**Interface-Homme Machine / Travaux pratiques**

S02 – Application avec interface graphique – MineHunt





17.03.2015

Adriano De Almeida Silva – T-1f

Alex Travasso – T-1f

**Tables des matières**

[1. Structure MVC de notre code 3](#_Toc417383762)

[2. Arborescence 3](#_Toc417383763)

[3. Code 3](#_Toc417383764)

[a. Contenu de la classe modèle proposée (IMineHuntModel) 3](#_Toc417383765)

[b. Contenu de la classe modèle (MineHuntModel) 5](#_Toc417383766)

[c. Contenu de la classe case (Case) 12](#_Toc417383767)

[d. Contenu de la classe de test (TestMineHuntModel) 13](#_Toc417383768)

[e. Contenu de la spécialisation de la classe Button(CellButton) 13](#_Toc417383769)

[f. Contenu de la classe view (MineHuntView) 13](#_Toc417383770)

[g. Contenu de la classe controller (CtrlMineHunt) 17](#_Toc417383771)

[4. Remarques 18](#_Toc417383772)

# Structure MVC de notre code

|  |  |  |
| --- | --- | --- |
| **Model** | **View** | **Controller** |
|  |  |  |
| Taille de la grille | Application | Gestion de événements |
| Génération random des mines | Grille |  |
| Nombre de mines | Boutons |  |
| Comptage d’erreurs |  |  |
| Comptage de clicks |  |  |
| Gestion du voisinage |  |  |

# Arborescence

|  |  |  |
| --- | --- | --- |
| **Model** | **View** | **Controller** |
|  |  |  |
| Case.java | CellButton.java | CtrlMineHunt.java |
| IMineHuntModel.java | MineHuntView.java |  |
| MineHuntModel.java |  |  |
| TestMineHuntModel.java |  |  |

# Code

# Contenu de la classe modèle proposée (IMineHuntModel)

//==========================================================================

// HEIA-FR / Jacques Bapst

//==========================================================================

package s02.model;

public interface IMineHuntModel {

/\*\*

\* Initialize a new game

\*

\* Reset game state (all grid's cells, error-counter, ...) and place a new

\* random arrangement of mines

\*

\* @param minesNb

\* Number of mines

\*/

void initNewGame(int minesNb);

/\*\*

\* Number of columns in grid

\*/

int gridWidth();

/\*\*

\* Number of rows in grid

\*/

int gridHeight();

/\*\*

\* Number of mines in grid

\*/

int mines();

/\*\*

\* Error-counter state.

\*/

int errors();

/\*\*

\* Number of mines in the neighborhood of a cell

\*

\* @param row

\* Cell row index [ 0..gridHeight()-1 ]

\* @param col

\* Cell column index [ 0..gridWidth()-1 ]

\*/

int neighborMines(int row, int col);

/\*\*

\* Return true if cell has been opened (clicked)

\*

\* @param row

\* Cell row index [ 0..gridHeight()-1 ]

\* @param col

\* Cell column index [ 0..gridWidth()-1 ]

\*/

boolean isOpen(int row, int col);

/\*\*

\* Return true if cell has been flagged (supposed mine position)

\*

\* @param row

\* Cell row index [ 0..gridHeight()-1 ]

\* @param col

\* Cell column index [ 0..gridWidth()-1 ]

\*/

boolean isFlagged(int row, int col);

/\*\*

\* Check if game is over. Return true only if all non-mined cells have been

\* opened (clicked)

\*

\* @return game-over state

\*/

boolean isGameOver();

/\*\*

\* Open a cell (user clicked on it).

\*

\* No effect if cell is already open.

\*

\* If a mine is in the cell, the internal error-counter is incremented.

