Raphael DORÉ
Samuel THÉRIAULT-HALL
Gsroupe 0001

Dossier de programmation

Benjamin Lemelin
Programmation objet
420-310-SF

Département d'informatique
Techniques de l'informatique
Cégep de Sainte-Foy
11 novembre 2014

Table des matières

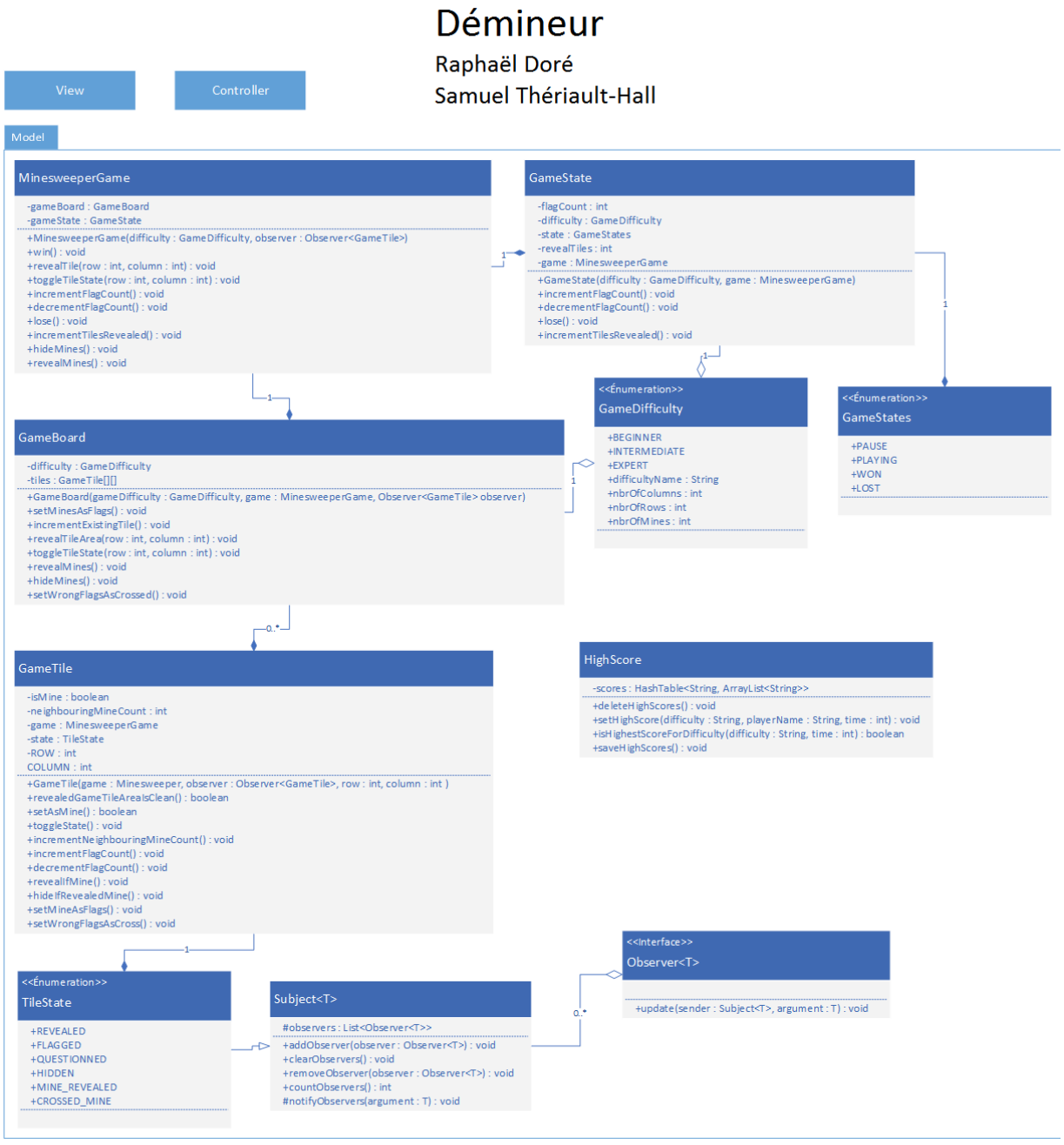
Contents

Complétion des « user stories »

