Langage Orienté Objet / Java

Partie 7 – JavaFX

Avant de continuer

Cette partie vous demandera de créer des interfaces graphiques en JavaFX et d'en gérer les évènements. Du coup, avant de continuer la programmation du jeu, lisez le cours et assurez-vous d'avoir bien compris les notions abordées.

The Learning Souls Game

Ce TD doit être réalisé à partir du code créé dans la partie précédente.

Nous avons bien avancé sur les mécanismes internes au jeu et il est grand temps que nous lui fournissions une interface graphique ! Nous allons la baser sur JavaFX, qui va nous fournir les widgets nécessaires, faciliter leur organisation à l'écran, et nous fournir les moyens de réaliser des animations basées sur des sprites (successions d'images de type « dessin animé »).

L'organisation générale que nous allons adopter consistera à créer une application JavaFX (qui remplacera notre fameux programme principal **LearningSoulsGame**) contenant une fenêtre et une scène sur laquelle nous superposerons un ensemble de « calques » avec chacun leur fonction : un titre animé, un splash screen permettant au joueur de donner son nom, un HUD (Head Up Display) qui présentera les informations principales en cours de jeu (barres de vie, de stamina, compétences, score, etc.) et un panneau d'animation dans lequel les représentations graphiques des personnages seront animées.

Il est à noter que cette organisation est arbitraire (le but étant d'organiser au mieux le TP pour comprendre les concepts objets sous-jacents) et que, contrairement à ce que nous devrions faire dans un réel développement de jeu, nous ne nous intéresserons pas beaucoup aux optimisations (poids des fichiers, performances d'animations, etc.).

Il faut enfin souligner que les ressources graphiques (png, fonts, etc.) proposées pour réaliser ce TP sont des versions de démonstration (freebies, etc.) utilisables mais qui restent sous licence (les fichiers d'information vous seront fournis avec les ressources).

1. LearningSoulsGameApplication

Notre application principale sera à présent gérée par la classe **lsg.LearningSoulsGameApplication** créée sous la forme d'une sous-classe de **javafx.application.Application**, selon la norme JavaFX.

1.1. Créez la classe **lsg.LearningSoulsGameApplication** en tant que sous-classe de **javafx.application.Application**

- Créez un attribut privé **scene** de type **javafx.scene.Scene** destiné à recevoir une référence vers la scène de l'application.
- Créez un attribut privé **root** de type **javafx.scene.layout.AnchorPane** (cf. doc)

1.2. Implémentez la méthode **start :**

- Donnez un titre à la fenêtre (**stage**) : "Learning Souls Game"
- Instanciez **root**
- Instanciez **scene** en la liant à **root** et en fixant ses dimensions à 1200x800 px.
- Liez la **scene** à la fenêtre (**stage**)
- Faites en sorte que la fenêtre ne soit pas redimensionnable
- Faite appel à une méthode **buildUI()** (qui n'existe pas encore…)
- Faites afficher la fenêtre grâce à **stage.show()**

1.3. Créez la méthode **private void buildUI()** en la laissant vide pour l'instant.

1.4. Testez : vous devriez avoir une fenêtre vide de 1200x800 pixels.

2. CSSFactory

Comme vous le savez, il est possible d'associer des feuilles de style aux interfaces JavaFX (cf. cours). Nous allons utiliser une telle feuille de style pour donner un joli fond à notre application.

Les feuilles de style correspondent à des fichiers css qu'il faut charger et lier au nœud JavaFX désiré (**stage** dans notre cas) : il faut donc indiquer le nom du fichier css et faire en sorte qu'il soit bien toujours trouvé par l'application.

Pour faciliter l'accès à ce fichier, et aux autres que nous créerons plus tard, nous allons créer la classe **lsg.graphics.CSSFactory** dont le seul « travail » sera de faire le lien entre un nom de fichier, et le chemin (sur le disque) qui permet d'y accéder.

2.1. Créez (et comprenez) la classe **lsg.graphics.CSSFactory** dont le code est :

```
package lsg.graphics ;

public class CSSFactory {

    private static String CSS_DIR = "css/" ;

    public static String getStyleSheet(String filename){
        return CSSFactory.class.getResource(CSS_DIR + filename).toExternalForm() ;
    }

}
```

2.2. Copiez le dossier **images** fourni dans le dossier **lsg/graphics**
(pour que les images se trouvent dans **lsg/graphics/images**)

2.3. Créez le fichier **lsg/graphics/css/LSG.css** avec le code suivant :

```
.root {
    -fx-background-image: url("../images/BACKGROUND.png");
    -fx-background-repeat: no-repeat;
    -fx-background-size: 100% 100%;
}
```

2.4. Utilisez **CSSFactory** dans **buildUI()** pour associer la feuille de style **LSG.css** à la fenêtre de l'application (**stage**), puis testez : l'image doit apparaître en fond de la fenêtre.

