Object-Oriented Programming

Project Statement

Academic Year 2016-2017

This project is individual and must be submitted to `oop@montefiore.ulg.ac.be` *for May 8th at the latest. Projects returned after the deadline will not be corrected. Plagiarism is not tolerated, in line with the policy of the university (*`http://www.ulg.ac.be/plagiat`*).*

This year's project consists in implementing in Java a simplified version of the *Minesweeper* game. *Minesweeper* is a classic single-player game played with the mouse on a rectangular grid of tiles, where the player has to discover mines without activating them.

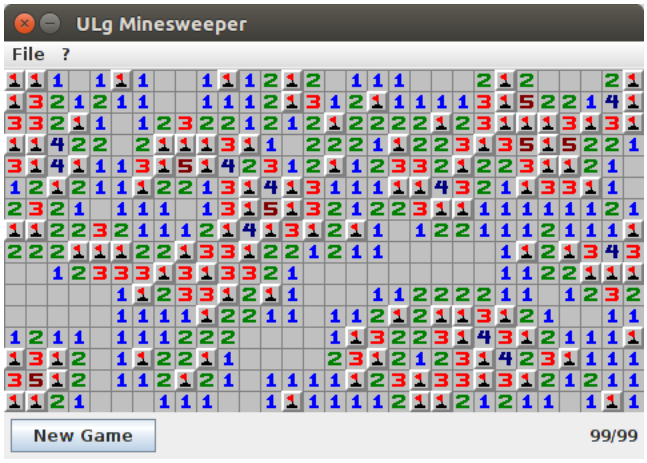


Figure 1: Example of a completed *Minesweeper* game.

Goal, rules and features At the start of a Minesweeper game, all tiles are hidden and can be clicked with the mouse. Left-clicking on a tile reveals its content. There are three types of content:

- empty tile.
- numbered tile.
- mine tile.

The goal of the game is to reveal the content of all tiles except those that contain a mine. Left-clicking on a mine tile will result in the player losing the game. Reciprocally, a game is won when the player has left-clicked on all tiles that do not contain a mine. To achieve this goal, the player has to take advantage of the following game mechanisms:

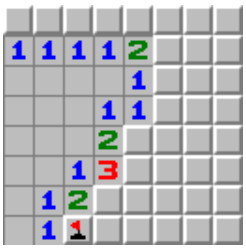
- A numbered tile indicates how many mines are located in the 8 tiles surrounding it (or 5 if the tile is at the border of the grid, and 3 in a corner). For instance, a tile labeled by 1 hints at the presence of a single mine within the surrounding tiles.
- Right-clicking on a hidden tile places a flag on it. Flagging is a way of keeping track of tiles suspected of containing a mine. Right-clicking again on a such a tile will remove the flag. However, the player can only place as many flags as there are mines in the grid.

- Upon clicking on an empty tile, all neighboring tiles that are empty as well are automatically revealed, along with all numbered tiles that border the corresponding empty area.

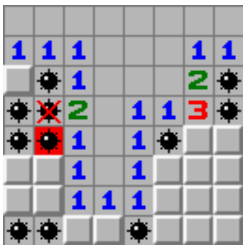
Typical implementations of the Minesweeper game also provide several levels of difficulty. A level of difficulty is characterized by the dimensions of the grid and a specified number of mines. In our simplified version of the game, we will consider three levels of difficulty:

- *easy*: 8×8 grid with 10 mines,
- *medium*: 16×16 grid with 40 mines,
- *hard*: 30 (width) $\times$ 16 (height) grid with 99 mines.

Figure 2a shows a partially revealed grid shortly after starting a new game (easy mode). One flag has been put by the player on a hidden tile, since the label of its left neighbor indicates the presence of a mine.



(a) An easy game, with a first flagged hidden tile.



(b) A lost easy game. All mines are shown.

Figure 2: Examples of games

Upon failing a game, a Minesweeper always reveals the location of all mines, as shown in Figure 2b (easy mode). The mine that caused the player to lose is highlighted, as well as incorrectly flagged tiles.

Statement The project consists in developing a Java program for playing a simplified version of Minesweeper through a graphical user interface *(GUI)*. Unlike regular implementations of Minesweeper, you are not required to create random grids (we however leave this feature as a bonus problem), but should instead initialize the grid at the beginning of the game from the contents of a configuration file (see next section). The program display at all times the total amount of mines along with the total amount of flags used by the player. It should also write an error on the standard error output stream when it is unable to load the configuration file or when this file is not properly formatted.

The GUI will be handled by a library that will be made available to you (in other words, you do not have to program it by yourself). This library contains mechanisms for signaling that the player performs game actions, to which your program should respond in the appropriate way. For instance, when the player left-clicks on a hidden tile, the program receives a notification, which should then trigger operations such as checking if the tile was hidden, if the tile is safe (i.e., it does not contain a mine), revealing its content (and the one of the neighboring safe tiles if necessary), refreshing the GUI, etc.

