Final Report Contextproject TI2806

Felix van Doorn favandoorn 4299566
Roy Graafmans rgraafmans 4299442
Millen van Osch mjevanosch 4299426
Emiel Rietdijk earietdijk 4205383
Davey Struijk dstruijk 4289080

25 June 2015

Abstract

This application was created in order to offer a service that provides matched play-list to the user. This music is carefully picked from the user's music own music collection. We identified the primary elements that make tracks compatible and implemented these into our sorting algorithms. Eventually we went further than just ordering the tracks, we also identified the points where the Tracks should transition in order to make Tracks sound even better. This enables our application to make its own mixes like a real DJ would. This solution would be a great addition to every user that likes to listen to EDM on their computers at home. In the end we managed to build a decent audio player that creates very decent mixes. At the end of the development cycle we staged a Turing Test with over 200 participants, with promising results. We were surprised by the fact that the participants had such a hard time distinguishing our mixes from those created by actual DJ's. That does not mean we are finished with this application! We believe that this application has a lot of potential and we are highly motivated to keep working on it in the future.

Contents

1 Introduction

To many people, music is an essential part of life. People often have big music libraries and compile their own play-lists. Some people choose their music with care, knowing which tracks sound good when following up on each other, but most people just throw some tracks they like in a play-list. This often leads to a play-list where its potential is unused. Music often stops because of intros/outros and sometimes the mood gets killed because tracks are just too different.

DJs, the better ones at least, do this with their mixes. Songs have to be compatible with each other for them to follow up on each other. This is based on a universal fact all humans share: every tone has a set of tones that must follow up or it will sound flat or just ugly.

Although everyone can decide if they like certain songs or mixes, not a lot of people can create them. People can choose the songs they want to use but actually mixing them together is something the average person can't do or is too lazy to do. Being programmers we thought this could be an area where a computer, with the right software, could vastly outperform the average human. That would give every person who installs this software the experience of having a personal assistant that takes their music and mixes it accordingly so the user just has to relax and listen to a smooth play-list. That is just one of the things that we offer to the user. The others being :

- Intuitive GUI for non-developers
- CLI for developers
- Option to listen to play-list in-App
- Being able to export a constructed play-list to a different media-player.
- Saving play-lists for later use.

We hope to have succeeded in combining this into clean and easy to use application to make EDM mixes like a human DJ would do at a party, club or festival. This document will further contain an overview of the developed and implemented software followed by a reflection on the product. Next it will give a description of developed functionalities and then a section on interaction design. After this is a section on the evaluation of the product and a failure analysis and finally our outlook on the future of our product.

2 Overview

The main feature of our application are the algorithms we have created and implemented. We actually have two sorting algorithms that we can switch between, a greedy algorithm and a Maximum-Flow algorithm.

2.1 Track analyzing

We use different attributes of a song to calculate the compatibility score of a given track. These include the BPM of a song, the Musical Key and the energy levels. Upon loading a track into our program we calculate these properties and set them in the file's ID3 Tag, these properties can then easily be accessed by our program. For the analyzing part we integrated a BPM and Key analyzer into our project, so that a user could input any .mp3 file and our program will still obtain the required data for the matching process.

2.2 Play-list

Once one of our algorithms is done running, we have a matched play-list. This play-list can be exported in .M3U format and played in a different media player like iTunes or Windows Media Player but without transitions. The most important feature for our program was the mixing of the play-list itself. The program creates transitions between the tracks so the music never stops because of intros or outros. If users prefer the transitions they can listen to the music in our application. The play-list is added to a Library, that can hold multiple play-lists and are stored in a XML-file. Our playlist keep playing unless they are paused or the application is stopped. Upon reaching the final song, the application will make it transition back to the first song and the playlist will loop. That way your play-list will remain available when reopening our program. To implement the music player of our program, we used the TarsosDSP library to set up the audio and playing the music.

2.3 GUI

For the users that choose to use our application through the GUI, we have a clean, simple and easy to use interface implemented in JavaFX and styled with CSS. Interaction with our GUI is self explanatory. All we need is a small menu-bar and a Play/Pause button. When users first start our application, they faced with a directory explorer that requires them to load their desired music into our application. The music then shows up in our GUI and you can start playing music. Any other actions you would like to take, such as importing a new directory or exporting your play-list can be found in the menu. This is exactly what we wanted for our users, we want the focus to be on the music and that the users can just sit back and enjoy!

