Mount'n Fall

Werner BEROUX
werner@beroux.com

14 mai 2005

1 Introduction

On veut réaliser un jeu des tours en Java2. Les règles sont définies clairement dès le départ. Les éléments nécessaires, et les éléments rajoutés en plus, ont été décrit dans les cas d'utilisation UML. Un papier, un crayon, et il restait à réfléchir sur le comment présenter le jeu avant de voir comment le programmer.

Le projet avait été planifié pour deux personnes. La 2e personne ayant voulu faire le projet à part pour mieux comprendre, je n'ai pas réalisé tout ce qui avait été planifié. Comme j'ai réalisé tout ce qui est dans ce projet (planification comprise), je ne ferai plus référence à ce détail.

Nous verrons donc ce qui a été planifié et comment cela l'a été. Ensuite le design objet et l'implémentation par phase. Et enfin nous verrons les tests effectués, les correctifs, rajouts...

2 Planification

J'avais déjà réalisé un jeu assez similaire : **iPuissance 4D**. Ce jeu est disponible sur mon site à www.beroux.com/ ?id=2. J'avais donc des bases et j'ai principalement réfléchi aux différences avec ce jeu.

La planification à commencé sur papier. Des petits **croquis** pour les éléments graphiques et les idées et des **diagrammes** au crayon pour pouvoir les modifier facilement. Les feuilles papier du projet sont en annexe. J'ai choisi le style du jeu à ce moment la. Le jeu étant assez simple, j'ai voulu l'adresser à une population jeune. J'ai donc choisis des éléments très colorés, tons pastels, beaucoups d'animations et peu de texte. Les gouts changent et je n'ai pas prévu ce projet pour les correcteurs mais pour les joueurs. Il me semble que c'est eux qui priment pour la réalisation d'un jeu.

Ensuite avant de coder j'ai complété l'UML sur l'ordinateur. J'avais encore prévu le faire avec quelqu'un à cette époque, vous trouverez donc des classes tel que **Tournament** qui n'ont pas été réalisées. Ca pourra être une amélioration future possible. A partir de la, j'avais tout pour commencer à coder.

3 Construction

Je n'avais pas fait beaucoup de Java donc j'ai commencé par tester les éléments critiques liée au JDK. C'est à dire **MainFrame** et tout **beroux.game**. Le packaque **beroux.game** est en fait un couche Middleware pour les jeux. Elle permet d'adapter plus facilement le jeu sur une autre plateforme (par exemple les téléphones portables) et elle simplifie la réalisation de jeux en fournissant des outils génériques. J'ai rajouté au cours du projet des éléments dans le package. Pour tester j'ai aussi du réaliser les bases de la classe **InGame** qui en fait le controlleur de la partie.

Une fois que j'avais les bases, il me restait à compléter **InGame** par le plateau (**Board** avec ses sous-classes), les joueurs (**Player**, **HumanPlayer** et **AiPlayer**), la logique de jeu (**GameLogic**) et d'autres éléments graphiques (**GUI**).

A la racine les évènements et le requètes sont passées par MainFrame vers le GameCanvas qui est en fait la classe abstraite de base d'un jeu. Le **SceneController** est la classe noo-abstraite de base pour **Mount'n Fall**. Elle reçois donc les évènements souris, des requête de rendu (**render()**) et mise à jour au cours du temps (**update(dt)**). Ici chaque **Scene** est en fait une boite de dialogue animée. Le **SceneController** est une machine à état qui choisis quelle scène afficher à quel moment et qui initilalise chacune d'elle si nécessaire.

Durant la partie c'est la scène **InGame** qui est affichée. Sa struture globale est en MVC (clairement visible sur l'UML).

Une fois une partie pour 2 joueurs fonctionnelle, je l'ai testé. Ensuite j'ai réalisé l'A.I. qui utilisait au début un MinMax. C'est un moyen efficace de vérifier les bugs. MinMax étant très gourmant j'ai chercher à accélérer les calculs des points critiques - éléments que j'ai identifié à l'aide d'un timer précis non fourni dans le JDK. Par exemple le clone de la grille était très long. J'ai donc découvert une utilité des Immutable : Pouvoir copier uniquement la référence suffit pour garder cloner les cases du plateau. Je passe les détails que vous pourrez trouver dans **AI Benchmarks.log**. Bien plus tard j'ai utilisé AlphaBeta et j'ai inventé le **RecallAlphaBeta** qui est détaillé dans le AiPlayer et dont je vous parlerai volontier.

Il ne restait plus qu'à réaliser les autres scènes et tester.

4 Au final

Je ne saurait dire le temps total j'y ai consacré. J'ai réalisé les images sur PhotoShop. Les animations sur Flash (beaucoup d'animation ne sont pas visibles dans le jeu car je n'ai pas eu le temps de les incorporer). Les diagrammes avec Visual Paradigm (très bon outil que je conseil, avec une licence pour les écoles possible). Le code en lui même je l'ai fait sous VIM et j'ai compilé en ligne de commande. J'ai pour cela écrit un petit **make.bat** qui permet de compiler facilement.

Il y a un peu plus de 5000 lignes de code réparties dans 37 classes. Pour une métrique plus détaillée voir doc/.cccc/cccc.html dans le projet (il n'est tout à fait à jour mais très proche du final).

La structure globale des fichiers est détaillée dans ReadMe.txt. Je n'ai pas incorporé les fichiers PDF (PhotoShop) car ils sont trop gros ainsi que quelques autres éléments trop volumineux.

5 Annexe

Les pages annexes seront inclus dans le rapport de la version papier. Pour la version électronique : les diagrammes UML sont dans doc/VP-UML Report/index.html et les croquis dans doc/Sketches.

6 Références

- Space Invaders 101 : An Accelerated Java 2D Tutorial
  http ://www.cokeandcode.com/info/tut2d.html
- Java2D : An Introduction Tutorial
  http ://www.apl.jhu.edu/ hall/java/Java2D-Tutorial.html
- Java2D Transformations
  http ://www.glyphic.com/transform/applet/1intro.html
- Java2D : Text
  http ://www.glyphic.com/transform/applet/7wiggly.html
- Java2D Samples
  http ://www.vorlesungen.uos.de/informatik/javaapi/Java2D/README.html
- Better compiler : jikes
  Opfuscator : JODE

– Java Code Optimization
http ://www.protomatter.com/nate/java-optimization/