Licence 3 Informatique	Programmation des IHM 2020-2021	X. Skapin A. Bergey

Projet *Interface d'un jeu d'exploration*

Le but est de développer une interface graphique à partir du jeu d'aventure développé en POO. Le descriptif de ce projet est rappelé en Annexe.

1 Noyau fonctionnel (modèle)

Vous reprendrez le noyau fonctionnel développé dans le jeu original. Il n'est pas demandé de le modifier (sauf pour corriger des bogues) ou de l'étendre ; vous ne serez pas notés la-dessus.

2 Interface (vue)

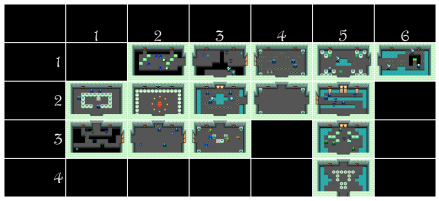
Dans le jeu original, l'interface était purement textuelle (sur console). L'objectif principal est de concevoir et mettre en place une interface graphique qui remplace l'originale. Le format de cette nouvelle interface est libre, mais on peut imaginer :

- des images représentant des flèches de déplacement pour de changer de pièce ;
- la description d'une scène dans un `TextField` ;
- des pièces représentées par des régions rectangulaires, contenant des formes simples (carrés, rectangles, disques) ou plus complexes pour symboliser les objets de ces pièces ;
- que lorsque la souris passe sur un objet intéressant, elle change de forme ou bien l'objet change de couleur ;
- de remplacer chaque commande textuelle (`go`, `help`...) par une icône appropriée ;
- que l'état (vie, pouvoirs...) de votre personnage est affiché sous forme de barres ;
- que la carte répertoriant les pièces visitées ou bien l'inventaire du héros sont accessibles *via* des onglets.

Les images ci-dessous sont des exemples d'interface (on ne vous demande pas des graphismes aussi sophistiqués ni des animations, mais n'hésitez pas à vous faire plaisir – images, musiques, vidéos...).

(a) The Secret of Monkey Island

(b) Eye of the Beholder II



(c) Golden Axe Warrior - map

3 Contrôleur

Le contrôleur assure la liaison entre le modèle et la vue. Toute action *via* l'interface graphique passera par le contrôleur pour appeler une ou des méthodes du modèle. Réciproquement, la modification du modèle peut entraîner une mise à jour de l'interface.

4 Travail demandé

Ce projet doit être réalisé en groupe de deux à quatre personnes[1]. Pour l'interface, vous ne devez utiliser que les classes de la librairie `JavaFX` (pas d'autres composants tous faits) et `SceneBuilder`, si vous le souhaitez, pour construire l'interface.

Le code doit être propre, modulaire, bien indenté, documenté. Les consignes données en début de semestre s'appliquent.

En particulier, vous devrez :

— Réutiliser au maximum les fonctionnalités de `JavaFX` étudiés en CM et en TP (les composants graphiques et les `bindings` notamment).
— Séparer le plus distinctement possible les parties "Modèle", "Vue" et "Contrôleur".
— Définir la partie "Vue" sous forme d'un ou plusieurs fichiers `FXML`. vous pouvez utiliser `SceneBuilder` pour concevoir ces fichiers `FXML`.
— Associer chaque fichier `FXML` à un fichier `Java` qui représentera son contrôleur.
— Veiller au redimensionnement de l'application : si vous ne fixez pas ces dimensions, le redimensionnement doit pouvoir être réalisé jusqu'à un certain point sans provoquer de catastrophe.

L'ensemble des sources sera regroupé dans le package `NomDeVotreApplication`[2], le noyau fonctionnel dans le package `NomDeVotreApplication.model`, l'IHM dans le package `NomDeVotreApplication.view` et les fichiers contrôleurs dans le package `NomDeVotreApplication.controller`. Libre à vous d'ajouter autant de sous-packages que nécessaires.

Vous rendrez un rapport d'au plus 20 pages, au format `pdf` (**à l'exclusion de tout autre**), expliquant comment est structurée votre application, comment vous avez résolu les problèmes de conception rencontrés, et comment votre application peut être étendue pour produire l'application rêvée.

Votre travail sera rendu sur UPdago dans une archive dont **le nom concatène le nom de chacun des membres du groupe**, au format `tar.gz`. Cette archive contiendra le rapport et votre code source. Si ce code est contenu dans des répertoires, vous devez intégrer ces répertoires dans le rendu.

Nous nous réservons le droit de convoquer certains groupes à un oral pour d'éventuelles précisions sur le travail rendu. Il est bien entendu qu'on attend un travail original (c'est à dire personnel, tout plagiat sera très fortement pénalisé), et équitablement réparti au sein de chaque groupe.

À noter enfin que les différentes consignes données dans les documents déjà distribués s'appliquent à ce projet.

5 Évaluation

L'échéance pour les dépôts des projets est fixée au vendredi 30 avril 2021 23h59, sur le cours UPdago "*Programmation des IHM, JavaFX*", section "*Projet IHM jeu d'exploration*".

Les critères pris en compte pour la notation :

— les différentes consignes sont respectées (structuration et format de l'archive rendue, encodage des caractères, etc...) ;
— le code est propre, modulaire, lisible, documenté... ;
— la séparation Modèle/Vue/Contrôleur est bien respectée ;
— un maximum de composants graphiques sont gérés au format `FXML` ;
— le code rendu compile sans erreur ni warning avec les options `-Xlint:all -Xdiags:verbose` ;
— l'application est ergonomique ;
— si vous avez repéré des erreurs dans votre application, sans avoir le temps de les corriger, elles sont documentées dans le rapport ;
— le rapport donne les informations nécessaires pour comprendre votre démarche et utiliser votre application.

