Rapport de projet
Jeu de société

Médéric HURIER
mederic.hurier@etu.univ-lorraine.fr

6 mai 2013

Année 2012-2013

1 Présentation du sujet

Testez vos compétences en programmation lors d'une série d'histoires et de défis !

Vous incarnez un commandant de vaisseau spatial du nom de Capitaine Duke, et vous êtes en charge d'une périlleuse mission qui vous emmènera au confins du système solaire. Pour mener à bien votre objectif vous devrez répondre aux attentes de votre équipage grâce aux outils informatiques dont vous disposez. Les demandes peuvent vous semblez farfelues, mais du bien être de l'équipage dépend la réussite de la mission.

Votre adresse ne sera pas mesurée uniquement par votre rapidité, mais aussi par votre précision et votre assurance. Vous devez faire le moins d'essais possibles pour arriver à une réponse acceptable, et vos performances seront mesurées pour que vous puissiez vous en vanter.

2 Utilisation

2.1 Lancement du jeu

Le jeu se lance en ligne de commande à partir du dossier d'archive :
- dans un terminal, placer vous dans le répertoire "dist"
- taper la commande suivante pour lancer le programme :
  - java -jar CaptainDuke.jar

Attention ! la musique démarre une seconde après l'affichage de la fenêtre. Pour désactiver la musique, ajouter le drapeau "-nomusic" :
- java -jar CaptainDuke.jar -nomusic

Il n'y a pas de paramètres supplémentaires.

En cas de problème, contacter moi à l'adresse suivante : mederic.hurier@freaxmind.pro.

(option) Compiler vous même le programme

- dans un terminal, placer vous dans le répertoire "src"
- compiler la classe principale en tapant :
  - javac CaptainDuke.java
- copier le répertoire "DATA" de l'archive dans le dossier "src"
- lancer le programme en ajoutant les bibliothèques nécessaires au classpath lors de l'exécution :
  - java -cp "../lib/sqlite-jdbc-3.7.2.jar :." CaptainDuke

(option) Lancement du serveur de scores

Le serveur est écrit en Python et nécessite la bibliothèque "bottle" pour fonctionner.

Vous pouvez vous référer à cette page pour obtenir la procédure d'installation :
http ://bottlepy.org/docs/dev/tutorial.html#installation

Il suffit alors de lancer le programme en tapant dans le dossier "server" :
– python scoreboard.py
– consulter la page "http ://127.0.0.1 :9000" pour voir si le serveur est bien lancé.

Le serveur n'est PAS nécessaire au fonctionnement du jeu, mais vous ne pourrez pas récupérer ou poster des scores.

2.2 Déroulement d'une partie

Au lancement du programme, la fenêtre suivante apparaît :

FIGURE 1 – Fenêtre de lancement

De cette fenêtre, vous pouvez lancer une nouvelle partie une ou continuer la dernière partie jouée. Vous pouvez également afficher les meilleurs scores (si le serveur est lancé), et consulter la page d'aide.

La difficulté change le nombre d'essais de départ et les bonus en fin de mission :
– facile = 20 essais au départ mais aucun bonus
– normal = 10 essais au départ, bonus de 20 essais en fin de scénario
– difficile = 5 essais au départ, bonus de 50 essais en fin de scénario

Le jeu est difficile ! Je vous invite à commencer avec la difficulté "FACILE".

Une fois la partie chargée, la fenêtre suivante remplace le lanceur :

FIGURE 2 – Fenêtre de jeu

Les éléments sont :

1. Zone de texte éditable ou vous devrez coder la (les) fonction(s) permettant de passer les tests
2. Test unitaire servant à vérifier votre fonction
3. Scénario nécessitant de créer une fonction
4. Boutons pour exécuter votre code :
   (a) "Faire un essai" diminue votre nombre d'essai de 1
   (b) "J'ai de la chance" diminue votre nombre d'essai de 5, mais double votre score final
5. Indicateurs de performances :
   (a) le premier nombre est votre nombre d'essais restants
   (b) le deuxième nombre est un chronomètre en seconde
6. Boutons pour sauvegarder ou quitter la partie

Le chronomètre commence quand vous cliquez sur le bouton "Commencer" (qui prend la place du bouton "Sauvegarder" en attendant votre signal). A chaque essai, une boîte de dialogue apparaît avec une erreur ou un message de félicitation si vous avez réussi le scénario.

Vous pouvez afficher des variables en utilisant "System.out.println" comme vous le feriez en Java. Le contenu de sortie est alors ajouté à la boîte de dialogue.

