Manipulations de base, quelques composants, et utilisation de panneaux : suite.

Attention lorsque vous programmez, apportez un soin tout particulier à la «propreté» de votre code : déclarez ce qu'il faut à l'endroit qu'il faut, utilisez à bon escient l'héritage et les variables statiques, créez les classes et les méthodes qu'il faut de manière à éviter la duplication de code, etc...

Gardez toujours une fenêtre ouverte sur la documentation JavaFX de la classe que je vous demande d'utiliser, ou que vous manipulez. Il faut absolument balayer cette documentation, vous aurez un aperçu des possibilités de la classe, et ensuite vous trouverez plus facilement les propriétés et méthodes dont vous aurez besoin. Vous vous repèrerez également plus facilement dans la hierarchie des classes javaFX.

Exercice 7 Border et Box.

1. On voit sur la figure 1 une représentation d'un conteneur dont le panneau principal est un `BorderPane`. Écrire une application qui reproduise cette fenêtre avec 5 Strings différents. Pour information :
   - la plupart des panneaux ou composants élémentaires sont encadrés pour être visibles (mais évidemment, on ne voit pas tout) ;
   - chacun des textes composés normalement sont placés dans un Label et centrés dans ce composant ;
   - les textes composés verticalement sont placés dans une boîte verticale.

   Tout cela doit être placé de manière automatique, indépendamment du texte que l'on voit sur la figure, et du redimensionnement de la fenêtre.

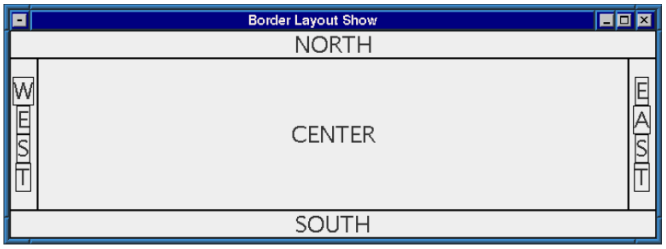

Figure 1: des piles de lettres

2. Sur la figure 2 on voit le même type d'agencement mais la zone centrale du `BorderPane` est composée de manière bien plus complexe. Sans aide particulière (i.e. sans layout-pane ad hoc), le travail est diffile : ou bien placer «à la main» avec des `setLayoutX`, et `setLayoutY` les différentes lettres du texte placé au centre, on peut avoir besoin pour cela de récupérer les dimensions ou positions avec les méthodes `getHeight`, `getWidht`, etc des différents composants (panneaux ou Label) ; ou bien utiliser des boîtes et «ressorts» comme on en a fabriqué dans la question Exo4.Q3 de la feuille précédente. Donc ceci n'est à faire que pour ceux qui sont en avance.