2.4 CLI

Since our software is open source, we also thought it was important that there was an option to use our program from the command line. It offers less interaction to the users, you can't pause and resume the play-list from the command line. The usage of the CLI is just intended for developers who want to test our program using the CLI. It does however offer quicker interaction for these users.

3 Reflection

One of the things that we might do differently in a future product is to choose a different library for the audio. In this case TarsosDSP was easy to implement in our project and it offered a lot of good things when it came to analysis of audio files. But when it comes to real-time playback of the song we believe that it would be better if we wrote our own code to handle the audio.

A lower-level language like C is probably far better suited to handle these kind of operations than Java. We did not do this during our project due to the rather limited development time for this project and the fact that we are not experienced C developers. This would have allowed us to play track with a better audio quality than we have at the moment. So if we were to improve our project in the future, we would first spend some time porting the contextproject.audio package to C.

Also we feel as a team that we structured the development process very well. Our architecture was very clear and it was fairly easy to implement changes in our project. Even when the project became larger in size, the structure of our project remained very clear and turned out to be of great help in the end.
On the flip side, as a team we feel we feel we could still improve our GUI. Just adding little details would make the entire application that much slicker and it is a pity that we didn't have time to fully implement all the styling features that we envisioned.

Another thing that we did not do well enough is user-testing throughout the development process. We did do them at the end and the results were good, but it would still have been better had we done this throughout the entire development cycle. This would have given us important feedback, we could have used to build a better product. This loss of potential serves as a valuable lesson for the future.

Coming back on the user tests we did do, the reception of our application exceeded our wildest expectations. We spread the Turing Test we created through Facebook and over 200 people took this test. Most of the participants really liked the idea and had a hard time distinguishing our mixes from those created by an actual DJ. Chapter 5 contains more detailed info about this test.

4 Description of Developed Functionality

There are a few complex algorithms programmed to solve the main functions. For example the audio playing, play-list sorting and track analyzing are parts with complex code. In the paragraphs below each of them will be explained.

4.1 Sorting algorithms

At first there was the greedy play-list algorithm. The greedy algorithm works by assigning compatibility scores between tracks and then going through the list and picking the ones with the highest score. This sorter added the best song pairs first and goes so on. This will lead to a order where in the beginning the compatibility is best but decreases when going further in the play-list.

The maximum-flow algorithm works by reducing the problem of mixing tracks to a multi-source multi-sink problem, where we use the inverse of the compatibility score for the edge weights. After all we want the ones with the highest scores to be part of the optimal path. These algorithms were fully implemented in Java and for the maxflow algorithm we used the JGraphT library. This allows us to quickly and efficiently mix the play-lists and that the waiting time is kept to a minimum. This is a variant of the maximum-flow problem with the addition to find the best flow trying every point as start and finish. The maximum flow problem involves finding a feasible flow through a flow network. For the maximum flow problem itself we use a external library which already includes a build structure for this. On top of this library we extend some classes for more functionality and build a multi-source, multi-sink method around it. When done calculating the best maximum flow the undirected graph will be split into a tree. To limit the computation time and speeding up the algorithm only the best 10 per level will be added. At the end the best path will be path from down to top will be used as the play-list order.

4.2 Track analyzing

Our program analyzes 3 main components of a track. Beats per minute, musical key and energy levels. The BPM and musical key are analyzed by an external jar file as this was not as a library available. This program runs as a sub process when called. The energy levels are detected every 16 beats by calculating the root mean square (RMS) of the float buffer of the audio. Before the energy is actually calculated the onset of the track is first analysed and from that time the energy is being detected. This energy is used for calculating the average energy of a track and the differences of the energy levels between 16 beats. This difference is then used for calculating times to switch between songs. For calculations on the Musical Key we use something from musical theory called the Camelot Wheel. This makes the calculations possible and easily implementable in Java. The Camelot wheel can be found in Appendix A.

4.3 Audio playing

In terms of audio playback, we have built a component that manages two separate track players. Most of the time, one single track is playing. In the background, the next track is then prepared so it can be played instantly when necessary. Whenever the system thinks the new track should be mixed, it starts a nice transition between the songs, and finally the GUI updates to reflect that the new track is playing.

