Morel Mathieu
Couchot Jaden

Preuves de validation des compétences

DOCUMENTATION

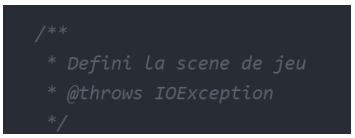
Je sais concevoir un diagramme UML intégrant des notions de qualité et correspondant exactement à l'application que j'ai à développer.

Cf. it-flows.mdj

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

Cf. descriptionDiagramme.pdf

Je sais documenter mon code et en générer la documentation.



```
/**
 * Defini la scene de jeu
 * @throws IOException
 */
```

La javaDoc est dans le dossier /javaDoc/.

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

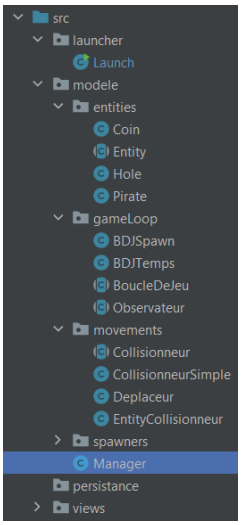
Cf. Contexte.docx

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

Cf. UCDiagram.mdj

CODE

Je maîtrise les règles de nommage Java.



Les packages sont écrits en minuscule et les classes avec la première lettre en majuscule.

Les constantes sont écrites en majuscules

```
private final int OFFSET = 8;
```

Je sais binder bidirectionnellement deux propriétés JavaFX.

```
textFieldUsername.textProperty().bindBidirectional(manager.getActualPlayer().nameProperty());
```

Nous utilisons le binding bidirectionnel pour afficher le nom par défaut du joueur dans le champ de texte lors du lancement, celui-ci peut le changer en cours de route.

Je sais binder unidirectionnellement deux propriétés JavaFX.

```
private void bindAll(){
    //Bind rectangle -> Pirate
    rectanglePirate.heightProperty().bind(manager.getPirate().hauteurProperty());
    rectanglePirate.widthProperty().bind(manager.getPirate().largeurProperty());
    rectanglePirate.xProperty().bind(manager.getPirate().xProperty());
    rectanglePirate.yProperty().bind(manager.getPirate().yProperty());
```

Bind du rectanglePirate aux propriétés de l'entité Pirate du Manager (personnage incarné par le joueur)

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

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

```
private void buttonActivator(){
    buttonGame.disableProperty().bind(textFieldUsername.textProperty().isEmpty());
}
```

La propriété disable d'un bouton est contrainte par la propriété isEmpty d'un TextFiled avec un binding FXML.

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

```
laListe.setCellFactory(__ ->
        new ListCell<Player>(){
            @Override
            protected void updateItem(Player item, boolean empty) {
                super.updateItem(item, empty);
                if (!empty) {
                    textProperty().bind(Bindings.format("%s : %s points", item.nameProperty(),
                } else {
                    textProperty().unbind();
                    setText("");
                }
```

Notre CellFactory se met à jour au changement du modèle car elle est bindée

Je sais éviter la duplication de code.

Les parties de code utilisées fréquemment sont regroupées dans des méthodes uniques afin d'être réutilisées.

Je sais hiérarchiser mes classes pour spécialiser leur comportement.

```
public abstract class BoucleDeJeu implements Runnable{
```

```
public class BDJTemps extends BoucleDeJeu {
```

Nous utilisons des classes qui héritent de classes plus hautes dans la hiérarchie

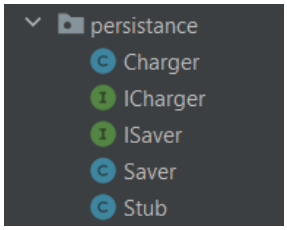
Je sais intercepter des évènements en provenance de la fenêtre JavaFX.

Pour récupérer les déplacements de notre personnage par exemple.

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

Toutes les classes de notre code respectent le principe de single responsability en ayant le moins de méthodes possibles.

Je sais gérer la persistance de mon modèle.

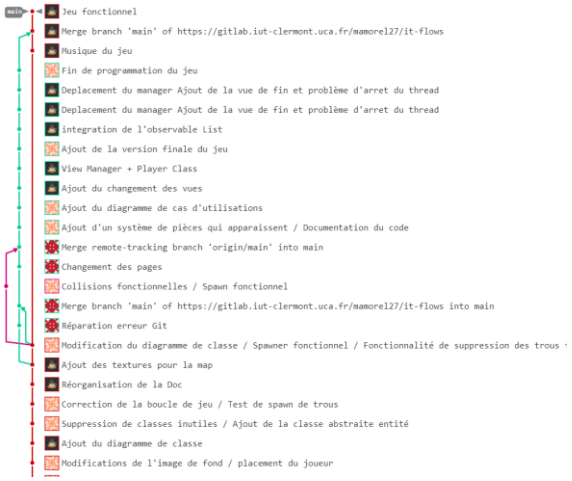


Nous avons attaqué la persistance mais elle n'est pas finie.

Je sais utiliser à mon avantage le polymorphisme.

Nous utilisons le polymorphisme avec nos différentes boucles de jeu.

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



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

```
private Player actualPlayer
```

Pour l'ensemble de nos variables nous choisissons le type adéquat.

