DELMAS Jean-Pacôme
FAUVIN Arthur
ROS Yann
DUBERNET Juliette
FITTIPALDI Sofiane

L3 Ingénierie informatique

Projet Sokoban
Enseignant : Jean-Michel COUVREUR

Année 2018-2019

Nota Bene : Le projet est réalisé en Java 9.

2) Travail réalisé

Fonctionnalités réalisées :

- Dans la fenêtre du menu :

  - Il y a 4 boutons (bouton pour charger le niveau, 2 boutons de sélection du niveau et un bouton jouer avec gestion d'erreur (on ne peut pas lancer de partie ni sélectionner un niveau si on n'a pas chargé les niveaux).
  - Le bouton charger charge le fichier texte "MicroCosmos.txt".
  - Quand on clique sur le bouton jouer, la fenêtre du menu se ferme puis la fenêtre du jeu s'ouvre.

- Dans la fenêtre du jeu :

  - Affichage du nombre de poussées (poussées de caisses), de déplacements de Soko et affichage du temps écoulé depuis le début de la partie.
  - Il y a les boutons de déplacements, d'undo-redo, de replay et de reset, qui sont également disponibles par raccourcis clavier.
  - Le bouton close permet de fermer la fenêtre du jeu (si par exemple on n'arrive pas à terminer le niveau) et d'ouvrir la fenêtre du menu.

- Dans la fenêtre de la fin de partie :

  - Si on termine un niveau, la fenêtre du jeu se ferme pour ouvrir cette fenêtre de fin de partie. Affiche un résumé de la partie :
    - Affichage du niveau gagné.
    - Affichage du temps pris pour terminer le niveau.
    - Affichage du nombre de déplacement et de poussées réalisées.
  - Bouton close qui permet de fermer la vue de fin de partie et ouvre la vue du jeu avec le niveau suivant.

Fonctionnalitées non réalisées :

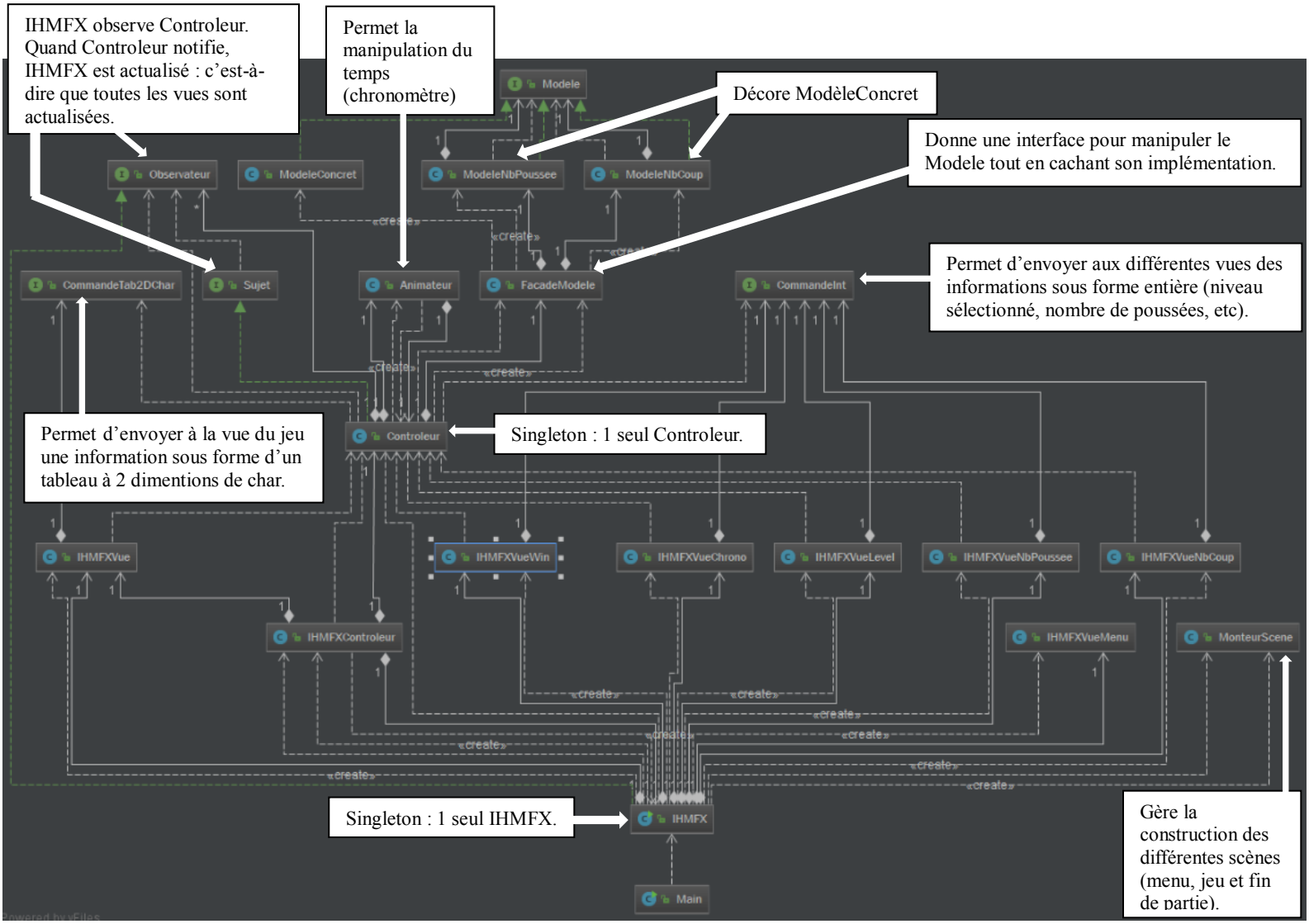
- Affichage du nom et de l'auteur du tableau, je n'ai pas trouvé l'information et vous nous avez dit que ce n'était pas nécessaire.

