Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Série 2

Mine Hunt - Création d'un démineur

David Alvarez Luca Moos

15 avril 2017

Soucieux de l'environnement, nous avons réduit les marges de ce document au minimum afin d'économiser quelques feuilles A4

Table des matières

1 Introduction **1**
1.1 Objectifs 1

2 Application **1**
2.1 Vues . . . 1
2.1.1 Vue principale . . . 1
2.1.2 Settings . . . 1
2.1.3 Vue principale avec un terrain plus grand en cours de partie . . . 2
2.2 Graphes de scène . . . 2
2.2.1 Vue principale . . . 2
2.2.2 Settings . . . 3
2.3 Diagramme UML . . . 3
2.4 Description des classes . . . 3
2.5 Fonctionnalité implémentées . . . 4
2.6 Code (fichiers *.java*) . . . 4
2.6.1 Cellbutton.java . . . 4
2.6.2 IMineHuntController.java . . . 4
2.6.3 MineHuntController.java . . . 4
2.6.4 MineHuntModel.java . . . 4
2.6.5 MineHuntView.java . . . 5
2.6.6 NewGameController.java . . . 5
2.6.7 SettingController.java . . . 5
2.6.8 SettingView.java . . . 5
2.6.9 ShowMinesController.java . . . 5
2.6.10 TerrainController.java . . . 5

3 Conclusion **5**
3.1 Difficultés rencontrées . . . 5
3.2 Idées d'amélioration . . . 5

1 Introduction

1.1 Objectifs

— Créer une application *Java* comportant une interface utilisateur graphique
— Réaliser une vue en créant un graphe de scène et en configurant les conteneurs et les composants
— Insérer des images (ressources externes) dans une application
— Structurer l'application en basant la conception et le codage sur l'architecture *MVC*[1]. Créer une interface du modèle permettant des variantes d'implémentation
— Gérer les actions de l'utilisateur en traitant des événements de type **ActionEvent** et **MouseEvent**
— Gérer un menu et créer des boîtes de dialogue simples pour informer ou questionner l'utilisateur

