PacmanFX – preuves compétences

Documentation

Je sais concevoir un diagramme UML intégrant des notions de qualité et correspondant exactement à l'application que j'ai à développer.

Le diagramme de classes est dans *pacmanfx\Doc\ClassDiagram.mdj*

Je sais décrire un diagramme UML en mettant en valeur et en justifier les éléments essentiels

Le diagramme ne contient que les classes du modèle qui sont importantes à la compréhension du fonctionnement de l'application.

Je sais documenter mon code et en générer la documentation.

La documentation javadoc du code est complète et déjà générée, elle est située dans *pacmanfx\Doc\javadoc*

Elle peut également être générée depuis Idea en sélectionnant le niveau « privé » ainsi que les tags @author (pour savoir qui a écrit ce fichier) et les tags @deprecated (pour les fichiers inutilisés)

Je sais décrire le contexte de mon application, pour que n'importe qui soit capable de comprendre à quoi elle sert.

Le contexte de l'application est décrit dans *pacmanfx\Doc\pacmanContext.md* (il date cependant du début du projet) et décrit les fonctionnalités du jeu.

Je sais faire un diagramme de cas d'utilisation pour mettre en avant les différentes fonctionnalités de mon application.

Le diagramme de cas d'utilisation est situé à *pacmanfx\Doc\UseCaseDiagram.mdj* il permet de faire ressortir les intentions métier de l'application.

Code

Je maîtrise les règles de nommage Java.

Les classes, variables et packages de l'applications ont été nommés en accord avec les règles de nommage Java décrites par Oracle ici :
https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

Je sais binder bidirectionnellement deux propriétés JavaFX.

Du binding bidirectionnel est utilisé dans views.MenuView.java à la ligne 44 :

```
levelsList.itemsProperty().bindBidirectional(levels);
```

Ce n'était pas strictement requis d'utiliser un binding bidirectionnel ici mais cela a été fait pour la démonstration de ce type de binding.

Je sais binder unidirectionnellement deux propriétés JavaFX.

Le binding unidirectionnel est très utilisé dans ce projet. Un exemple serait views.GameView.java à la ligne 217 et 221 :

```
IntegerBinding xpos = pac.getPositionLogique().CaseColProperty()
                        .multiply(scaleFactor).add(58).add(2);
```

On créé un binding d'int et on le bind à la position X du pacman :

```
pacman.centerXProperty().bind(xpos);
```

(unidirectionnel modèle -> vue)

Je sais coder une classe Java en respectant des contraintes de qualité de lecture de code.

Cela est laissé à l'appréciation du correcteur, mais les classes utilisent des noms assez évidents, sont indentées correctement, il n'existe qu'une classe par fichier java dont le nom est le même que celui du fichier.

Je sais contraindre les éléments de ma vue, avec du binding FXML.

La réponse serait similaire à « Je sais binder unidirectionnellement deux propriétés JavaFX » mais pour donner un autre exemple, les vies restantes au joueur sont bindées via un propriété booléenne qui vérifier qu'une propriété int est supérieure à X :

```
    public void bindCompteurs(CompteurScore cs, CompteurVie cv){
        BooleanBinding alive1 = cv.Viesproperty().greaterThanOrEqualTo(1);
        BooleanBinding alive2 = cv.Viesproperty().greaterThanOrEqualTo(2);
        BooleanBinding alive3 = cv.Viesproperty().greaterThanOrEqualTo(3);

        Life1.visibleProperty().bind(alive3);
        Life2.visibleProperty().bind(alive2);
        Life3.visibleProperty().bind(alive1);

        StringBinding sb = cs.Scoreproperty().asString();
        scoreCounter.textProperty().bind(Bindings.format("Score: %s",sb));
    }
```

(views.GameView.java ligne 200)

Je sais définir une CellFactory fabriquant des cellules qui se mettent à jour au changement du modèle.

