Projet IHM - Banquise

Ce projet est à réaliser avant le 30 avril 2021, dans le cadre de l'Unité d'Enseignement "Programmation des Interfaces Homme-Machine". Il porte sur la reprise d'un projet proposé dans le cadre de l'Unité d'enseignement "Programmation Orientée Objet" dont les informations sont détaillées dans le fichier `README_POO.md`. Plus particulièrement, ce projet cible le développement d'une interface graphique à partir d'un jeu d'aventure déjà développé.

Composition de notre groupe :

- Yann Berthelot
- Vincent Commin
- Louis Leenart.

Sommaire

Installation

- Version du JDK de java : `15.0.1`
- Version du JDK de javafx : `15.0.1`
- Le lancement de l'application se fait via la classe `src/Main.java`.

Utilisation

Pour lancer une partie de jeu, il faut suivre les étapes suivantes :

1. Le jeu se lance via le fichier `src/Main.java`
2. Entrer un nom pour votre personnage dans la zone d'entrée en haut à gauche
3. Ouvrez le menu et cliquez sur `Launch Game` pour démarrer la partie
4. Maintenant que la partie est lancée, vous pouvez faire plusieurs actions :
    - Lister les passages disponibles en cliquant sur `List Crossings`. Vous pouvez ouvrir un passage fermé en cliquant dessus dans la liste des passage sur la partie droite de l'écran.
    - Lister les personnages présents sur la case en cliquant sur `List Personnages`. Vous pouvez parler avec un personnage en cliquant sur son nom dans la liste des personnages affichés sur la partie droite de l'écran.
    - Lister les objets sur la case en cliquant sur `List Items`. Vous pouvez ramasser un objet pour le mettre dans votre inventaire en cliquant sur son nom dans la liste des objets affichés sur la partie droite de l'écran.

    - Lister les objets dans l'inventaire de votre personnage en cliquant sur `Inventory`. Vous pouvez utiliser un objet en cliquant sur son nom dans la liste des éléments, puis en sélectionnant le personnage sur lequel utiliser l'objet.
    - Déplacer votre personnage à une case adjacente en cliquant sur le bouton flèche correspondant (si le passage vers la case est fermé, il faut l'ouvrir).
5. Le but du jeu est de trouver le personnage `Chief Scientist` qui est présent sur une des case du jeu. Il faut ensuite lui parler pour gagner la partie.

Notre approche de conception en partant de la conception existante

Notre séparation modèle/vue/contrôleur

Pour séparer correctement nos fichiers, nous nous sommes servis de la séparation modèle/vue/contrôleur vu en cours. Nous avons donc géré nos fichiers de la sorte :

