

Langage Orienté Objet / Java

Partie 8 – Liaison au modèle

The Learning Souls Game

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

1. Instanciation des personnages

Nous avons laissé la méthode **play** de **LearningSoulsGameApplication** avec le démarrage d'une démo de l'**animationPane** : il est temps de remplacer cette démonstration par le démarrage réel du jeu, c'est à dire l'instanciation des personnages au niveau du modèle (un **Hero** et un **Zombie**).






1.1. Zombie

Les monstres que nous avons créés jusqu'à présent sont un peu trop forts pour débiter le jeu. Nous allons donc créer une nouvelle classe de monstre qui permettra à notre héros de s'entraîner sur des ennemis un peu plus faibles.

Créez la classe **lsg.characters.Zombie** dont les instances auront les caractéristiques :

[Zombie]	Zombie	LIFE: 10	STAMINA: 10	PROTECTION: 10.00	BUFF: 0.00	(ALIVE)
WEAPON : Zombie's hands (min:5 max:20 stam:1 dur:1000)						

1.2. LearningSoulsGameApplication

-  Créez l'attribut **hero** de type **Hero**
-  Créez l'attribut **heroRenderer** de type **HeroRenderer**
-  Créez l'attribut **zombie** de type **Zombie**
-  Créez l'attribut **zombieRenderer** de type **ZombieRenderer**
-  Créez la méthode **createHero()** qui :
 - instancie **hero** en lui donnant le nom entré par le joueur
 - donne à **hero** une instance de **Sword**
 - initialise **heroRenderer** en utilisant la méthode **createHeroRenderer** de **animationPane** (fournit une instance qui sera bien placée dans la scène)
 - lance une animation pour faire entrer le **heroRenderer** en scène

NB : vous pouvez adapter la ligne de code utilisée dans **startDemo** de **AnimationPane**

- ✎ Dans **play**, remplacez l'appel à **animationPane.startDemo()**, par un appel à **createHero()**
- ✎ Testez : seul le héros doit (pour l'instant) entrer en scène.
- ✎ Créez la méthode **createMonster(EventHandler<ActionEvent> finishedHandler)**
 - inspirez-vous de **createHero()** pour initialiser **zombie** et le faire entrer en scène
 - le paramètre **finishedHandler** est un *handler* qui sera appelé lorsque l'exécution de **createMonster** est totalement terminée, c'est à dire, lorsque **zombieRenderer** est bien arrivé au centre de la scène (animation terminée) : il s'agit donc de faire remonter l'événement généré par la méthode **goto** de **CharacterRenderer**
- ✎ Complétez la méthode **play** par un appel à **createMonster** (juste après l'appel à **createHero**).
 - placez un *handler* qui affiche « FIGHT ! » dans la console lorsque **createMonster** est terminée
- ✎ Testez : le héros et le zombie doivent maintenant entrer en scène, et « FIGHT ! » doit s'afficher dans la console seulement une fois le zombie arrivé à sa place

2. HUDPane et MessagePane

Nous venons d'afficher « FIGHT ! » dans la console mais cela ne sera pas très utile au joueur (qui n'est pas censé la regarder...). De plus, au cours du jeu, nous aurons divers messages à lui faire passer comme par exemple des indications liées à la levée des exceptions que nous avons mise en place dans la partie 6 (manque de stamina, arme cassée, etc.). Pour supporter ces affichages, nous allons développer la classe **MessagePane**.

Cependant, les messages ne sont pas les seules informations que nous devons fournir à l'utilisateur en cours de jeu. On voudra par exemple aussi afficher ses barres de vie et de stamina, celles de son adversaire, ses compétences (attaquer, etc.), son score, son équipement, le contenu de son sac, ... De manière à organiser l'ensemble de ces informations (barres en haut, compétences en bas, messages au centre, etc.), nous allons les répartir dans un pane nommé **HUDPane** (Head Up Display).

2.1. Créez la classe **lsg.graphics.panes.MessagePane**

- ✎ sous-classe de **javafx.scene.layout.VBox**
- ✎ avec la méthode **showMessage(String msg)**
 - pour l'instant : une version simple qui ne fait qu'ajouter une instance de **GameLabel** contenant le message aux *children* du pane

2.2. Créez la classe **lsg.graphics.panes.HUDPane**

- ✎ sous-classe de **javafx.scene.layout.BorderPane**
- ✎ un attribut privé **messagePane** du type **MessagePane**
- ✎ un *getter* pour cet attribut
- ✎ une méthode **buildCenter()** (appelée dans le constructeur) qui :
 - instancie **messagePane**
 - l'ajoute au centre du pane