Une cellFactory est définie dans views.MenuView.java tel que :

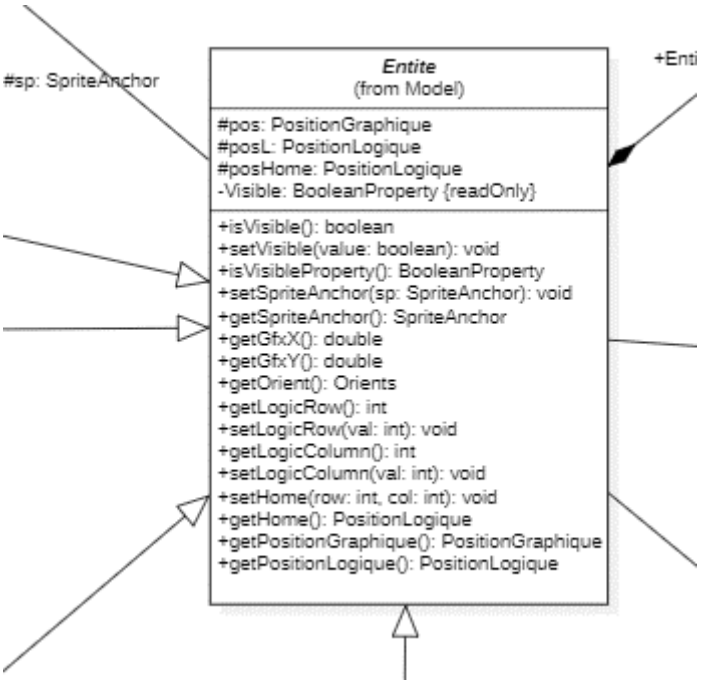
```
        levelsList.setCellFactory(__ ->
                new ComboBoxListCell<LevelFile>(){

                    @Override
                    public void updateItem(LevelFile item, boolean empty) {
                        textProperty().unbind();
                        textProperty().set("test");
                        super.updateItem(item, empty);
                        if (!empty) {
                            textProperty().bind(Bindings.format("Nom du
niveau: %s ,taille: %d x %d", item.FilenameProperty(),
item.RowProperty(),item.ColumnProperty()));
                        } else {
                            textProperty().unbind();
                            setText("");
                        }
                    }
                }
        );
```

Elle va contenir la liste des niveaux valides découverts par le modèle (sous la forme d'une collection de LevelFile).

Je sais éviter la duplication de code.

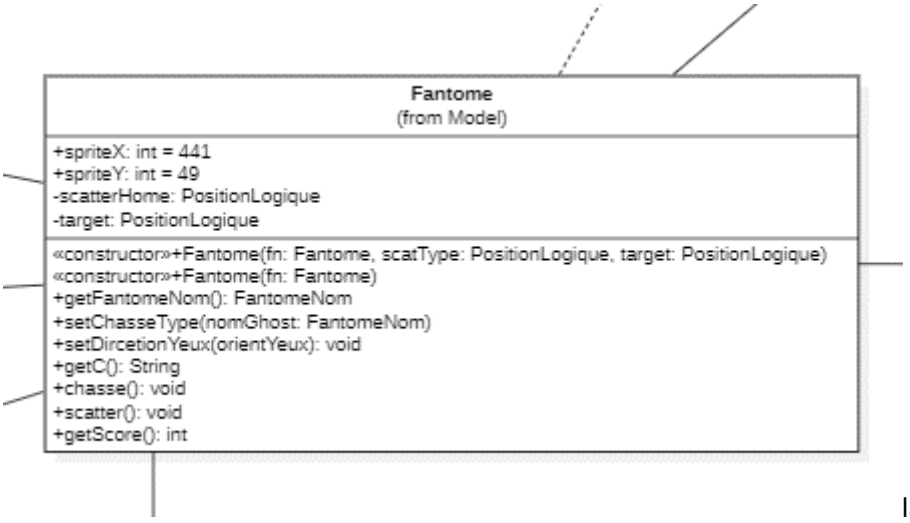
De l'héritage est utilisé pour limiter la duplication du code, par exemple pour les entités :



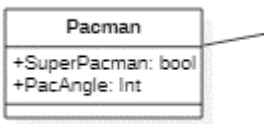
Les classes qui héritent de cette classe abstraite auront toutes les variables requises pour être utilisées comme des entités non-statiques par le modèle, juste avec l'héritage.

Je sais hiérarchiser mes classes pour spécialiser leur comportement.

Pour reprendre l'exemple précédent, des classes Pacman et fantôme (notamment) héritent de la classe Entité et se comportent de manière différente :



la classe fantome



et pacman

Je sais intercepter des évènements en provenance de la fenêtre JavaFX.

La classe controller.GameController implémente

```
EventHandler<KeyEvent>
```

Et peut donc intercepter les touches du clavier détectées par la fenêtre principale.

Je sais maintenir, dans un projet, une responsabilité unique pour chacune de mes classes.