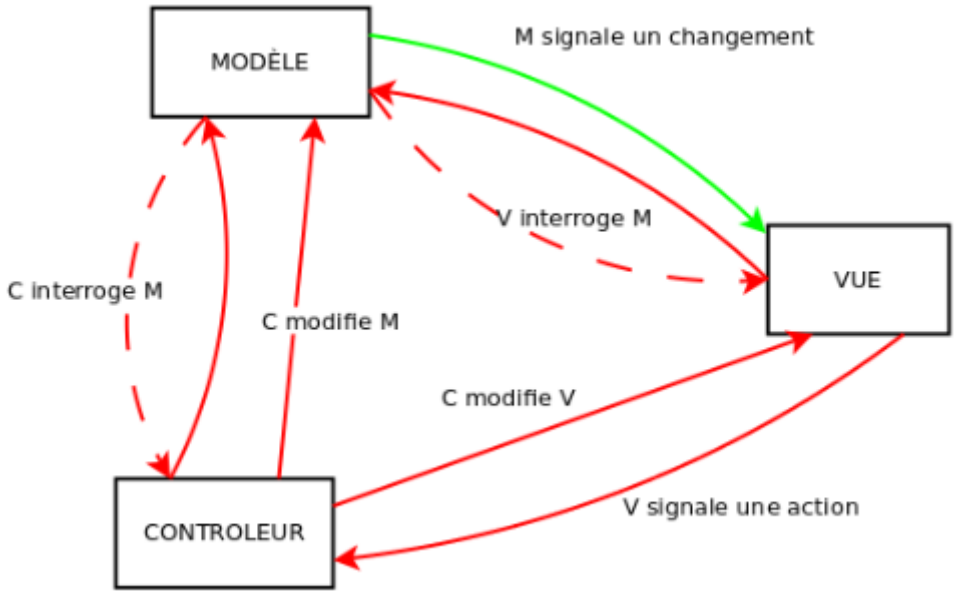
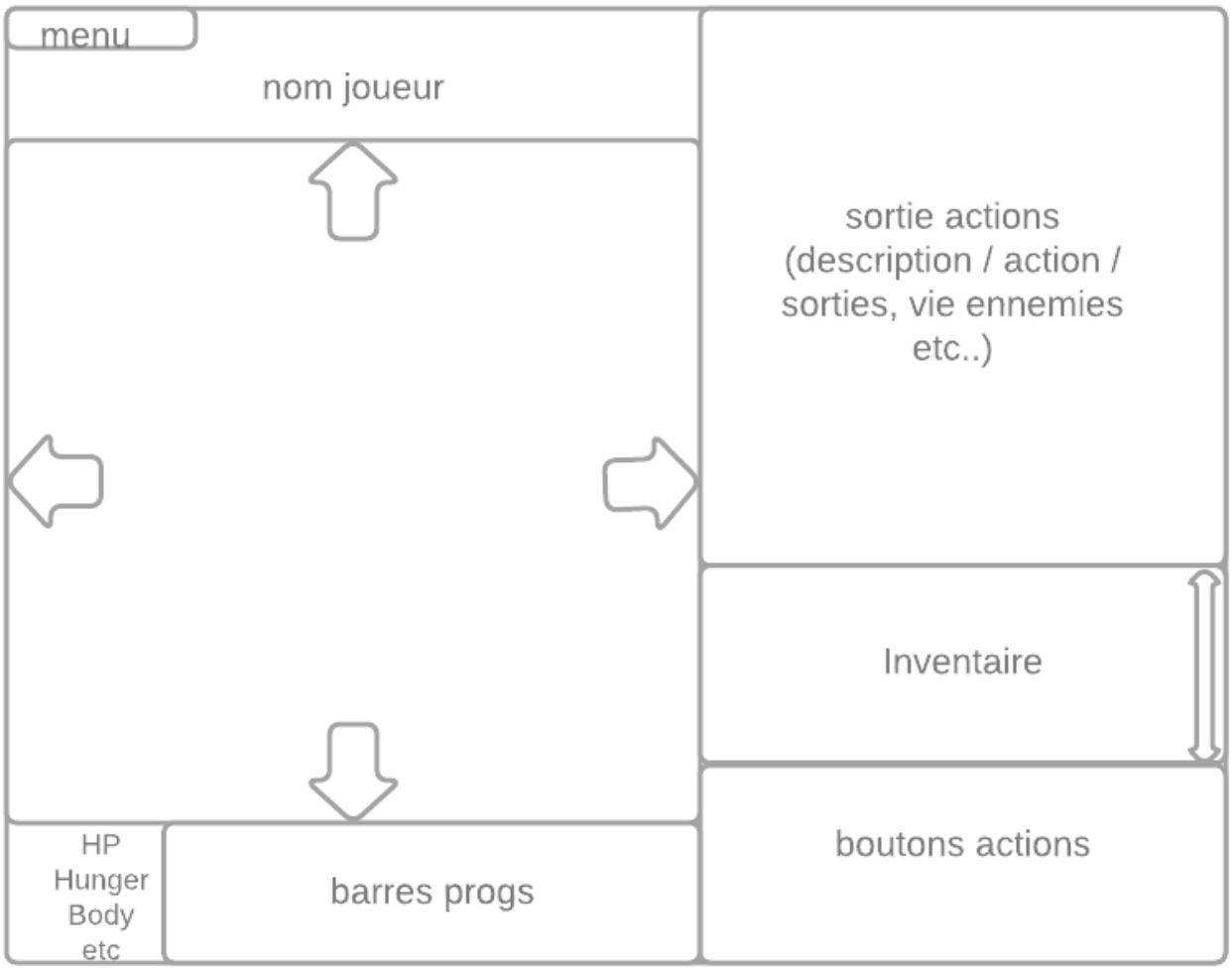