3. GameLabel

De manière à donner de l'identité à notre jeu, nous allons créer un widget de texte stylé que nous pourrons utiliser un peu partout : titre, nom du joueur, etc. Pour ce faire, nous allons créer une sous-classe de **javafx.scene.control.Label** que nous allons associer à un style particulier lors de son instanciation.

3.1. Copiez le fichier de police **Arkham_reg.TTF** dans un dossier **lsg/graphics/fonts**

3.2. Créez la feuille de style **lsg/graphics/css/LSGFont.css** :

```
@font-face {
    font-family: 'Arkham';
    src: url('../fonts/Arkham_reg.TTF');
}

.game-font{
    -fx-font-family: 'Arkham';
    -fx-font-style: 'reg';
    -fx-font-size: 45px;

}
```

3.3. Créez la classe **lsg.graphics.widgets.texts.GameLabel**

- Sous classe de **javafx.scene.control.Label**

- Surchargez les 3 constructeurs de **Label** (avec appel à **super**) pour qu'ils :
  - Associent **GameFont.css** au composant
  - Lui applique la classe css « **game-font** » :

```
this.getStyleClass().addAll("game-font") ;
```

3.4. Vous pouvez tester **GameLabel** par un ajout à **root** dans **buildUI()** :

```
root.getChildren().addAll(new GameLabel("Test d'utilisation du Label")) ;
```

3.5. C'est sympa mais ça manque de peps !

- Ajouter les définitions suivantes à **LSGFont.css** :

```
.game-font-fx{
    -fx-text-fill:
            linear-gradient(from 0% 0% to 50% 50%, repeat, ghostwhite,  red) ;
    -fx-effect: dropshadow(three-pass-box, purple, 10, 0, 0, 0);
    -fx-stroke: black;
    -fx-stroke-width: 1.5px;
}

.game-font-fx .text{
    -fx-stroke: black;
    -fx-stroke-width: 1.5px;
}
```

- Ajoutez la classe css « **game-font-fx** » dans la construction d'un **GameLabel**
- Re-testez ! La classe, non ?

4. TitlePane

Nous allons faire mieux qu'un simple ajout de texte à **root** pour l'affichage du titre du jeu, et créer un panneau dans le quel le titre s'animera !

4.1. Créez **lsg.graphics.panes.TitlePane**, sous-classe de **javafx.scene.layout.VBox**

- Avec les attributs privés :
  - **Scene scene**
  - **GameLabel titleLabel**
- Avec un contructeur **TitlePane(Scene scene, String title)** qui :
  - initialise **this.scene** avec la scène passée en paramètre
  - instancie **titleLabel** (grâce à **title**) et l'ajoute au pane NB : **this.getChildren.add(titleLabel) ;**

4.2. Dans **LearningSoulsGameApplication** :

- Créez un attribut privé **TitlePane gameTitle**
- Instanciez le dans **buildUI()** (avec le titre du jeu), et ajoutez le aux *children* de **root** en l'ancrant à la fois au haut (top), à gauche, et à droite.

  Ex. d'ancrage à gauche :

```
AnchorPane.setLeftAnchor(gameTitle, 0.0);
```

4.3. Retouchez le constructeur de **TitlePane** pour que **gameLabel** soit centré et à 10 pixels du haut :

NB : Notez que pour tester/vérifier vos placements, vous pouvez faire appel à la méthode **setStyle** sur les composants, avec par exemple :

```
setStyle("-fx-border-color: red");
```

4.4. Un peu d'animation !

JavaFX permet de créer assez facilement diverses animations grâce à des classes prédéfinies. Nous allons donc faire en sorte que lors de sa 1ere apparition, notre titre subisse un effet de zoom grâce à la classe **ScaleTransition**.