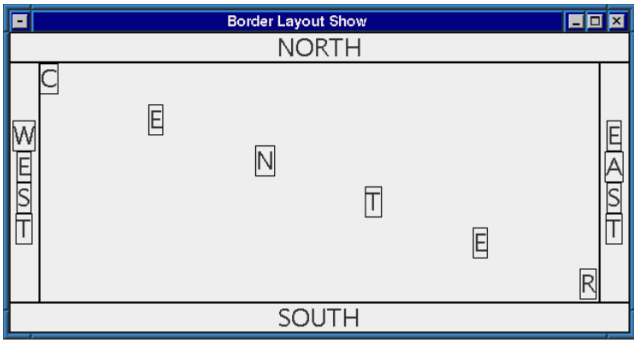

Figure 2: des lettres décalées

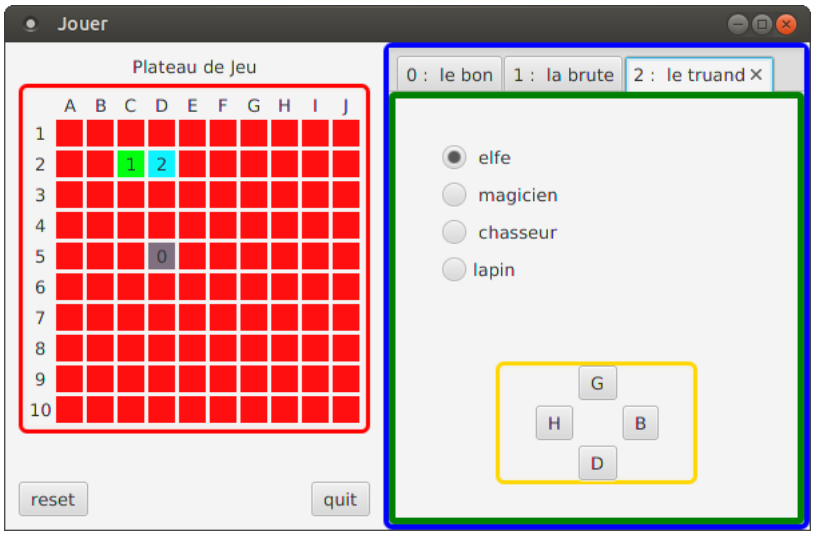

Figure 3: une petite interface de jeu

Exercice 8 GridPane, RadioButton et ToggleGroup, TabPane et Tab.

Le but de cet exercice est au final de faire l'interface présente dans la figure 3. Cette interface fournira un ensemble de méthodes permettant de la manipuler : afficher un caractère dans une case du plateau, sélectionner un rôle pour un joueur, récuperer le bouton UP d'un joueur, selectionner un onglet, etc. L'API demandée ici pour cette interface, ainsi que pour les composants interne est minimale, on pourrait la compléter si besoin, mais elle suffit ici.

Pour chacune des questions, vous ferez bien de tester le composant concerné avec une fenêtre ne contenant que ce seul composant. Par ailleurs, vous aurez souvent besoin de "ressorts" pour remplir tout l'espace entre composants dans une Box, nous avons vu dans l'exercice 4 comment le faire et pour ces ressorts je vous conseille de créer une classe `Spring` héritant de Region, vous gagnerez du temps si vous devez utiliser souvent ce mécanisme dans cet exercice ou dans un autre.

1. *Board* : le plateau (panneau avec bordure rouge).

   Ce composant hérite d'un `GridPane` dans lequel vous placerez des `Label`. Vous réaliserez ainsi le plateau de jeu que l'on voit sur la figure 3, chaque case (Label) du plateau pourra contenir un caractère sur un fond dont on pourra choisir la couleur. Une case vide sera affichée en rouge. Penser à fixer une taille minimum aux labels.

   • Écrire cette classe `Board` qui fournira :
   - le constructeur `Board(int size)` construisant un damier de la taille souhaitée,
   et les méthodes :
   - `setCell(char val, int x, int y, String numberColor)` qui positionne le caractère `val` avec la couleur `numberColor` (sous format CSS) à la position (x, y) du damier,
   - `resetCell(int x, int y)` qui vide la case (x, y) du damier.

   • Essayer de faire la même chose en utilisant un `TilePane`. Que devient le panneau lorsqu'on redimensionne le composant ?

2. *Direction* : un panneau de commande (panneau avec bordure gold).

   Ce panneau contient 4 boutons placés comme sur la figure. Si vous le souhaitez vous pouvez mettre une image sur les boutons à la place du texte, il suffit pour cela de :
   1) trouver 4 images de flèches (on peut utiliser une seule image, mais dans ce cas, il faudra la transformer pour 3 des boutons), et les placer dans un répertoire (par exemple `Icons`) dans le même répertoire que vote fichier java qui les utilise.
   2) créer l'image `Image imageF = new Image(getClass().getResourceAsStream("Icons/fleche.jpg"))` (attention, le chemin peut-être différent suivant si l'on est sous windows ou Linux),
   3) créer l'icone `ImageView iconeF = new ImageView(imageF)`
   4) enfin créer votre bouton avec un texte vide `Button b = new Button ("", iconeFleche)`, on peut également positionner l'icone avec un `b.setGraphic(iconeFleche)`

• Écrire une classe `Direction` qui héritera de ce qui vous semble le plus pratique (Box, GridPane, Pane ?). Cette classe devra fournir les accesseurs `getButtonUp`, `getButtonDown`, `getButtonLeft`, `getButtonRight`, retournant respectivement les boutons haut, bas, gauche et droit.