Schéma du cours expliquant le MVC (modèle/vue/contrôleur)

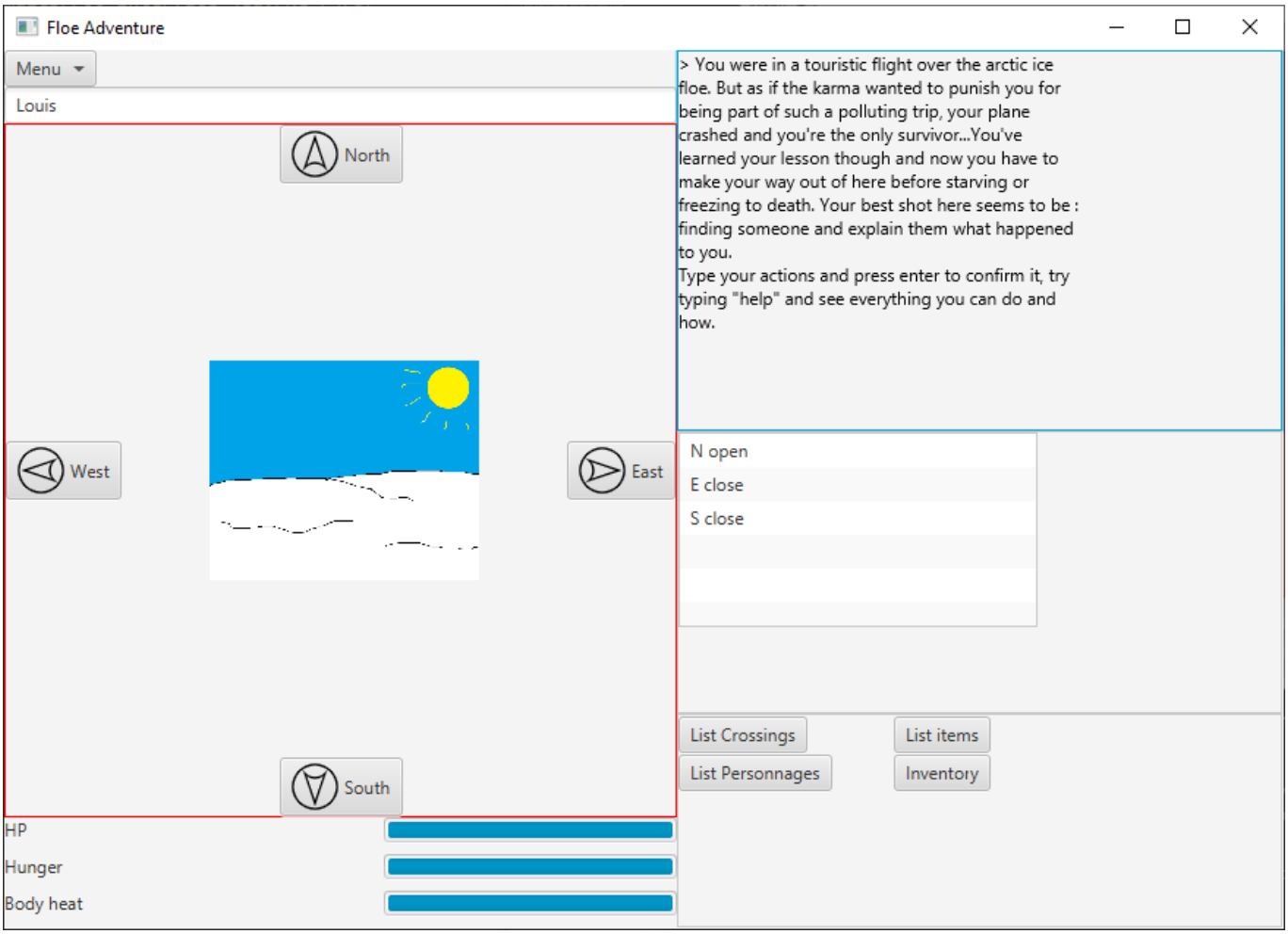
- Nos fichiers de base venant du projet de POO étaient ce qui définissait notre jeu, que ce soit les items ou bien le comportement des personnages ainsi que la génération des tuiles. Nous avons donc mis ces fichiers dans le répertoire "modèle".
- Pour la vue, nous y avons mis notre fichier .fxml ainsi que sa classe appelant le fxmlLoader permettant de le charger dans l'application.
- Enfin, pour le contrôleur, nous avons mis la classe liant la vue au modèle.

