Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

A Novel Music Browsing Framework: The AMBIF Project

Supervisor: Prof. Augusto SARTI
Assistant supervisor: Dr. Massimiliano ZANONI

Master graduation thesis by:
Stefano Cherubin ID. 787061

Academic Year 2013–2014

Nell'ombra vedo te, dolce follia

Musa... indicami la via

Folkstone - «Vortici Scuri»

Acknowledgments

I would like to thank:

Professor Augusto Sarti and Dott. Massimiliano Zanoni. They patiently gave me the possibility to work on this project.

My friend and colleague Michele Buccoli. He introduced me to this research field and supported me in the development of this work.

Ing. Daniele Ciminieri. He was a precious source of knowledge for data visualization and software evaluation research fields.

The rest of the Image and Sound Processing Group. I found there a stimulating place to work with very nice colleagues. Bruno, Dejan, Paolo, Antonio and all other guys of the lab... I still owe you some coffee.

Politecnico Open unix Labs, every current and past members of this association. I learned a lot from them, I had a lot of fun and I found there true friends. I will never forget you Lapo, Otacon22, Venom00, guy#1, guy#2 and all other guys whose name I already forgot.

My parents, my grandparents and my entire family. Without their aid I would not be here.

All my collegues I met in these years of MSc. It was a pleasure to meet you, to study with you and to work with you.

All people, monkeys and penguins who took my test and filled out my surveys.

All my friends I did not mention above, especially those who had the patience to wait me for the time I needed to write this thesis before getting drunk together another time.

Abstract

In recent years, great changes has involved music from a social and a technological point of view. The increase of the capacity of memory devices led an impressive increase of the size of both personal and industrial music repositories. This process brought the size of collections to a level where users get lost in their own music. From here comes the need of new browsing systems to help the user in finding suitable musical content. However, mainstream music browsers did not change that much and still rely only on lists of context-based metadata. This approach has many issues, the most important is that no information about the audio content is available. Music Information Retrieval (MIR) is the research field that deals with the retrieval of useful information from music content. It can also be applied to music browsing to provide a content-based music browsing experience.

The purpose of this thesis is to propose a new framework to build hybrid content-based and context-based music browsers using a multidimensional space.

We developed AMBIF (Advanced Music Browsing Interactive Framework) using Unreal Engine 4.7. It allows users to browse up to five song dimensions at the same time in a mixed content-based and context-based space. Each song in this space is represented by a spherical placeholder and its position and color depend on the song descriptors that the user chose.

We proposed a music browser instance made with AMBIF that relies on content-based high-level music descriptors to 33 users and we evaluated its usability. The feedback was great and results are promising.

Sommario

Negli ultimi anni grandi cambiamenti hanno coinvolto la musica da un punto di vista sociale e tecnologico. L'incremento della capacità di memoria dei dispositivi elettronici ha portato ad un aumento della dimenisone di collezioni musicali private e industriali. Questo processo ha portato le dimensioni delle collezioni musicali ad un livello tale per cui gli utenti si sentono persi nella loro stessa musica. Da qui nasce la necessità di un sistema di navigazione che aiuti gli utenti a trovare contenuto musicale adatto a loro. Tuttavia i più popolari music browser non sono cambiati poi molto e tuttora si basano su liste di metadati basati sul contesto. Questo approccio presenta molti problemi, uno su tutti è che non è disponibile alcuna informazione sul contenuto del file audio. Music Information Retrieval (MIR) è il campo di ricerca che si occupa di estrarre informazioni utili dal contenuto musicale. Può essere anche applicato alla navigazione di una libreria musicale per fornire un'esperienza di navigazione basata sul contenuto musicale.

Lo scopo di questa tesi è proporre un nuovo framework per costruire un sistema di navigazione musicale in uno spazio multidimensionale secondo un approccio ibrido basato su contenuto e contesto.

Abbiamo sviluppato AMBIF (Advanced Music Browsing Interactive Framework) con l'utilizzo di Unreal Engine 4.7. Questo sistema permette di navigare fino a cinque dimensioni di una canzone contemporaneamente in uno spazio basato su contenuto e contesto. Ogni canzone nello spazio è rappresentata da un segnaposto sferico la cui posizione e il cui colore dipendono dai descrittori della canzone che l'utene ha scelto.