#	« User story »	%
1	En tant qu'utilisateur, je désire pouvoir commencer une nouvelle partie de « Démineur » dans le mode de difficulté sélectionné. Pour plus d'efficacité, le raccourci « F2 » doit être assigné à cette fonction. Le bouton sourire doit lui aussi démarrer une nouvelle partie lorsqu'on clique sur ce dernier.	100
2	En tant qu'utilisateur, si je clique avec le bouton gauche de la souris sur une case non découverte du jeu, celle-ci doit se découvrir. Si la case possède des mines en guise de voisin, elle doit en afficher le nombre. Par contre, dans le cas contraire, elle doit aller découvrir les cases voisines. Notons que cette opération s'effectue en cascade en suivant la précédente règle.	100
3	En tant qu'utilisateur, je désire savoir le nombre de mines restant sur le champ de mines via un champ à gauche du bouton sourire.	100
4	En tant qu'utilisateur, je désire connaître le temps écoulé, en secondes, depuis la découverte de la première case, via un champ placé à droite du bouton sourire.	
5	En tant qu'utilisateur, si je perds le focus sur la fenêtre du jeu de démineur (par exemple pour consulter une page web dans mon navigateur), le décompte du temps écoulé doit s'arrêter.	100
6	En tant qu'utilisateur, si je clique avec le bouton droit de la souris sur une case non découverte du jeu, celle-ci doit alterner entre les états suivants : Vide – Marquée d'un drapeau – Marquée d'un point d'interrogation.	100
7	En tant qu'utilisateur, lorsque je marque une case d'un drapeau, le nombre de mines doit décrémenter de 1. Il est bien entendu possible d'avoir plus de drapeaux que de mines sur la grille, ce qui veut dire que le compteur peut très bien aller négatif.	100
8	En tant qu'utilisateur, je dois être incapable de découvrir une case marquée.	100
9	En tant qu'utilisateur, je dois perdre la partie si je clique sur une case minée.	100
10	En tant qu'utilisateur, lorsque j'ai perdu, je désire savoir où se trouvait les mines.	100
11	En tant qu'utilisateur, lorsque j'ai perdu, je désire savoir quels sont les emplacements marqués de drapeaux où il n'y avait pas de mine.	100
12	En tant qu'utilisateur, je dois gagner la partie s'il ne reste à découvrir que des cases avec des mines.	100
13	En tant qu'utilisateur, lorsque je gagne, je désire que les cases non marquées d'un drapeau où il y a une mine soit marquées d'un drapeau.	100
14	En tant qu'utilisateur, lorsque je gagne, si je bats le meilleur score pour le niveau de difficulté courant, l'application doit me permettre d'enregistrer mon nom pour le placer dans le tableau des meilleurs scores.	100
15	En tant qu'utilisateur, lorsque j'ai gagné ou perdu, je ne peux plus cliquer sur aucune case. Je n'ai pas d'autres choix que de commencer une nouvelle partie.	
16	En tant qu'utilisateur, en cours de partie, je veux que le bouton sourire affiche l'état « Normal ».	100