Configuration file format A configuration file should start with the level of difficulty, written on a single line (either "`easy`", "`medium`", or "`hard`"). From the second line and up to the end of the file, each line should give the coordinates, written as "`x,y`", of a single mine. The coordinate system is illustrated in Figure 3 and defined as follows: `x` is a horizontal coordinate, `y` is a vertical coordinate, and "`0,0`" corresponds to the tile at the top-left corner. Note that your program must check whether an input configuration file is incorrectly formatted, and throw exceptions accordingly.

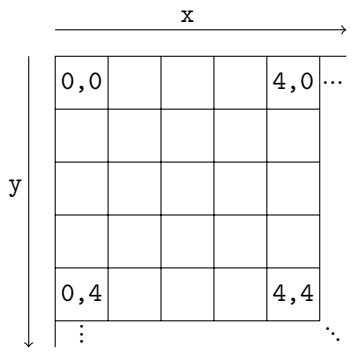


Figure 3: Illustration of the coordinate system

For example, the grid shown in Figure 2b can be loaded with the following configuration file:

```
easy
1,2
7,2
0,3
7,3
0,4
1,4
5,4
0,7
1,7
4,7
```

On the exercises website[1], some example configuration files will be provided in order to test your solution. You should, at least, make sure that your program is able to handle correctly these files before submitting your project. You can also, of course, easily create your own files to test your program in depth.

GUI library The GUI library will be provided as an archive file `graphics.jar`[2]. The classes contained in this file will only be given as bytecode (in other words, their source code will not be available). The documentation of this library is given in the appendix.

Mouse events In order to manage the game, your program has to react to the following events:

- The player left-clicks on a tile.
- The player right-clicks on a tile.
- The player request a new game with the current level of difficulty (if the bonus is attempted).

Program organization Each class of your program should belong to the `be.ac.ulg.montefiore.oop` package. You are allowed to create new packages within this one, provided that they do not conflict with the `be.ac.ulg.montefiore.oop.graphics` package used by the GUI library. Finally, your program must implement the `main()` method in a public class called `Minesweeper`.

[1] `http://www.montefiore.ulg.ac.be/~blaugraud`
[2] The technical details of this file format will be explained in an upcoming exercise session.

Program execution You program should expect to receive two runtime arguments, returning an error otherwise. In a typical execution (i.e., with an input configuration file), these arguments will respectively contain the string "*load*", and a path to a configuration file. For example:

```
$ java Minesweeper load my_grid.txt
```

If you attempt the bonus (for one extra point!), your program should also be able to receive "*random*" as first argument, and a level of difficulty as second argument. It should then generate a random grid at the specified level of difficulty.

Evaluation criteria The evaluation of the project will take into account the fact that the program compiles successfully and is functionally correct, the appropriate usage of object-oriented mechanisms, and the structure and the readability of the source code. Your program will also be tested with erroneous scenarios to check the correct usage of the exceptions mechanism. (In the case of any error, the program should display an appropriate error message.) To avoid bad surprises, you are strongly recommended to check that your program compiles and works properly on the university computers *("Network 8")*.

Submission rules Your program must be submitted for May 8th at the latest using a **ZIP** archive named:

```
oop_lastname_firstname.zip
```

where `lastname` and `firstname` have been respectively replaced by your last and first names. This archive must be sent to `oop@montefiore.ulg.ac.be` in an email message with the subject:

```
OOP - 2017 - lastname firstname
```
.

At the root of the archive, you must place:

- An Apache Ant file `build.xml` for building the project.
- An empty `bin` folder in which the resulting bytecode should appear after a build.
- A `src` folder containing the source code of your program.
- The provided `graphics.jar` file.

Note Developing your own GUI instead of the provided library, or using any classes provided by other graphical libraries (such as Swing, AWT, JavaFX, . . .) is **forbidden**!

Appendix to the Project Statement

The GUI library is provided on the exercises website, as a `graphics.jar` file to be included into your project. This GUI is able to render all the graphical elements of the game in a window, to handle mouse events, and to detect when the player closes the window.

Contents of the library The GUI library contains the following items, in addition to exception classes and internal components:

- `MinesweeperEventsHandler:` An interface that must be implemented in order to react to player actions such as left or right clicking on a tile.
- `MinesweeperBonusEventsHandler:` An interface, subclassing `MinesweeperEventsHandler`, that must be implemented in order to react to additional player actions. The interface is needed for implementing the bonus feature. Note that your projet must contain a class implementing either the interface `MinesweeperEventsHandler`, or `MinesweeperBonusEventsHandler`, but not both.
- `MinesweeperSwingView:` A class that represents a graphical view of the game. It must be instantiated once by your program, passing to its constructor a level of difficulty, and a reference to an object whose class implements either `MinesweeperEventsHandler`, or `MinesweeperBonusEventsHandler`. This object will then automatically receive messages signaling that the player performs game actions.
- `MinesweeperView:` An interface that must be employed for accessing the instance of `MinesweeperSwingView`, in order to update the view displayed in the game window.

