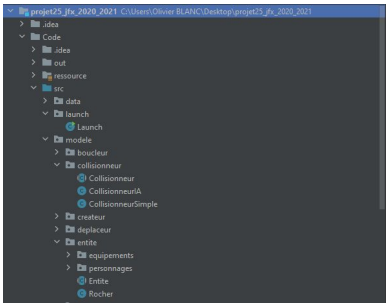
Escape the Coronavirus: Olivier Blanc & Victor Mommalier

Fichier de preuves

- Je maîtrise les règles de nommage Java.



- Je sais binder bidirectionnellement deux propriétés JavaFX.

  Nous utilisons le bind bidirectionnelle pendant la configuration de la partie, entre le manager et la vue de la configuration de la partie « SetupPartie »:

```
public void initialize() { saisieSurnom.textProperty().bindBidirectional(PartieVue.m.pseudoProperty()); }
```

- Je sais binder unidirectionnellement deux propriétés JavaFX.

  Nous utilisons le bind unidirectionnel dans le Manager afin d'afficher les données tels que le Temps, le nombre de Kills, le Pseudo et le nombre de PV restant:

```
public void initialize(){
    kill.textProperty().bind(m.killProperty());
    pseud.setText(m.getPseudo());
    temps.textProperty().bind(m.secondesProperty());
    vie.textProperty().bind(m.vieProperty());
```

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

  Oui, dans chacune de nos classes nous respectons une certaine nomenclature afin de garder un code lisible et compréhensible rapidement autant pour nous que pour une personne qui découvre le projet:

```java
public class BoucleurSimple extends Boucleur{
    @Override
    public void run() {

        while (super.actif){
            timeBeep();
            try {
                Thread.sleep( millis: 100);
            } catch (InterruptedException e) {
                actif = false;
            }
        }
    }
}
```

```java
public abstract class CreateurEntite {

    public abstract PersoPrincipal creerPersoPrincipal(Carte carte);
    public abstract void creerIA(Carte carte);
    public abstract void creerRocher(Carte carte);
    public abstract void creerCombinaison(Carte carte);
    public abstract void creerMasque(Carte carte);
    public abstract void creerVisiere(Carte carte);

}
```

```java
public class Score implements Serializable {

    private transient IntegerProperty score = new SimpleIntegerProperty();
        public Integer getScore(){return score.get();}
        public IntegerProperty scoreProperty(){return score;}
        public void setScore(Integer score){this.score.set(score);}


    private transient StringProperty pseudo = new SimpleStringProperty();
        public String getPseudo(){return pseudo.get();}
        public StringProperty pseudoProperty(){return pseudo;}
        public void setPseudo(String pseudo){this.pseudo.set(pseudo);}

    private LocalDateTime date;

    public Score (int score, String pseudo, LocalDateTime date){
        this.score.set(score);
        this.pseudo.set(pseudo);
        this.date = date;
    }

    @Override
    public String toString() {
        return pseudo.get() + ": " + score.get() + " fait le " + date.getDayOfMonth() + "/"+date.getMonth() + "/"+date.getYear() + "
    }

}
```

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

  Nous avons fait le biding FXML en affichant le nom du joueur saisie dans le TextField et en désactivant le bouton start si le test n'est pas saisie:

```
48  <HBox alignment="CENTER" prefHeight="100.0" prefWidth="200.0">
49      <Label text="Bonne chance "  textFill="WHITE"/>
50      <Label text="${saisieSurnom.text}"  textFill="WHITE"/>
51  </HBox>
52  <Button disable="${saisieSurnom.text == ''}" onAction="#clickStart" text="Start" textAlignment="CENTER" BorderPane.alignment="CENTER">
```

- Je sais définir une CellFactory fabriquant des cellules qui se mettent à jour au changement du modèle.
  - Création d'une listeCell pour pouvoir afficher le toSting() d'un score

```
listeScores.itemsProperty().bind(tabScore.lesScoresProperty());
listeScores.setCellFactory((param)->new CellScore());
```

```
public class CellScore extends ListCell<Score> {
    @Override
    protected void updateItem(Score score, boolean b) {
        super.updateItem(score, b);
        if (score != null){
            textProperty().setValue(score.toString());
        }
    }
}
```

- Je sais développer une application graphique en JavaFX en utilisant FXML.

Toutes nos pages de vues sont basées en FXML:

```
            <Font size="24.0" />
         </font>
         <BorderPane.margin>
            <Insets top="15.0" />
         </BorderPane.margin>
      </Text>
   </top>
   <center>
      <VBox alignment="TOP_CENTER" styleClass="boutons">
         <Button alignment="CENTER" contentDisplay="CENTER" layoutX="259.0" layoutY="144.0" mnemonicParsing="false"  onAction="#onNewGame"
            <padding>
               <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
            </padding>
         </Button>
         <Button alignment="CENTER" contentDisplay="CENTER" layoutX="259.0" layoutY="187.0" mnemonicParsing="false" onAction="#onTabScore"
            <padding>
               <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
         </padding>
         </Button>
         <Button alignment="CENTER" contentDisplay="CENTER" layoutX="259.0" layoutY="236.0" mnemonicParsing="false" onAction="#onInformatic
            <padding>
               <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
            </padding>
         </Button>
         <padding>
            <Insets top="90.0" />
         </padding>
      </VBox>
   </center>
</BorderPane>
```

- Je sais éviter la duplication de code.