2 Application

2.1 Vues

2.1.1 Vue principale

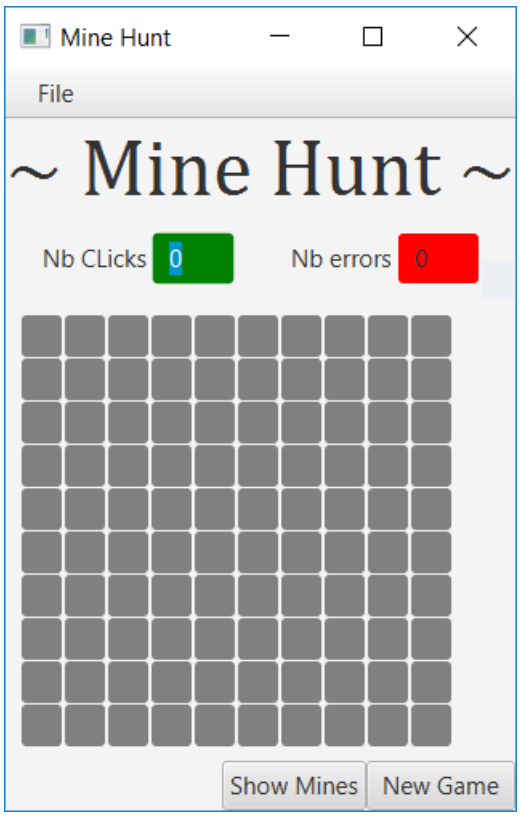

FIGURE 1 – Vue principale

1. Model-View-Controller

2.1.2 Settings

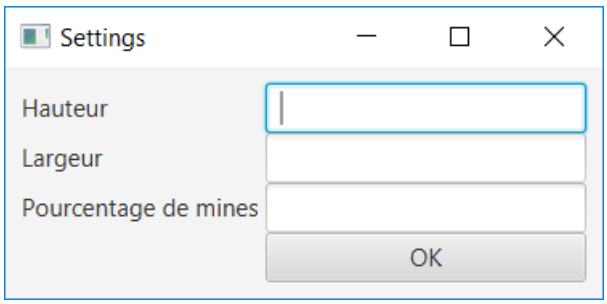

FIGURE 2 – Vue *Settings*

2.1.3 Vue principale avec un terrain plus grand en cours de partie

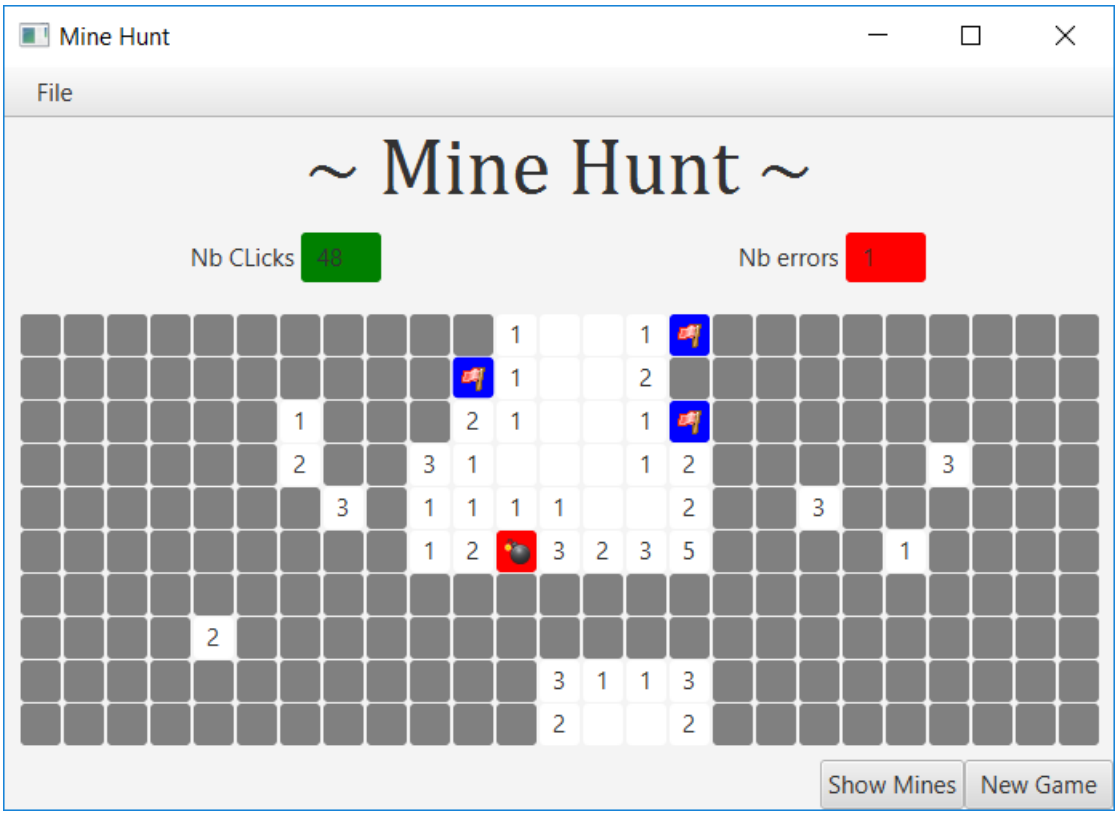

FIGURE 3 – Partie en cours

2.2 Graphes de scène

2.2.1 Vue principale

Voici le graphe de scène de la vue principale :

