






PREUVES MAITRISE DES COMPÉTENCES

Documentation : (dossier doc)

- 1- Je sais concevoir un diagramme UML intégrant des notions de qualité et correspondant exactement à l'application que j'ai à développer. 
- 2- Je sais décrire un diagramme UML en mettant en valeur et en justifier les éléments essentiels 
- 3- Je sais documenter mon code et en générer la documentation.  
(Dossier javadoc)
- 4- Je sais décrire le contexte de mon application, pour que n'importe qui soit capable de comprendre à quoi elle sert.
(Voir README)
- 5- Je sais faire un diagramme de cas d'utilisation pour mettre en avant les différentes fonctionnalités de mon application. 

Code :

- 1- Je maîtrise les règles de nommage Java 

- Packages : écrits en minuscules

```
> launch
> model
> view
```

- Classes, Interfaces et Constructeurs : La première lettre est en majuscule. Si le nom est composé de plusieurs mots, la première lettre de chaque mot doit être en majuscule, ne pas mettre de caractère underscore '_'

```
public class ClassicTower extends Tower
public interface IAttacker
public ClassicTower(int x, int y)
```

- Méthodes : Leur nom devrait contenir un verbe. La première lettre est obligatoirement une minuscule. Si le nom est composé de plusieurs mots, la première lettre de chaque mot doit être en majuscule sans mettre de caractère underscore '_'. Les méthodes pour obtenir la valeur d'un champ doivent commencer par get suivi du nom du champ. Les méthodes pour mettre à jour la valeur d'un champ doivent commencer par set suivi du nom du champ

```
public boolean updateLocations()
public int getDirection() {return direction;}
public void setDirection(int direction) {this.direction = direction;}
```

- Les variables : La première lettre est obligatoirement une minuscule. Si le nom est composé de plusieurs mots, la première lettre de chaque mot doit être en majuscule, ne pas mettre de caractère underscore '_'.

```
private boolean dead;
private boolean pathFinished;
private int movementSpeed;
```

- Les constantes : Toujours en majuscules, chaque mot est séparés par un underscore '_'. Ces variables doivent obligatoirement être initialisées lors de leur déclaration.

```
private static final int BUILD_TIME_SECONDS = 2;
private static final int DEFAULT_SELL_COST = 35;
```

2- Je sais binder bidirectionnellement deux propriétés JavaFX

view/MainMenu -> ligne 55

```
pseudoField.textProperty().bindBidirectional(manager.pseudoProperty());
```

Binding sur la propriété pseudo dans la classe Manager

3- Je sais binder unidirectionnellement deux propriétés JavaFX.

view/MainMenu -> ligne 118

```
Text liveText = new Text();
liveText.textProperty().bind(gameManager.getGame().livesProperty().asString());
```

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

- Des commentaires sont appliqués au-dessus des méthodes :

model/gamelogic/map -> ligne 22

```
/**
 * Fonction qui permet de lire un format de map sur un fichier texte
 * @return int [][] map
 */
public int[][] loadMap() {
```

- Notre code est indenté et les espacements sont respectés :

model/gamelogic/map -> ligne 96

```
for (int y = 0; !scanSwitch; y++) {
    if (map[y][x] > 2 & map[y][x] < 7 & y != previousY) {
        pathXY.add(new Coordinate(x, y));
        scanSwitch = true;
        previousY = y;
    }
}
```

- Les propriétés ainsi que ce qui leurs sont liés sont disposés sur 4 lignes avec une indentation améliorant la lisibilité


model/gamelogic/ GameState -> ligne 18

```
private 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 IntegerProperty timeSeconds = new SimpleIntegerProperty();
public int getTimeSeconds() {return timeSeconds.get();}
public IntegerProperty timeSecondsProperty() {return timeSeconds;}
public void setTimeSeconds(int timeSeconds) {this.timeSeconds.set(timeSeconds);}
```

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

ressources/fxml/MainMenu -> ligne 22

```
<Label fx:id="helloLabel2" text="{pseudoField.text}" maxWidth="300"/>
<TextField fx:id="pseudoField"/>
```