Abbiamo proposto un'istanza di AMBIF basata su descrittori di alto livello del contenuto musicale a 33 utenti e ne abbiamo valutato l'usabilità. Il riscontro è stato decisamente positivo e i risultati promettenti.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Digital audio is technology that encode audio signals in digital format. It can be used to reproduce, store, manipulate, record, distribute sound. On the wave of advances in digital technology during the 1970s, digital audio replaced analog audio technology in most areas of sound engineering. Digital audio is based on the fact that all audible-analog-audio signals can be encoded as digital signals without significant variation. The theory behind this idea dates back to the Fourier Transform, first described in 1822 [2], and its derived transform functions. One of those, the Discrete-Time Fourier Transform reduces a continuous signal to a discrete sequence of values (samples) that can be interpolated to rebuild the original signal. This process ensure that for each given signal there exist a digital approximation of the original analog signal and this approximation can be indistinguishable or almost indistinguishable to the human ear when sample frequency is sufficiently high (at least double of the maximum audible frequency).

Comparing digital audio to analog audio it can be said that there are no disadvantages in term of quality of the music, but there are a lot of vantages in term of dematerialization. This is the primary reason why digital replaced analog as main format to edit, transfer and store music. The diffusion of the internet in the 1990s and 2000s acted as main amplifier of the diffusion of the digital audio format. Nowadays the digital audio is almost everywhere: cars, smartphones, personal computers, ornaments, even wearable devices can play digital music. Reaching music content in digital format is very easy.

As Bauman [3] described the concept of liquid society, we can nowadays use the term liquid to describe music. From a sociological point of view, this means that there is not a well-defined shape of what music is because it is in constant change. Music structure is hard to define, recent contamination of music are going to destructure models hitherto known. New music genres and subgenres born every

day: some of them exist only in a few works, others become rapidly popular. Music evolves quickly.

From a technological point of view, recent cloud technologies increased the level of abstraction in content management. Once the concept of music was related to a physical object (e.g. a vinyl disk, a musicassette, a CD-ROM), then it was related to a file stored in a computer or a portable device, now the age of data streams is coming: a song is only a label on the screen and how music collections are organized are no longer a matter of the user. Streaming music from a cloud repository to a personal computer, from a content provider to a laptop or a smartphone, user have no idea about where exactly the audio file is, he just enjoys the music flowing to his/her ears. This is another way to see liquid music: user get the audio, but he no longer knows where it resides.

1.2 About the project

1.2.1 Reasons

For both society and music these are times of changes. Therefore, also the way to listen to, look for and buy music is changed in recent years and it is still changing.

The increase of the capacity of memory devices led an impressive increase of the size of both personal and industrial music repositories. This process brought the size of collections to a level where users get lost in their own music. From here comes the need of new browsing systems to help the user.

A good browsing system must fit different kinds of music it is used with. It should adapt to every known music genre but also to all those genres that will be invented in the future. Music is a constant-evolving art and this is a fact that can't be ignored.

The author of [4] states that music browsing and listening are activities with unique and distinct characteristics that motivate the creation of domain specific software and control interfaces. In a more general way it can be said that every user has its own needs and preferences. A music browser should allow the end user to personalize the software interface to make him more comfortable.

Although personalization is main aspect of browsing every good software must be intuitive and ready to use by every possible user. This means that the default interface has to be simple, clear and per se meaningful.

Therefore, a good music browser have to be:

- **adaptive**
- **personal**
- **intuitive**

1.2.2 Proposed solutions

Science community and the market already dealt with this problem. Their solutions make use of different approaches: lists, metadata, semantic spaces.

The list-based approach is the elder and simplest way to display information about music content. It is easy to implement, it can be used with every genre of music and the average user is already used to browse a list. This approach is no more suitable when content grows in size and the list becomes a wall of text. An average user gets tired very quickly of a wall of text, he doesn't really look at the list and rely on other auxiliary systems (for example a search engine). Here, the problem is that there's no visual match between the text shown on the list and the content of every list item. Furthermore, by looking at this kind of interface there is no way to gain information about the content of the music.

Starting from these system, it is possible to add levels of abstraction. Humans are used to describe music with specific terms. Using a context-based system, which relies only on metadata, it is possible to assign several terms (tags) to every audio file and search for them. In this way it is possible to query a library using defined tags. Tags can improve personalization, but it is still complex to define meaningful relations between tags. There is still no scientific correlation between tags and audio content. It also has high management complexity and does not scale very well. Moreover, this has a cost of $O(N * T)$ where N is the number of audio files and T is the number of desired tags.