4.4 Exporting Play-list

Exporting works flawless, the play-list that are matched and constructed in our program can be exported as a .M3U format to a different media player. We implemented this by simply creating a class that constructs files in this format and store the tracks in this file in the correct order.
When playing them in iTunes and Windows Media Player this works great. A user would do this, because these music players offer better audio quality than our own player. However this does demand a little bit of extra work and skill from the user and comes at the cost of losing the transitions.

4.5 GUI

The code behind the GUI consists out of multiple parts, three to be exact. There are six controllers containing the code how to set things up and handling interactions with the user like when buttons are pressed. The second part of the GUI are the views. These views are FXML files and contain the code initialising the components of the GUI like the buttons and connecting these to their respective code via ids. In the view you give the components unique id and in the controllers you initialise them with the same name, this way all the attributes are already initialised and saves a lot of coding in the controllers. Finally there are CSS files. These contain CSS code meant to style the GUI like images and and sizes of text.

The GUI itself consist of multiple sections. In the top there is menu bar meant for importing, exporting and deleting play-lists. There is also an about and help button. On the left are all the loaded play-lists, when a play-list is clicked the right, much larger field shows the songs in the play-list. When a song is clicked the music starts playing. On the Bottom of the screen the is a bar we call the status bar. It contains a progress bar and a play/pause button. It also displays the track that is currently playing and the track that will be played next.

4.6 File storage

Our program needs to analyze all the tracks that are loaded for the first time. This is a waste of time when a user has to do this every time the program starts.

Therefor our program writes all the important information of the tracks away to a XML file. On the program start all the information from the XML will be loaded and the play-lists will be restored. The export works when a object in the track class is serializable. Only a getter, setter and a empty constructor have to be implemented. This has the advantage that the export is scalable for many arguments.

5 Interaction Design

For our 'human computer interaction module', we thought it would be very cool how the mixes produced by our program compare to those made by human DJs. Therefore we set up a kind of Turing test where we would let the user listen to audio files and see if the test subjects could distinguish which mixes are made by people and which are made by our program.

In order to do this we took some transitions that occur when a DJ mixes a track and some that were generated by our program. We then let the test subjects listen to these audio samples and asked if it was produced by a human or a computer. We had 10 samples, with 5 human generated transitions and 5 computer generated ones. It was unknown to the test subjects how many of the samples were mixed by a human or by our program and the order of the samples.

The results, which can be found in Appendix B, were quite pleasant. We made a distinction between people who do like EDM and people who don't in the graphs. We witnessed something quite remarkable when analyzing the test results. Our robot was mostly picked to be human and vice versa. It would be too simple to say that that means our bot is just very well made.

One of the problems with our test is that people went into the test knowing that there was a computer in play. Most participants made a choice at that point; either the DJ would be good or the computer. This made them pay attention to the wrong aspects, as not all human-created were of the same quality.
We found a weak correlation in exposure to EDM and ability to distinguish our mixes from those created by human DJ's. With some of the samples, non-EDM listeners actually performed better than EDM-listeners. The problem was that, we let our application mix some tracks and it sounded almost the same as mixes recently released by popular DJ's. In these cases the EDM-listeners were tricked a little bit, as they thought it had to be a human DJ. Eventually the participants were able to identify our bot in less than half the cases. So it turns out even the majority of EDM-listeners thought they were listening to a real DJ when they were actually listening to our application!

From this we can still conclude that we built an application that can mix tracks like a decent DJ. Participants in our user-tests and Turing Test were surprised by the quality of our audio-transitions which is more than enough reason to be content with our application.

6 Evaluation

When looking at our final product, the GUI is as easy to use as we initially envisioned. It does not necessarily look great, but our application is all about the music anyway.

Our analysis tools work great with just about any form of EDM, which is a genre where mixing plays a rather important part. However when exposed to classical music or jazz, our program often comes up with rather odd results. On one hand this is a shame, because we were actually pretty curious what this would sound like if done properly. On the other hand, this is a feature that is not relevant in these music genres so it would be of no use anyway.

We would also look for ways to distinguish vocals from the instruments. A human DJ would never mix to another track when vocals are present, however our program sometimes does. This is a small mistake that makes our program less effective at a human DJ, so in order to outperform them all the time this has to be improved. However this is likely to be very hard for us to do at this point, because of time constraints and lack of experience in the field of multimedia analysis.

