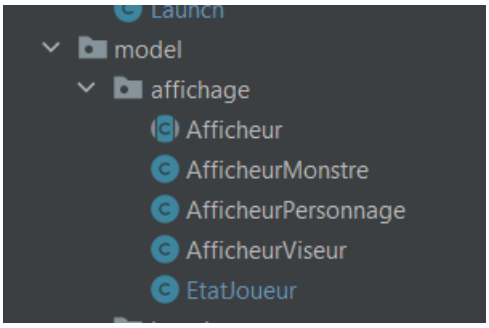
Je maîtrise les règles de nommages Java :

Package minuscule, Classe qui commence en Majuscule, constante en majuscule, méthode (update) première lettre en minuscule, variable ont leur première lettre en minuscule




```java
private Manager manager;
//Vitesse de déplacement des monstres
private double STEP = 2.5;

public DeplaceurMechant(Collisionneur collisionneur, Manager manager)
{
    super(collisionneur);
    this.manager=manager;
}

/**
 * Méthode update du déplaceur des monstres
 * Les monstres se déplacent vers le centre du joueur
 */
@Override
public void update() {
    //On vérifie que la liste monstre n'est pas vide
```

Je sais binder bidirectionnellement deux propriétés JavaFX :

```java
Bindings.bindBidirectional(ptsDeVie.textProperty(), manager.getJoueur().pvProperty(), converter);
Bindings.bindBidirectional(timer.textProperty(), manager.getTimer().tempsProperty(), converter);
```

Je sais binder unidirectionnellement deux propriétés JavaFX :

```java
joueurVue.xProperty().bind(manager.getJoueur().posXProperty());
joueurVue.yProperty().bind(manager.getJoueur().posYProperty());
```

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

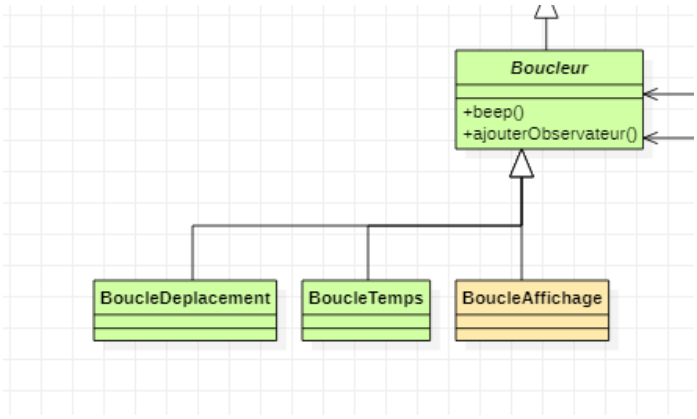
Je sais contraindre les éléments de ma vue, avec du binding FXML :

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

Je sais éviter la duplication de code :

```java
if(joueur.getPv()==0){
    for(Thread thread: listeThread){
        thread.stop();
    }
```

Je sais hiérarchiser mes classes pour spécialiser leur comportement :



Je sais intercepter des évènements en provenance de la fenêtre JavaFx :

```
//Appel de la méthode supprimerTouche quand une touche est lachée
Launch.getPrimaryStage().addEventFilter(KeyEvent.KEY_RELEASED, new EventHandler<KeyEvent>() {
    @Override
    public void handle(KeyEvent event) { manager.getDeplaceur().supprimerTouche(event.getCode()); }
});
```

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

Voir diagramme de classe

Je sais gérer la persistance de mon modèle :

Non fait

Je sais utiliser à mon avantage le polymorphisme :

Voir dans le code du projet

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

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

Voir dans le code du projet

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

```
private List<Thread> listeThread;

//liste de rectangle monstres
private List<Rectangle> listeRectangle;
//listeObservable contenant les monstres
private ObservableList<Personnage> oListeMonstre;
```

Je sais utiliser les lambda-expression :

Voir code

Je sais utiliser les listes observables de JavaFX.

```
//listeObservable contenant les monstres
private ObservableList<Personnage> oListeMonstre;
```

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

```
//Convertisseur string en Number
StringConverter<Number> converter = new NumberStringConverter();
```

```
Bindings.bindBidirectional(ptsDeVie.textProperty(), manager.getJoueur().pvProperty(), converter);
```

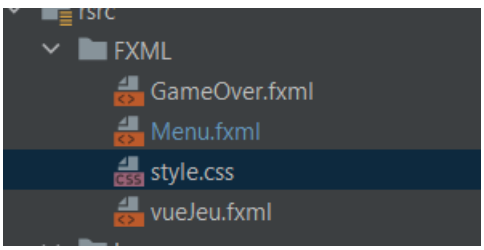
Je sais utiliser un fichier CSS pour styler mon application JavaFX :

```
#jeu{
    -fx-background-image: url("/Image/Background.png");
    -fx-background-repeat: stretch;
    -fx-background-size: 1200 800;
    -fx-background-position: top left;
    -fx-effect: dropshadow(three-pass-box, black, 30, 0.5, 0, 0);
}
```

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

Voir code

Je sais développer un jeu en JavaFX en utilisant FXML :



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

```java
*/
public abstract class Boucleur implements Runnable {

    private List<Observeur> observateur = new ArrayList<>();


    public void ajouterObservateur(Observeur o) { observateur.add(o); }

    /**
     * Méthode qui envoie le signal à tout les Observeurs
     */
    public void beep(){
        for(Observeur o : observateur){
            Platform.runLater(() -> o.update());
        }
    }
```

```java
boucleDeplacement = new BoucleDeplacement();
boucleDeplacement.ajouterObservateur(deplaceurJoueur);
boucleDeplacement.ajouterObservateur(deplaceurMechant);
boucleDeplacement.ajouterObservateur(deplaceurProjectile);
```