Rapport d'implémentation

Projet de Génie Logiciel

Activité d'Apprentissage S-INFO-015

Groupe numéro : 3
Membres du groupe :
DOM Eduardo , DHEUR Victor, AMEZIAN Aziz

Année Académique : 2017 - 2018
BAC 2 en Sciences Informatiques

Faculté des Sciences, Université de Mons

30 mars 2018

1 Introduction

Une vidéo explicative du fonctionnement du jeu est disponible sur Youtube à l'adresse `https://www.youtube.com/watch?v=W9WH--D0s7w`.

La phase d'implémentation s'est globalement bien passée. Nous avons suivi ce que nous avions prévu dans le diagramme de Gantt, excepté que nous avons développé les extensions couleurs et intensité en même temps que le modèle, car il était plus facile pour nous de l'intégrer directement. Les extensions *Level Checker* et *Editor* ont en revanche été créées après le jeu de base.

L'extension *Sauvegarde + Authentification de différents utilisateurs + Hall of Fame* a été implémentée en plus des autres extensions.

2 Exécution du projet

Les commande suivantes peuvent être utilisées à la racine du projet pour l'exécuter :

```
gradlew desktop:run
gradlew core:test
gradlew compile
gradlew javadoc
```

De plus, une commande permet de compiler le jeu ainsi que de lancer les tests et créer la JavaDoc.

```
gradlew generate
```

3 Modèle UML

Nous avons directement structuré notre projet selon le diagramme de classe. Certains classes ont été ajoutées de manière à rendre le code plus structuré. Une classe `Hint` a été utilisée afin de trouver les mouvements nécessaires pour atteindre une solution. Nous avons aussi choisi d'ajouter une classe `Modifier` héritant d'Hexagon dont héritent à leur tour les modifiers afin de regrouper le comportement de ceux-ci.

Nous avons ajouté certaines sous-classes lors du développement, afin de spécialiser certains comportements en surchargeant des méthodes. Par exemple, les classes `GameLevel` et `EditorLevel` héritent de `Level` et `GameMapRenderer` hérite de `MapRenderer`.

Les diagrammes d'activité d'envoi du laser et de détection de victoire ont bien été respectés. Nous avons utilisé notre diagramme d'activité montrant le déplacement de l'hexagone mais celui-ci a dû être complexifié afin de respecter les contraintes et le déplacement entre la map principale et les containers.

Le diagramme de séquence décrivant le déroulement d'un `BasicGame` a été respecté.

Enfin, le Level Checker a respecté son diagramme de séquence, en ajoutant quelques éléments afin de l'améliorer.

4 Maquette

En général, nous avons suivi l'interface que nous avions définie dans la maquette. Nous avons modifié les miniatures des niveaux car nous ne réussissions pas à disposer les images de niveaux de manière consistante.

Dans l'éditeur, plusieurs champs ont été ajoutés afin de modifier le nom, le temps limite, la difficulté et de pouvoir charger le niveau de l'éditeur à partir d'un niveau existant.

Deux boutons ont été ajoutés dans l'éditeur. Le premier est *Fill*, il sert à mettre des hexagones laissant passer le laser là où il n'y a rien. Le deuxième est *Show solutions*, il permet de voir toutes les solutions disponibles du niveau actuel de l'éditeur.

Dans le mode démonstration, une barre de chargement a été ajoutée.

5 Choix d'implémentation

Nous avons publié notre code sur GitHub durant toute l'implémentation, ce qui a fortement facilité le développement en équipe.

Nous avons implémenté le jeu de base de façon à faciliter le plus possible de développement des extensions. Par exemple, l'interface et le controleur du jeu de base ont pu être réutilisés dans l'éditeur.

Les packages indiqués sur le diagramme de classe ont été utilisés et un package `gui` contenant toutes les classes créant les menus a été ajouté.

Nous avons utilisé le test-driven development lors de la création du model, mais peu pour les autres parties. Les tests créés concernent le modèle, le vérificateur de niveau et la lecture des fichiers XML.

Afin de nous assurer que les niveaux sont correctement structurés dans leur fichier XML, nous avons créé un fichier XSD décrivant de manière précise le format attendu du niveau.

6 Design patterns

Nous avons choisi l'utilisation du Design Pattern MVC pour représenter le jeu. Il nous a permis de travailler de manière indépendante sur le modèle et le controlleur et la vue. Cette limitation des dépendances nous a aussi permis de détecter les bugs plus facilement car le malfonctionnement d'une des parties n'affecte pas les autres.

