



Document Preuves

Bastien Malvezin (G3)

Matthias Bizet (G4)

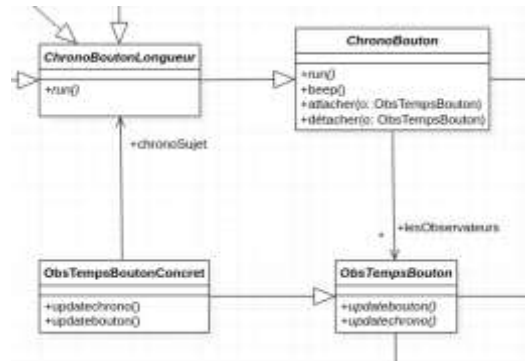
Documentation

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

Cf. « Diagrammes.mdj »

Notion de qualité : réalisation

patron de conception Observateur
de type comportemental



d'un

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

cf. « DescriptionDiagrammeUML.docx »

Je sais documenter mon code et en générer la documentation

```
/**
 * Méthode qui permet d'arrêter le mouvement, et qui vérifie si le carréjoueur peut sortir du niveau
 */
public void finMouvement(){
```

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

Cf. « contexte.docx »

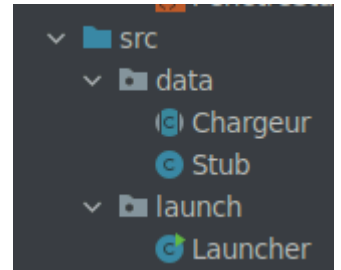
Je sais faire un diagramme de cas d'utilisation pour mettre en avant les différentes fonctionnalités de mon application

Cf. « Diagrammes.mdj »

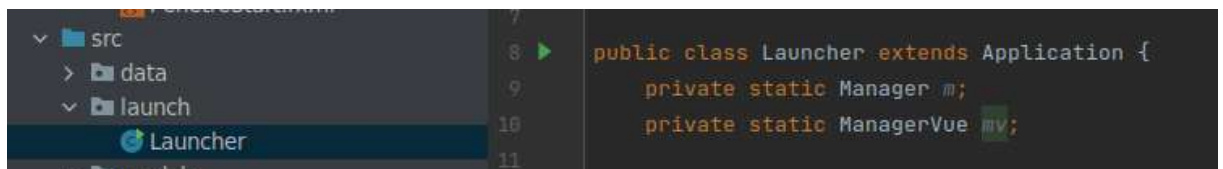
Code

Je maîtrise les règles de nommage Java

Les noms des classes sont en majuscule et les packages en minuscule :



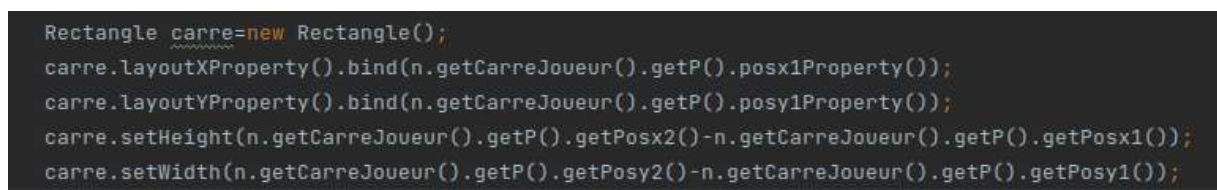
Les fichiers des classes ont le même nom que ces dernières :



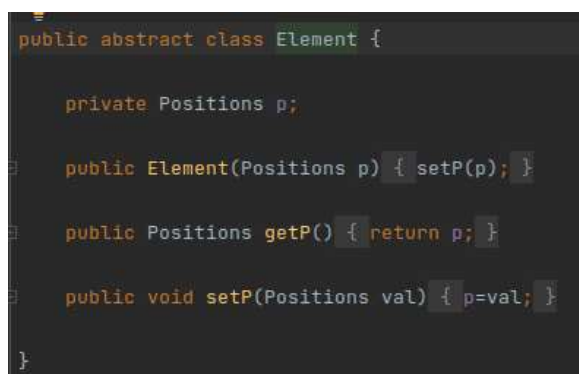
Je sais binder bidirectionnellement deux propriétés JavaFX

non

Je sais binder unidirectionnellement deux propriétés JavaFX



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



L'attribut Positions est accessible via un getter et on peut le modifier avec une méthode setP()