1. La note finale tiendra compte du nombre de participants
2. Nom à personnaliser, bien sûr.

6 Annexe - Description du jeu en mode texte

6.1 Contexte général

Le but de ce projet est de programmer un petit jeu d'aventure en mode texte à la manière de Colossal Cave Adventure 1. Ce jeu, créé dans les années 70 est considéré comme le premier jeu d'aventure interactif où le programme décrit une situation (i.e. par un affichage texte) et où le joueur indique ce qu'il souhaite faire en saisissant du texte. Le programme analyse la phrase entrée par l'utilisateur et réagit à son tour en affichant la nouvelle situation, et ainsi de suite. Colossal Cave Adventure se déroule dans une grotte, mais vous pouvez créer l'univers de votre choix.

6.2 Éléments principaux

6.2.1 Des lieux

Vous devez modéliser votre terrain de jeu (i.e. la carte de votre aventure) par un ensemble de lieux où le joueur peut se trouver (par exemple, si le jeu se déroule dans l'espace, chaque lieu peut correspondre à une planète ; alors que s'il se passe dans un bâtiment, chaque lieu pourra correspondre à une pièce). Au cours de l'aventure, le joueur se déplace d'un lieu à l'autre en franchissant des sorties.

6.2.2 Des sorties

Chaque lieu contient un certain nombre de sorties permettant de se rendre dans les lieux voisins. C'est vous qui définissez le plan de votre terrain de jeu. Il peut bien évidemment exister différents types de sorties (des portes toujours ouvertes, des portes qui nécessitent une clé ou un code secret...). Ainsi, chaque sortie connaît son lieu voisin. Notez que l'on l'on pourra tenter de franchir n'importe quel type de sortie, mais on ne franchira pas de la même manière une porte ouverte et une porte nécessitant une clé. Depuis un lieu donné, il sera donc possible de se rendre à un lieu voisin à condition d'être en mesure de franchir la sortie correspondante. Vous devez donc trouver un moyen d'associer les noms des lieux voisins avec leur sortie correspondante. Pour se rendre dans un lieu voisin, il faudra donc franchir la sortie associée à l'aide de la commande go (cf. section Commandes). Notez que s'il existe une sortie dans le lieu A permettant d'accéder au lieu B, cela n'implique pas qu'il existe nécessairement une cortie dans le lieu B permettant d'accéder au lieu A.

6.2.3 Des objets et/ou des personnages dans les lieux

Afin de pimenter un peu votre jeu, chaque lieu pourra contenir des objets (des armes, de la nourriture, des coffres qui peuvent contenir à leur tour des objets...) et/ou des personnages avec lesquels le joueur pourra interagir (discuter, combattre, ...) ou pas.

6.2.4 Pas d'aventure sans héros

Le jeu consiste principalement à donner des instructions au héros de votre aventure, pensez donc à établir ses caractéristiques (e.g. points de vie, liste d'objets qu'il possède...) qui pourront/devront évoluer au cours du jeu. Le joueur devra interagir avec le programme via une console. Vous devrez donc mettre en place un système d'analyse du flux de l'entrée standard. Plus précisément, le programme devra analyser mot par mot chaque ligne de texte entrée par l'utilisateur. Pour cela, vous utiliserez un objet Scanner instancié à partir de l'entrée standard System.in. Le programme devra "comprendre" au minimum les commandes décrites dans la section Commandes, mais vous pouvez ajouter toutes les commandes nécessaires au bon déroulement de votre jeu.

6.2.5 Pas d'aventure sans objectif

Bien entendu, s'il n'y a pas d'objectif à atteindre, un jeu perd vite de son intérêt. Il vous faut donc établir un but, une quête, ce que vous voulez qui permette de dire que la partie est gagnée, ou perdue.

6.2.6 Commandes

Votre jeu doit permettre à l'utilisateur d'effectuer des actions en saisissant des phrases d'un ou plusieurs mots. Votre programme devra comprendre au minimum les commandes GO, HELP, LOOK, TAKE, QUIT et USE. Vous êtes bien entendu libre d'en ajouter autant que vous le souhaitez. Voici un rapide descriptif du fonctionnement des commandes de base dans la console :

- `go location` : la commande go est suivie du nom du lieu voisin où l'utilisateur souhaite se rendre. Si le lieu location existe et que la sortie est bien franchissable, alors le personnage s'y rend, sinon il ne change pas de lieu et un affichage indiquera à l'utilisateur ce qui se passe.
- `help` : indique l'ensemble des commandes disponibles
- `look [object]` : affiche un descriptif du lieu courant si aucun argument n'est ajouté. Si l'argument object est précisé et qu'il existe, la commande affiche un descriptif de ce dernier.
- `take object` : ajoute (si cela est possible) l'objet à la liste d'objets du joueur.
- `quit` : quitte la partie.
- `use object1 [object2]` : utilisation de l'objet désigné par le premier argument. Si un second argument est renseigné, le premier objet est utilisé avec le second. Par exemple, on peut imaginer que l'instruction use `gun bullet` permette de charger `gun`, ce qui pourra être utile pour une future utilisation.