The main limitations of context-based systems are partially solved by the content-based systems. It is possible to extract descriptive features directly from the audio content; in this way it is possible to have a visual representation of the audio content without relying on the context. There is no more need of manual tag management, feature extraction can be automated. Content-based features can also be processed to obtain a content-based classification of music, an interesting overview about feature extraction and feature processing can be found in [5]. This totally-different approach exploits semantic spaces to visualize content-based features on a map. Abstraction level is increased: instead of describing music using terms, music is described using concepts. The most popular map is the Valence-Arousal plan (VA plan) where each song can be mapped as a point and each point on the map represents a different emotion. Whereas it is possible to automate content-feature extraction, it is also possible to automate the match between audio files and points. In this way it is possible to provide to the user an immediate feedback about the content.

Unfortunately, content-based approach is not (yet) very popular because it introduces a very-high-level of abstraction, which can confuse the average user. An hybrid content-based and context-based approach was introduced to improve the quality of these systems. Anyway, many mainstream music browser prefer to keep using a

context-based approach because user experience proved they still work better.

1.2.3 Goal

Each one of the previously described approaches has its own point of strength and point of weakness. The goal of this work is to describe a new hybrid system that aims to exploit the advantages of both content-based and context-based systems without their limitations.

In order to create such a hybrid system it is important to understand what a user expects from a music browser. Different people put different level of attention to different aspects of music. Someone primarily focuses his/her own attention on metadata like album title, artist name; someone else focuses on genre, others are more interested in bpm and so on. The range of potential interesting aspects is pretty large[1], to cover all these possible needs is not easy and requires big effort to maintain and a nontrivial visualization system. Though all these metadata are available, it is a best practice to show only a few of them at the same time; the reason lies in preserving clarity of the visualization. As an immediate consequence, it often happens that interesting metadata are not used.

Although existing music browsing systems based on semantic spaces are not widely appreciated, there is interest on the part of users to exploit visual spaces when browsing music[1].

The key of this project is personalization, to provide a framework for leaving the end user creates his/her own personal music browser. This system makes use of a hybrid approach. It provides both content-based and context-based support in a multidimensional space. This project exploits a subset of all available descriptors the user likes more by letting him chose which one to show and how. In this way the semantic-visual space is brought closer to what the user has in his/her own mind.

[1] Stefano Cherubin. Music browsing, raccontami la tua esperienza e le tue impressioni in merito alla navigazione di una raccolta di canzoni. `https://it.surveymonkey.com/results/SM-79SW6M87/`, May 2014.

Chapter 2

State of the art

2.1 State of the Art in Browsing Music Collections

Music browsing is a particular case of data visualization. Data visualization depends on user behavior no less than how much it depends on data. Therefore, the study of such a science is a matter of understanding humans as well as data and to address the music browsing problem it is important to set the focus also on the user. It is stated in [6] that how people perceive and classify music is strongly influenced by social factors such as age, geographical location. Furthermore, several people and companies around the world create different music browsing services that organize music many in different ways because they serve for different purposes and target different people. It is nearly impossible to have an overall view of all parameters that influence music browsing. For this reason, there are not clear metrics to describe performance of each music browser or music browsing technique. The behavior of the user is the key to understand if he/she likes or not a way to visualize data and metadata. Main classifications can be done by usage, target device and browsing purpose. Therefore, we are going to discuss the state of the art in music browsing with respect to these parameters.

There are a few main music browsers widely adopted on the market and a lot of minor and research projects. It is meaningful to mention also research projects because, though they are not (yet) widely adopted, they are interesting because they explore new approaches and/or new interfaces. Therefore, in this chapter we are going to analyze both major music browsers and research projects.

It is also worth to mention that browsing systems differentiate on the device. Indeed, controls and interfaces are different for any device. Browsing a music library from a smartphone has not the same issues as browsing from a personal computer or a tablet. Usually smartphone and tablet browsing applications are considered good enough for their purpose when they implement a subset of the features contained in some other desktop-reference systems. Other different type of interaction are

used with augmented reality (AR) or virtual reality (VR) devices. AR and VR data visualization projects are ad hoc-designed solutions for every use case, artistic installation [7]; out of some design installation [8, 9] this field is not yet widely explored. The [10] goes deeper in detail about the issues data visualization has with VR systems. AR and VR systems represent a possibility for the future of music browsing systems, but, at the moment, their lack of popular devices makes them not ready for a massive use for content browsing. In this thesis will be discussed desktop music browsing systems, since they actually represent the most advanced state of the art in this field.