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

non

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

non

Je sais éviter la duplication de code

```
for (Collisionneur col : lesCollisionneurs) {  
    lesPos.add(col.Collision(niveauEnCours.getCarreJoueur(), d, niveauEnCours));  
}
```

On utilise le fait d'avoir une interface Collisionneur afin de ne pas avoir besoin de dupliquer l'appel aux collisionneurs et de pouvoir les faire en une seule boucle.

Je sais hiérarchiser mes classes pour spécialiser leur comportement

```
public abstract class Collisionneur {  
  
    public abstract Positions Collision (Element e, char d, Niveau niv);  
}  
  
public class CollisionneurMur extends Collisionneur {  
  
    /*...*/  
  
    @Override  
    public Positions Collision(Element e, char d, Niveau niv) {
```

Je sais intercepter des événements en provenance de la fenêtre JavaFX

```
@FXML  
public void initialize() {  
    boutonLancer.setOnAction(lancer());  
    boutonQuitter.setOnMouseClicked(quitter());  
}
```

La méthode lancer() se lance quand le boutonLancer est appuyé.

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

Même si la classe
déplaceur a besoin de
connaître les positions de
la collision, une autre
classe (Collisionneur) s'en
charge afin de garder une
seule responsabilité

```
/**
 * Classe permettant de déplacer un élément dans un Niveau
 */
public abstract class Deplaceur {

/**
 * Interface permettant de gérer les collisions dans un niveau
 */
public interface Collisionneur {
```

Je sais gérer la persistance de mon modèle

non

Je sais utiliser à mon avantage le polymorphisme

```
for (Collisionneur col : lesCollisionneurs) {
    lesPos.add(col.Collision(niveauEnCours.getCarreJoueur(), d, niveauEnCours));
}
```

On utilise le polymorphisme afin de pouvoir appeler la méthode Collision des différents collisionneurs, même si ces dernières sont différentes.

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

803dfdb		03/12/2021 10:00	Matthias BIZET	Suppression du répertoire Code/out
8147d8c4		03/12/2021 09:52	Matthias BIZET	Ajout du répertoire Code contenant le projet JavaFX + Ajout d'un package modele contenant les classes Element, CarreJoueur, Positions, Mur, Deplaceur et DeplaceurCarre
26f0fc1c		01/12/2021 12:01	Bastien MALVEZIN	Debut de l'explication du Use Case, et avancées sur le modèle
2f9018c3		01/12/2021 11:29	Bastien MALVEZIN	Création du dépôt + ajout des diagrammes, du contexte et d'une maquette

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

```
public abstract class Deplaceur {

    protected Element e;
```

Même si on ne déplace jamais un Element, mais seulement des objets en héritant, on utilise le type statique élément afin de pouvoir créer un DeplaceurMur par exemple, si l'on veut modifier notre programme pour le rendre plus complet

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

non

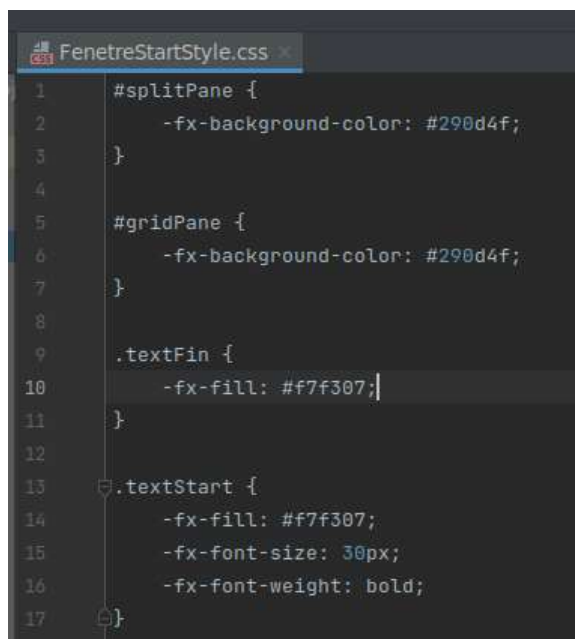
Je sais utiliser les lambda-expression. Je sais utiliser les listes observables de JavaFX