  Nous évitons au maximum la duplication de code, par exemple dans la classe PartieVue, nous utilisons une méthode update() qui évite la duplication de code.

```
    m.getListeEntite().addListener((ListChangeListener.Change<? extends Entite> change) -> {
       change.next();
        for (Entite e : change.getAddedSubList()) {
            update(e);
        }


        for (Entite e : change.getRemoved()) {
            Iterator<Node> unIterateur = map.getChildren().iterator();
            while (unIterateur.hasNext()) {
                Node leNode = unIterateur.next();
                if (leNode.getUserData() == e) {
                        unIterateur.remove();
                    }
                }
            }
        }
    );

    m.getLeCollisionneur().widthProperty().bind(map.widthProperty());
    m.getLeCollisionneur().heightProperty().bind(map.heightProperty());
}

public void onStart(ActionEvent actionEvent) {
    for (Entite entite : m.getListeEntite()) {
        update(entite);
    }
```

```
private void update(Entite e){
    ImageView entiteAAfficher = new ImageView();
    entiteAAfficher.setUserData(e);
    entiteAAfficher.setImage(new Image(getClass().getResource(e.getImage()).toExternalForm()));
    entiteAAfficher.layoutXProperty().bind(e.xProperty());
    entiteAAfficher.layoutYProperty().bind(e.yProperty());
    entiteAAfficher.setFitHeight(e.getMaxHeight());
    entiteAAfficher.setFitWidth(e.getMaxWidth());
    map.getChildren().add(entiteAAfficher);
}
```

- Je sais hiérarchiser mes classes pour spécialiser leur comportement.
    - Par exemple pour le déplacement, nous utilisons une hiérarchie qui diffère les comportement des IA du comportement du PersonnagePrincipal.
      Pour le déplacement du personnage principal:

```
public void deplacerHaut(Entite e){
    if (leCollisionneur.canMove(e.getX(), y: e.getY()-pas)){
        e.setPosiPerso(e.getX(), y: e.getY()-pas);
    }
}
```

      Pour le déplacement de l'IA:

```
public void deplacerHaut(Entite e){
    if (leCollisionneur.canMove(e.getX(), y: e.getY()-pas)){
        e.setPosiPerso(e.getX(), y: e.getY()-pas);
        ((CollisionneurIA)leCollisionneur).contaminerAuContacte(e);
    }
    else{
        ((IA) e).resetDest();
    }
}
```

- Je sais intercepter des événements en provenance de la fenêtre JavaFX.

  Les événements que nous récupérons sont les clics de l'utilisateur lorsqu'il souhaite changer de page, quand il consulte les score pu veut lancer une nouvelle partie:

```
public void onRetour(ActionEvent actionEvent) throws IOException {
    Parent container = FXMLLoader.load(getClass().getResource( name: "/MainWindow.fxml"));
    container.getStylesheets().add("css/style.css");
    Launch.fenetrePrincipale.setScene(new Scene(container));
}
```

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

  Oui nous avons dissocié chaque classe pour qu'elle respecte au maximum le "Single responsibility Principle":

- Je sais gérer la persistance de mon modèle.

  Oui nous sauvegardons des scores pour les restituer ensuite dans une vue dans une classe "TableauScore":

```java
public class SauvegarderFile extends Sauvegarder {

    @Override
    public void SauvegarderDonnee(Score score) {
        TableauScore lesScores = new TableauScore();

        try (var in = new ObjectInputStream(new FileInputStream( name: "scores.bin"))) {
            while (in.available() > 0){
                lesScores.ajouterScore((Score) in.readObject());
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }

        lesScores.ajouterScore(score);

        try (var out = new ObjectOutputStream(new FileOutputStream( name: "scores.bin"))) {
            for (Score sc: lesScores.getLesScores()) {
                out.writeObject(sc);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
@Override
public TableauScore chargerDonnee(){
    TableauScore lesScores = new TableauScore();

    try{
        File test=new File( pathname: "scores.bin");
        if (test.createNewFile()) {
            System.out.println("fichier crée");
    }
    }catch (Exception e){
        System.out.println(e.getMessage());
    }


    try (var in = new ObjectInputStream(new FileInputStream( name: "scores.bin"))) {
        while(in.available() > 0){
            var sc = (Score) in.readObject();
            lesScores.ajouterScore(sc);
        }
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }

    return lesScores;
}
```

- Je sais surveiller l'élément sélectionné dans un composant affichant un ensemble de données.

- Je sais utiliser à mon avantage le polymorphisme.

