Révisions

1 Apprentissage

L'objectif de cette séance est de faire le point sur ce que vous avez appris. Il ne devrait donc pas (trop) y avoir de nouveauté(s)…

1.1 GIT

Reprenez votre dépôt git et créez une branche "tp5" depuis la branche `master`.
Ajoutez-y le projet que vous trouvez sur eCampus. Il correspond à un jeu de démineur dont je fournis le modèle.

N'oubliez pas de tester votre application et de *commiter* vos fichiers régulièrement!

1.2 La vue

La vue de l'application est assez simple. Construisez-la avec :
- un **BorderPane**
- une **VBox** centrée en haut, avec :
  - une **MenuBar** avec :
    - deux **Menu** "Jeu" et "Difficulté"
    - un **MenuItem** "Quitter"
    - trois **RadioMenuItem** "Facile", "Moyen" et "Difficile".
      On associera à ces trois balises un **ToggleGroup** commun et des `userData` respectivement égaux à "`10;10;10`", "`20;20;50`" et "`30;20;100`".
      Note : tous les composants Java-Fx peuvent contenir un `userData` à usage personnel…
      Attention, l'attribut `userData` ne peut pas être renseigné dans SceneBuilder et doit être mis directement dans le fichier FXML ou dans le code Java.
  - une **HBox** avec :
    - deux **Label** "Nombre d'inconnues" et "Nombre de marques" et les deux **TextField** associés.
      Les deux **TextFields** ne seront ni `editable` ni `focusTraversable`.
- un **GridPane** au milieu, sans ligne ni colonne, avec son `gridLinesVisible` positionné à `true`.

N'oubliez pas de donner un `fx:id` à tous les éléments avec lesquels on va interagir…(**ToggleGroup**, **TextField**, **GridPane**)

1.3 Le contrôleur

- Créez un contrôleur **ControleurDemineur** qui sera **Initializable** ; ajoutez lui une instance du **ModeleDemineur** et importez les composants graphiques recensés ci-dessus.
- Liez la propriété `text` des deux **TextField** aux propriétés `nbInconnues` et `nbMarques` du modèle.
  N'oubliez pas que la méthode `asString()` permet de convertir une propriété quelconque en **StringExpression**.
- Ajoutez à la propriété du **ToggleGroup** un observateur (*listener*) qui appellera une nouvelle méthode "initGrille" avec le `userData` du **Toggle** choisi.

1.4 La grille

Cette partie est assez difficile, car la grille est dynamique : elle change de taille en fonction de la difficulté choisie.
- Commencez par supprimer les contraintes de lignes de de colonnes de la grille. (méthodes `clear`).
- Découpez le `userData` en trois entiers (la fonction **ModeleDemineur**`.parseUserData` fait déjà le travail) : le nombre de colonnes, de lignes, et de mines à placer dans la grille.
- Créez/Modifiez un **ModeleDemineur** avec ces paramètres.
- Créez une contrainte par ligne et par colonne, avec une largeur de 32 pixels.

Testez votre application, et *commitez* vos fichiers.
Essayez différents niveaux de difficulté et voyez la grille changer d'aspect.

1.5 La grille (suite)

Il est maintenant nécessaire de créer les cases de la grilles.
Chaque case va être remplie par un **Label** qui aura un texte et un fond selon son statut. Préparez donc quatre fonds (**Background**) basés sur un **BackgroundFill** différent :

inconnu une couleur claire (AQUA, par exemple) avec des coins arrondis (20% par exemple) pour les cases non révélées et non marquées (texte " ?")

libre une couleur foncée (LIGHTGRAY, par exemple) sans arrondi, pour les cases révélées et sans mine (texte "0"..."8")

echec une couleur flashie (RED, par exemple) pour la case révélée qui contient une mine (fin de partie, texte "X")

marquée une couleur neutre (LEMONCHIFFON, par exemple) pour les cases marquées d'un drapeau (texte "P")

Pour chaque case, créez ensuite un **Label** :

- avec une taille préférée de $31x31$
- avec un fond inconnu
- avec un texte centré
- avec un texte lié au texte de la case correspondante du modèle
- avec un *handler* pour l'événement `MOUSE_CLICKED` :
  - qui appelle la méthode `revele` du modèle si le bouton primaire est enfoncé
  - qui appelle la méthode `marque` du modèle si le bouton secondaire est enfoncé
  - qui met à jour le fond du **Label** selon le résultat obtenu.

