**, copiez le contenu des méthodes éponymes de **Controleur**, sauf les coordonnées X et Y. La figure courante pourra être déplacée dans l'outil...
- Dans **OutilEtoile**, remplissez les méthodes de façon à ce que le glissé de la souris dessine des rayons depuis le centre de l'étoile (où le bouton a été enfoncé) vers chacun des points de la figure.
- Dans **Controleur**, créez une variable de type **Outil** (initialisée à **OutilCrayon** et déléguez-lui le code des fonctions liées à la souris après avoir lis à jour les variables `precX` et `precY`.

Testez votre application. Le comportement doit être le même qu'avant.

- Dans `menus.fxml`, ajoutez un `fx:id` aux **RadioMenuItem** "Crayon", "Étoile" ainsi qu'à leur **ToggleGroup**.
- Dans **MenusController**, créez les variables d'instance liées à ces trois `fx:id`.
- Dans la méthode `initialize` de **MenusController**, ajoutez à la propriété **ToggleGroup** ci-dessus un **ChangeListener**.
- Dans ce *listener*, si la nouvelle valeur est le menu "crayon", appelez la méthode `onCrayon` de **Controleur**.
- Similairement, appelez `onEtoile` si la nouvelle valeur est le menu "étoile".
- Dans **Controleur**, définissez les deux méthodes et changez-y l'outil courant.
- Profitez-en pour modifier le texte du **Label** outil dans la barre d'état!

Testez votre application et commitez vos fichiers!
Félicitations, vous venez de créer une application qui change de contrôleur souris à la demande de l'utilisateur!
Vous verrez l'an prochain que vous venez de mettre en place une *stratégie*...

Il reste bien sûr à corriger la méthode `dessine` de **Controleur** pour redessiner les étoiles...
Pour le moment, un `instanceof` suffira pour différencier les deux types de figures.
L'an prochain, vous verrez qu'on peut utiliser pour cela un *visiteur*...

3 Rendu

N'oubliez pas de commiter vos fichiers régulièrement, de fusionner vers la branche stable et de faire un *push* à la fin!