Henintsoa Andrianarivony
Samuel Goubeau
Yannis Sauzeau

Licence 3 Informatique
Programmation des IHM
2020-2021

Projet *Interface d'un jeu d'exploration*

[Gypsy's carnival]

-

Rapport

Table des matières

I - Structure de l'application

Package **gypsysCarnival.model**

Ce package contient les sources de notre jeu d'exploration initial dont la structure est représentée ci-dessous :

- **character**
  - *Character.java*
  - *NPC.java*
  - *Player.java*
- **command**
  - *Command.java*
  - *Go.java*
  - *Help.java*
  - *Interpreter.java*
  - *Inventory.java*
  - *Look.java*
  - *Play.java*
  - *Quit.java*
  - *Take.java*
  - *Unlock.java*
  - *Use.java*
- **item**
  - *Food.java*
  - *Item.java*
  - *Key.java*
- **place**
  - **exit**
    - *Exit.java*
  - **game**
    - **copper**
      - *FindNumber.java*
      - *QTE.java*
      - *RockPaperScissors.java*
    - **gold**

        - *HanoiTower.java*
        - *Riddle.java*
        - *TicTacToe.java*
    - **platinum**
        - *Hangman.java*
        - *Karaoke.java*
        - *Questions.java*
  - *Ending.java*
  - *Game.java*
  - *Place.java*
  - *Shop.java*
- *Describable.java*
- *Gameplay.java*
- *Level.java*

Le sous-package **character** est lié aux personnages du jeu qu'ils soient joueurs ou non.

Le sous-package **command** s'occupe d'interpréter chaque commande disponible pour faire avancer le jeu.

Le sous-package **item** gère les objets avec lesquels le joueur peut interagir.

Le sous-package **place** permet de créer les différents lieux du jeu. Dans ce sous-package, il y a aussi le sous-package **exit** qui gère les connexions entre les lieux et le sous-package **game.<level>** qui contient les mini-jeux de niveau **<level>**.

Le fichier *Gameplay.java* est la classe exécutable du jeu d'origine.

Package **gypsysCarnival.view**

Ce package contient des images, une musique, des fichiers FXML et des classes dérivées des composants JavaFX. Ci-dessous la structure de ce package :

- **design**
  - **image**
    - *apple_candy.png*
    - *...*
    - *volume_on.png*
  - **sound**
    - *theme.mp3*
- **place**
  - **game**
    - **copper**
      - *findNumber.fxml*
      - *qte.fxml*
      - *rockPaperScissors.fxml*
    - **gold**
      - *hanoiTower.fxml*
      - *riddle.fxml*
      - *ticTacToe.fxml*
    - **platinum**
      - *hangman.fxml*
      - *karaoke.fxml*
      - *questions.fxml*
  - **hub**
    - *copperHub.fxml*
    - *goldHub.fxml*
    - *platinumHub.fxml*
  - **shop**
    - *foodShop.fxml*
    - *keyShop.fxml*
  - *carnival.fxml*
  - *ending.fxml*
- *ClickableImage.java*
- *CustomAlert.java*
- *CustomButton.java*
- *help.fxml*
- *main.fxml*
- *map.fxml*
- *playerInfos.fxml*
- *start.fxml*

Dans le dossier **design,** se trouvent toutes les images au format png ou gif pour les animations. Ces images ont été trouvées sur les sites flaticon.com ou giphy.com ou ont été créées à l'aide du site piskelapp.com. La musique provient de la vidéo suivante : www.youtube.com/watch?v=7l_x9R6yAxQ.

Dans le dossier **place** se trouvent les différentes vues propres à chaque lieu du jeu.

À la racine du package se trouvent trois fichiers Java qui sont les composants personnalisés JavaFX utiles à notre jeu. Il y a aussi les trois scènes du jeu qui sont : la scène du début avant de commencer le jeu, la scène d'aide accessible à tout moment une fois dans le jeu et la scène principale divisée en trois fichiers FXML : un pour la scène de jeu, un autre pour la carte et le dernier pour les informations du joueur.