Dans **TitlePane** :

- Créez des constantes que nous utiliserons pour fixer la durée d'une animation, ainsi que l'échelle du zoom :

```
private static final Duration ANIMATION_DURATION = Duration.millis(1500) ;
private static final double ZOOM_SCALE = 1.5 ;
```

- Créez la méthode **ZoomIn** comme suit :

```
public void zoomIn(){
    ScaleTransition st = new ScaleTransition(ANIMATION_DURATION) ;
    st.setNode(titleLabel);
    st.setToX(ZOOM_SCALE);
    st.setToY(ZOOM_SCALE);
    st.setCycleCount(1); // nombre de répétitions de l'effet
    st.play();
}
```

Dans **LearningSoulsGameApplication :**

- Créez **public void startGame()** :

```
private void startGame(){
    gameTitle.zoomIn();
    System.out.println("Animation lancée !");
}
```

- Appelez **startGame()** à la fin de la méthode **start()**
  (…après l'affichage de la fenêtre)
- Admirez…

Cette animation est un peu simpliste : nous allons la complexifier en la parallélisant avec une autre : un déplacement du texte qui accompagnera le zoom. Nous allons donc transformer **zoomIn()** pour y ajouter une animation de translation (**TranslateTransition**), et utiliser la classe **ParallelTransition** pour lancer nos 2 animation (scale + translate) en même temps.

- Créez la constante statique :

```
private static final double ZOOM_Y = 0.25 ;
```

- Modifiez **zoomIn()** comme suit :

```
public void zoomIn(){
    ScaleTransition st = new ScaleTransition(ANIMATION_DURATION) ;
    st.setToX(ZOOM_SCALE);
    st.setToY(ZOOM_SCALE);

    TranslateTransition tt = new TranslateTransition(ANIMATION_DURATION) ;
    tt.setToY(scene.getHeight()*ZOOM_Y);

    ParallelTransition pt = new ParallelTransition(tt, st) ;
    pt.setNode(titleLabel);
    pt.setCycleCount(1); // nombre de répétitions de l'effet
    pt.play();
}
```

- Testez et admirez…

4.5. Animation terminée…

Si vous êtes attentifs, vous pouvez constater que le lancement de l'animation est non bloquant. En effet, le message « Animation lancée ! » apparaît dans la console avant que l'animation ne soit terminée ! (Vous pouvez momentanément changer la hauteur de la fenêtre si vous ne voyez pas la console…). Ceci peut s'avérer gênant lorsque l'on veut séquencer les actions et attendre qu'une animation soit terminée avant de lancer l'instruction suivante.
Pour répondre à ce besoin, les animations de JavaFX fournissent un mécanisme d'évènements permettant de placer un écouteur sur une animation grâce à la méthode **setOnFinished(EventHandler<ActionEvent> event).**
Nous allons utiliser ce mécanisme pour permettre à notre application de savoir quand le **zoomIn** est terminé.

Dans **TitlePane** :

- Pour tester, modifiez **zoomIn** et ajouter un écouteur d'événement grâce à **setOnFinished** sur la transition **pt** juste avant l'appel à **pt.play()** : affichez simplement un message du type « Animation terminée ! ».
  Testez.

- Notre but est de récupérer cet événement et de permettre de le retransmettre au niveau de **zoomIn** :
  - transformez la signature de la méthode **zoomIn** en :
    public void zoomIn(EventHandler<ActionEvent> finishedHandler)

  - au lieu d'afficher « Animation terminée ! » dans le *handler* enregistré sur **pt**, appelez la méthode **handle(event)** sur le **finishedHandler** passé en paramètre de **zoomIn**.

Dans **LearningSoulsGameApplication** :

- Transformez la méthode **startGame** en fournissant un *handler* à **zoomIn** afin d'afficher un message :

```
private void startGame(){
    gameTitle.zoomIn(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            System.out.println("ZOOM terminé !");
        }
    });
}
```

NB : grâce aux ***expressions lambda***, ceci peut aussi s'écrire :

```
private void startGame(){
    gameTitle.zoomIn((event -> {
        System.out.println("ZOOM terminé !");
    }));
}
```

5. CreationPane

Une fois le titre apparu, l'idée est d'afficher un champ de texte permettant au joueur de créer/saisir le nom de son héro. Nous allons faire cet affichage dans un nouveau « calque » nommé **CreationPane**.