There is also a difference between browsing a well-known personal music library and browsing an online music store database. In the first case the user is most likely looking for something to listen to in his/her own collection. This means that the music collection to be browsed contains only music the user has chosen to import in his/her own library; it is not a strong assumption to say the user likes all the available tracks in the collection and he/she is just looking for tracks compatible with his/her mood. In the case of online music stores there are two main scenarios: the user may either be interested in identifying a specified artist/album/track he/she would like to purchase or he/she is interested in following suggestions to find new content he/she may like. Nowadays users no longer cares about the size of his/her own music collection and it is possible that he/she may need to discover or rediscover content even inside his/her own music collection. However, content discovery and recommendation is a topic that deserves more than a thesis by its own. Our work can be applied to both systems for browsing a known music library and systems for browsing an unknown music library.

Music browsing systems must adapt to the proper use case they have been inserted into. Benefit and malus of each different approach will be explored later in this chapter. In order to contextualize the analysis, the discussion is grouped by music browser instance.

2.1.1 Major browsers

2.1.1.1 iTunes

iTunes[1] is the most popular software used to organize personal music libraries and synchronize them with Apple portable devices. It also integrates browsing feature with the intent of finding known music to play or to buy. After experiencing different interfaces, this software currently offers a set of layouts through which can be browsed the local music library. A text box is always available to allow querying by keywords on text metadata. Each user can switch between different layout at any time, even when music is playing. Available layouts actually are:

[1] Apple Inc. iTunes. `https://www.apple.com/itunes/`.

- Single list
- Wall of album covers
- List grouped by artist
- List grouped by composer
- List grouped by genre

In each of these layouts are always available the following metadata: Artist, Title, Album Title, year, length, user rating. It is also possible to group tracks by Album Title.

Single list Single list layout reserve all the available browsing area on the screen for a single list of tracks. It allows to display a larger set of metadata than any other layout available for this system. However, this layout is heavily defective: when the number of tracks to list is higher than half screen, it becomes disturbing to the eye and creates confusion to the user. The result is a wall of text. In addition, metadata are based only on context and gives no information about the content.

Wall of Images The wall of album covers is one of the most popular layouts and it can be found also on many other competitors. It is very simple and poor of metadata, but it fits very well on mobile devices. It groups all tracks with the same *Album Title* and in place of all of them show only the album front cover. Users recognize the album cover and can select the album by clicking anywhere on the image. It represents a portable layout, i.e. a layout that can be used on different types of device.

iTunes, like many of its competitors, allows the user to complete the music collection by retrieving missing album covers form a centralized database. Although iTunes cover database is one of the most populated in the world, sometimes it happens that album covers are unavailable; the reasons may vary: album metadata may contain typos, album may not be popular enough to be included in iTunes database, etc. In these situations this layout become almost useless because it fails most of the feedback that the user should receive from the browser.

Grouped Lists Grouped-list layouts leave on the left about one sixth of the size of the screen for the top level list (list of artists, list of composers, list of genres) and the rest of the view is used to display a list of tracks for the selected top level aggregator. This kind of layout allows the user to see a hierarchic organization of his/her own music library. Grouped-list layouts are the most common layouts that can be found in music players and music browsers for personal computer and laptop

devices. Several main competitors[2,3] of this software use this kind of layout by default to browse a personal music library.

Using aggregators may help in reducing the list size and restricting the context, but this only lightens and does not remove the problems of the Single list layout.

Success and problems of this system The success of this browser dates back to early 2000s when the Apple iPod became popular [11]. This music player device requires iTunes as a mandatory software installed on every personal computer used to synchronize music with. Year by year iPod became more and more popular and Apple decided to invest on iTunes to keep their user base. In 2001 *iTunes digital jukebox* managed up to 1,000 songs synchronized with portable device, two years later this limit was 25 million and iTunes became also an online music store.

Apple made a great marketing campaign with their products and created a huge song database, thanks to its online store; these reasons brought many users to adopt and keep using iTunes.