Testez votre application.
Vous noterez qu'on ne peut pas marquer une case révélée, et qu'on ne peut pas jouer après avoir perdu.
Corrigez votre *handler* pour obtenir un fonctionnement satisfaisant.

Commitez et *pushez* vos fichiers.

2 Réutilisation

Revenez à la branche "gribouille_stable" et créez une nouvelle branche "gribouille_tp5".
Vous allez aujourd'hui compléter l'application avec un nouveau contrôleur et deux fonctionnalités :
Le clavier, l'épaisseur et la couleur.

2.1 L'épaisseur

Dans le menu "outils" se trouve un sous-menu "épaisseur" dans lequel sont rangés 9 **RadioMenuItem** associés à un même **ToggleGroup**.

- Associez un *listener* au groupe de façon à modifier la propriété `epaisseur` du **Controleur**.
  (Créez une méthode `setEpaisseur` dans **Controleur**).
- Lors de la création d'une nouvelle figure, utilisez cette épaisseur.
- Modifiez également le **DessinController** en lui ajoutant une méthode `setEpaisseur` qui appellera la méthode `setLineWidth` du contexte graphique du **Canvas**.
- Ajoutez l'appel à cette fonction aux méthodes de dessin (les deux outils et `dessine`).

Testez votre application et *Commitez* vos fichiers.

2.2 La couleur

La couleur du tracé est définie par le panneau latéral. On ignorera pour le moment le **ColorPicker**.

— Ajoutez un *handler* pour l'événement `MOUSE_CLICKED` de la **VBox**.
— Testez si le `target` de l'événement est bien un **Rectangle**.
— Appelez la (nouvelle) méthode `setCouleur` du **Controleur** en lui passant la couleur du rectangle cliqué.
— Modifiez le rectangle pour avoir des `arcWidth` et `arcHeight` à 10 au lieu de 5.
— Modifiez le rectangle pour avoir un `strokeWidth` à 5 au lieu de 1.
— Remettez le rectangle précédent à ses attributs d'origine

Testez votre application.

Il reste bien sûr à tenir compte de cette couleur...

— Ajoutez une méthode `setCouleur` au **DessinController**.
Elle appellera la méthode `setStroke` du contexte graphique du **Canvas**.
— Modifiez la construction des figures et les fonctions de dessin pour utiliser la bonne couleur.
Attention, les couleurs sont stockées sous forme de **String**. Il faudra donc utiliser les méthodes **Color**.`valueOf` et `toString` pour faire la conversion.

Testez votre application et *commitez* vos fichiers.

2.3 Un contrôleur clavier

Changer les outils par le menu n'est pas efficace. C'est plus rapide au clavier :

— Ajoutez à la fenêtre (**Stage**) un *handler* pour l'événement `KEY_PRESSED`.
— Appelez la (nouvelle) méthode `onKeyPressed` du **Controleur** en lui passant le `text` de l'événement.
— Dans cette méthode, un gros `switch` pas beau permettra de changer d'épaisseur, de couleur ou d'outil en fonction de la touche enfoncée.
On verra l'an prochain une utilisation du patron *commande* permettant de se débarrasser de ce `switch`...

Testez votre application et *commitez* vos fichiers.

2.4 Changement d'outil en cours de dessin - difficile

L'avantage du clavier est que l'on peut changer d'outil, de couleur ou d'épaisseur sans interrompre le tracé!

Cela pose cependant un souci : Il faut créer une nouvelle figure à la volée.
La classe **Figure** propose pour cela les méthodes `changeCouleur` et `changeEpaisseur` qui font le travail. Il faut cependant encore configurer l'outil pour changer de figure, et le dessin pour stocker celle-ci!

Le changement d'outil demande la modification de l'outil en cours, mais aussi la création d'une nouvelle figure en cours pour cet outil.

Ces fonctionnalités ne sont pas essentielles pour la suite. Elles représentent toutefois un challenge intéressant pour les étudiants qui veulent progresser!

3 Rendu

N'oubliez pas de *commiter* vos fichiers régulièrement, et de faire un *merge* (→ branche stable) et des *push* à la fin!