Requirements and Analysis Document for Melody Generator (Group 19)

Version: 2.0
Date: 2014-05-22
Authors: David Oskarsson, Kristofer Yffén, Emma Nyborg

This version overrides all previous versions.

Contents

1 Introduction

This project will create a simple melody generator where the user can choose between a few settings and generate a melody, which can be saved to the computer and played later.

1.1 Purpose of application

The project aims to create an application that lets the user generate new melodies and save the files in a music-format. There will be no difference for users with or without prior knowledge in music theory.

1.2 General characteristics of application

The application will be a desktop, standalone (non-networked) application with a graphical user interface for the Windows/Mac/Linux platforms. Also, the GUI logic will be easy to follow.

1.3 Scope of application

We turn to the user that do not know how or have the time to make their own music, or just want to have some fun. It will be possible to add different filters to the generated tunes to alter the outcome. The program will have the possibility of extending the filter feature with adding new ones. There is no filehandling within the application.

1.4 Objectives and success criteria of the project

The project is considered successful when it is possible to generate, filter the tune after personal specifications and save melodies.

1.5 Definitions, acronyms and abbreviations

GUI = Graphical User Interface
MIDI = Musical Instrument Digital Interface
ABC = A music coding language that can be parsed to MIDI
MVC = Model-View-Controller

2 Requirements

In this section we specify all requirements for our project.

2.1 Functional requirements

Click *Generate* to generate a tune
Change title/genre/signature/key/length/(Major/Minor) for specific parameters
Click *To player* to open the tune player
Click *Save song* to save the melody locally
Click *Play* to play the tune now
Click *Stop* to stop playing
Add Regexp to set a new filter

2.2 Non-functional requirements

NA

2.2.1 Usability

The application will be easy enough for any user.

2.2.2 Supportability

The program will be maintained for three years after product release. This can be extended with a new agreement with the developers.

2.2.3 Implementation

Melody Maker is implemented in a way that makes it easy for both developers and users to follow and expand. Only very clear and usuable functions has been implemented which removes unnecessary complicated functions that can confuse the user. There are also ways for more advanced users to specify extra details to the tune.

2.2.4 Packaging and installation

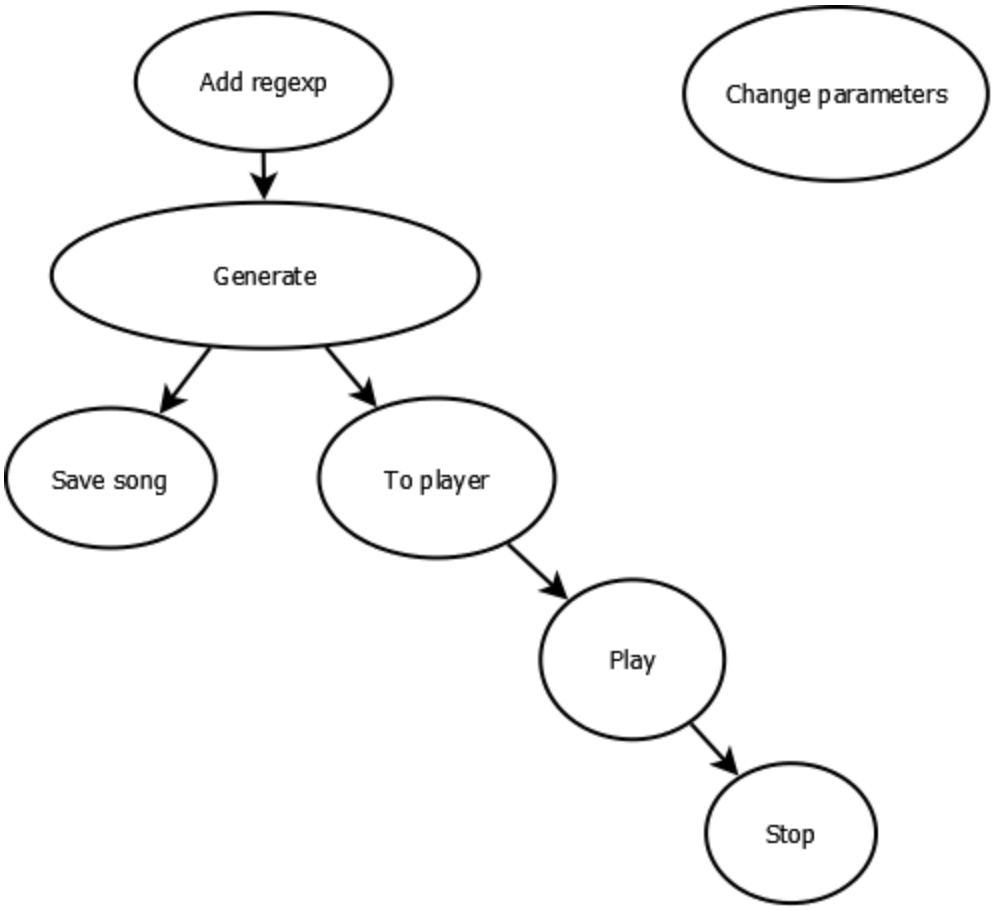
There is no need to install anything with this program. Just start the program.