Beyond this success, iTunes remains context-based music browser mainly based on text lists. Unresolved problems in this music browser include:

- no information about **content** is available (e.g. there is no way to visualize any emotional descriptors for each song);
- the visual feedback provided to the user is limited to simple **text** and (in come cases) album images;
- user can only choose one of the presented layouts and no more **personalization** is allowed.

2.1.1.2 Spotify

Spotify is a system for music streaming. It became very popular for streaming digital-rights-management-restricted (DRM-restricted) music on mobile devices. A web-based version of this service is also available[4]. It allows the user to search for his-own-favorite music or browse the online library by

- Radio streams
- Categories
- Suggestions
  - most populars

[2] Amazon.com. Amazon Music for PC and Mac. `http://amazon.com/getamazonmusic`.

[3] Microsoft. Windows Media Player. `http://windows.microsoft.com/en-us/windows/windows-media-player`.

[4] Spotify AB. Spotify web player. `https://play.spotify.com/`.

- latest releases
- liked by the user
- profile-based suggestions

Graphically, all of these options show an album cover or a wall of album covers. Suggestions are based on different recommendation systems and do not represent a novel feature. However, it is interesting to analyze this system because it introduces some interesting concepts in music browsing.

Radio stations This service defines as *radio station* a thematic-endless playlist. This playlist is generated with tracks similar to a user-specified tag, artist or song and improved song by song by providing a positive (like) or negative (dislike) feedback to each proposed tracks. The user can change radio station to stop the playlist and start another one with another theme. It is basically a special case of recommendation system that users like a lot. This is why this system can be classified as a popular browsing system.

Tag categories This service suggests a categorization based on moods and not only on genres. According to Spotify music category is not a taxonomy, but it is a tag classification. These tags are indexed in a two or three levels hierarchical structure, depending of each category; leaves of this structure are playlists. This categorization allows the user to run a playlist built to match a daily-life activity (such as workout, dinner, party) without caring about which industry-defined genre is suitable for this activity.

Success and problems of this system This system represents a little improvement in context-based systems. Categories and genres are no longer synonyms in the music field. Users appreciate the possibility to browse and play music using a mood or another abstract concept as a search key.

Although the DRM technology brings many limitations[5], users seem to like this system. The main reason is behind Spotify's recommendation systems and category-based playlists features.

This user interface for this system has been designed for mobile devices and fits very well there. However, the web-based version is very similar to the mobile application and although a desktop screen would allow it, Spotify does not show more metadata than it shows on the mobile interface. This makes the web-based version of this service seem very poor of information, indeed it is based on the Wall of Images layout.

[5]for an extended discussion, see `www.DefectiveByDesign.org`

Spotify make use of recommendation systems as the only personalization mechanism in music browsing. Our project increases the user abilities to interact with the system by bringing personalization to the visualization layer.

2.1.2 Other browsers and research projects

Although they do not represent the most popular approaches to music browsing, from now the analysis continues describing projects which use interesting approaches. Some of them doesn't even allow to browse a personal music collection, but they deserve a mention for the peculiarity of the methods of interaction or feedback provided.

2.1.2.1 Nightingale

Nightingale[6] is a fork of the open source project Songbird. Songbird[7] was a free music player focused on fans communities; in 2013 it was shut down. Users decided to fork this project and continue the development under the name *Nightingale*. Nightingale is a music organizer based on Firefox web browser and designed as an open source alternative to iTunes. It offers high extensibility: Firefox plugin system is a well-known development paradigm and this allowed community developers to easily add features to the main project.

Two layouts are available by default: single list and multi-filter. In each one of those layouts all available metadata can be displayed as text. Another interesting feature of this software is the ability to perform complex queries on the music library via a special playlist generation.

Multi-filter This layout is similar to Grouped Lists, but it allows the user to group by several levels of aggregation based on context-based metadata. By default the set of filter groups by genre, then by artist, then by album name. It is possible to add, remove change grouping levels.

This layout represents one of the most-advanced-text-based views in the context-based world. However, it is fully context-based and suffers from all problems of this approach: it is still complex to define meaningful relations between tags, there is still no scientific correlation between tags and audio content and it also has high management complexity.

Like all the context-based systems, it has difficulties in dealing with missing data. When the user applies a filter on a given field and an element has no avilable metadata for that field, that element will not be shown in results. Therefore filter results may be incomplete and it is a problem. In addition, each introduced-aggregation level is a multiplier for problems caused by missing values in metadata.

