LIFAPOO – Projet Printemps 2023-2024

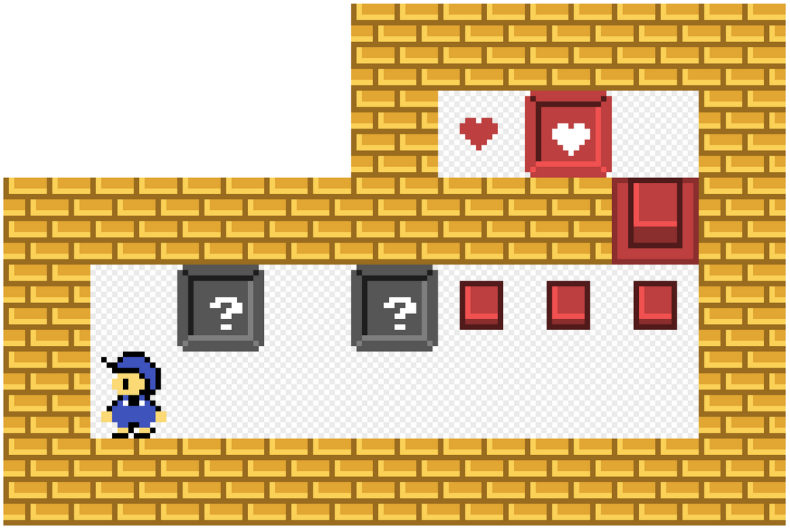
Projet Sokoban

Modélisation Objet, Modèle Vue Contrôleur, Swing

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

1 Sokoban

Ce projet consiste à développer le jeu de puzzle Sokoban (**https://www.sokobanonline.com/play/tutorials**)



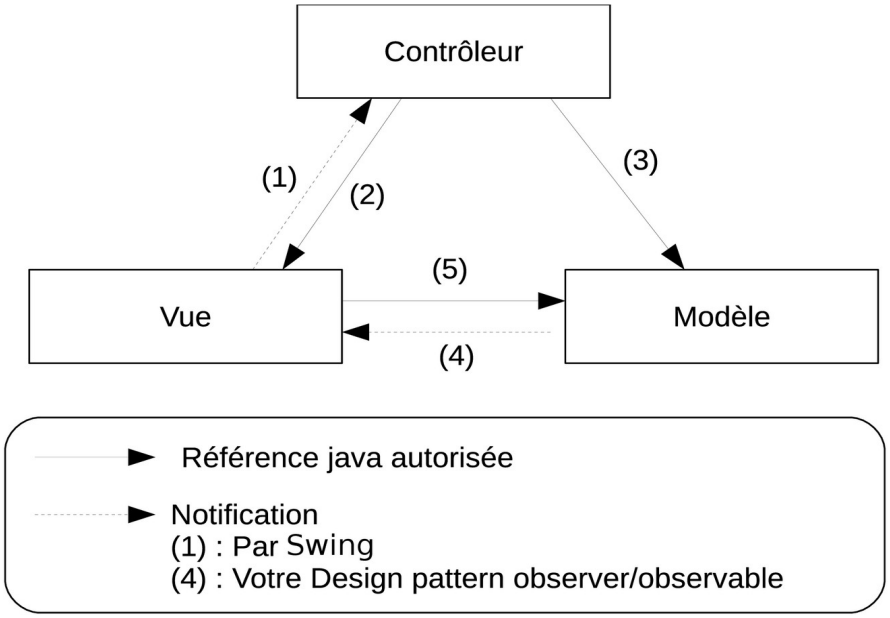
1.1 Travail en binôme obligatoire (ID binôme à renseigner sur Tomuss)

- Travail personnel entre les séances ;
- Évaluations individuelles ;
- Rapport par binôme : 6 pages au maximum, listes de fonctionnalités et extensions (indiquer la proportion de temps associée à chacune d'elles), copies d'écran, 1 diagramme UML de votre choix.
- Démonstration lors de la dernière séance encadrée (présence des deux binômes obligatoire).
  - Les 2 binômes doivent connaître et comprendre l'ensemble du code.

1.2 Travail à réaliser

Développer une application graphique Java la plus aboutie possible de l'application, en respectant le modèle MVC Strict. Vous devez utiliser le code fourni, sur lequel vous ajoutez des fonctionnalités. Veillez à proposer ces fonctionnalités incrémentalement, afin d'avoir une démonstration opérationnelle le jour de la soutenance. **Le code doit être le plus objet possible. Privilégiez donc une programmation objet plutôt qu'un algorithme central long et complexe.**

2 *Rappel : MVC Strict*



MVC Strict :
(1) Récupération de l'événement Swing par le controleur
(2) Répercutions locale directe sur la vue sans exploitation du modèle
(3) Déclenchement d'un traitement pour le modèle
(4) Notification du modèle pour déclencher une mise à jour graphique
(5) Consultation de la vue pour réaliser la mise à jour Graphique

Application Calculette :
(1) récupération clic sur bouton de calculette
(2) construction de l'expression dans la vue (1,2...9, (,))
(3) déclenchement calcul (=)
(4) Calcul terminé, notification de la vue
(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 vu en cours (utilisation de classes anonymes ou de classes internes).

2.1 Modèle

Données :

Cases fixes (Vide, Mur, Glace, etc.), Entités mobiles (Bloc, Héros), Grille, etc.

Fonctionnalités :

- Paramétrage de la partie (démarrer, quitter, pause, etc.)
- Initialiser : charger un puzzle
- Déroulement du jeu :
  - Chaque évènement de déplacement du héros agit sur l'environnement du puzzle
  - La partie se termine lorsque tous les blocs Cible sont bien placés

3 Étapes de l'analyse et de l'implémentation

3.1 Comprendre le code MVC fourni

Ce code ne propose pas de modèle détaillé, mais permet :
- de faire bouger le héros dans une grille
- de récupérer des évènements (bouton ou clavier)
- d’afficher une vue de la grille

3.2 Ajouter incrémentalement les fonctionnalités (tel que vu en CM)

- Initialiser puzzles
- Modéliser les comportements des pièces
- Détecter la fin de partie
- etc.

3.2 Ajouter une ou plusieurs extensions suivant votre avancement

4 Extensions, suivant votre avancement

Undo/Redo

Charger niveau à partir d’un fichier

Sauvegarde des meilleurs scores

Rendu graphique et sonore

Une extension que vous souhaitez proposer (à valider avec votre encadrant)