2.2.5 Legal

The ABC-files that comes with the program are all anonymous work, meaning that any MIDI-file generated from these can be used anywhere. However, Melody Maker offers the possibility of adding your own ABC-notation files. If these files are derived from original work the generated MIDI-files is considered to be derivative work; as defined under 17 U.S.C. § 101. The law about derivative work may differ from country to country. The creators of Melody Maker do not take responsibility for any copyright infringement followed by a derivative work.

2.3 Application models

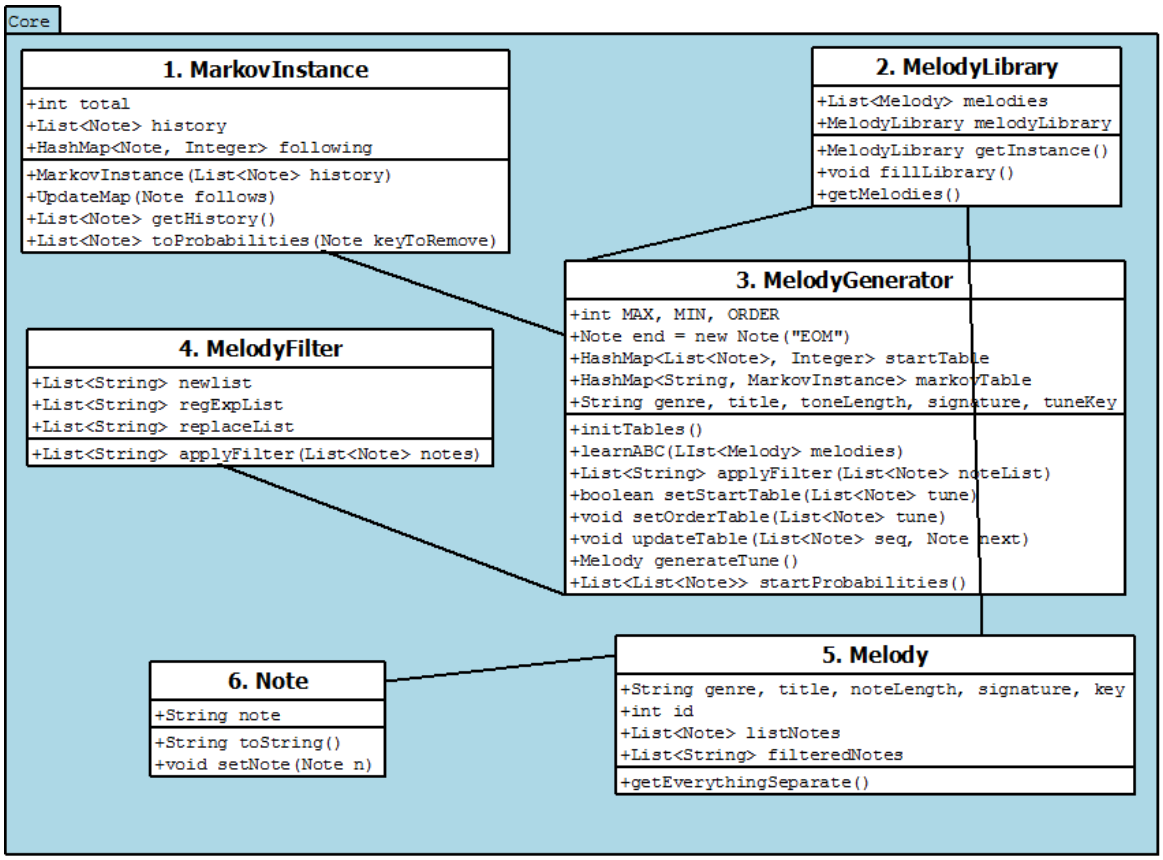
2.3.1 Use case model



2.3.2 Use cases priority

1. *Generate*
2. Change parameters
3. *Add Regexp*
4. *To player*
5. *Play*
6. *Stop*
7. *Save song*

2.3.3 Domain model



1.
* The probability tables in MelodyGenerator is filled with MarkovInstances. An
* object of this class stores information of for a sequence of notes and the
* probabilities for other notes to follow this sequence.

2.
Holds all Melody-objects

3.
* MelodyGenerator can generates new tunes with Markov chains. The probabilities
* is based on the note succession in the melodies in MelodyLibrary.