  Dans le manager nous utilisons le polymorphisme pour pouvoir changer rapidement de Deplaceur ou de Serializer sans refaire tout le code à chaque fois:

```java
private Boucleur leBoucleur = new BoucleurSimple();
private Boucleur leBoucleurIA = new BoucleurIA();
private Spawner leSpawner = new SpawnerSimple();
private Collisionneur leCollisionneur = new CollisionneurSimple(carte, m: this);
private Collisionneur leCollisionneurIA = new CollisionneurIA(carte, m: this);
private Ramasseur leRamasseur = new RamasseurSimple(carte);
private Deplaceur leDeplaceur = new DeplaceurSimple(leCollisionneur);
private Deplaceur leDeplaceurIA = new DeplaceurIA((CollisionneurIA) leCollisionneurIA);
private SauvegarderFile leSerializer = new SauvegarderFile();
```

- Je sais utiliser certains composants simples que me propose JavaFX.

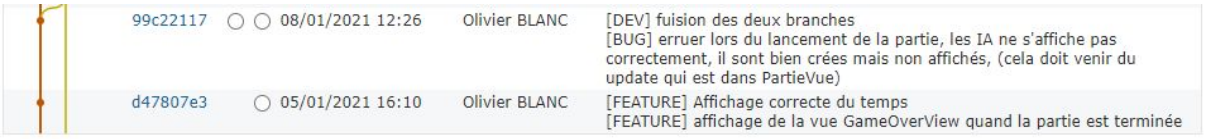
  Oui nous avons utilisé des composants tel que des BorderPane, des comboBox, des boutons ou encore des Label:

```xml
<BorderPane prefHeight="213.0" prefWidth="279.0" styleClass="background" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://ja
   <top>
      <Label text="Game Over!" textFill="WHITE" BorderPane.alignment="CENTER">
         <BorderPane.margin>
            <Insets top="20.0" />
         </BorderPane.margin>
      </Label>
   </top>
   <bottom>
      <Button onAction="#onRetour" mnemonicParsing="false" text="Retour au menu principal" BorderPane.alignment="CENTER">
         <BorderPane.margin>
            <Insets bottom="20.0" />
         </BorderPane.margin>
      </Button>
   </bottom>
   <center>
      <VBox alignment="CENTER" prefHeight="200.0" prefWidth="100.0" BorderPane.alignment="CENTER">
         <children>
            <Label contentDisplay="CENTER" text="Votre score:" textFill="WHITE" />
            <Label fx:id="score" text="Label" textFill="WHITE" />
         </children>
      </VBox>
   </center>
</BorderPane>
```

- Je sais utiliser certains layout que me propose JavaFX.

Oui nous avons utilisé essentiellement BorderPane, Pane mais aussi VBox ou HBox:

- Je sais utiliser GIT pour travailler avec mon binôme sur le projet.
    - Nous travaillons depuis le début du projet avec GIT sur la forge Clermont-Ferrand. Nous utilisons les mots clé tel que “[DEV]”, “[FEATURE]”, “[BUG]”, … pour être compréhensible au maximum, de plus chacun d’entre nous à travaillé sur ça branche pour éviter au maximum les conflits:



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

  Oui car pour pouvoir accéder au manager dans toute nos vues nous avons dû le rendre static, c’était surtout pour notre vue GameOver pour pouvoir récupérer le score du joueur :

```
public static final Manager m = new Manager();
```

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

Nous utilisons beaucoup la liste ObservableList mais nous avons aussi utilisé une ChoiceBox pour pouvoir choisir notre niveau de difficulté:

```
@FXML
public ChoiceBox choiceDifficulty;
```

- Je sais utiliser les lambda-expression.

  Oui principalement dans la vue lors d'une mise à jour pour éviter d'écrire une première boucle for nous utilisons " ->" :

```
m.getListeEntite().addListener((ListChangeListener.Change<? extends Entite> change) -> {
```

- Je sais utiliser les listes observables de JavaFX.

  Oui car notre carte utilise une liste observable d'entité pour connaître le nombre d'ia ou de rochers, liste observé par la vue pour modifier l'affichage:

```
public class Carte {


    private ObservableList<Entite> lesEntites = FXCollections.observableArrayList();
```

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

  Oui dans notre vue nous affichons le temps en secondes hors on utilise un label qui possède une textProperty donc on fait un convertisseur d'un int à une string:

```
temps.set(String.valueOf(tps));


if (tps%10 == 0){
    secondes.set(String.valueOf(Integer.parseInt(secondes.get())+1));
}
```

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

Oui pour notre background de l'application nous avons utilisé un fichier css.

```css
.background{
    -fx-background-image: url('/images/bg/background.png');
    -fx-background-repeat: stretch;
    -fx-background-position: center center;
    -fx-background-attachment: fixed;
    -fx-background-size: cover;
}

.map{
    -fx-background-image: url('/images/bg/map.png');
    -fx-background-repeat: stretch;
    -fx-background-position: center center;
    -fx-background-attachment: fixed;
    -fx-background-size: cover;
}

.boutons{
    -fx-spacing :25;
}
.blocBlanc{
    -fx-background-color: rgba(255,255,255,0.70);
}
```

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