Package gypsysCarnival.controller

Ce package contient les sources principales du jeu qui sont les contrôleurs qui assurent la communication entre le modèle et la vue de l'application. Son arborescence, qui suit les packages du dessus, est visible ci-dessous :

- **place**
  - **game**
    - **copper**
      - *FindNumberController.java*
      - *QTEController.java*
      - *RockPaperScissorsController.java*
    - **gold**
      - *HanoiTowerController.java*
      - *RiddleController.java*
      - *TicTacToeController.java*
    - **platinum**
      - *HangmanController.java*
      - *KaraokeController.java*
      - *QuestionsController.java*
  - **hub**
    - *CopperHubController.java*

        - *GoldHubController.java*
        - *PlatinumHubController.java*
    - **shop**
        - *FoodShopController.java*
        - *KeyShopController.java*
    - *CarnivalController.java*
    - *EndingController.java*
- *HelpController.java*
- *MainController.java*
- *MapController.java*
- *PlaceController.java*
- *PlayerInfoController.java*
- *SceneController.java*
- *StartController.java*

Tous les fichiers sont des contrôleurs associés à un fichier FXML à part les fichiers *SceneController.java* et *PlaceController.java* qui gère respectivement les changements de scène et les changements entre les lieux.

II - Résolution des problèmes rencontrés

Les problèmes

Créer une interface graphique pour le jeu que nous avons développé en POO nous a posé certains problèmes de conception.

Nous voulions modifier le moins possible le noyau fonctionnel original, c'est-à-dire le code du package **controller**. Nous nous sommes aussi donné comme contrainte que le jeu puisse toujours fonctionner en ligne de commande, exactement comme le jeu initial.

Un premier problème était que le modèle n'avait pas besoin de contrôleur pour fonctionner, car le jeu initial compilait seulement avec le contrôleur. Un second problème était les affichages sur la sortie standard qui étaient présentes sur quasiment toutes nos fonctions dont certaines renvoyaient autre chose que **void** ce qui était compliqué pour renvoyer le **string** qui sera affiché à l'écran après la fonction appelée.

Le dernier problème se trouvait en phase principale du jeu, soit sur les mini-jeux que le joueur doit gagner pour finir le jeu. Le problème est que chaque classe représentant un mini-jeu avait une méthode play qui se lançait quand le joueur tapait la commande *play*. Et tout l'essence du mini-jeu se déroulait dans cette fonction *play* qui lisait sur l'entrée standard.

Les solutions

Pour le premier problème du modèle qui ne doit pas connaître les contrôleurs, nous avons transformé les attributs de classes qui devaient être visibles pour l'interface en **Property** JavaFX. Cette solution est idéale puisque le modèle peut continuer de fonctionner seul sans interface graphique ni contrôleurs. De plus, les contrôleurs pouvaient facilement accéder à ces attributs et leurs modifications grâce au **Bindings** de JavaFX.

Le second problème des affichages sur la sortie standard a été résolu par le contrôleur principal *MainController.java* qui redirige la sortie standard vers un **Label** dans un **ScrollPane.**

Concernant le dernier problème des mini-jeux, il a été le plus complexe à résoudre puisque nous avons dû extraire du code des fonctions *play* en plusieurs fonctions auxiliaires **public** pour être accessible au contrôleur.

III - Comment étendre l'application

Bien que nous soyons contents du rendu final de l'application et du code que nous avons rendu, il nous a manqué du temps pour arriver à l'application rêvée.

Le premier point améliorable est l'interface graphique qui est très sobre et qui manque de fond dynamique pour plus facilement rentrer dans le jeu. Au niveau de l'interaction avec l'utilisateur, nous avons tout fait pour que l'interface soit la plus intuitive possible, et après avoir fait tester le jeu à quelques personnes les retours sont plutôt positifs.

La partie que nous voulons améliorer le plus, mais qui nous a fait défaut par manque de temps est la modularité du code. En effet, nous avons décidé de faire une vue pour chaque lieu du jeu alors que nous aurions très bien pu faire une vue unique pour les lieux de la classe **Shop** et ceux de la classe **Hub** (impossible cependant pour les mini-jeux qui sont trop différents).
Nous voulions aussi plus distinctement séparer les vues et les contrôleurs en mettant les **Controller** JavaFX dans le package **view** et faire en sorte qu'il n'y ait que le code nécessaire à la vue dans ces contrôleurs. Nous aurions donc créé nos propres contrôleur dans le package **controller** qui aurait appelé des méthodes de la vue par le biais des **Controller** JavaFX du package **view.**