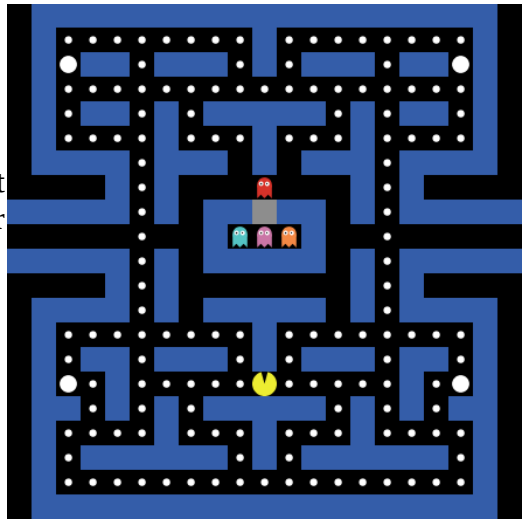
Projet Encadré - POO en java

Pac-Man

Critères d'évaluation :
- Qualité de l'analyse et du code associé
- Respect du Modèle Vue Contrôleur Strict
- Modularité
- Extensions proposées

1 Sujet

1.1 Pac-Man



Description du jeu : https://fr.wikipedia.org/wiki/Pac-Man

Précisions concernant l'implémentation :

- Le plateau est représenté par une grille de cases du côté de la vue et une grille de cases du côté du modèle. Les murs sont matérialisés par des cases pleines, les couloirs par des cases vides.

- Les mouvements sont discrétisés : on avance case par case, pas d'autres positions intermédiaires (pixel, etc.)

- Les entités (Fantômes et PacMan) sont associés à un processus indépendant. Du côté modèle, nous avons un code équivalent à :

```
abstract class ModèleEntité extends Observable implements Runnable {
    public void run() {
        while(actif) {
            réaliserAction() ;
            setChanged(); // notification de la vue, (4) sur le schéma MVC ci-dessous
            notifyObservers();
            sleep(tempsEntreActions) ; /* par exemple, Pac-Man est plus rapide durant quelques secondes
            après avoir mangé une super-pac-gomme,  tempsEntreActions peut varier */
        }
        grille.retirerDeLEnvironnement(this) ;
        setChanged(); // notification de la vue
        notifyObservers();
    }
}
```

Soit une classe Pac-Man héritant de ModèleEntité définira réaliserAction() comme le calcul de la nouvelle position de Pac-Man dans la grille suite à *(1) puis (3), sur le schéma MVC ci-dessous. Concernant les Fantômes, prévoir une IA simple (définir une direction à suivre, changer aléatoirement de direction si l'entité rencontre un mur).*

1.2 Travail en binôme

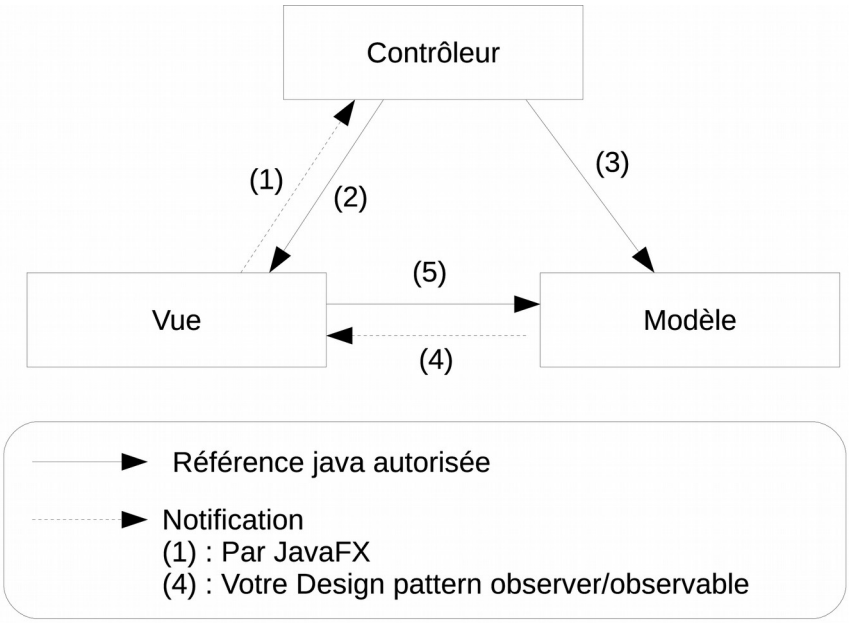
- Travail personnel entre les séances
- Évaluations individuelles
- Rapport par binôme (8 pages au maximum, listes de fonctionnalités et extensions (proportion de temps associée à chacune d'elles), documentation UML, justification de l'analyse, copies d'écran)
- Démonstration lors de la dernière séance encadrée (présence des deux binômes obligatoire)