\*

\* @param row

\* Cell row index [ 0..gridHeight()-1 ]

\* @param col

\* Cell column index [ 0..gridWidth()-1 ]

\* @return true if a mine was in the cell (error-counter incremented)

\*/

boolean open(int row, int col);

/\*\*

\* Set flag state of a cell

\*

\* @param row

\* Cell row index [ 0..gridHeight()-1 ]

\* @param col

\* Cell column index [ 0..gridWidth()-1 ]

\* @param state

\* Flag-state of the cell

\*/

void setFlagState(int row, int col, boolean state);

}

# Contenu de la classe modèle (MineHuntModel)

package s02.model;

import java.util.Random;

public class MineHuntModel implements IMineHuntModel {

int gridraw;

int gridcol;

int mines;

int errors;

int leftToOpen;

Case[][] grid;

Random random;

// Constructeur qui reçoit en paramètre le nombre de lignes, de colonnes et

// de mines.

public MineHuntModel(int gridraw, int gridcol, int mines) {

this.gridraw = gridraw;

this.gridcol = gridcol;

this.mines = mines;

random = new Random();

}

// Retourne un emplacement aléatoire dans le tableau pour une mine.

private String randomizeMine() {

int randRawMine = random.nextInt(gridraw);

int randColMine = random.nextInt(gridcol);

return randRawMine + ";" + randColMine;

}

// Vérifie si l'emplacement de la mine générée est un emplacement libre,

// sinon regénère des mines jusqu'à qu'il trouve un emplacement libre.

// Retourne l'emplacement de la mine.

private int[] randomMineOK() {

String[] tab = new String[2];

int[] tabResultat = new int[2];

boolean res = true;

while (res) {

tab = randomizeMine().split(";");

int rawi = Integer.valueOf(tab[0]);

int coli = Integer.valueOf(tab[1]);

if (!grid[rawi][coli].getContainMine()) {

res = false;

tabResultat[0] = rawi;

tabResultat[1] = coli;

}

}

return tabResultat;

}

// Teste si la case contient une mine, si oui elle incrémente les erreurs et

// retourne true.

public boolean isMine(int row, int col) {

boolean resultat = false;

if (grid[row - 1][col - 1].getContainMine()) {

errors++;

resultat = true;

}

return resultat; }

// Initialise une nouvelle partie, en créant la grille et en posant des

// mines aléatoirement.

@Override

public void initNewGame(int minesNb) {

int tab[] = new int[2];

grid = new Case[gridraw][gridcol];

for (int i = 0; i < grid.length; i++) {

for (int j = 0; j < grid[0].length; j++) {

grid[i][j] = new Case(false, false, false);

}

}

for (int i = 0; i < minesNb; i++) {

tab = randomMineOK();

grid[tab[0]][tab[1]].setMine(true);

}

leftToOpen = (gridraw \* gridcol) - mines;

}

// Retourne le nombre de colonnes.

@Override

public int gridWidth() {

return gridcol;

}

// Retourne le nombre de lignes.

@Override

public int gridHeight() {

return gridraw;

}

// Retourne le nombre de mines.

@Override

public int mines() {

return mines;

}

// Retourne le nombre d'erreurs.

@Override

public int errors() {

return errors;

}

// Retourne le nombre de mines voisine de la case à l'emplacement donné.

// parametre row va de 1 à rowMax, idem pour col.

@Override

public int neighborMines(int row, int col) {

int neighborMines = 0;

if (row - 1 == 0 && col - 1 == 0) {

if (grid[row][col - 1].getContainMine()) {

neighborMines++;

}

if (grid[row - 1][col].getContainMine()) {

neighborMines++;

}

if (grid[row][col].getContainMine()) {

neighborMines++;

}

} else if (row - 1 > 0 && col - 1 == 0) {

if (grid[row - 2][col - 1].getContainMine()) {

neighborMines++;

}

if (grid[row - 2][col].getContainMine()) {

neighborMines++;

}

if (grid[row - 1][col].getContainMine()) {

neighborMines++;

}

if (grid[row][col].getContainMine()) {

neighborMines++;

}

if (grid[row][col - 1].getContainMine()) {

neighborMines++;

}

} else if (row - 1 == gridraw - 1 && col - 1 == 0) {

if (grid[row - 2][col - 1].getContainMine()) {

neighborMines++;

}

if (grid[row - 2][col].getContainMine()) {

neighborMines++;