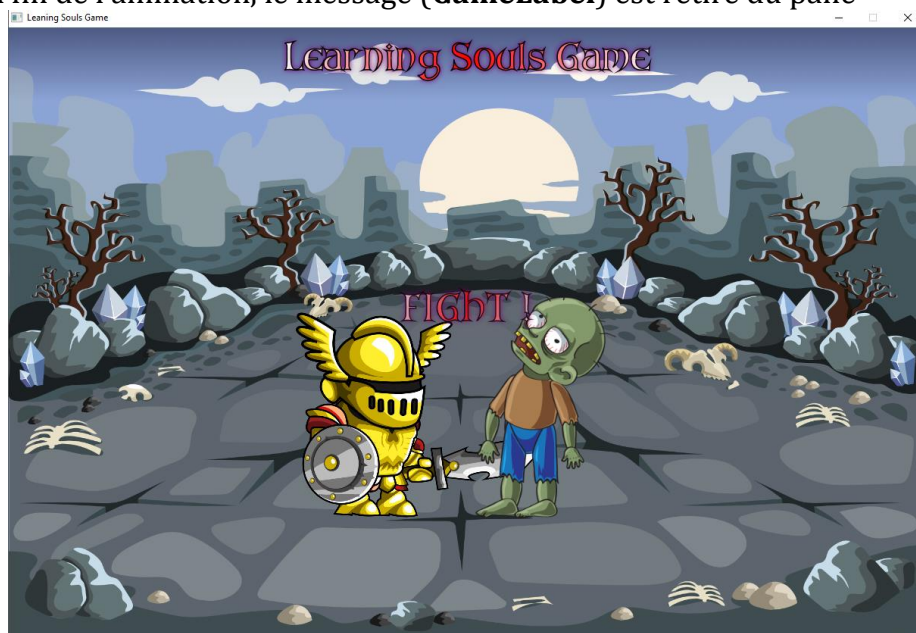
2.3. Dans **LearningSoulsGameApplication**

- ✎ Ajoutez un attribut privé **hudPane** du type **HUDPane**
- ✎ Dans **buildUI()** : instanciez **hudPane** et ancrez-le pour qu'il occupe tout l'espace
- ✎ Dans **play()** :
 - ajoutez **hudPane** aux *children* de **root** juste après l'ajout de **animationPane**
 - remplacez l'affichage de « FIGHT ! » dans la console par un **showMessage(« FIGHT ! »)** sur le **messagePane** (utilisez le getter) de **hudPane**
- ✎ Testez :



2.4. Modifiez **MessagePane**

- ✎ Modifiez le constructeur **MessagePane()** pour faire en sorte que les messages soient centrés dans le pane
- ✎ Transformez **showMessage** :
 - le message apparaît (au centre donc)
 - il s'anime en subissant une translation de 200 pixels vers le haut et un *fade out* (il disparaît petit à petit) en 3 secondes
 - à la fin de l'animation, le message (**GameLabel**) est retiré du pane



3. StatBar

Nous allons maintenant créer la classe **StatBar** qui permet d'afficher le nom, l'avatar et les barres de vie et de stamina du personnage. Ce *widget* sera utilisé dans **HUDPane**.

3.1. Créez la classe **lsg.graphics.widgets.characters.statbars.StatBar**

Une sous-classe de **javafx.scene.layout.BorderPane**

Implémentez le constructeur **StatBar()**

- fixer la taille préférée du composant à 350 pixel de large et 100 pixels de haut
- nous allons construire le composant pas à pas donc dans un premier temps on le laissera vide, tout en faisant néanmoins en sorte de pouvoir le voir en affichant ses bords : utilisez **this.setStyle("-fx-border-color: red");**
(la couleur est au choix 😊)

3.2. Dans **HUDPane**

Créez les attributs **heroStatBar** et **monsterStatBar**, tous deux de type **StatBar**

Fournissez les *getters* correspondants

Ajoutez la méthode **buildTop()** qui :

- place un **BorderPane** au top de **HUDPane**
- instancie et ajoute **heroStatBar** à la gauche de ce **BorderPane**
- instancie et ajoute **monsterStatBar** à la droite de ce **BorderPane**

Appelez **buildTop()** dans le constructeur

Testez : vous devez voir apparaître les 2 cadres :



3.3. Dans StatBar

✎ Créez les attributs privés :

- **ImageView** avatar
- **GameLabel** name
- **ProgressBar** lifeBar
- **ProgressBar** stamBar

✎ Implémentez les *getters* correspondants

Dans le constructeur, instanciez et organisez ces composants sachant que :

✎ Pour **avatar** :

- l'image par défaut de avatar sera celle correspondant à **ImageFactory.SPRITES_ID.HERO_HEAD**
- le ratio de l'image doit être préservé
- son **fitHeight** est de 100 pixels (la hauteur de **StatBar**)