Par héritage et découpage de classes, les classes ne devraient pas avoir plus d'une responsabilité à la fois dans le projet.

Je sais gérer la persistance de mon modèle.

Dans ce projet, la persistance n'est pas gérée en sérialisant une classe (manque de temps), elle est implémentée en écrivant le score actuel dans le fichier de score du niveau à la fin de la partie.

Dans tools.files.ScoreSaver :

```
    public static void saveScore(String levelName, String playerName, int
points){
        String toWrite = "\n"+playerName+";"+points;
        try {
            Files.writeString(Paths.get("./out/production/pacmanfx/Cartes/"+le
velName+"/"+levelName+".score"),toWrite + System.lineSeparator(),CREATE,
APPEND);
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Ce fichier est lu à chaque démarrage de l'application pour afficher les scores du niveau sélectionné.

Je sais utiliser à mon avantage le polymorphisme.

Le polymorphisme est utilisé dans les runnables mais plus clairement dans les déplaceurs :

Dans model.mouvement.Deplaceurs.Deplaceur :

```
protected abstract Case deplacement();
```

Cette méthode est @Override dans les classes héritantes pour que le déplacement soit personnalisé.

Je sais utiliser GIT pour travailler avec mon binôme sur le projet.

Git a été utilisé pendant tout le projet, avec des commits réguliers.

Je sais utiliser le type statique adéquat pour mes attributs ou variables.

Pour obtenir des performances correctes tout en ayant une gestion de la mémoire facile, nous avons utilisé des ArrayList pour la plupart de nos collections qui ne sont pas accédées par de multiples threads (vector sinon).
Les classes Case utilisent des ArrayList pour garder en mémoire les entités qu'elles contiennent.

Je sais utiliser les différents composants complexes (listes, combo...) que me propose JavaFX.

On utilise une ComboBox dans le menu, ainsi que des Arc pour représenter le pacman et ses vies, ils viennent tous deux de jfx (idem pour les objets mangeables qui sont des cercles)

Je sais utiliser les lambda-expression

Les lambda-expressions sont surtout utilisées pour facilement créer des runnables et Threads quand nécessaire, en économisant du code :

Ligne 283 de views.GameView

```
Platform.runLater(() -> {EndGame();});
```

Je sais utiliser les listes observables de JavaFX

Ligne 36 de views.MenuView, on utilise une SimpleListProperty (collection observable) pour contenir les différents niveaux valides qu'on a découvert.

```
private ListProperty<LevelFile> levels = new SimpleListProperty<LevelFile>();
```

Je sais utiliser un convertisseur lors d'un bind entre deux propriétés JavaFX.

Les convertisseurs ne sont pas réellement utilisés dans ce projet. Ils permettent de changer le type de variable d'un binding. Un exemple serait :

```
StringConverter<Number> converter = new NumberStringConverter();
```

Puis il suffit d'ajouter ce converter en troisième argument de la méthode bind().

Je sais utiliser un fichier CSS pour styler mon application JavaFX.

Toutes les vues FXML utilisent des fichiers de style CSS qui sont présentes dans les ressources. Exemple (Menu.fxml):

```
<ListView fx:id="ScoreList" layoutX="197.0" layoutY="108.0" prefHeight="236.0"
prefWidth="407.0" styleClass="menuBtn" />
```

(on utilise la classe menuBtn de la feuille de style active ici)

Je sais utiliser un formateur lors d'un bind entre deux propriétés JavaFX.

Le titre du tableau des scores fait usage d'un formateur de String :

```
textProperty().bind(Bindings.format("Nom du niveau: %s ,taille: %d x %d",
item.FilenameProperty(), item.RowProperty(),item.ColumnProperty()));
```

(views.MenuView ligne 55)

Je sais développer un jeu en JavaFX en utilisant FXML

A l'appréciation du correcteur, la plupart des actions qu'on peut réaliser dans le jeu original de pac-man fonctionnent dans cette implémentation en JavaFX.

Je sais intégrer, à bon escient, dans mon jeu, une boucle temporelle observable.

Le jeu dispose d'un système de boucles observables timées assez complexe, car il permet d'avoir plusieurs boucles en même temps avec des périodes différentes et ce assez facilement.

Ce système de boucles est décrit dans le package model.boucles

Exemple d'une boucle (méthode clé) :

```
    @Override
    public void run() {     //boucle de jeu
        while(running) {
            try {
                Thread.sleep(periode);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            notifyAbonnes();
            }
        }
    }
```