3. *RoleChoice* : des boutons radio pour choisir le rôle du joueur.

   RoleChoice hérite de VBox et contient des `RadioButton` que l'on peut cocher. On souhaite que la sélection d'un des boutons déselectionne tout autre bouton sélectionné : cela se fait automatiquement en mettant ces boutons radios dans un `ToggleGroup`. Il faut donc :
   1) créer un `ToggleGroup tg = new ToggleGroup();`
   2) créer les boutons radio, par exemple `RadioButton rb1 = new RadioBouton(role1);`
   3) dire que chaque radiobouton fait partie de ce toggleGroup (avec `rb1.setToggleGroup(tg)`).
   Le ToggleGroup assure ensuite qu'un seul radiobouton de son groupe est selectionné à la fois.

   • Écrire la classe `RoleChoice` qui fournira :
   -le constructeur `RoleChoice(String roles[])` ayant en parametre les différents types de personnage.
   et les méthodes
   - `void setRole(int i)` permettant de selectionner le $i^{ème}$ RadioButton,
   - `String[] getRoles()` retournant la liste des différents rôles possibles pour ce panneau,

4. *PlayerTab* : le panneau avec bordure verte.

   Cette classe héritera d'une VBox, elle contient un panneau permettant de choisir un rôle ainsi qu'un panneau de déplacement.

   • Écrire une classe `PlayerTab` fournissant :
   - un constructeur `PlayerTab(String roleNames[])` avec les différents rôles possibles,
   et les méthodes :
   - `void setRole(int i))` permettant de sélectionner le $i^{ème}$ RadioButton
   - `void setRoleChoiceDisable(boolean b)` permettant de désactiver le choix du personnage,
   - les différents `Button getButton`Xxx`()` retournant les boutons up, down, left right de ce panneau.

5. *PlayerTabs* : un ensemble d'onglets, le composant avec bordure bleu.

   PlayerTabs hérite de TabPane, un TabPane contient des onglets (`Tab`) ; chaque onglet contenant un PlayerTab (composant avec bordure verte programmé précédemment). Il faut donc :
   1) créer le panneau d'onglets `TabPane tp = new TabPane();` (ou `super()` si on hérite)
   2) créer autant d'onglets que nécessaires : `Tab onglet = new Tab("Mon onglet");`
   3) remplir l'onglet avec le noeud (Graphics, Control, Pane, ou autre...) désiré ; cela se fait avec : `onglet.setContent(...)`
   4) et enfin mettre l'onglet dans le TabPane avec l'appel `getTabs().add(onglet)`

   L'utilisateur pourra choisir lui même l'onglet, mais on peut faire cela depuis l'application avec `tp.getSelectionModel().select(i)` qui permet de selectionner le $i^{ème}$ onglet d'un tabPane `tp`.

   Attention ! Un `TabPane`, malgré son nom, et le fait qu'il puisse contenir d'autres noeuds, hérite de `Control` et pas de `Pane`. Quant au `Tab`, bien qu'il contienne un composant (ici ce sera un PlayerTab), il n'hérite pas de `Pane` mais directement d'Object) ! Comme d'habitude, je vous laisse consulter toutes les propriétés et méthodes de `TabPane` et `Tab` dans la documentation officielle d'Oracle.