Le design pattern Abstract Factory a été utilisé afin de créer des widgets LibGDX de manière uniforme, et de pouvoir changer l'apparence de l'interface facilement. La classe `WidgetFactory` déclare l'Abstract Factory et `ScaledTextWidgetFactory` implémente cette classe.

Le pattern Observer a été un peu utilisé de manière à ce que `AutomaticGameScreen` (la classe dessinant le mode démonstration), puisse être informée de la fin des niveaux par `AutomaticController` (la classe prenant en charge le fonctionnement du mode démonstration) et passer au niveau suivant.

La classe `UserManager` a été construite selon le pattern Singleton.

7 Algorithmes

Les algorithmes principaux du projet sont l'algorithme utilisé pour l'envoi du laser, celui permettant de trouver les solutions d'un niveau et celui permettant de donner des indices afin de se diriger vers une solution.

L'envoi du laser respecte son diagramme d'activité, et n'a pas changé pendant l'implémentation.

L'algorithme permettant de trouver les solutions d'un niveau, implémenté dans `LevelChecker`, a subit quelques ajouts.

Une idée précédente était de placer les hexagones par permutation distincte, une par une. Les permutations sont maintenant parcourues par l'algorithme lui-même. On peut visualiser l'algorithme de placement des hexagones comme un arbre dont chaque noeud a comme enfant les prochains hexagones distincts pouvant être placés, ou aucun. Cette modification a permis d'éviter de parcourir plusieurs permutations amenant le même résultat, enlevant des possibles duplications.

De plus, l'algorithme vérifie aussi s'il peut placer les hexagones en tant que modifiers, et prend en compte les contraintes. Les solutions comprenant des hexagones "inutiles" (n'interagissant avec aucun laser) ne sont pas trouvées par l'algorithme dû à son fonctionnement, ce qui est avantageux.

Une estimation de l'avancement du calcul des solutions est faite, permettant d'ajouter une barre de chargement dans le mode démonstration. Cette estimation est calculée en donnant un poids initial au noeud considéré comme la racine des récursions du Level Checker. A chaque fois que des récursions sont effectuées, le poids est partagé équitablement. Une fois qu'une feuille a été atteinte, on peut ajouter son poids au chargement. Plus le chargement, se rapproche du poids initial, plus le parcours de l'arbre est proche de la fin.

L'algorithme final permet de trouver les solutions distinctes de n'importe quel niveau, la seule limite étant le temps que cela prendra. Il permet de trouver assez rapidement les solutions de tous les niveaux inclus au jeu, et en général de tous les niveaux raisonnables en taille et en nombre d'hexagones à placer.

Enfin, la classe `Hint` comprend un algorithme permettant d'appliquer des indices sur la `Map`. En comparant les objets `Map` actuels et les objets `Map` de la solution, elle trouve les positions intiales et finales d'un mouvement, en vérifiant que la prochaine solution sera bien plus proche de la résolution du niveau. Il est ainsi garanti qu'une solution sera atteinte en appliquant continuellement des indices.

8 Difficultés rencontrées

La principale difficulté que nous avons rencontreé est l'apprentissage de LibGDX. Le manque de documentation et l'abondance de tutoriaux obsolètes nous ont obligé à apprendre à utiliser certaines fonctionnalités de LibGDX par "essai et erreur".

9 Librairies utilisées

Nous avons utilisé la librairie Java inclue dans le JRE, car elle remplit nos besoins et permet de vérifier que nos fichiers XML respectent la structure imposée par notre fichier XSD.

Une classe que nous avons trouvé sur un dépôt github à l'adresse `https://github.com/TomGrill/gdx-testing` a été adaptée afin de permettre le lancement des tests avec LibGDX. Elle est sous license Apache 2.

10 Conclusion

La planification du projet nous a été utile afin de convenir entre les membres du groupe sur les différents délais imposés.

La modélisation a été une étape avantageuse pour nous car elle nous a permis de nous mettre d'accord sur la structure du projet avant l'implémentation, et de pouvoir facilement mettre notre code en commun.

Plus d'expérience en création de projet et en LibGDX nous aurait permis de faire une planification et une modélisation plus précises, les rendant encore plus avantageuses. Nous pensons que ce projet nous a permis d'acquérir de l'expérience en génie logiciel et qu'il nous aidera à mener à bien de futurs projets.