The rest of the appendix briefly documents these items. Note that a comprehensive documentation can be found in the `.java` files available in the `graphics.jar` file, or on the exercises website.

Interface constants The interface `MinesweeperView` defines *difficulty constants* that must be used for referring to the different levels of difficulty:

```
int EASY;           // Easy level of difficulty
int MEDIUM;         // Medium level of difficulty
int HARD;           // Hard level of difficulty
```

tiles constant for referring to the types of tiles which can be displayed on screen:

```
int TILE_HIDDEN;    // Hidden tile
int TILE_1;         // Tile surrounded by 1 mine
...
int TILE_8;         // Tile surrounded by 8 mines
int TILE_EMPTY;     // Unhidden tile with no mine
int TILE_FLAG;      // Flagged hidden tile
int TILE_EXPLODED;  // Mine exploded after a left-click
int TILE_INCORRECT; // Incorrectly flagged tile
int TILE_MINE;      // Unhidden tile with a mine
```

and other constants for specifying other useful information:

```
int EASY_HEIGHT;     // The height of the grid in easy mode
int EASY_WIDTH;      // The width of the grid in easy mode
int EASY_MINES;      // The number of mines to find in easy mode
...
int HARD_MINES;      // The number of mines to find in hard mode
```

Note: These constants must be used in every part of your program needing this information.

Documentation of the `MinesweeperEventsHandler` interface

- ```
  void leftClickTile(final int x, final int y);
  ```

  Signals that the player has left-clicked on a tile at coordinates (`x`,`y`).

- ```
  void rightClickTile(final int x, final int y);
  ```

  Signals that the player has right-clicked on a tile at coordinates (`x`,`y`).

- ```
  String getStudentName();
  ```

  When invoked, this method should return the name of the student implementing the project. This name will be credited in the "About" box.

Exceptions These methods do not throw checked exceptions.

Documentation of the `MinesweeperBonusEventsHandler` interface

It subclasses `MinesweeperEventsHandler`. The following method must additionally be implemented:

- ```
  void newGame();
  ```

  Signals that the player has started a new game with the current level of difficulty.

Exceptions The method above does not throw checked exceptions.

Documentation of the `MinesweeperView` interface

- ```
  void updateGrid(final int[][] grid)
    throws NullArrayException, BadHeightException, BadWidthException,
           BadTileConstantException;
  ```

  Updates the graphical state of the grid. The parameter `grid` contains a two-dimensional array with the same size as the grid. Each cell of this array must be filled with a tile constant provided by this interface.

- ```
  void updateFlagsNumber(final int number);
  ```

  Updates the number of flags currently placed by the player on the grid.

- ```
  void win();
  ```

  Announces to the player that the game has been won. After invoking this method, mouse events are ignored until the player starts a new game (if the bonus is implemented).

- ```
  void lose();
  ```

  Announces to the player that the game has been lost. After invoking this method, mouse events are ignored until the player starts a new game (if the bonus is implemented).

- ```
  void refreshWindow();
  ```

  Updates the graphical representation of the game according to all the modifications that have been performed since the last update. This method must be invoked in order for modifications to become visible to the player.

Implementation of this interface The class `MinesweeperSwingView`, provided with the library, implements the `MinesweeperView` interface. However, the messages sent to a `MinesweeperSwingView` instance must always be sent through a `MinesweeperView` reference. The constructor of the `MinesweeperSwingView` class is presented as follows:

- ```
  public MinesweeperSwingView(int difficulty, MinesweeperEventsHandler handler)
    throws BadDifficultyException, NullHandlerException
  ```

  The `difficulty` parameter must be a difficulty constant, and the `handler` parameter must be a reference towards an instance of a class implementing either the `MinesweeperEventsHandler`, or `MinesweeperBonusEventsHandler` interface. A grid of appropriate dimensions will be created according to the given difficulty.

Exceptions The methods above and the constructor of the `MinesweeperSwingView` class can throw the following exceptions:

Exception	**Context**
`BadDifficultyException`	The given difficulty does not come from a difficulty constant.
`BadHeightException`	The given array has a height different than the one of the grid.
`BadTileConstantException`	A cell in the given array does not contain a valid tile constant.
`BadWidthException`	The given array has a width different than the one of the grid.
`NullArrayException`	The given array is `null`.
`NullHandlerException`	The given events handler is `null`.