Le jeu comporte 6 scénarios, mais il est possible d'en rajouter en éditant la base de données contenue dans "DATA/scenario.sqlite". Une fois le jeu terminé, une invite vous propose de poster votre score. Ils sont ensuite consultable sur le site ou via l'interface de jeu depuis le lanceur.

3 Modélisation

3.1 Couche modèle

La couche modèle comporte les données de l'application : une partie, des scénarios, des récompenses et des scores. Toutes ces données sont sérialisables pour être sauvegardé.

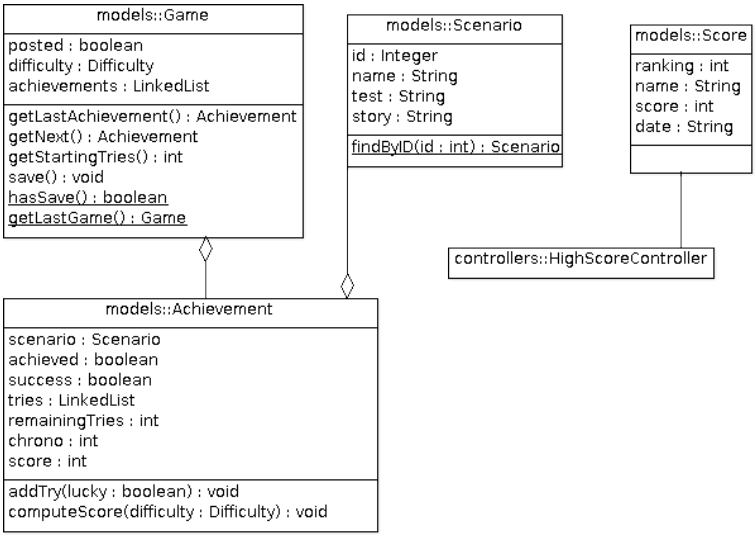


FIGURE 3 – Diagramme de classe de la couche modèle

Scenario est la description d'une mission avec son histoire et le test associé. Les scénarios sont stockées dans une base de données SQLite avec le composant JDBC et ont peut les récupérer en utilisant la méthode findById.

Achievement est une récompense obtenue ou en cours d'acquisition pour un scénario. La classe contient des attributs pour qualifier si la récompense est un succès ou non, si elle est terminée, le chronomètre actuel et les essais qui ont été fait. Une fois la récompense obtenue, on calcule le score que le joueur a obtenu avec la fonction computeScore.

Game est une partie du joueur, elles contient les récompenses. C'est cet objet qui est sauvegardé, et qui provoque l'enregistrement de tous les attributs grâce aux classes ObjectInputStream et ObjectOutputStream.

Enfin, la classe **Score** contient les informations du tableau des scores mais n'est pas sérialisable.

3.2 Couche composant

La couche composant regroupe des classes utilitaires et des données non persistantes de l'application.

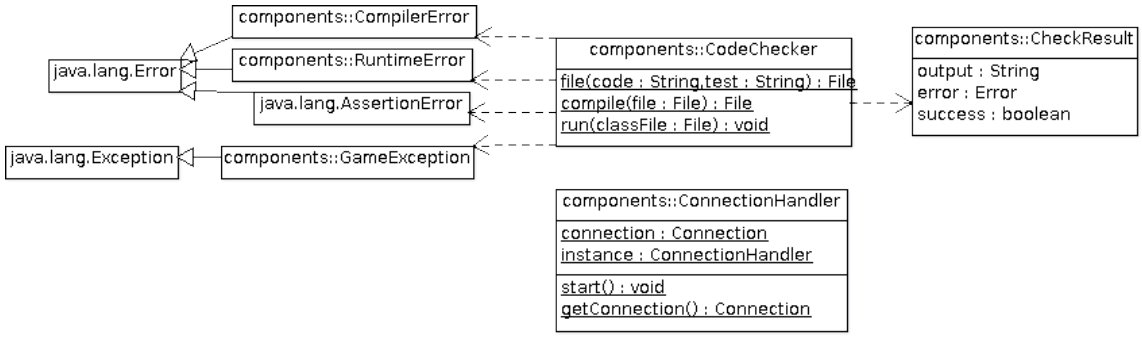


FIGURE 4 – Diagramme de classe de la couche composant

Le coeur du jeu est la classe **CodeChecker** qui permet de vérifier le code de l'utilisateur grâce à la méthode **check** en revoyant une instance de **CheckResult**. Si l'utilisateur a fait une erreur, l'objet contient une erreur (compilation, exécution ou assertion) qui sera affichée à l'utilisateur. On ajoute également le contenu de la sortie du programme.
Si le programme rencontre un cas qu'il ne sait pas gérer, ou si un composant est défaillant, la classe envoie une exception de type GameException qui provoque l'arrêt immédiat de l'application.