We failed to deliver high-quality music to our user. The library we used to play audio, does not really work with stereo and streams to the operating system in mono sound. This gives us audio quality that is actually pretty much below the standard that we envisioned

7 Outlook

In our opinion this product is a nice proof of concept, but far from fulfilling the potential we believe it has. We do have the intention to keep working on this as a team. As mentioned earlier we would port the contextproject.audio to C to enhance the quality of our audio and the manipulation of the signals.

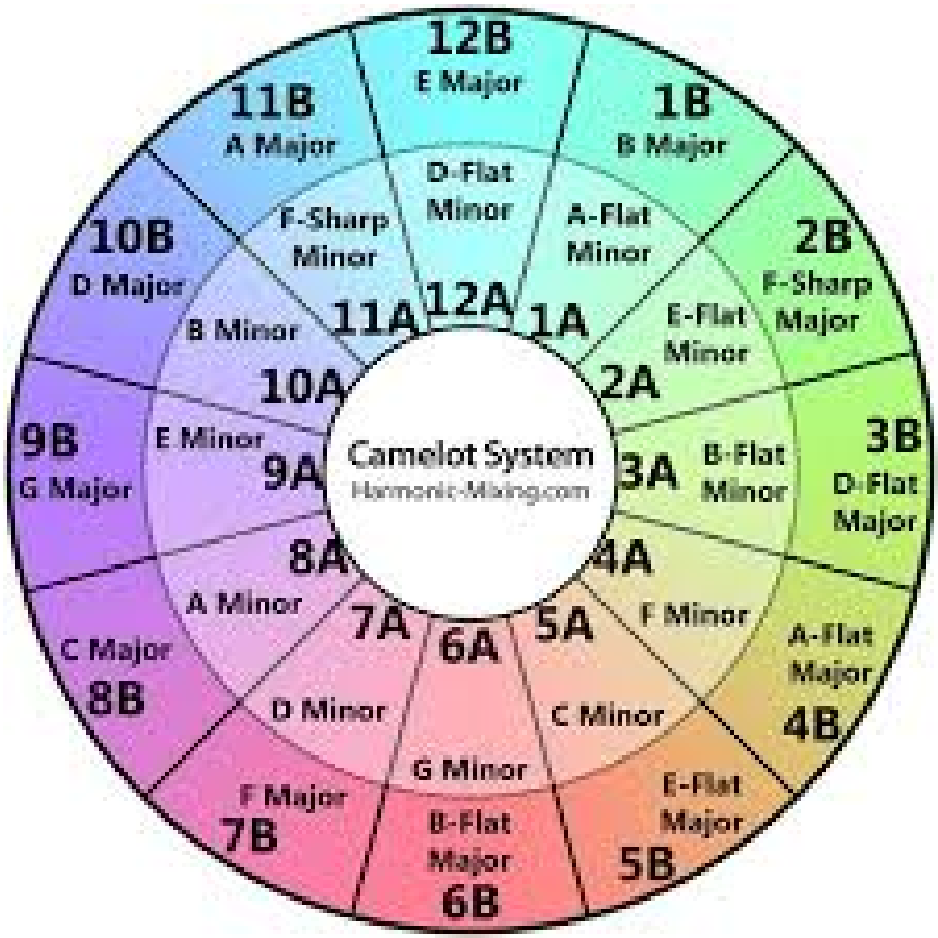
Further we want to improve on a couple of things:

- We want to make the KeyBpmFinder OSX compatible. As of now it only works properly on Windows.
- we want to expand the amount of genres it can handle. Right now it works best with EDM. It can handle most pop songs as well. However genres were these mixing techniques would be of great use, such as Drum & Bass and Techno do not go as well as they should.
- the program has troubles with lyrics, it can't detect them and sometimes transitions in the middle of a sentence.

We would like to make a mobile version as well. As our application is written in Java, we should be able to port it to Android without too much hassle. This would be worth doing, because most of our target audience have huge collection of MP3's on their mobile phone's. These are often used to play music out loud when mounted on a speaker. This is often done at parties or social gatherings to play music, our application could also bring an enhanced musical experience in these scenarios.

Maybe in the distant future when our application has taken off, there was an idea to cooperate with a bigger musical service like spotify. The idea is that you can log into spotify via our program and then load your play-list in our program. if we could pull that one off it could be a big hit for us.

A The Camelot Wheel



B Turing test results