17	En tant qu'utilisateur, lorsque je perds, je veux que le bouton sourire affiche l'état « Mort ».	100
18	En tant qu'utilisateur, lorsque je gagne, je veux que le bouton sourire affiche l'état « Lunette de soleil ».	100
19	En tant qu'utilisateur, je désire pouvoir choisir je désire pouvoir choisir entre 3 niveaux de difficulté, à savoir « Débutant » avec une grille de 9x9 de 10 mines, « Intermédiaire » avec une grille de 16x16 de 40 mines et « Expert » avec une grille de 30x16 de 99 mines.	100
20	En tant que professeur, pour accélérer la correction, et pour accélérer vos tests, je désire pouvoir afficher la position des mines dans le champ de mines sans même avoir eu à découvrir des cases. Cette option devrait être activable via le menu « Partie ».	100
21	En tant qu'utilisateur, je désire pouvoir consulter les meilleurs scores pour chaque niveau de difficulté. Si je ferme l'application, et que je l'ouvre à nouveau, les scores doivent être préservés.	100
22	En tant qu'utilisateur, je désire pouvoir effacer les meilleurs scores.	100
23	En tant qu'utilisateur, je désire pouvoir consulter une fenêtre « À propos » donnant des informations à propos de l'application, à savoir le nom de cette dernière et le nom des programmeurs.	100
24	En tant qu'utilisateur, je désire pouvoir consulter une fenêtre « Aide » m'indiquant comment jouer au démineur.	100

Diagramme UML



Architecture de l'application

Notre application est conçue selon l'architecture MVC et le patron « observer ». Il y a donc des classes contrôleurs pour chacune des différentes fenêtres qui peuvent être ouvertes par l'utilisateur (Contrôleur), des fichiers fxml détaillant le positionnement de chacun des éléments de l'interface (Vue) et finalement, des classes qui gère une partie de « Démineur » (Modèle). Cette dernière est la plus intéressante et mérite le plus d'explication sur son fonctionnement.

Le modèle

Lorsqu'une partie commence, un objet MinesweeperGame est créé. Celui-ci sert principalement à communiquer entre différentes classes du modèle et à relayer les demandes du contrôleur. Il comporte une instance d'un GameState qui lui garde en mémoire la difficulté, le nombre de drapeaux que l'utilisateur a placé, le nombre de tuiles qui ont été révélées ainsi qu'un attribut concernant l'« état » du jeu (perdu, gagné, en cours). Il s'assure que lorsqu'assez de cases ont été révélées, le joueur gagne.

Le MinesweeperGame a aussi un objet du type GameBoard qui représente l'ensemble des cases du jeu. Il s'occupe de la création des cases, détermine au hasard lesquelles qui ont des mines et s'assure que les cases environnantes en soient averties. Il s'occupe de révéler les mines si le joueur décide de tricher ou s'il perd et met des drapeaux sur les mines en parcourant le tableau de cases. Finalement, il gère en partie la révélation des mines dans le sens où c'est lui qui dira aux autres mines de se découvrir elles aussi si jamais la première mine n'est pas une mine et n'a pas de voisin qui est une mine. Cette fonction utilise la récursivité et s'appelle en boucle jusqu'à ce qu'il n'y aille plus lieu de révéler d'autres cases (cela inclut d'avoir atteint le bord du tableau).

Les cases en tant que telles, ou GameTile, ont non seulement une fonction de révélation (qui provoque une défaite si la case est minée) qui dit au GameBoard s'il faut révéler d'autres cases comme dit plus haut, elles ont aussi la possibilité de changer d'état (drapeau, « ? », cachée), de se montrer comme mine et de se mettre un drapeau si elles sont des mines. Chaque fois qu'elle passe de son état « drapeau » à autre chose ou qu'elle passe à l'état « drapeau », elle en avertit le MinesweeperGame qui avertit le GameState pour qu'il sache toujours combien de drapeaux il y a sur le jeu. L'état d'une case représente en fait son apparence (ce que le ToggleButton de l'interface doit montrer) et chaque fois que celle-ci change, il y a inévitablement un appel de la méthode update sur le contrôleur GameWindowController.