✎ Pour **name** :

- fixer la font à 33px (utilisez un **setStyle** sur la propriété **-fx-font-size**)
- mettre la chaîne « name » par défaut

✎ Pour **lifeBar** et **stamBar** :

- utilisez **lifeBar.setMaxWidth(Double.MAX_VALUE)** pour que le composant prenne toute la largeur de son parent
- idem pour **stamBar**
- utilisez **setStyle("-fx-accent: red")** pour colorier la barre de progression (**red** pour **lifeBar** ; **greenyellow** pour **stamBar**)

✎ Visuel à obtenir par défaut :



✎ Retirez le bord (rouge) du composant dès qu'il vous semble inutile !

3.4. Dans LearningSoulsGameApplication

- ✎ Complétez **createHero** et **createMonster** en appelant les méthodes de **hudPane** pour que les personnages aient bien leur nom et leur image

NB : il est possible d'appliquer une rotation à un **ImageView** avec **setRotate**



3.5. Peaufinage

On peut remarquer que la barre du monstre qui se trouve à droite gagnerait à être affichée en « miroir » (avatar à droite et barres à gauche).

- ✎ Dans **StatBar**, créez la méthode **flip()**
 - l'effet miroir peut être créé en appelant : **this.setScaleX(-this.getScaleX())** ;
 - notez que lors d'un **flip()**, le texte se retrouvera aussi (logiquement) lui-même inversé, et devient donc illisible : il faut donc doubler l'effet miroir sur le label...
- ✎ Dans **HUDPane** :
 - utilisez **flip()** sur la **monsterStatBar**
 - ajoutez un peu de *padding* pour que les composants ne touchent plus les bords de la scène



4. Liaison au modèle

Il nous reste maintenant à faire en sorte que les barres affichent en temps réel les montants de vie et de stamina des personnages.

Dans **StatBar**, les attributs **lifeBar** et **stamBar** sont des instances de **ProgressBar** : la taille de l'indicateur y est gérée par une propriété nommée **progressProperty**.

Sur ce point, *JavaFX* repose sur la technologie *JavaBeans* qui met en œuvre le pattern observer/observable... que vous verrez en détails l'an prochain. Pour l'instant, nous noterons simplement que grâce à ces mécanismes, il est possible de lier des propriétés entre elles, ce qui a pour effet de rendre une propriété dépendante d'une autre. En d'autre terme, la modification d'une propriété peut entraîner la mise à jour automatique d'une autre propriété qui lui est liée.

C'est ce mécanisme que nous allons utiliser en ajoutant à la classe **lsg.characters.Character** des propriétés qui représentent les taux restants de vie et de stamina. Il ne nous restera plus ensuite qu'à lier les **progressProperty** des barres à ces nouveaux attributs de **Character** pour que les barres soit automatiquement tenues à jour.

4.1. Dans la classe **Character** :

- ✎ Créez l'attribut **lifeRate** de type **SimpleDoubleProperty** et instanciez le
- ✎ Implémentez le *getter* **SimpleDoubleProperty lifeRateProperty()** pour cet attribut (on ne mettra pas le préfixe « get » pour les propriétés)
- ✎ Gérez le calcul de la valeur de **liferate** dans les méthodes **setLife** et **setMaxLife** avec l'instruction : **lifeRate.set((double)life/maxLife) ;**
[NB : on caste **life** en **double** pour utiliser le bon opérateur de division (à virgule)]

4.2. Dans **LearningSoulsGameApplication** , à la fin de **createHero()** :

- ✎ Liez la **progressProperty** du héros à sa **liferateProperty** :

```
hudPane.getHeroStatBar().getLifeBar().progressProperty().bind(hero.lifeRateProperty());
```

- ✎ Testez : la barre de vie du héros doit être fixe et pleine

4.3. Liez la barre de vie du monstre (**createMonster**)

4.4. Appliquez la même méthode (à partir du 4.1) pour lier les barres de stamina.



4.5. Vérification

De manière à vérifier le bon fonctionnement des barres, ajoutez quelques lignes de test dans le *handler* passé à **createMonster** dans **play** :

- ✎ Le lancement d'une attaque (méthode **attack** de **Character**) doit faire diminuer la stamina du personnage

NB : gérez simplement les exceptions en affichant leur message dans **messagePane**

- ✎ La réception d'un coup (méthode **getHitWith** de **Character**) doit faire diminuer la vie du personnage (si l'attaque n'est pas nulle....)

4.6. Peaufinez encore et encore

Vous pouvez encore parfaire l'interface. Par exemple, faites en sorte que l'affichage de la barre de stamina soit un peu plus courte et fine que la barre de vie :