[6] Nightingale Community. Nightingale web player. `http://getnightingale.com`.
[7] Pioneers of the Inevitable aka POTI Inc.. Songbird. `http://getsongbird.net/`.

Let m_t be the missing rate for the tag t, with $0 \leq m_t \leq 1$, defined as:

$$m_t = 1 - \frac{1}{S} \sum_{i=1}^{S} s_{i,t}$$

where S id the number of songs in the music library, $s_{i,t} = 0$ if song i has a value for t and $s_{i,t} = 1$ if not. The aggregation over T tags brings to a missing rate for the filter results m that is given by ther relation:

$$m = 1 - \prod_{t=1}^{T} (1 - m_t)$$

which is higher than every m_t with $t \in T$. Our project reduces this problem by proving a different missing values handling policy for each tag recognized by the system.

Complex queries This software allows the user to create a *smart playlist* with all the tracks which satisfy a given condition. Through the creation interface the user can specify a complex query using all available metadata. Nightingale with this feature creates an abstraction layer upon a database query.

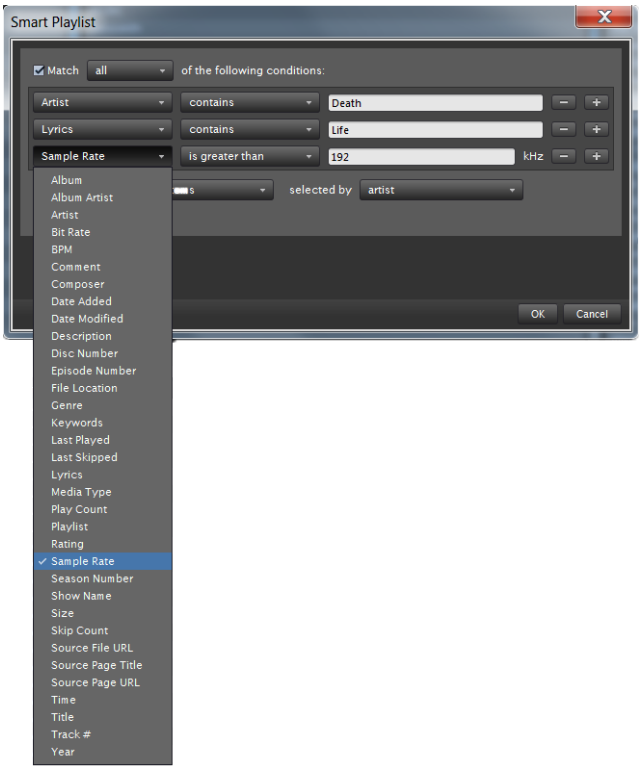


Figure 2.1: Nightingale *smart playlist* creation

It is the context-based browsing approach brought to the maximum level of expression for text-based interfaces: songs are browsed like element in a database. Another similar-open-source project (Quodlibet[8]) adds the possibility to use regular expressions to query the library and manage audio files.

These examples can give and idea of what an advanced user would do with context-based metadata. In a scalable project the design should allow to interact with a superset of these data and perform queries on them.

[8] quodlibet. `https://code.google.com/p/quodlibet/`.

2.1.2.2 Torch Music

Torch Music[9] is an online music browser and music streaming service based on Torch Web Browser, which is in turn based on Chromium web browser source code.

This service offers different browsing options:

- Suggestions
  - most populars
  - liked by the user
  - profile based suggestions
- Genre categories
- Radio streams
- Torch Music Map

Suggestions provides playlists based on recommendation systems, genre categories allow to browse the online library by a rough classification of musical genres, radio streams are endless playlists inspired by the Radio stations concept described in 2.1.1.2. These options do not represent any novel feature and will not be discussed any longer.

Torch Music Map Torch Music is an online service and could track the location of users around the world. The *Music Map* feature shows who is listening what and where. This is done by plotting tokens on a planisphere (Google Maps); every token is a user and its position on the map is obtained by geolocalizing the IP address of the user, clicking on the token allows to show details about which song that user is currently listening.

The interesting part of this approach is the use of a bidimensional space. In addition, the position of the point has a meaning: it is relate to the location where someone is listening to that song.