5.1. Créez **lsg.graphics.panes.CreationPane**, sous-classe de **javafx.scene.layout.VBox**

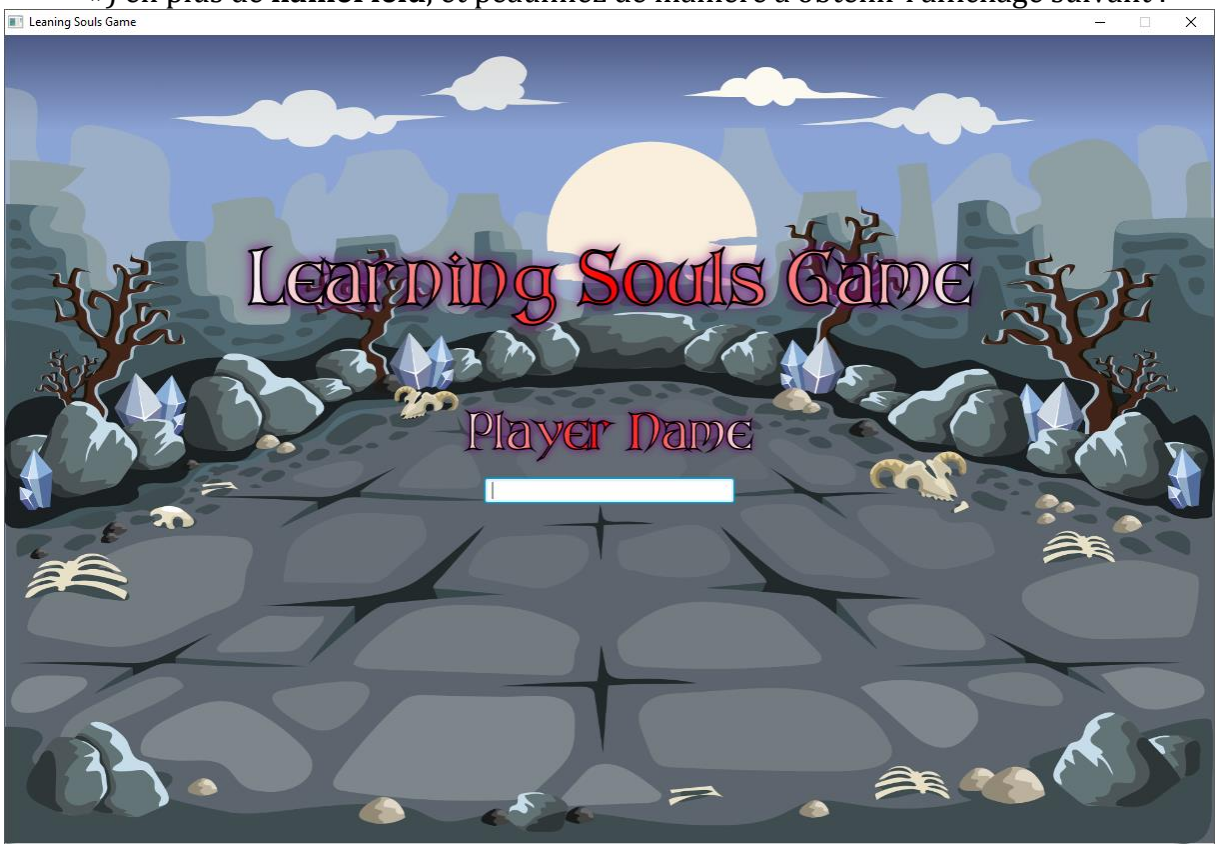
- Définissez un attribut privé **nameField** du type **javafx.scene.control.TextField**
- Créez un getter publique pour cet attribut
- Surchargez le constructeur sans paramètre :
  - instancier **nameField**
  - l'ajouter aux *children* de l'objet

Dans **LearningSoulsGameApplication** :

- Ajoutez un attribut privé **creationPane** de type **CreationPane.**
- Complétez **buildUI()** :
  - instancier **creationPane**
  - ancrer **creationPane** aux 4 bords de **root** (pour qu'il prenne tout l'espace disponible)
  - ajouter **creationPane** aux *children* de **root**
    NB : cette dernière instruction fera apparaître **creationPane** dans la fenêtre avant que **gameTitle** ait fini son **zoomIn** : nous réglerons ce problème plus tard !

Dans **CreationPane** :

- Modifiez le constructeur afin d'ajouter un **GameLabel** (contenant « Player Name ») en plus de **nameField**, et peaufinez de manière à obtenir l'affichage suivant :



5.2. L'affichage de **creationPane** est un peu abrupte ! Nous allons l'adoucir.

Dans **CreationPane** :

- Inspirez-vous de la méthode **zoomIn** de **TitlePane** pour créez la méthode **fadeIn(EventHandler<ActionEvent> finishedHandler)**
  NB : mettez en œuvre une **javafx.animation.FadeTransition**

Dans **LearningSoulsGameApplication** :

- Modifiez **buildUI()** (ajouter 1 ligne) pour fixer l'opacité de **creationPane** à 0 lors de son instanciation.