- Résolution des niveaux, c'est très complexe et surtout on n'avait pas de temps nécessaire pour le faire.
- L'image de Soko reste la même quand on se déplace (par exemple si on va à droite l'image n'est pas celle de Soko allant à droite, l'image de Soko est toujours la même).
- La disposition des bouttons n'est pas très ergonomique, on aurait pu faire mieux en modifiant le MonteurScene.

Aucun bug non corrigé

3) Diagramme de conception

(La qualité de l'image n'est pas très bonne mais on a pas réussi à faire mieux, on joint l'image du diagramme avec le rendu car la qualité est meilleure)



Capture d'écran du digramme de conception depuis IntelliJ

Le diagramme si dessus représente la conception globale de notre projet. Celui-ci regroupe un certain nombre de patrons de conceptions (ils seront cités après). Grâce à ce diagramme nous sommes capables de retracer l'intégralité de notre code et ainsi permettre de faire évoluer notre jeu plus efficacement.

Bien sûr, des nouvelles classes ainsi que d'autres patrons de conception peuvent être ajoutés afin d'optimiser l'ensemble de notre projet, c'est une possible évolution. Ici nous avons 8 designs patterns qui sont reliés les uns aux autres.

Afin de faire évoluer notre jeu, notre diagramme nous permettant d'avoir le recul nécessaire pour bien choisir les futurs patrons de conceptions possibles pour réaliser nos améliorations. De plus, au niveau de la maintenabilité du code c'est beaucoup plus pratique d'avoir à notre disposition le diagramme représentant l'ensemble du projet.

4) Patrons de conceptions

- Modèle Vue Controleur (MVC) : nous nous en sommes servi afin de pouvoir séparer le code, le modèle représentant les données à afficher à l'écran, la vue l'Interface Homme Machine (IHM), et le Controleur les liens entre les vues ou lorsque l'on clique sur des boutons. L'intérêt de ce pattern est que le code ainsi écrit est facilement modifiable du fait de la séparation que ce pattern impose.

- Singleton : Il nous permet de restreindre une classe à une seule instance d'elle-même afin d'éviter des conflits. Dans notre projet, nous en avons 2 : un Controleur et un IHMFX. Le Controleur sert à ce que toutes les actions passent par lui et IHMFX sert à être sur qu'il n'y a qu'une seule IHM.

- Décorateur : Ce pattern a une utilité ressemblant à celle de l'héritage. En effet, il permet de rajouter des fonctionnalités de manière plus simple que l'héritage. Dans notre projet, nous avons 2 décorateurs : ModèleNbCoup et ModèleNbPoussées. Ils servent à rajouter les fonctionnalités du nbCoup joués et du nombre de poussées.

- Sujet-Observateur : Il permet une gestion simplifiée d'observateur multiples sur un même objet observable. Dans notre projet, le Controleur implémente sujet, l'IHMFX implémente observateur. Le sujet permet de mettre à jour les observateurs.

- Façade : Il permet de cacher une conception et une interface complexe difficile à comprendre en fournissant une interface simple du sous-système. Dans notre programme, la façade cache le modèle.

- Monteur : Il permet de créer plusieurs objets à partir d'un objet source. Du coup, un même processus de construction peut créer différentes représentations. Dans notre projet, le monteur sert à monter les scènes.

- Commande : Il permet de séparer le code d'une action quelconque du code initiateur de cette même action. Un objet commande sert donc à communiquer une information.

- Animateur : Ça permet de gérer le temps, dans notre projet on gère un chronomètre.