En plus de cette classe, le ConnectionHandler est un singleton permet de stocker une instance unique de connection à la base de données SQLite.

3.3 Couche contrôle

La couche contrôle regroupe les règles du jeu et l'enchaînement des actions.

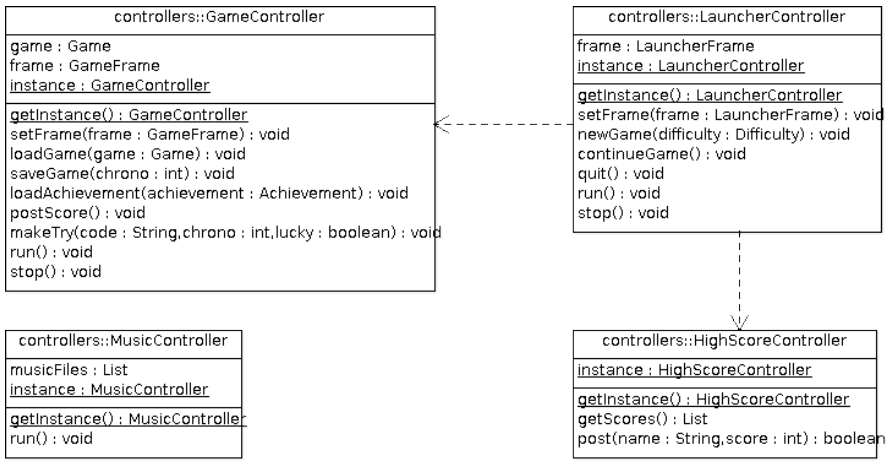


FIGURE 5 – Diagramme de classe de la couche contrôle

Le premier contrôleur appelé est **LauncherController** qui gère la première fenêtre du programme.

Il peut créer une nouvelle partie ou relancer la dernière jouée en passant ensuite le relai à la classe GameController. Il gère une interface graphique LauncherFrame (voir couche interface), qui est affichée lors de l'appel à la méthode run et stoppé à l'appel de la méthode stop.

Le contrôleur principal est **GameController**, qui permet d'effectuer toutes les actions à partir de l'interface de jeu : charger/sauvegarder une partie, charger un scénario, faire un essai ou poster un score.

Les deux autres contrôleurs **MusicController** et **HighScoreController** contrôlent respectivement l'enchaînement des musiques jouées et la récupération et l'envoie de score au serveur de score.

3.4 Couche interface

La couche interface est responsable de l'affichage des composants et de l'interception des événements. Note : les attributs SWING sont masqués pour la clarté du diagramme.

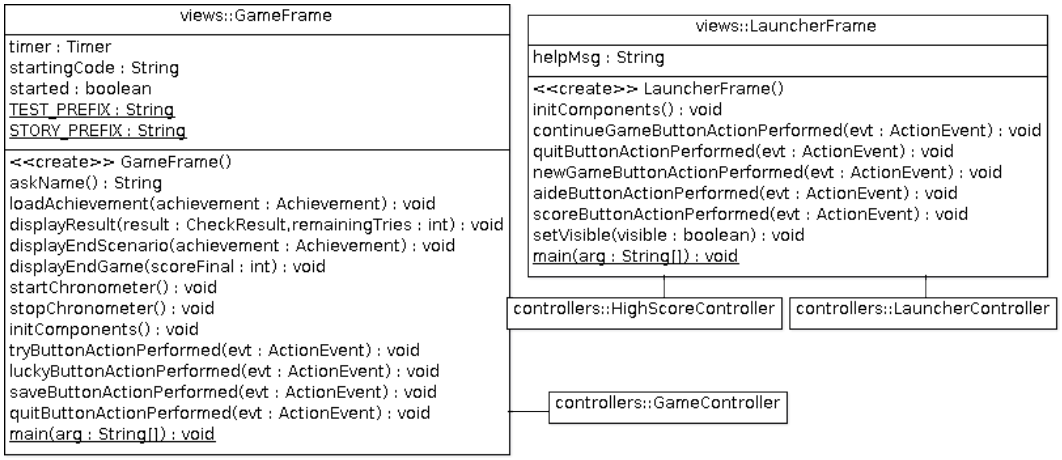


FIGURE 6 – Diagramme de classe de la couche interface

Le jeu comporte deux fenêtres, la première est **LauncherFrame** qui comportent les événements liés aux boutons de la première fenêtre. La seconde est **GameController**, qui s'occupe de la fenêtre de jeu. Cette fenêtre dispose également de méthodes pour afficher les résultats (fin de jeu, fin de scénario, résultat d'un test).