}

if (grid[row - 1][col].getContainMine()) {

neighborMines++; }

} else if (row - 1 == gridraw - 1 && col - 1 > 0) {

if (grid[row - 1][col - 2].getContainMine()) {

neighborMines++;

}

if (grid[row - 2][col - 2].getContainMine()) {

neighborMines++;

}

if (grid[row - 2][col - 1].getContainMine()) {

neighborMines++;

}

if (grid[row - 2][col].getContainMine()) {

neighborMines++;

}

if (grid[row - 1][col].getContainMine()) {

neighborMines++;

}

} else if (row - 1 == gridraw - 1 && col - 1 == gridcol - 1) {

if (grid[gridraw - 1][gridcol - 2].getContainMine()) {

neighborMines++;

}

if (grid[gridraw - 2][gridcol - 2].getContainMine()) {

neighborMines++;

}

if (grid[gridraw - 2][gridcol - 1].getContainMine()) {

neighborMines++;

}

} else if (row - 1 > 0 && col - 1 == gridcol - 1) {

if (grid[row - 2][col - 1].getContainMine()) {

neighborMines++;

}

if (grid[row - 2][col - 2].getContainMine()) {

neighborMines++;

}

if (grid[row - 1][col - 2].getContainMine()) {

neighborMines++;

}

if (grid[row][col - 2].getContainMine()) {

neighborMines++; }

if (grid[row][col - 1].getContainMine()) {

neighborMines++; }

} else if (row - 1 == 0 && col - 1 == gridcol - 1) {

if (grid[row - 1][gridcol - 2].getContainMine()) {

neighborMines++;

}

if (grid[row][gridcol - 2].getContainMine()) {

neighborMines++;

}

if (grid[row][gridcol - 1].getContainMine()) {

neighborMines++;

}

} else if (row - 1 == 0 && col - 1 > 0) {

if (grid[row - 1][col - 2].getContainMine()) {

neighborMines++;

}

if (grid[row][col - 2].getContainMine()) {

neighborMines++;

}

if (grid[row][col - 1].getContainMine()) {

neighborMines++;

}

if (grid[row][col].getContainMine()) {

neighborMines++;

}

if (grid[row - 1][col].getContainMine()) {

neighborMines++;

}

} else if (row - 1 > 0 && col - 1 > 0) {

if (grid[row - 2][col - 2].getContainMine()) {

neighborMines++;

}

if (grid[row - 2][col - -1].getContainMine()) {

neighborMines++;

}

if (grid[row - 2][col].getContainMine()) {

neighborMines++;

}

if (grid[row - 1][col - 2].getContainMine()) {

neighborMines++; }

if (grid[row - 1][col].getContainMine()) {

neighborMines++; }

if (grid[row][col - 2].getContainMine()) {

neighborMines++;

}

if (grid[row][col - 1].getContainMine()) {

neighborMines++;

}

if (grid[row][col].getContainMine()) {

neighborMines++;

}

}

return neighborMines;

}

// Dit si une case à l'emplacement donné est ouverte.

@Override

public boolean isOpen(int row, int col) {

return grid[row - 1][col - 1].isOpened();

}

// Dit si une case à l'emplacement donné est flagguée.

@Override

public boolean isFlagged(int row, int col) {

return grid[row - 1][col - 1].isFlagged();

}

// Teste si le jeu est terminé.

@Override

public boolean isGameOver() {

if (leftToOpen == 0)

return true;

return false;

}

// Ouvre une case à l'emplacement donné en paramètre.

@Override

public boolean open(int row, int col) {

boolean resultat = false;

if (!grid[row - 1][col - 1].isOpened()) {

grid[row - 1][col - 1].setOpened(true);

leftToOpen--;

resultat = true;

}

return resultat;

}

// Met le flag selon la spécification à une case de l'emplacement donné en

// paramètre.