- Modifiez **startGame** pour déclencher le **fadeIn** de **creationPane** une fois le **zoomIn** de **gameTitle** terminé :

```
private void startGame(){
    gameTitle.zoomIn((event -> {
        creationPane.fadeIn(null);
    }));
}
```

- Testez : vous devez maintenant voir le zoom du titre, puis l'apparition (en fade) de l'invite pour la création du personnage.

5.3. Récupération du nom du héro.

Grâce à nos animations, le joueur est maintenant invité à entrer le nom de son héro. La validation sera faite par un appui sur la touche *Entrée*, ce qui correspond à mettre en œuvre un *handler* enregistré sur le **setOnAction** de **nameField** (instance de **TextField**).

Dans **LearningSoulsGameApplication** :

- Créez un attribut **heroName** de type **String**

- Créez la méthode **private void addListeners()** qui place un *handler* sur le **setOnAction** du **nameField** de **creationPane** pour récupérer le contenu du champ et le mettre dans **heroName** :

```
private void addListeners(){
    creationPane.getNameField().setOnAction((event -> {
        heroName = creationPane.getNameField().getText() ;
        System.out.println("Nom du héro : " + heroName);
    }));
}
```

- Appelez **addListeners** dans la méthode **start** et juste après l'appel à **buildUI**.

- Testez : le message doit s'afficher lorsque vous tapez sur *Entrée* dans le champ.

5.4. On passe à la suite !

Lorsque le joueur a saisi (et validé par *Entrée*) un nom de héro qui n'est pas vide, le **creationPane** doit disparaître (on le retire des *children* de **root**), et **gameTitle** doit retourner à sa place initiale (celle avant le **zoomIn**).

Dans **addListeners()** de **LearningSoulsGameApplication** :

- Juste après le **System.out.println("Nom du héro : " + heroName)** du *handler* :
  Si **heroName** n'est pas vide, retirer **creationPane** de la liste des *children* de **root**

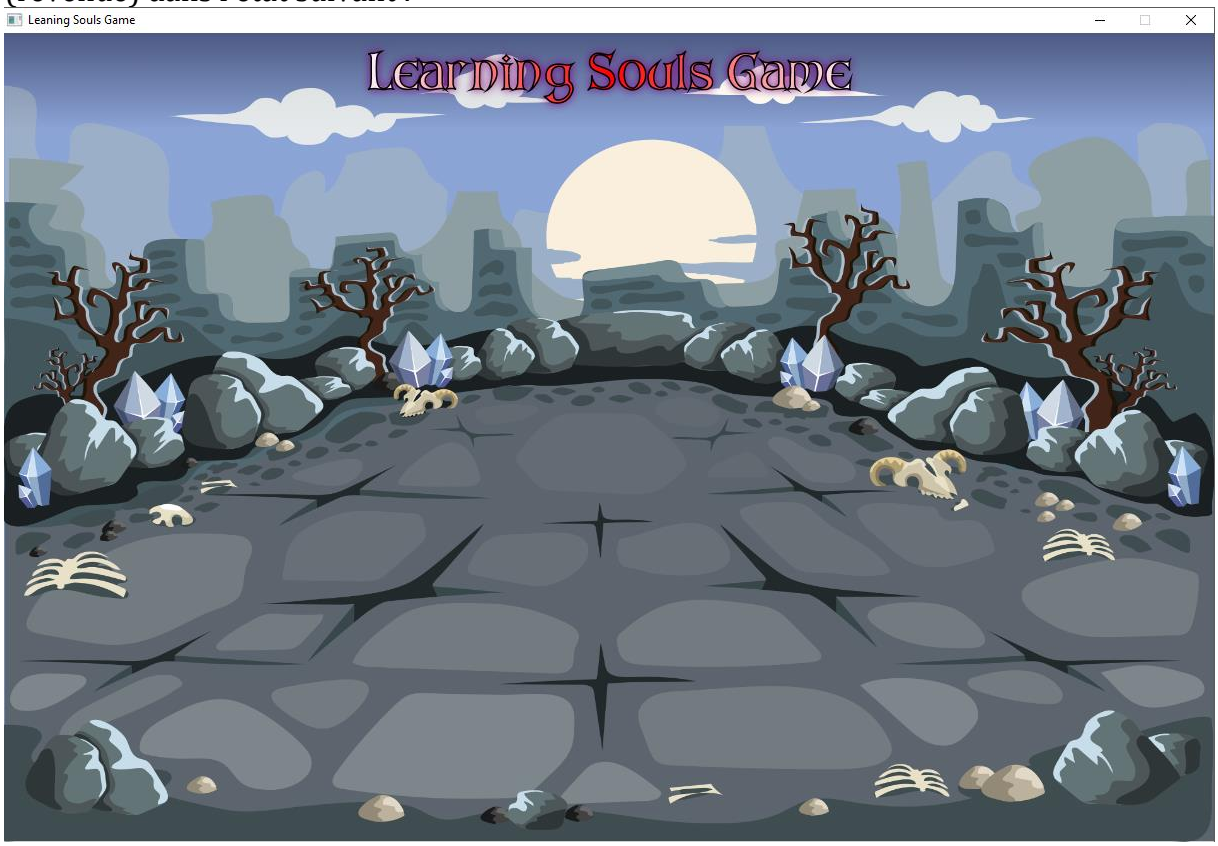
Dans **TitlePane** :