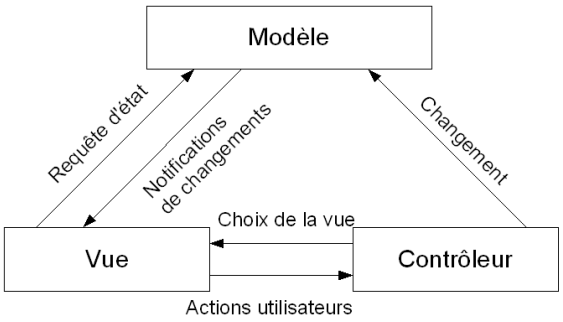
4 Implémentation

4.1 Choix

Architecture

Le projet est découpé en suivant l'architecture 3-tiers MVC (Model-View-Controller) pour répartir logiquement le rôle de chaque classe. Il n'y a pas d'interfaces permettant de diminuer les dépendances entre couches, mais ces techniques ne sont pas requises pour un projet de ce type.

Les dépendances avec les autres bibliothèques Java ont été maîtrisées. A de rares exceptions, chaque bibliothèque ne dépend que d'une seule couche (modèle pour java.io, composants pour java.lang.reflect ...) afin de limiter les contextes d'appel.



Persistance des données

Les données sont sauvegardées de deux manières :
- la partie du joueur est sauvegardée via sérialization en utilisant les classes ObjectInputStream pour la lecture et ObjectOutputStream pour l'écriture
- les scénarios sont stockées dans une base de données au format SQLite grâce à JDBC

Dans le cas des scénarios, le but était de rendre la base des missions extensible sans avoir besoins des classes du programme pour les manipuler. Grâce à SQL et JDBC, ces données sont indépendantes du programme. Il suffit d'utiliser un outil comme SQLitebrowser pour ajouter de nouvelles histoires, qui seront proposées au lancement de la partie. On peut également envisager un éditeur de scénario, comme cela se fait dans les jeux vidéos.

Pour la sauvegarde de la partie, l'indépendance n'est pas requise et est au contraire indésirable. Même si l'utilisateur peut facilement changer les données de sérialisation, cela n'est pas évident pour un novice en informatique. De plus, la sérialisation est très facile à implémenter et ne informatique pas d'association entre les objets et leur stockage.

Les deux solutions sont simples et faciles à maintenir. Elles répondent très bien au problème de ces deux types d'information.

Test du code de l'utilisateur

Tester un code dynamiquement n'est pas naturel pour un langage statique comme Java. C'est pourtant une fonctionnalité essentielle pour l'application.

Afin de réaliser ce module, j'ai utilisé les outils de la bibliothèque java.lang.reflect et javax.tools.JavaCompiler. Ces deux outils manipulent le compilateur Java et la structure des classes pour créer un fichier temporaire, le compiler, l'importer, instancier la classe qu'il comporte et enfin exécuter les tests. Les deux conditions sont que le nom du fichier soit unique et que l'utilisateur dispose d'un JDK installé.

Une autre solution de secours serait l'utilisation de scripts Bash qui remplissent le même rôle. Toutefois, la solution entièrement en Java a été préférée car elle ne demande pas de manipuler du texte, mais des composants du paradigme objet.

4.2 API Java

Au cours du projet, ces élements de l'API Java ont été utilisés :
- java.util.logging dans les contrôleurs pour afficher les exceptions et l'avancement
  - classe principale, tous les contrôleurs
- **java.io.Serializable** pour les modèles dont la classe Game est le point d'entrée
  - tous les modèles sauf Score
- **java.io.ObjectOutputStream et java.io.ObjectInputStream** pour sauvegarder une partie
  - classe models.Game
- **java.sql** (dont JDBC-SQLite) pour sauvegarder les scénarios et en ajouter facilement
  - classe models.Game
- java.net pour récupérer et poster les scores via HTTP
  - classe controllers.HighScoreController
- **javax.tools et java.lang.reflect** pour tester le code de l'utilisateur
  - classe components.CodeChecker
- javax.sound pour jouer de la musique de manière asynchrone
  - classe controllers.MusicController
- **javax.swing** pour l'interface graphique
  - dossier views
- **java.util.timer** pour gérer le chronomètre de manière asynchrone
  - classe views.GameFrame
- Math.log pour le calcul des scores en fonction du temps
  - classe models.Achievement
- org.junit pour testers les composants et les scénarios
  - dossier test
- java.util.LinkedList et java.util.ArrayList pour les collections