4.
* A class that makes you generate your tune after your own specifications.
* You can apply this filter on the generated tune

5.
An object of this class stores all information we need for a melody

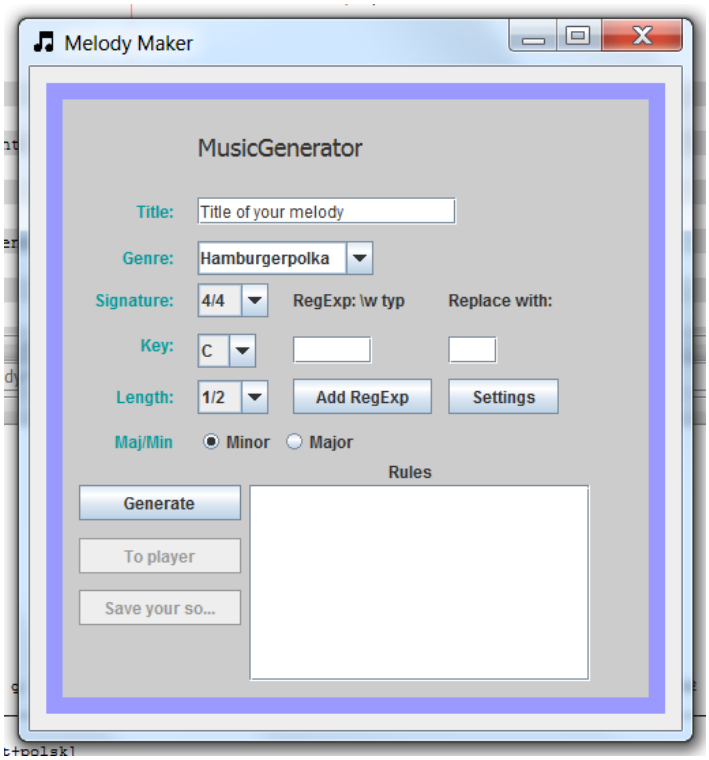
6.
* A simple wrapper class that holds a note, represented as a String of various
* length

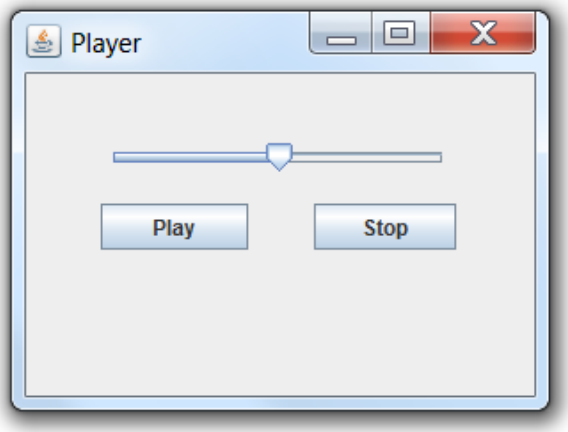
2.4 References

Copyright Act, http://www.law.cornell.edu/uscode/text/17/101

APPENDIX I

Screenshots of the GUI





USE CASES

Use Case: Change title/genre/signature/key/length/(Major/Minor) for specific parameters
Summary: User shall be able to set the parameters to create a song.
Priority: Medium
Participators: User, System

1	**Picks an alternative**	
2		**Puts the options into system**
3		**Waiting for generation**

Use Case: Click To player to open the tune player
Summary: A tune player window will be opened when clicking a button “To player”.
Priority: Medium
Participators: User, System

1	**Clicks on button**	
2		**Button Listener**
3		**Window opens**

Use Case: Click Generate to generate a tune
Summary: The tune with the predefined settings will be generated and stored in the local active database.
Priority: High
Participators: User, System

1	**Clicks on generate button**	
2		**Read input data & settings**
3		**Use markov chain on data and output string**
4		**Use regexp on string**
5		**Store the generated song**

Use Case: Click Save song to save the melody locally
Summary: User will be able to save a song locally on the computer.
Priority: Medium
Participators: User, System

1	**Clicks the save button**	
2		**Saves the generated song locally on the computer**
3		**Outputs a .txt file with data**

Use Case: Click Play to play the tune now
Summary: User will be able to play the generated tune to hear how it sounds.
Priority: Medium
Participators: User, System

1	**Click the play button**	
2		**Parse the generated tune into midi**
3		**Use a midi player that presents the data in shape of a tune**

Use Case: Click Stop to stop playing
Summary: A song that is already playing will be stopped when the user clicks this button.
Priority: Medium
Participators: User, System

1	**Click the stop button**	
2		**Stops the current tune that is playing.**

Use Case: Add Regexp to set a new filter
Summary: User will be able to add regular expressions to manipulate the generated result.
Priority: Medium
Participators: User, System

1	**Write a regexp**	
2	**Write a “replace with”**	
3	**Press the regexp button**	
4		**Regexp is stored in a list for latter use**

Analysis Model