1.3 Travail à réaliser

Développer une application graphique java la plus aboutie possible (utilisant JavaFX) de l'application, en respectant le modèle MVC Strict. Vous êtes responsables de votre analyse, et pouvez proposer des fonctionnalités. Veillez à proposer ces fonctionnalités incrémentalement, afin d'avoir une démonstration opérationnelle le jour de la démonstration. Il est recommandé de suivre les consignes de développement données dans la partie « Étapes de l'implémentation ». **Le code doit être le plus objet possible. Privilégiez donc une programmation objet plutôt qu'un algorithme central long et complexe.**

2 Rappel Modélisation MVC Strict

2.1 MVC Strict



MVC Strict :	**Application Calculette :**
(1) Récupération de l'événement JavaFX par le controleur	(1) récupération clic sur bouton de calculette
(2) Répercutions locale directe sur la vue sans exploitation du modèle	(2) construction de l'expression dans la vue (1,2...9, (,))
(3) Déclenchement d'un traitement pour le modèle	(3) déclenchement calcul (=)
(4) Notification du modèle pour déclencher une mise à jour graphique	(4) Calcul terminé, notification de la vue
(5) Consultation de la vue pour réaliser la mise à jour Graphique	(5) La vue consulte le résultat et l'affiche

Remarque : le code associé au contrôleur et à la vue peut être réalisé dans le même fichier .java tel que dans l'application *Calculette* (utilisation de classes anonymes ou de classes internes)

2.2 Modèle

Données :

Grille, Cases (états), Entités, Pac-Gommes, Super-Pac-Gommes, sablier, etc.

Processus :

Initialiser
Réaliser Actions
Tests la fin de partie
etc.

2.3 Vue Plateau

Pour commencer la vue de votre projet, inspirez-vous de l'application *Calculette* en JavaFX qui est disponible sur Spiral, qui respecte le MVC Strict et faites évoluer son code.

Ne pas utiliser FXML de JavaFX, ce n'est pas la priorité pour le projet (à moins de connaître très bien JavaFX, et de vouloir approfondir vos connaissances à ce sujet de façon autonome)

3 Étapes suggérées pour l'analyse et l'implémentation

3.1 Analyse sur papier : Modélisation Objet du problème (classes, périmètres des fonctionnalités, principaux traitements) : étudié en CM

3.2 Connexion MVC – Modèle presque vide

Utiliser l'application calculette comme base (MVC + javaFX), et faite la évoluer (chaque le type de cases graphique suivant vos besoins, changer le modèle, etc.).

Pour cette première phase, on souhaite simplement connecter la vue à un modèle « vide », afin d'obtenir le comportement suivant :

- Contrôleur : Réceptionnez les événements utilisateur
- Modèle : grille de positions et d'états à mettre à jour.
- Vue : mettre à jour l'affichage de la grille (suivant la grille du modèle). Pour commencer colorier les cases suivant le type de cases du modèle (ensuite seulement utiliser des images pour les cases si vous le souhaitez). Par exemple, pour un rafraîchissement global, on parcourt les cases côté Vue et Modèle en parallèle pour mettre à jour graphiquement.

Ceci permet de valider un comportement MVC Strict (avec modèle très simple) opérationnel pour la suite du développement. À cette étape les processus métiers ne sont pas implémentés.

Remarque : Il est normal d'avoir une structure de grille côté modèle et une structure de grille côté vue (gérées par javaFX), c'est nécessaire pour garantir l'indépendance du modèle et de la vue. Les rôles des grilles sont différents, il ne s'agit pas d'une redondance.

3.3 Écrire les traitements du modèle

3.4 Ajouter une ou plusieurs extensions suivant votre avancement

- Une extension de vous proposez
- IA Fantômes améliorée
- Éditeur/Générateur de Niveau
- Meilleurs scores
- Effet de wraparound
- Jeu collaboratif sur un même plateau ou en réseau
- etc.