Notre interface

Notre jeu étant de base assez basique, nous n'avons pas eu besoin d'une multitude de scènes. Il n'y avait pas d'états particuliers pour les dialogues ou bien les combats, ce qui nous a simplifié la tâche et permis de tout mettre dans la même interface.



Interface imaginée au départ via Lucidchart



Interface rendue du jeu

On peut constater que ces deux interfaces sont quasiment les mêmes à l'exception de l'inventaire et de la sortie des actions.

A l'origine, l'inventaire devais uniquement être affiché dans la ListeView mais au fil du projet nous en avons décidé autrement. Maintenant, chaque action listant des objets/personnages/passages sont affichés dans la liste. Ceci nous permet en plus d'effectuer des actions sur les objets listés via la méthode OnMouseClicked.

Pareil pour la sortie des actions. On pensait y faire apparaître les points de vie des personnages mais on a préféré garder la description des actions ainsi que la sortie des dialogues.

Autres points

Affichage des actions dans l'interface

Une des difficultés rencontrée était due à notre conception. En effet tous nos affichages se faisaient sur la sortie standard. Cependant, nous devions écrire les sorties dans le TextFlow sur l'interface. Il a donc fallut changer la signature des méthodes en les faisant retourner leurs chaînes de caractères au lieu de les afficher.

Lancement du jeu

Un de nos problèmes venait du lancement du jeu. L'interface faisant appelle à nos méthodes, il était possible de jouer au jeu sans le lancer via le menu. Nous avons donc décidé de désactiver les boutons au lancement de

l'application et de les activer uniquement au lancement du jeu.

Fermeture du jeu lors de la fin

Lorsque le jeu est terminé (par exemple le personnage meurs), alors on modifie l'image du jeu pour l'indiquer au joueur, cependant on ferme la fenêtre au même moment, l'affichage de l'image n'est donc même pas visible. Pour palier à ce problème, il faudrait un peu avant de fermer la fenêtre ou attendre que l'utilisateur notifie la modification.

Mise en place des tours de jeu

Une des mécaniques principales de notre jeu était la gestion de tours pour les actions des personnages non-joueurs. L'interface appelant directement les méthodes nécessaires en fonction du déclencheur a rendue la tâche plus difficile. Pour résoudre ce problème, nous appelons donc le tour suivant à chaque action le nécessitant.

De plus, dans notre conception de base, chaque tour appelait le suivant jusqu'à la mort ou la victoire du joueur. Nous avons donc dû modifier la méthode en retirant l'appelle récursif ainsi que l'appelle de notre interpréteur de commandes.

Redimensionnement de la fenêtre

Pour rendre notre jeu le plus agréable à utiliser, nous avons opté pour mettre en place une fenêtre dont la taille des éléments s'adaptent à la taille de la fenêtre. Jusque-là assez peu expérimentés, et après avoir tenté plusieurs approches, nous avons décidé de nous appuyer sur les propriété des `HBox` et `VBox` et notamment `HBox.hgrow` et `VBox.vgrow`.
L'utilisation de ces propriétés nous permet alors d'avoir une interface dynamique quand les dimensions évoluent.
Cependant, nous n'avons pas réussi à implémenter le redimensionnement de certains éléments tel que le `TextFlow`, la `ListView` et l'`ImageView`.

Liste du contexte

Une des difficultés était la liste des personnages, passages et items. La ListView que nous utilisons fait appelle à des `ObservableList`. Or, nos méthodes retournant les personnages, passages et items nous retournaient uniquement des listes. Nous devons donc convertir les listes en `ObservableList` à chaque action du joueur demandant de lister.

Un autre problème était la sélection dans la liste. Vu que nous utilisons des `ObservableList<String>` Il nous faut récupérer l'item et non la référence de l'objet dans la liste.