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

view/MainMenu -> ligne 59

```
scoreList.setCellFactory(_ ->
    (ListCell) updateItem(gameState, empty) -> {
        super.updateItem(gameState, empty);
        if (!empty) {
            Label l1 = new Label();
            l1.setStyle("-fx-font-weight: bold;");
            l1.textProperty().bind(gameState.pseudoProperty());
            Label l2 = new Label();
            if (gameState.isVictory()) {
                l2.setTextFill(Color.GREEN);
            } else {
                l2.setTextFill(Color.RED);
            }
            l2.textProperty().bind(Bindings.format("Level : %s",gameState.levelProperty().asString()));
            Label l3 = new Label();
            l3.textProperty().bind(Bindings.format("Score : %s",gameState.scoreProperty().asString()));
            Label l4 = new Label();
            l4.textProperty().bind(Bindings.format("Time (seconds) : %s",gameState.timeSecondsProperty().asString()));
            HBox myHboxContent = new HBox(l1, l2, l3, l4);
            myHboxContent.setSpacing(5);
            setGraphic(myHboxContent);
        } else {
            textProperty().unbind();
            setText("");
            setGraphic(null);
        }
        setStyle("-fx-background-color: transparent;");
    }
);
```

7- Je sais éviter la duplication de code.

model/ ScoreRanking

Sans duplication de code

```
if (!rankingObservable.isEmpty()) {  
    if (rankingObservable.size() >= getNumberScores()) {  
        Collections.sort(rankingObservable);  
        GameState lowerState = rankingObservable.get(rankingObservable.size() - 1);  
        if (lowerState != gameState) {  
            rankingObservable.remove(lowerState);  
        }  
    }  
    rankingObservable.add(gameState);  
    if (rankingObservable.size() > 1) {  
        Collections.sort(rankingObservable);  
    }  
}
```

(méthode sort() utilisé deux fois car la première fois c'est pour récupérer la plus petite statistique de partie sans avoir à boucler)

Au lieu de :

Avec duplication de code

```
if (!rankingObservable.isEmpty()) {  
    if (rankingObservable.size() >= getNumberScores()) {  
        Collections.sort(rankingObservable);  
        GameState lowerState = rankingObservable.get(rankingObservable.size() - 1);  
        if (lowerState != gameState) {  
            rankingObservable.remove(lowerState);  
        }  
        rankingObservable.add(gameState);  
        Collections.sort(rankingObservable);  
    }  
    else {  
        rankingObservable.add(gameState);  
        Collections.sort(rankingObservable);  
    }  
}  
else {  
    rankingObservable.add(gameState);  
}
```

8- Je sais hiérarchiser mes classes pour spécialiser leur comportement

Package : model/serialization

```
public abstract class AdministratorPersistence implements ISaveStates, ILoadStates {  
    @Override  
    public abstract void save(ScoreRanking scoreRanking);  
  
    @Override  
    public abstract void load(ScoreRanking scoreRanking);  
}
```

```
public class AdministratorPersistenceBinary extends AdministratorPersistence{  
  
    private static final File fileSerialization = new File( pathname: System.getProperty("user.dir") + "/co  
  
    @Override  
    public void save(ScoreRanking scoreRanking) {  
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fileSerialization))) {  
            ScoreRankingSerializable srs = new ScoreRankingSerializable();  
            StateSerializable gameStateSerialization;  
            for (GameState game : scoreRanking.getRanking()) {  
                gameStateSerialization = new StateSerializable(  
                    game.getPseudo(),  
                    game.getLevel(),  
                    game.getScore(),  
                    game.getTimeSeconds(),  
                    game.isVictory()  
                );  
                srs.getRanking().add(gameStateSerialization);  
            }  
            oos.writeObject(srs);  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

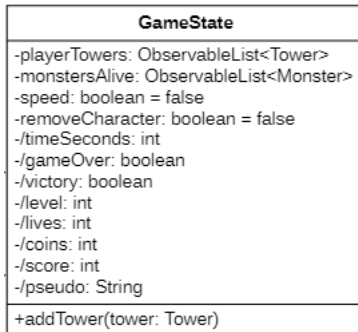
```
@Override  
public void load(ScoreRanking scoreRanking) {  
    if(fileSerialization.length() == 0) return;  
  