   • Écrire la clase `PlayerTabs` fournissant :
   - le constructeur `PlayerTabs(String playerNames[], String roleNames[])` dans lequel on fournit les différents noms des personnages, ainsi que les différents rôles qu'ils peuvent avoir.
   et les méthodes
   - `void setSelected(int i)` permettant de selectionner un onglet précis,
   - `void setPlayerRole(int i, int role)`, permettant de choir un rôle sur l'onglet i,
   - `void setPlayerRoleChoiceDisable(int i, boolean b)` permettant d'activer/désactiver le choix du rôle pour l'onglet i,
   - les quatre accesseurs `Button getPlayerButtonxxx(int i)` permettant de récupérer les boutons Up, Down, Left et Right de l'onglet i.

6. *GamePane* : le panneau contenant le jeu complet.

   Le plus difficile est maintenant fait, un GamePane contient sur la partie gauche un label, un Board, et deux boutons reset et exit ; à droite un trouve un playerTabs.

   • Écrire la classe `GamePane` fournissant :
   - le constructeur `GamePane(int size, String playerNames[], String roleNames[])` permettant de

choisir la taille du plateau, les différents personnages, et les différents rôles possibles
et les méthodes permettant de :
- selectionner un onglet précis,
- selectionner un rôle sur un onglet,
- désactiver le choix du rôle sur un onglet,
- afficher un caractère sur une case du plateau,
- réinitialiser une case du plateau,
- récupérer les boutons reset et exit,
- récupérer les différents boutons d'un onglet.

7. Vous pouvez maintenant afficher l'interface de votre jeu. On s'en servira plus tard.

Exercice 9 une rampe paramétrable.

En utilisant des `Labels`, des panneaux, et des marges correctement choisies, faire la construction que l'on voit sur la figure 4.

Le programme récupére le nombre de cases N à placer sur la ligne de commande. L'alignement fait que le sommet, de la première «boîte» que l'on voit est sur la même ligne horizontale que la base de la dernière boîte, les autres boîtes étant réparties régulièrement ; l'ensemble est indéformable (on ne peut redimensionner la fenêtre). POur le reste on voit dans les «boites» les nombres écrits de 0 à $N-1$ écrits sur deux chiffres, le chiffre de poids fort en haut, et celui de poids faible en bas.

Exercice 10 des ComboBox.

On voit sur la figure 5 un panneau permettant d'effectuer des conversions entre deux systèmes d'unités. Dans un premier temps il s'agit d'écrire une petite application JavaFx qui reproduit le placement de différents composants.

Les contraintes à respecter sont les suivantes : l'arrangement des composants est constitué de trois parties : gauche, milieu et droite. Les parties de gauche et de droite sont tout à fait analogues. Dans la partie de gauche ou celle de droite :
- tous les composants sont alignés (au fer) à gauche, séparés verticalement d'une dizaine de pixels environ ;
- la *ComboBox* contient un certain nombre d'items représentant des unités ;
- dans le champ de saisie en bas, il est agréable que les nombres soient formattés (i.e. représentés toujours de la même façon, ici avec 2 chiffres après la virgule) ;

Dans la partie centrale : les boutons de conversion seraient bien plus jolis (et certainement plus petits) si l'on y plaçait une image de texte au lieu du texte que l'on voit. Ces deux boutons sont centrés verticalement dans l'espace disponible, et séparés par un espacement fixe.

Enfin, et c'est très important, lorsqu'on redimensionne la fenêtre, l'arrangement des composants ne change absolument pas. On peut faire le placement avec des boîtes, mais il existe certainement d'autres solutions.

Figure 4: une rampe de numéros verticaux

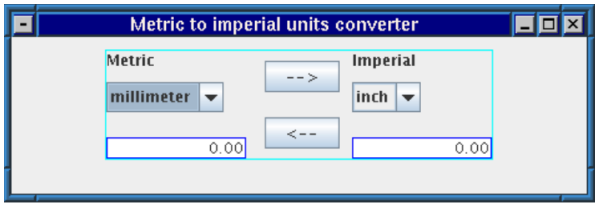

Figure 5: Placement des composants d'un convertisseur