Gredit

Developer Manual

Group 10

Mohammad Alkhufash, Simon Rafeck, Alexander Adrio

July 9, 2018

Contents

Introduction

Since this project has been developed in Java certain Java specific vocabulary has been used to ensure a clean communication. Overall the work-process of Gredit can be broken down into three different parts. * Starting Gredit, resuming an old project and the creation of a new project * Further explained in Starting Gredit * Editing the opened project * Further explained in Editing the Project * Saving the work-progress and exporting the finished editing as a PNG-file * Further explained in [Saving and Exporting](#saving and exporting)

A detailed explanation of how to use Gredit can be found in the User Manual or a quick reference in the Readme

Starting Gredit

The starting process of Gredit is handled via the Main-class (as per usual in Java). First necessary libraries are load and directories created if not already existing. Afterwards an instance of the DataManager-class is created either resuming the last opened project or if no such project exists or the save-file is unusable an empty instance is created. Lastly the GUI is loaded and the user can start editing.

Config

In order to resume the last project or to load any saved project a configuration file formatted as a YAML-file is needed. This file is represented by the Config class which holds all information about the notes,lines and keys as well as the path to the image it has been created from and a checksum to verify that the file has not been altered manually. Its most notable methods are listed below: * `init()` This method is used to properly initialize the Config-File read in via the DataManager and ensures that the checksum is correct. * `saveAll()` This methods saves the current state of all opened tabs in a designated Yaml-File at the specified path and makes use of the DataManager to convert the elements. * `save()` This methods saves the current state of the selected tab in a designated Yaml-File at the specified path and makes use of the DataManager to convert the elements. * `export(File file)` Exports the current state of the selected tab as a PNG-Picture

Furthermore the Config makes use of utility classes which are used to convert to the YAML-Format and visa versa: * `Note` Represents a note in the stave and has its own coordinate. * `Line` Represents a line in the stave and has its own coordinates and width. * `Key` Represents a key in the stave and has its own coordinate and type. * `Stave`Represents a stave consisting of notes,lines, keys and a path to an image

YAML File

In general a YAML file to load a project consists of a list of staves, each of which is made up of a list of notes, a list of lines, a list of keys and a path to an image. Each file also has its own checksum to prohibit unwanted editing.

```
checksum:
staves:
- keys: []
  lines: []
  notes: []
  path: []
- keys: []
  lines: []
  notes: []
  path: []
...
```

Opening a new Project

A new project can be created by using the Open dialog from the Files menu and choosing a picture to be opened. Doing this will create a new Config File via utilizing the OMR library. This library will scan the image and provide results which are converted to the Config format by the DataManager.

Editing the Project

After starting Gredit and initializing the DataManager with the project that the user wishes to edit, the Config File is converted to graphical elements which are loaded into the GUI. This conversion and presentation is handled by the Controller.

DataManager

This class is used to manage conversion processes whilst also providing access to all data currently used by Gredit. It acts as a pseudo Singleton to ensure that all necessary information can be accessed in a timely and efficient manner.

Important methods:

- `save()` Saves the current state of the given tab. The tab is given to the method by Lists of the elements the tab contains. These elements are produced by the Config class and given to this method respectively.
- `convertToConfig()` This method converts an element represented in the GUI into a Config element by extracting type and positioning information related to the image used.
- `buildConfigFromResult()` The result of the scan of an image produced by the OMR library is converted into Config elements by this method. To this end the positioning information produced by OMR are adapted to the GUI information.
- `loadConfigfromPath()` This method is used to load a Config from a specified path. However also image files are loaded in this manner. To achieve this the file format is being looked at and a decision is made. If the file is an image it is scanned via OMR, if it is a YAML Config it is converted into a Java Config Object.

Controller

The Controller is used to handle all things GUI related. This furthermore includes ensuring that all actions taken in the GUI are transmitted to DataManager and all other classes in need of the information. While its main purpose is handling the users actions and editing-needs it is also tasked with providing crucial information to other classes in a timely fashion.

Notable methods:

- `initialize()` This method initializes the GUI functions. This includes setting up all listeners on GUI elements like buttons, textfields and menus. Also it is used to give vital initializing information to other classes while also gathering information needed for the start up from other classes.
- `createTabFromConfig()` This method is used to translate the information represented in the Config into one tab for each stave. In order to do so, it iterates through each stave and creates nodes and images accordingly.
- `createNode()` All GUI elements used to represent the music sheet are created with this method.
- `createTranscriptionText()` This method is used to compute the String representation of the nodes currently present in the selected tab, which is then presented in the designated textfield and utilized for the search functionality. To achieve this the Transcription algorithm is used.

Transcription

TO ensure an easy-to-use process a transcription of all notes in the current tab is provided to make the stave easier to read. This will convert the position of each note within the stave into a String representation of the notes names, e.g. 'C D E E'.

Short Overview:

- `CoreUtils` This class is used to compute the name of a note depending on its position within the stave.
  - `computeNotePosition()` This method determines the position of a note relativ to the lines. It also makes sure whether a note is between or on lines.
  - `correctNotePosition()` The new coordinate for a note is computed and the given Note instance is moved to this position. Also the NoteName of this instance is determined.
- `NoteName` This enum represents the actual name of a note, e.g. 'C', 'D', etc.
  - `getNoteName()` This method is required for the search function. It tries to translate a string representation of a note value to an equivalent NoteName instance.

Saving and Exporting

The progress of the current project can either be saved in a YAML-file or exported as an PNG-picture.

Saving

When saving the current state of the project a path is chosen via a FileChooser-Dialog and given to the DataManager. There the elements of the currently selected tab are given to the Config as a list of Staves and written to the YAML-file specified with the chosen path. In order to be used in the Config however, the elements of the tab have to be converted into Config elements to ensure a working YAML-format. This conversion is handled by the DataManager and is a simple extraction of positioning and type information.

Exporting

When exporting the currently selected tab a PNG-picture is created with the JavaFx snapshot-method and saved at the path specified in the FileChooser-Dialog.

Support

For more support contact us:


```
Mohammad Alkhufash: mohammad.alkhufash@stud-mail.uni-wuerzburg.de
```

```
Simon Raffeck: simon.raffeck@stud-mail.uni-wuerzburg.de
```

```
Alexander Adrio: alexander.adrio@stud-mail.uni-wuerzburg.de
```