@Override

public void setFlagState(int row, int col, boolean state) {

grid[row - 1][col - 1].setFlag(state);

}

}

# Contenu de la classe case (Case)

package s02.model;

public class Case {

private boolean containMine;

private boolean flag;

private boolean opened;

public Case(boolean containMine, boolean flag, boolean opened) {

this.containMine = containMine;

this.flag = flag;

this.opened = opened;

}

public boolean getContainMine() {

return containMine;

}

public void setMine(boolean containMine) {

this.containMine = containMine;

}

public boolean isFlagged() {

return flag;

}

public void setFlag(boolean flag) {

this.flag = flag;

}

public boolean isOpened() {

return opened;

}

public void setOpened(boolean opened) {

this.opened = opened;

}

}

# Contenu de la classe de test (TestMineHuntModel)

Pas réalisé

# Contenu de la spécialisation de la classe Button(CellButton)

package s02.view;

import com.sun.prism.paint.Color;

import javafx.scene.control.Button;

public class CellButton extends Button{

int x;

int y;

this.setStyle("-fx-background-color: grey;");

public CellButton(int x, int y) {

this.x = x;

this.y = y;

}

public int getx() {

return this.x;

}

public int gety() {

return this.y;

}

}

# Contenu de la classe view (MineHuntView)

package s02.view;

import s02.controller.CtrlMineHunt;

import javafx.application.Application;

import javafx.event.ActionEvent;

import javafx.event.Event;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.input.MouseButton;

import javafx.scene.layout.\*;

import javafx.scene.paint.Color;

import javafx.scene.text.Font;

import javafx.stage.Stage;

public class MineHuntView extends Application {

// Boxes

private VBox root = new VBox(20);

private VBox vgrid = new VBox(30);

private HBox htitle = new HBox(30);

private HBox hinfo = new HBox(30);

private HBox hclick = new HBox(30);

private HBox herror = new HBox(30);

private HBox hbutton = new HBox(30);

// Position des cases

int px = 0;

int py = 0;

// GridPane

private GridPane grid = new GridPane();

// Bottom buttons

Button mines = new Button("Show Mines");

Button newgame = new Button("New Game");

// Labels

Label title = new Label("MINEHUNT");

Label click = new Label("NB Clicks: ");

Label error = new Label("NB Errors: ");

private CtrlMineHunt ctrl;

public MineHuntView() {

ctrl = new CtrlMineHunt(this);

}

public void start(Stage mainStage) throws Exception {

// events sur le bouton pour afficher les mines

mines.setOnMouseClicked(ActionEvent -> {

showMines();

});

// events sur le bouton pour rÃ©initialiser le jeu

newgame.setOnMouseClicked(ActionEvent -> {

// rÃ©initialise le jeu

// init new game

});

// Stage

mainStage.setTitle("MineHunt App");

mainStage.setMinWidth(500);

mainStage.setMinHeight(500);

mainStage.setResizable(true);

// Appropriation

root.getChildren().add(htitle);

root.getChildren().add(hinfo);

hinfo.getChildren().add(hclick);

hinfo.getChildren().add(herror);

root.getChildren().add(vgrid);

root.getChildren().add(hbutton);

// Positions

htitle.setAlignment(Pos.CENTER);

hinfo.setAlignment(Pos.CENTER);

hclick.setAlignment(Pos.CENTER\_LEFT);

herror.setAlignment(Pos.CENTER\_RIGHT);

grid.setAlignment(Pos.CENTER);

hbutton.setAlignment(Pos.BOTTOM\_RIGHT);

// Creation of the grid

createGrid(10, 10);

// Space between buttons in the grid

grid.setHgap(1);

grid.setVgap(1);

// background

Background bgMine = new Background(new BackgroundFill(

Color.LIGHTYELLOW, CornerRadii.EMPTY, null));

root.setBackground(bgMine);

// Police et ajout du text

title.setFont(Font.font("SansSerif", 44));

title.setTextFill(Color.RED);

// Insertion

htitle.getChildren().add(title);

hinfo.getChildren().add(click);

hinfo.getChildren().add(error);

hbutton.getChildren().add(mines);

hbutton.getChildren().add(newgame);

vgrid.getChildren().add(grid);

// show

mainStage.setScene(new Scene(root));

mainStage.show();

}

public void setCtrl(CtrlMineHunt ctrl) {

this.ctrl = ctrl;

