Matrix Explorer - Bertin-Style Matrix Permutations in Java

Okan Erat, Jakob Schweighofer

706.057 Information Visualisation SS 2016
Graz University of Technology

03 Jun 2016

Abstract

This project report aims to give an overview about the restoration and improvement of an already existing Java application called "Matrix Explorer". The Matrix Explorer is a software implementation of Bertin's Reorderable Matrix, a mechanical device that was built to visualise data and to interact with it. The authors of the report will explain how and why the application was modified.

Contents

Contents **i**

1 Introduction **1**

1.1 Bertin's Reorderable Matrix . . . 1

1.2 The Initial State of the Matrix Explorer . . . 1

1.3 The Task . . . 1

2 Project Report **3**

2.1 Conversion to Maven Project . . . 3

2.2 JDK Update . . . 3

2.3 Scalable UI . . . 3

2.3.1 Minimum Width and Height Principle . . . 4

2.4 CSV Support and File Memory . . . 4

2.5 Various Improvements and General Bug Fixes . . . 4

2.5.1 Row Movement Bug . . . 5

2.5.2 Performance Improvement . . . 5

Chapter 1

Introduction

1.1 Bertin's Reorderable Matrix

The innovative mechanical device was built by Jaques Bertin, a french cartographer, in the mid 1960s to analyse tabular data. It was possible to take out and swap rows and columns of his matrix and reposition them. This interaction method enhanced the possibility for knowledge gain from the data, as it was easier to see clusters or patterns in the data. He used distinct building blocks with different looks for the cells of his matrix, so that he was able to encode information in them. The same principle is used by the Matrix Explorer.

1.2 The Initial State of the Matrix Explorer

The Matrix Explorer was made as Java desktop application back in 2010 by students of the same lecture as this project emerged from. Since then it had several problems with newer hard- and software, that made it impractical and unusable. One of the goals of the project was to fix the game breaking problems and to improve the usability of the application.

Problems of the Matrix Explorer:

- Could not move Rows farther than one Index
- User Interface and Fonts very small on High Resolution Displays
- Poor Performance when moving Colums or Rows
- Could not remember the last Location of loaded Files
- Used custom File Format, which must be constructed before use

1.3 The Task

The project task was to make the Matrix Explorer usable again and pay special attention to the sclability of the application so that it also works on new high resolution displays. It should have a slider to give the user an option for adapting the scaling of the fonts and the content to his or her needs. The Matrix Explorer was designed and implemented a couple of years ago around the year 2010 and therefore did not utilise the newer versions of the JDK. Furthermore it did not use any build system and had to be compiled either by an IDE or with Java commands. One of the explicit tasks was to make a Maven project out of the application.

List of Major Tasks:

- Convert to Maven Project
- Update to latest JDK
- Make UI scalable
- Support CSV Format
- Fix Bugs
- Various Visual Improvements

Chapter 2

Project Report

2.1 Conversion to Maven Project

The focus at the beginning was to establish a solid base that can be used for subversion software, thus the decision was made to start with the conversion to a Maven project, so that later on no changes on the folder structure have to be made and it is easier to collaborate on the code. To convert an existing Java project to a Maven project, one has to remove all JAR dependencies from the existing project. After this is done, the project will obviously not compile anymore so it is, as always, a good choice to make a backup before the modification. When tried to compile the application after the removal of the JARs, a lot of different error messages will be shown. These can be used to carefully look up the correct packages with respect to their versions from the Maven Repository. The new packages from the repository must be added to the central element of every Maven project: the pom.xml file. This file holds the whole build information and defines the dependencies. If a dependency is added correctly with the so called "artifactId" from the Maven Repository, the packages will automatically be downloaded when building the project with for example the command "mvn clean install".

2.2 JDK Update

The update to the latest JDK, which at this time is JDK 8u91, was uneventful and only a couple of adaptations had to be made. For example it was not allowed anymore to add an override annotation to interface methods. After downloading and installing the JDK and after the removal of the override annotation of all interface method implementations, the Matrix Explorer compiled fine and the port to the new JDK was complete.

2.3 Scalable UI

Involved Classes:

- MatrixExplorerView.java
- ui.MatrixTable.java
- ui.MatrixTableHeaderRenderer.java
- ui.MatrixTableFirstColumnRenderer.java (added)
- ui.MainComponentListener.java (added)

The class MatrixExplorerView.java was the central element in all ui-related modifications, as it holds the view information and structure. For the UI adaptations the Eclipse WindowBuilder plugin was used. With this way it was easier to understand the layout and as there were sections in the code that stated that they are auto generated code that should not be modified by hand, it seemed to be the best choice to go with. First, a new popup menu was added to the top menu bar. It contains a slider which defines a scaling value in percent, ranging from 100 to 200 percent. The slider was added with the visual editor and the functionality was then bound with an added listener. When the scaling slider value is modified the listener registers that event and calls a scaling function. The problem with the scaling of the complete window was that each element must be scaled differently. There is a main loop that loops trough all menu elements, fetches their font objects if present and specify a new font size that is multiplied with the scaling slider value. The content section is scaled by one function: doLayout(), which is called by several triggers.

One of the said triggers it the MainComponentListener class, which implements a listener for window resize events. If such an event is fired, the listener then registers that and calls the doLayout() function again.

2.3.1 Minimum Width and Height Principle

For the scaling of the table cells and their content the following decisions were made:

- The Table always resizes to fill the whole content area on startup and on every window resize event
- Every cell has a defined Minimum Width
- Every cell has a defined Minimum Height
- If one of these Constraints would be violated by the Resizing of the Application Window - Do not make Cells smaller and show Scrollbars instead
- Cells grow with their Content

This behaviour gives the user control over the size of the cells as well as their content, by simply changing the window size and allows to see larger data sets than before. Furthermore it guarantees that the user is always able to see the content of the table, unlike before where they would shrink to unreadability when a dataset with many columns was loaded.

2.4 CSV Support and File Memory

Before this modification was implemented, the Matrix Explorer was only able to read and write files that must be constructed specially for the Matrix Explorer, which is time consuming. In the the current Version of the Matrix Explorer it is possible to load standard CSV files as well as modifying and saving them. The first row of the file is automatically interpreted as head line and is expected to contain the column names as strings. Following this principle, the first column of the file is expected to be another set of strings to be used as row labels. After a file is loaded the built in Java preferences are used to persistently remember where the user loaded the file from and on the next usage of the Matrix Explorer the folder browser is automatically navigated to the remembered location. This is a small enhancement but makes a big difference in usability.

Modified Classes:

- MatrixExplorerView.java
- data.DataSetReader.java
- data.DataSetWriter.java

2.5 Various Improvements and General Bug Fixes

One of the tiniest changes was the production of labels for the header row in the MatrixTableHeaderRenderer, which enabled the possibility to center the header text.

2.5.1 Row Movement Bug

As mentioned in 1.2 the Matrix Explorer had some problems that made it unusable. One of the first things that were fixed, was the bug where it was not possible to freely move rows. After the row was moved by one index, an unhandled Index-Out-Of-Bounds Exception was thrown because the calculation of the new index of the row did not take the first column containing the row labels into account and was off by one.

2.5.2 Performance Improvement

In the MatrixTable class every cell gets a renderer assigned. Depening on the selected display mode a different renderer is used. In this way it is possible to render the data as circles or numbers and so on. Before the modification every cell had its very own renderer object, meaning that if a lot of cells are used, a lot of objects are instantiated, all doing the same. The project members decided to use only one renderer object for all cells of the same type, which is held and distributed by the MatrixTable class. This change made a noticeable difference in the fluidness of the row and cell movement.