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fileSerialization))) {  
        ScoreRankingSerializable scoreRankingSerializable;  
        scoreRankingSerializable = (ScoreRankingSerializable) ois.readObject();  
        for (StateSerializable stateSerializable : scoreRankingSerializable.getRanking()) {  
            GameState gameState = new GameState(stateSerializable.getPseudo());  
            gameState.setLevel(stateSerializable.getLevel());  
            gameState.setScore(stateSerializable.getScore());  
            gameState.setTimeSeconds(stateSerializable.getTime());  
            gameState.setVictory(stateSerializable.isVictory());  
            scoreRanking.getRanking().add(gameState);  
        }  
    } catch (ClassNotFoundException | IOException e) {  
        e.printStackTrace();  
    }  
}
```

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

model/Manager -> ligne 24

```
ScreenController.getStage().setOnCloseRequest(event -> administratorPersistence.save(scoreRanking));
```

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



Responsable des statistiques de la partie



Responsable de contenir les statistiques des

11- Je sais gérer la persistance de mon modèle.

model/Manager

```
public Manager(ScoreRanking scoreRanking){
    this.scoreRanking=scoreRanking;
    administratorPersistence = new AdministratorPersistenceBinary();
    ScreenController.getStage().setOnCloseRequest(event -> administratorPersistence.save(scoreRanking));
    administratorPersistence.load(scoreRanking);
}
```

Sauvegarde dans fichier : code/ressources/serialization/saveScores.ser

```
saveScores.ser
aced 0005 7372 002c 6d6f 6465 6c2e 7365
7269 616c 697a 6174 696f 6e2e 5363 6f72
6552 616e 6b69 6e67 5365 7269 616c 697a
6162 6c65 12bc 7e52 8e74 7ecf 0200 014c
0007 7261 6e6b 696e 6774 0010 4c6a 6176
612f 7574 696c 2f4c 6973 743b 7870 7372
0013 6a61 7661 2e75 7469 6c2e 4172 7261
794c 6973 7478 81d2 1d99 c761 9d03 0001
4900 0473 697a 6578 7000 0000 0177 0400
0000 0173 7200 256d 6f64 656c 2e73 6572
6961 6c69 7a61 7469 6f6e 2e53 7461 7465
5365 7269 616c 697a 6162 6c65 d8bf 49b3
b084 5af2 0200 0549 0005 6c65 7665 6c49
0005 7363 6f72 6549 0004 7469 6d65 5a00
0776 6963 746f 7279 4c00 0670 7365 7564
6f74 0012 4c6a 6176 612f 6c61 6e67 2f53
7472 696e 673b 7870 0000 0001 0000 0024
0000 0011 0174 0003 7567 6f78
```

12- Je sais utiliser à mon avantage le polymorphisme. 🍴

model/Manager -> ligne 23

```
administratorPersistence = new AdministratorPersistenceBinary();  
ScreenController.getStage().setOnCloseRequest(event -> administratorPersistence.save(scoreRanking));  
administratorPersistence.load(scoreRanking);
```

Polymorphisme

Le polymorphisme nous permet l'avantage d'utiliser le patron de conception 'Stratégie'. Cela nous permet de changer l'algorithme des méthodes save et load à l'exécution. C'est-à-dire que les méthodes save et load qui seront appelées dépendront de la classe fille d'administratorPersistence instanciée.

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

Voir historique gitlab

Nous avons utilisé Git Bash nous permettant d'utiliser GIT. Nous avons commit, push, pull, créé des branches. Nous avons réglé des conflits.

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

```
private static final int BUILD_TIME_SECONDS = 2;  
private static final int DEFAULT_SELL_COST = 35;
```

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

view/MainMenu

```
@FXML  
private ListView scoreList;
```

16- Je sais utiliser les lambda-expression. 🍴

view/MainMenu

```
task.setOnFinished(event -> bar.setVisible(false));
```

view/creators/ CreatorProjectiles

```
Platform.runLater(() -> tilemapGroup.getChildren().add(projectileView));
```


17- Je sais utiliser les listes observables de JavaFX. 🍴

model/ ScoreRanking

```
private final ObservableList<GameState> rankingObservable = FXCollections.observableArrayList();
```

model/gamelogic/GameState