4.3 Packages

L'application est découpée logiquement et physiquement selon les couches de la partie modélisation :

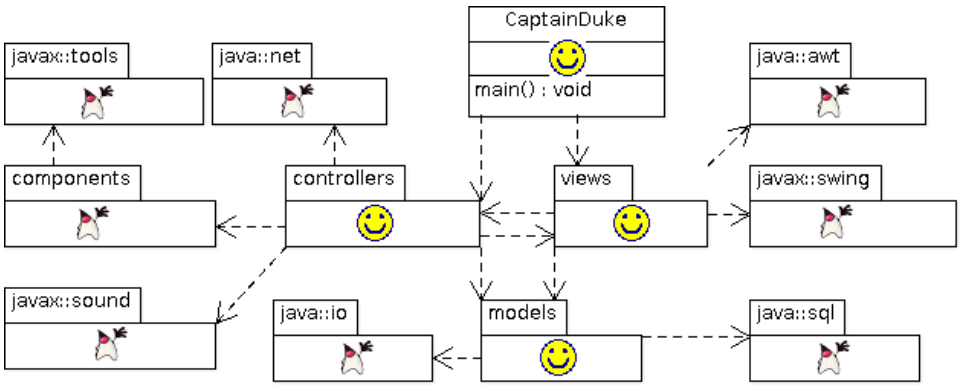


FIGURE 7 – Graphique des paquets utilisés

Les paquets du projet sont identifiés par un smiley. Les associations de dépendances montre les principaux paquets Java utilisés. Par exemple, le paquet "views" utilise les paquets "java.swing" et "java.awt" qui permettent de coder des interfaces graphiques en Java.

Le programme s'organise donc autour d'une architecture MVC très simple, où chaque paquet est responsable d'un composant du projet.

5 Conclusion

5.1 Bilan

J'ai appris le Java en 2e année de BTS, et j'en suis à ma 5e année d'étude avec Java. Je me suis aussi beaucoup intéressé aux langages orientés objets comme Python, Ruby ou Groovy.

Mon bilan personnel pour ce projet est plutôt bon : le jeu fonctionne et comporte de nombreux composants de l'API Java sans trop inclure de dépendances avec des bibliothèques externes. C'est donc un bon tour d'horizon de ce qu'offre naturellement Java (interface graphique, méta-programmation, multi-tâche, persistance, multimédia ...). J'ai beaucoup aimé coder la classe vérifiant le code de l'utilisateur (components.CodeChecker), qui m'a permis d'explorer des fonctions avancées de Java.

Je vois cependant deux points négatifs à mon jeu :
- Pour réellement être un jeu de société, il aurait fallu qu'il soit disponible en ligne pour que le code puisse être revu, commenté et noté. En se jouant seul, il perd sa dimension sociale.
- L'interface graphique n'est pas aussi jolie que ce que j'aurai souhaité. Ceci est dû à une difficulté que j'ai rencontré et que je détaille dans la partie suivante.

5.2 Difficultés rencontrées

Je prend toujours le pari d'utiliser des technologies innovantes dans mes projets afin de m'investir davantage et de capitaliser pour mon avenir. Je m'en suis toujours bien sortie, mais j'ai eu une déception avec ce projet.
En voulant utiliser l'API JavaFX pour la partie interface graphique, je me suis heurté à un gros manque de maturité et des lenteurs sur ma plateforme privilégiée (Linux). Les fonctions étaient pourtant intéressantes (style au format CSS, API plus cadrante), mais j'ai du faire machine arrière tardivement pour revenir à une solution plus fiable.

Suite à ce problème, je n'ai pas rencontré de difficultés majeures. Je comprend maintenant l'intérêt de Java pour une API de jeu comme sous Android : c'est un langage structurant avec lequel il est difficile d'avoir des erreurs. A mon goût, ce n'est cependant pas l'environnement le plus créatif. Je pense que les plateformes Web et les bibliothèques de développement rapide comme PyGame restent des technologies plus appropriées pour répondre au sujet.