non

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

non

Je sais utiliser un fichier CSS pour styler mon application JavaFX



```
1 #splitPane {
2     -fx-background-color: #290d4f;
3 }
4
5 #gridPane {
6     -fx-background-color: #290d4f;
7 }
8
9 .textFin {
10     -fx-fill: #f7f307;
11 }
12
13 .textStart {
14     -fx-fill: #f7f307;
15     -fx-font-size: 30px;
16     -fx-font-weight: bold;
17 }
```

```
<Text strokeType="OUTSIDE" strokeWidth="0.0" styleClass="textStart" text="ESCAPE CUBE" textAlignment="CENTER" />
```

```
Scene s = new Scene(r);
s.getStylesheets().add(getClass().getResource("CSS/FenetreStartStyle.css").toExternalForm());
stage.setScene(s);
```

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

non

Je sais développer un jeu en JavaFX en utilisant FXML

cf. dossier FXML de rsrc

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

La boucle temporelle de ce jeu réside dans le fait qu'un bouton se désactive au bout d'un certains temps.

```
public abstract class Bouton extends Element {
    protected boolean appuye;
    private List<ObsAppuye> lesObservateurs;

    private ObjectProperty<Color> c;
    public Color getc() {return c.get();}
    public void setc(Color c) {this.c.set(c);}
    public ObjectProperty<Color> cProperty() {return c;}

    public boolean isAppuye() {return appuye;}

    public void setAppuye(boolean appuye) {
        if(appuye) {
            setc(Color.GREEN);
            notifier();
        }
        else setc(Color.RED);

        this.appuye = appuye;
    }

    public Bouton(Positions pos, boolean appuye) {
        super(pos);
        c=new SimpleObjectProperty<>();
        setc(Color.RED);
        lesObservateurs=new ArrayList<>();
        lesObservateurs.add(new ObsTempsBoutonConcret());
        attacher(lesObservateurs.get(0));
        this.appuye = appuye;
    }
}
```

```
public Bouton(Positions pos, boolean appuye) {
    super(pos);
    c=new SimpleObjectProperty<>();
    setc(Color.RED);
    lesObservateurs=new ArrayList<>();
    lesObservateurs.add(new ObsTempsBoutonConcret());
    attacher(lesObservateurs.get(0));
    this.appuye = appuye;
}

public void attacher(ObsAppuye o){
    o.setBoutonSujet(this);
    lesObservateurs.add(o);
}

public void detacher(ObsAppuye o) { lesObservateurs.remove(o); }

public void notifier(){
    for (ObsAppuye lesObservateur : lesObservateurs) {
        lesObservateur.updatebouton();
    }
}
}
```

A chaque fois qu'on va appeler la méthode setAppuye(), on appelle la méthode notifier() qui va appeler la méthode updatebouton de tous ses observateurs.

```
public class ObsTempsBoutonConcret extends ObsTempsBouton {
    //
    // Méthode lancée en chrono une fois le bouton appuyé et qui attache cet observateur au chrono
    //
    @Override
    public void updatebouton() {
        ChronoBoutonLongueur csl=new ChronoBoutonCount();
        Thread t = new Thread(csl);
        t.start();
        csl.attacher(this);
    }

    //
    // Méthode qui désinscrit le bouton lorsque le chrono arrive à la fin
    //
    @Override
    public void updatechronometre() { boutonSujet.setAppuye(false); }
}
```

```
package modele.chrones;

import static java.lang.Thread.sleep;

public class ChronoSoutenCourt extends ChronoSoutenLongueur{

    /**
     * Notifie les observateurs après que 18 secondes sont passées
     */
    @Override
    public void run() {
        try {
            sleep(18000);
            beep();
        }
        catch (Exception e) {
            return;
        }
    }
}
```