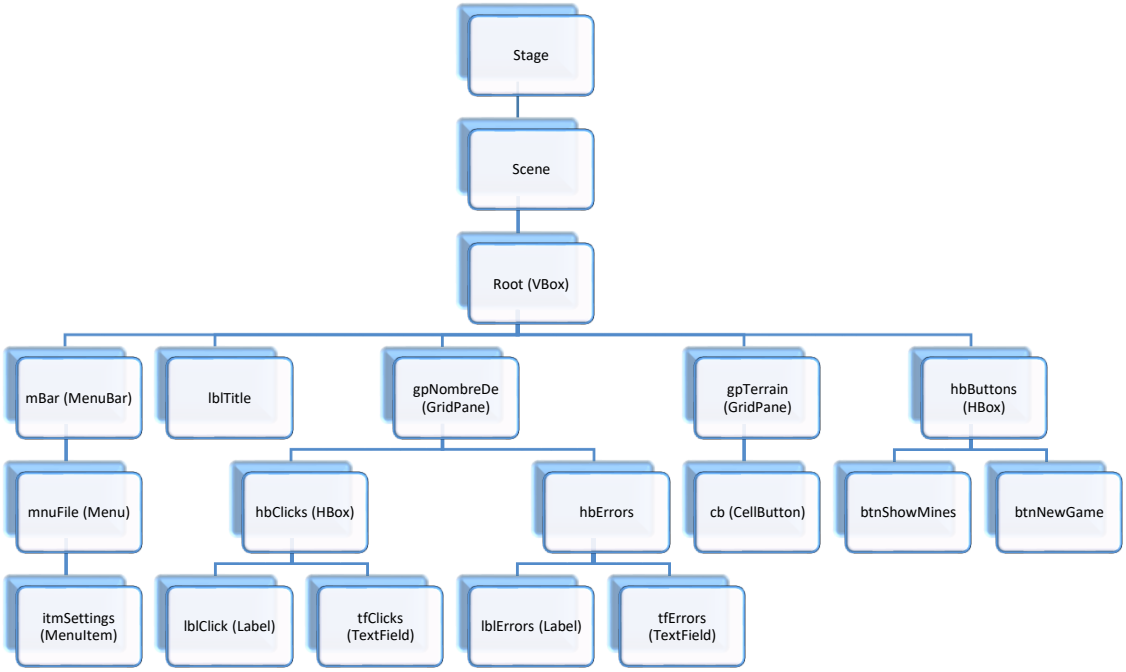

FIGURE 4 – Graphe de scène de la vue principale

Remarque : sur le schéma il n'y a qu'un seul *cb (CellButon.* Dans la réalité, il y a *hauteur* × *largeur* nombre de *Cellbuttons.*

2.2.2 Settings

Voici le graphe de scène de *Settings*

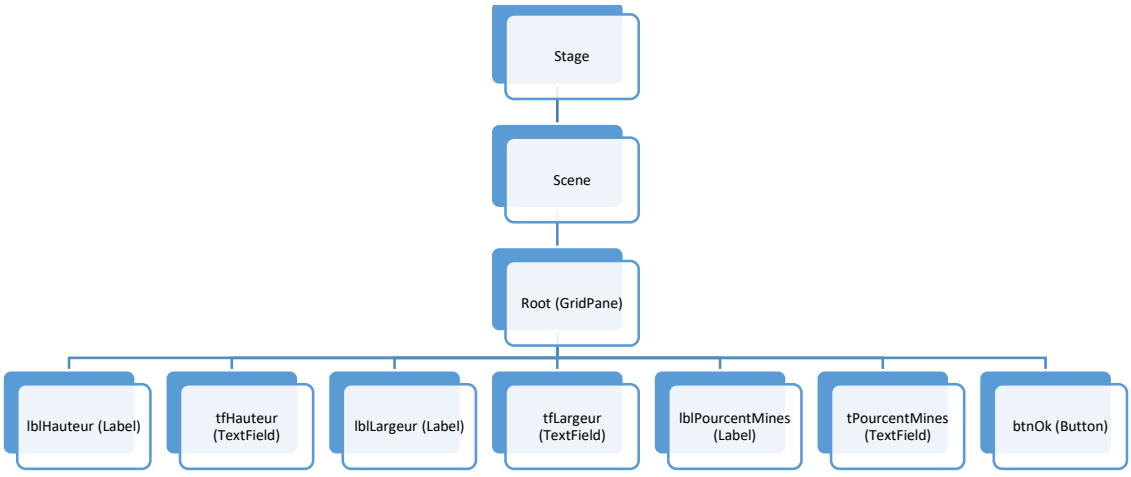

FIGURE 5 – Graphe de scène de *Settings*

2.3 Diagramme UML

2.4 Description des classes

— Cellbutton
— IMineHuntModel

— MineHuntController
— MineHuntModel
— MineHuntView
— NewGameController
— SettingController
— SettingView
— ShowMinesController
— TerrainController