```
private final ObservableList<Tower> playerTowers = FXCollections.observableArrayList();  
private final ObservableList<Character> charactersAlive = FXCollections.observableArrayList();
```

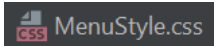
18- Je sais utiliser un convertisseur lors d'un bind entre deux propriétés JavaFX. 🍴

View/MainMenu -> ligne 57

```
nbScores.textProperty().bindBidirectional(manager.getScoreRanking().numberScoresProperty(), new NumberStringConverter());
```

19- Je sais utiliser un fichier CSS pour styliser mon application JavaFX. 🍴

Ressources/fxml/MenuStyle.css



launch/Main -> ligne 30

```
stage.getScene().getStylesheets().setAll(Main.class.getResource("fxml/MenuStyle.css").toExternalForm());
```

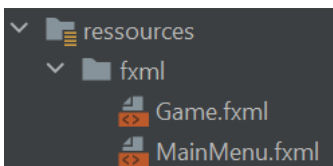
20- Je sais utiliser un formateur lors d'un bind entre deux propriétés JavaFX. 🍴

view/MainMenu -> ligne 74

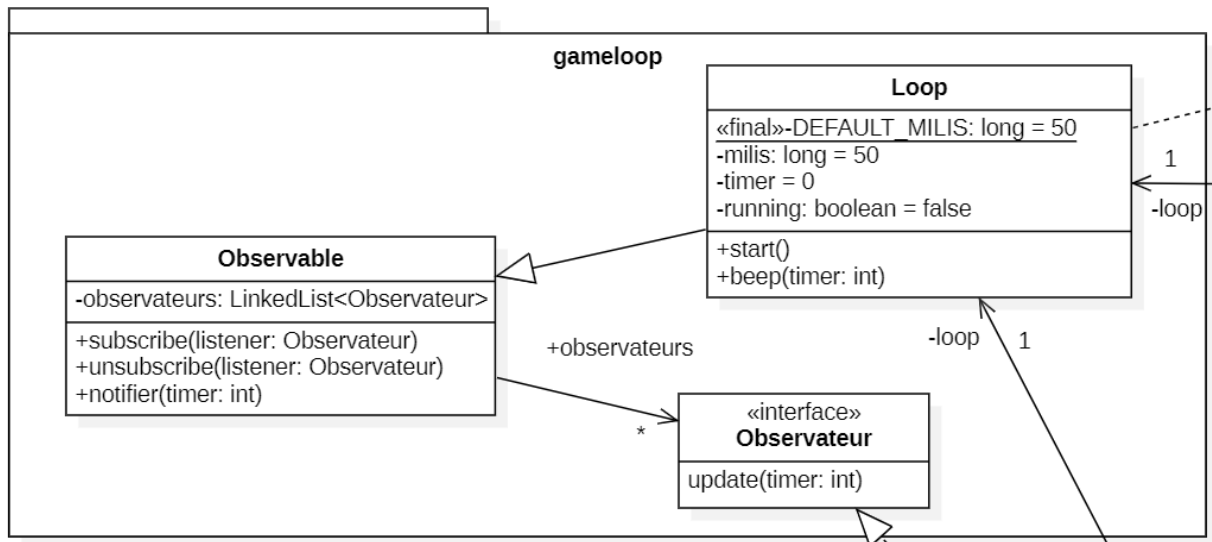
```
l2.textProperty().bind(Bindings.format("Level : %s", gameState.levelProperty().asString()));
```

21- Je sais développer un jeu en JavaFX en utilisant FXML. 🍴

Fichiers fxml utilisés :



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



Package : model/gameloop

```

public class Loop extends Observable implements Runnable {
    private static final long DEFAULT_MILLIS = 50;
    private long millis = 50;
    private int timer = 0;
    private boolean running = false;

    public static long getDefaultMillis() {return DEFAULT_MILLIS;}

    public long getMillis(){return millis;}
    public void setMillis(long millis){this.millis = millis;}

    public boolean isRunning(){return running;}
    public void setRunning(boolean run){running = run;}

```

```

public void start() {
    running = true;
    run();
}

```

```

@Override
public void run() {
    while(isRunning()) {
        try {
            sleep(millis);
            timer++;
            beep(timer);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public void beep(int timer) { Platform.runLater(() -> notifier(timer)); }

```