La classe HighScore

La classe HighScore est une classe un peu spéciale, et nous sommes franchement surpris que ça fonctionne parfaitement…Essentiellement cette classe contient un HashTable avec 3 clés (Une pour chaque difficulté). Chaque clé a comme valeur un ArrayList de strings, qui contient le nom du joueur aillant le meilleur temps pour la difficulté, et son temps. Pour sauver le tout sur le disque nous encodons notre HashTable en XML avec la classe XMLEncoder (du package `java.beans`). Pour lire le fichier, nous utilisons la classe inverse, c'est-à-dire la classe

XMLDecoder. Lors de l'instanciation de la classe, on regarde si le fichier existe, et s'il n'existe pas nous initialisons un nouveau fichier avec des valeurs par défauts. Dans le cas où le XMLDecoder rencontrerait une erreur, il ne se casse pas la tête et il initialise un nouveau fichier (Bye bye les scores!). Ce n'est pas vraiment le plus idéal au monde, mais en même temps, on ne veut pas charger un fichier XML non valide, et on ne veut pas paralyser notre application si jamais ça arrive. Donc on ne prend pas de chance, on y va *safe*.

Le contrôleur GameWindowController

Ce contrôleur est particulier puisque c'est le seul qui soit lié au modèle comme « observer », autrement dit, à chaque changement d'état d'une case, le GameWindowController en sera informé à travers la méthode update. Celle-ci prend l'état de la case et détermine l'apparence du ToggleButton approprié, il en profite aussi pour s'informer de si la partie a été gagnée ou perdu, afin de modifier l'apparence du bouton « sourire ».

Plus généralement, c'est lui qui gère l'interface, apparence de tout, quel clic provoque quelle fonction (à l'exception de ceux sur les ToggleButton, qui ont leurs EventHandler personnels servants à savoir quel clic s'est produit et sur quelle case qu'il s'est produit) et envoi ces messages au MinesweeperGame.

La classe WindowBuilder

Dans notre projet nous avons utilisé le patron de conception Builder. La classe WindowBuilder a initialement été conçue afin de contourner le problème du manque de paramètres par défaut dans les fonctions en Java. Finalement, on ne l'utilise pas vraiment à cause de cette raison, mais plutôt parce que ça rend le processus de création de fenêtres plus élégantes et moins ad-hock. Très intéressant comme concept.

Commentaire sur la production

La production de l'application s'est, en général, bien passée. Évidemment nous avons quelques problèmes, mais ceux-ci étaient plus des défis concernant la matière que des vrais blocages nous empêchant de réaliser l'application. Les tâches ont été assez bien séparées et chacun a appris de nouvelles choses.

Le fait que nous avons utilisé GIT pour le TP #1 nous a grandement aidé pour ce TP, puisque la plupart des problèmes qu'on a eus avec GIT on les avait déjà rencontrés, ainsi on ne perdait pas de temps inutile à se battre avec notre système de versions de fichiers.

Auto-évaluation du produit final

L'application répond à tous les « user stories », elle fonctionne donc très bien. Avoir plus de temps pour la peaufiner, on trouverait un moyen de faire que le joueur ne puisse pas mourir au premier clique et que quand il tient le bouton gauche enfoncé, le bouton « sourire » prenne un

air apeuré. Sinon, nous trouvons que notre GameWindowController est un peu monstrueux par sa taille tout comme sa fonction update et nous préférerions les sous-diviser. Finalement, nous n'avons pas trouvé de bonne (et belle, et fonctionnelle) solution pour gérer la révélation des cases. Celle que nous avons utilisée implique trop le GameBoard à notre goût, puisque de passer en paramètre les voisins d'une case lorsqu'on la révèle ne fonctionne que pour la première case, et que si chaque tuile connaissait ses voisins ça prendrait trop de mémoire inutile, etc. Bref, nous avons beaucoup l'impression d'avoir choisi le moindre mal plutôt que la bonne solution pour ce problème et ça nous chicote dans la région *chicoteuse*. Pour le reste, nous estimons que c'est globalement bien fait.