Et la description de chacune d'entre elles :

CellButton C'est la classe que nous avons utilisée pour les mines, elle hérite de la classe *Button*. Nous lui avons ajouté les attributs *ligIndex* et *colIndex*, afin de pouvoir savoir sur quel bouton nous cliquons (parmi tous les boutons présent dans notre "terrain" de mines

IMineHuntModel C'est l'interface de notre modèle. Elle contient les méthodes que nous pensions utiliser pour notre modèle. Finalement, notre interface ne contenait pas toutes les méthodes nécessaires, nous avons donc du en rajouter dans notre modèle

MineHuntController Cette classe permet de faire le lien entre entre la vue principale et le modèle

MineHuntModel C'est le modèle, elle contient toute l'intelligence de notre programme

MineHuntView La fenêtre principale

NewGameController Le controller faisant le lien entre le bouton *New Game* et le modèle

SettingController Le controller faisant le lien entre la fenêtre *Setting* et le modèle

SettingView La fenêtre secondaire permettant d'afficher les paramètre pour la partie suivante

ShowMinesController Le controller faisant le lien entre le bouton *ShowMines* et le modèle

TerrainController Le controller faisant le lien entre les *CellButtons* et le modèle

2.5 Fonctionnalité implémentées

Voici la liste des éléments que nous avons implémenter dans notre application :
— Possibilité de définir la hauteur, la largeur et le pourcentage de mines du terrain par l'utilisateur
— Nouvelle partie
— Montrer les mines
— Placer des drapeaux
— Click automatique sur toutes les cases autour d'une case n'ayant pas de mine(s) autour
— Compteur de clicks
— Compteur d'erreurs

2.6 Code (fichiers *.java*)

Ici, nous verrons le code de chacune des classes et expliquerons plus en détail leurs méthodes.

2.6.1 Cellbutton.java

2.6.2 IMineHuntController.java

2.6.3 MineHuntController.java

2.6.4 MineHuntModel.java

Le modèle contient trois attributs principaux :

terrain Tableau 2D de booléens représentant toutes les cases. *true* si une bombe est présente, sinon *false*

dejaClique Tableau 2D de booléens représentant l'état actuel des cases. *true* si la case à déjà été cliquée, sinon *false*

flagged Tableau 2D de booléens représentant les drapeaux sur le terrain. *true* si un drapeau est présent sur la case, sinon *false*

2.6.5 MineHuntView.java

2.6.6 NewGameController.java

2.6.7 SettingController.java

2.6.8 SettingView.java

2.6.9 ShowMinesController.java

2.6.10 TerrainController.java

3 Conclusion

Le développement de cette application fut très enrichissant. D'une part, cela nous a permis de nous familiariser avec beaucoup de composants *JavaFX* et d'autre part avec l'architecture MVC et la gestion des événements.

3.1 Difficultés rencontrées

Voici les éléments qui nous ont posé problème :

— Au départ, c'était difficile de coder toutes les fonctionnalités en MVC. Nous ne pensions pas qu'il était nécessaire de mémoriser les drapeau dans le modèle par exemple. Nous étions parti sur l'idée que puisqu'il ne s'agissait que de l'affichage d'un drapeau, cela aurait pu être placé dans la vue

— Nous sommes restés bloqués un long moment à cause des tableaux en deux dimensions. Nous en manipulons tellement, que nous avions des méthodes avec lesquelles nous prenions la hauteur à la place de la largeur et vis-versa, et cela menait à des problèmes.
Le simple fait que nous avons codés tout nos tableaux en mettant *[ligIndex][colIndex]* nous a donné des difficultés avec l'ajout des *CellButtons* dans le *GridPane* parce que la méthode permettant l'ajout d'éléments dans le *GridPane* prend d'abord en paramètre l'index de la colonne et ensuite l'index de la ligne.

3.2 Idées d'amélioration

— Faire en sorte de limiter le nombre de mines pour que le programme de soit pas plus grand que l'écran

— Dans la fenêtre *Settings*, n'autoriser que les numéros dans les champs texte, ou remplacer les champs texte par des *Spinners*

— Sauvegarder la partie en cours