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

```
<ListView id="laliste" fx:id="laListe" fixedCellSize="50.0" GridPane.halignment="CENTER"
```

```
laListe.itemsProperty().bind(manager.playerListProperty());
```

Pour l'affichage de nos joueurs et de leurs score sur l'écran de fin, nous utilisons une ListView.

Je sais utiliser les lambda-expression.

Je sais utiliser les listes observables de JavaFX.

```
private ObservableList<Player> list_players_obs = FXCollections.observableArrayList();
```

Pour la gestion de nos joueurs et des joueurs qui jouent actuellement, nous utilisons une ObservableList.

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

```
textProperty().bind(Bindings.format("%s : %s points", item.nameProperty(), Bindings.convert(item.scoreProperty())));
```

Pour l'affichage des items dans le ListView, nous utilisons un convertisseur.

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

style.css

```
@font-face{
    font-family: 'Press Start 2P';
    src: url("/font/PressStart.ttf")format("ttf");
}

#start_body{
    -fx-background-image: url("/images/startimage.png");
    -fx-background-size: cover;
}

```

```
<GridPane id="start_body"
```

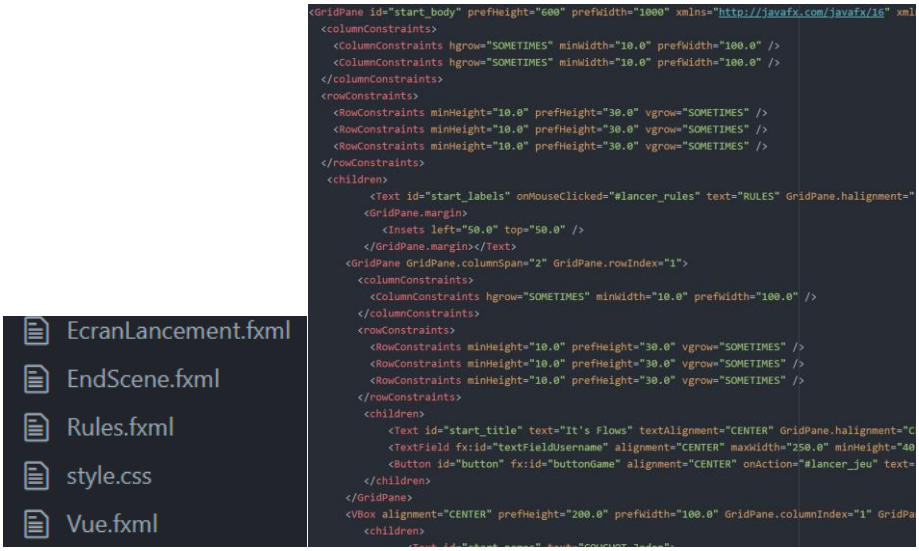
L'intégralité de nos fenêtres ont un composant qui fait appel au fichier .css.

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

```
textProperty().bind(Bindings.format("%s : %s points", item.nameProperty(), Bindings.convert(item.scoreProperty())));
```

Morel Mathieu
Couchot Jaden

Je sais développer un jeu en JavaFX en utilisant FXML.




```
<GridPane id="start_body" prefHeight="600" prefWidth="1000" xmlns="http://javafx.com/javafx/16" xml
  <columnConstraints>
    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
    <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
  </columnConstraints>
  <rowConstraints>
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
    <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
  </rowConstraints>
   <children>
         <Text id="start_labels" onMouseClicked="#lancer_rules" text="RULES" GridPane.halignment="
        <GridPane.margin>
           <Insets left="50.0" top="50.0" />
        </GridPane.margin></Text>
      <GridPane GridPane.columnSpan="2" GridPane.rowIndex="1">
        <columnConstraints>
          <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
        </columnConstraints>
        <rowConstraints>
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
        </rowConstraints>
         <children>
             <Text id="start_title" text="It's Flows" textAlignment="CENTER" GridPane.halignment="C
             <TextField fx:id="textFieldUsername" alignment="CENTER" maxWidth="250.0" minHeight="40
             <Button id="button" fx:id="buttonGame" alignment="CENTER" onAction="#lancer_jeu" text=
         </children>
      </GridPane>
      <VBox alignment="CENTER" prefHeight="200.0" prefWidth="100.0" GridPane.columnIndex="1" GridPa
         <children>
```

L'ensemble des fenêtres de notre jeu sont réalisées avec des fichier FXML.

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

```java
package modele.gameLoop;

import java.util.ArrayList;
import java.util.List;

/** Classe abstraite qui implemente Runnable afin de creer des Threads pour une boucle de jeu
 *
 */
public abstract class BoucleDeJeu implements Runnable{

    /** Liste d'observateurs de la boucle de jeu
     *
     */
    private List<Observateur> observateurList = new ArrayList<>();

    @Override
    public abstract void run();

    /** Permet de notifier les observateurs de la liste
     * Methode utilisee a chaque tour d'une boucle de jeu
     */
    public void notifyObservateurs(){
        for (Observateur o: observateurList) {
            o.update();
        }
    }

    /** Permet d'ajouter un observateur a la liste
     * @param o  un objet de type Observateur
     */
    public void addObservateur(Observateur o) {
        observateurList.add(o);
    }
}
```

Nous utilisons 3 boucles de jeu, une pour l'apparition des pièces, des trous et une pour le temps.