- Créez la méthode **zoomOut(EventHandler<ActionEvent> finishedHandler)** qui fait retourner le titre à sa place (et taille) initiale.

Dans **LearningSoulsGameApplication** :

- Complétez le *handler* pour que le **zoomOut** de **gameTitle** suive la disparition de **creationPane**.

Une fois le tout exécuté (donc après l'entrée d'un nom valide), l'application doit être (revenue) dans l'état suivant :



6. En scène !

Pour vous aider et gagner du temps, l'équipe a créé quelques classes qui vont vous permettre d'animer facilement des personnages.

6.1. **ImageFactory** (fournie)

La partie graphique du jeu requiert un certain nombre d'images. Pour en faciliter la gestion, et dans une approche similaires à celle mise en œuvre pour **CSSFactory**, la classe **ImageFactory** va vous permettre de récupérer les images dont vous aurez besoin.

- Intégrez la classe **lsg.graphics.ImageFactory** au projet.

Le chargement des images peut prendre du temps et bloquer l'exécution du jeu. C'est pourquoi **ImageFactory** propose la méthode **preloadAll** qui permet de pré-charger l'ensemble des images nécessaires au jeu en tâche de fond (exécution non bloquante).

NB : le paramètre **finishedHandler** permet d'associer un *handler* dont la méthode **run** est appelée quand le pré-chargement est effectivement terminé, mais nous ne l'utiliserons pas réellement pour l'instant (sauf pour afficher un message console).

Pour éviter toute saccade résiduelle visible (même si **preloadAll** n'est pas bloquante, elle peut ralentir l'exécution), nous allons déclencher le pré-chargement des images pendant que le joueur réfléchit : c.à.d. au moment où il doit trouver un nom pour son personnage !

Dans **LearningSoulsGameApplication** :

- Déclenchez le pré-chargement lorsque le **fadeIn** de **creationPane** est terminé :

```
private void startGame(){
    gameTitle.zoomIn((event -> {
        creationPane.fadeIn((event1 -> {
            ImageFactory.preloadAll((() -> {
                System.out.println("Pré-chargement des images terminé") ;
            })) ;
        }));
    }));
}
```

6.2. **AnimationPane** (fournie)

L'animation des personnages sera faite dans un « calque » de type **AnimationPane** qui fonctionne comme une sorte de théâtre dans lequel on dirigera des représentations graphiques des personnages.

- Intégrez dans le projet :
  - la classe **lsg.graphics.panes.AnimationPane**
  - le package **lsg.graphics.widgets.characters.renderers**

Dans **LearningSoulsGameApplication** :

- Créez l'attribut privé **AnimationPane animationPane**

- Instanciez **animationPane** à la fin de **buildUI** en lui donnant **root** pour parent

- Créez une méthode **private void play()** qui :
  - fait apparaître **animationPane**, c.à.d. qui l'ajoute aux *chidren* de **root**
  - *[pour le fun] appelle :* **animationPane.startDemo()**

- Lancez **play** lorsque la partie commence effectivement, c'est à dire lorsque le **zoomOut** de **gameTitle** est terminé !

- Admirez.