}

// Create the grid by the parameters

public void createGrid(int x, int y) {

// Parcourt selon la taille

for (int i = 0; i < x; i++) {

for (int j = 0; j < y; j++) {

// By the specification CellButton

CellButton btn = new CellButton(i, j);

btn.setMinSize(30, 30);

// btn.setrouge();

btn.setOnMouseClicked(event -> {

if (event.getButton() == MouseButton.PRIMARY) {

// Click gauche

leftClickAction(px, py);

} else if (event.getButton() == MouseButton.SECONDARY) {

// Click droit

rightClickAction(px, py);

}

});

// CrÃ©ation de la grille de bouton

grid.add(btn, i, j);

// IncrÃ©ment position y

py++;

}

// IncrÃ©mente position x

px++;

}

}

// Click gauche sur une case

// si un flag est prÃ©sent, enlÃ¨ve le flag de la case

// sinon, met un flag sur la case

public void leftClickAction(int i, int j) {

boolean mine = ctrl.open(i, j);

if (mine) {

// methode affiche mine sur la case et met à jour le label errors.

} else {

int minesNeighbor = ctrl.showNeighborMines(i, j);

// methode affiche bouton cliqué avec le nombre de mines voisines

// ou sans rien si aucune mines.

}

}

// Click droit sur une case

// si une mine est prÃ©sent --> ajoute une erreur

// sinon, affiches les mines voisines

public void rightClickAction(int i, int j) {

if (ctrl.containFlag(i, j)) {

ctrl.setFlag(i, j, false);

// methode qui enleve l'image du flag du bouton.

} else {

ctrl.setFlag(i, j, true);

// Methode qui met l'image du flag sur le bouton.

}

}

// Affiche l'emplacement des mines

private void showMines() {

// xxx

}

public void gameOver() {

// bloque tous les boutons et affiche une alerte indiquant que le jeu

// est terminé.

}

}

# Contenu de la classe controller (CtrlMineHunt)

package s02.controller;

import s02.model.MineHuntModel;

import s02.view.MineHuntView;

public class CtrlMineHunt {

private MineHuntView view;

private MineHuntModel model;

public CtrlMineHunt(MineHuntView view) {

super();

this.view = view;

model = new MineHuntModel(10, 10, 10);

}

// InsÃ¨re le flag (coleur bleue)

public void setFlag(int x, int y, boolean isflagged) {

model.setFlagState(x, y, isflagged);

}

// Ouvre une case, retourne true si la case contient une mine et false si la

// case est vide.

public boolean open(int x, int y) {

return model.open(x, y);

}

// retourne si la case est flag ou non

public boolean containFlag(int x, int y) {

return model.isFlagged(x, y);

}

// Retourne le nombre d'erreurs

public int GetErrors() {

return model.errors();

}

// Affiche le nombre de mines voisine

public int showNeighborMines(int row, int col) {

return model.neighborMines(row, col);

}

}

## Contenu de la classe MineHunt

package s02;

import s02.controller.CtrlMineHunt;

import s02.model.MineHuntModel;

import s02.view.MineHuntView;

import javafx.application.Application;

public class MineHunt {

public static void main(String[] args) {

Application.launch(MineHuntView.class, args);

}

}

# Remarques

Etant donné le nombre important de travaux écrits et travaux pratiques que nous avons eu au cours de ces 2 dernières semaines, nous n’avons pas réussi à coder notre démineur dans l’intégralité des fonctions demandées. Il va de soi que cela et pleinement un manque du à notre gestion du temps. Toutefois, nous avons essayé de faire au mieux avec nos connaissances.

Nous trouvons ce travail pratique très intéressant car il allie jeu et java ce qui est pour nous une source de satisfaction. En revanche, le travail contient peut-être un peu trop de contenu par rapport à nos connaissances actuelles en java. Ce ne sont pas les fonctionnalités demandées qui nous ont posé des problèmes et fait perdre du temps. Il nous a fallu pas mal d’essais, tests et d’aide afin d’apprivoiser la gestion entre les interfaces javafx et le code java en lui-même. C’est probablement ce qui nous a posé le plus de soucis.