2.1.2.3 MusicRainbow

MusicRainbow [12] is a simple interface for discovering artists. The whole browsing interface is based on a rainbow circle. Inside the circle there are high-level genre keywords; outside the circle more specific keywords are placed. Each ring of the rainbow (and thus color) corresponds to one high-level term. The user can rotate the rainbow via a knob to rotate labels. On the right side of the screen there is a

[9] Torch Media. Torch music. `http://music.torchbrowser.com/`.

list of the available artists; the selected artist is highlighted and lies in the middle of the right column.

This browsing interface allows to discover artists in the same way a radio can be tuned. Rainbow colors are mapped to different music genres however, which genres corresponds to which color is a piece of information that is not given explicitly to the user. There is no linear mapping between knob position and the circle position; this feature helps the user to discover new artists located in a given area, but it also makes difficult to perform direct access to a known artist.

This project was a first attempt to use a simple interface to perform context-based browsing using a non-textual interface.

2.1.2.4 MusicSun

MusicSun [13] is a graphical user interface for discovering artists. It is an evolution of MusicRainbow. Artists are recommended based on one or more artists selected by the user.

The interface shows a poorly-drawn sun: a circle in the middle and a set of triangles as rays. Inside the circle there are artists provided by the user as seed for a query. For each artist the system retrieves a set of keywords as descriptors. The most common descriptors of the user defined artists represent rays of the sun. The triangular shape of each ray encodes the following information with respect to the word it represents: if the side of the triangle facing the sun is longer, then the respective word describes the artists better; if the length of the ray is longer, then there are more artists in the collection which can also be described using the respective word. Once a ray is selected, it spins itself into the rightmost position, indicating that it is currently being used to modify the recommendations. Recommendations are shown as a list in the right column. Minor components of the interface such as little sliders and on/off switches allows the user to tune some weights of the recommendations.

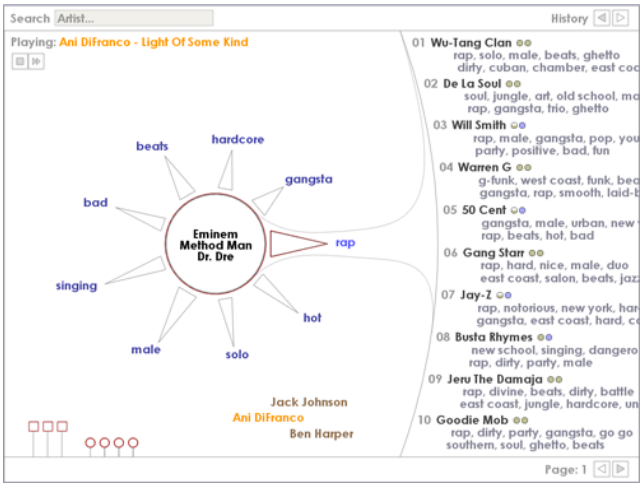


Figure 2.2: MusicSun. Elias Pampalk & Masataka Goto

With respect to MusicRainbow, authors make much less use of colors and started to use more shapes to provide visual feedback to the user. It remains a browsing interface specifically designed for authors discovery and it is not suitable for direct

access to known artists. It is context-based system with a non-textual interface that also allows minimal personalization via parameters tuning.

2.1.2.5 Liveplasma

Liveplasma[10] is a music discovery tool that shows Amazon.com similarities; it is useful to find relationship between artists and suggest to the user some new artist similar to a given one. This similarity captures the preferences of the users by counting the number of users that bought music from each artist; according to Amazon.com, two artists are similar if a high number of users bought music from both of them.

Each artist is a node in a graph; node size depends on artist popularity, edges represents the relationship *users who bought album from this artist also bought*. This system allows the user browse a music library using a graph instead of lists. It is an interesting method for discovering artists and several projects refer to this approach to develop new browsing interfaces (e.g. Discovr Apps [14]).

The key of this project is simplicity. It makes use of a very simple graph to display context-based metadata and connections between elements.

2.1.2.6 Music Timeline

Music Timeline[11] is a Google Research project for music browsing based on Google Play Music data. This project shows a timeline of music album and artist from 1950 to nowadays grouped by genres and subgenres. It uses different colors to separate genres and different shades of the same color to separate subgenres. Timeline width is an area diagram normalized on the popularity of shown categories. Under the timeline area, this system shows a wall of album covers of the genre (or subgenre or artist) currently focused.

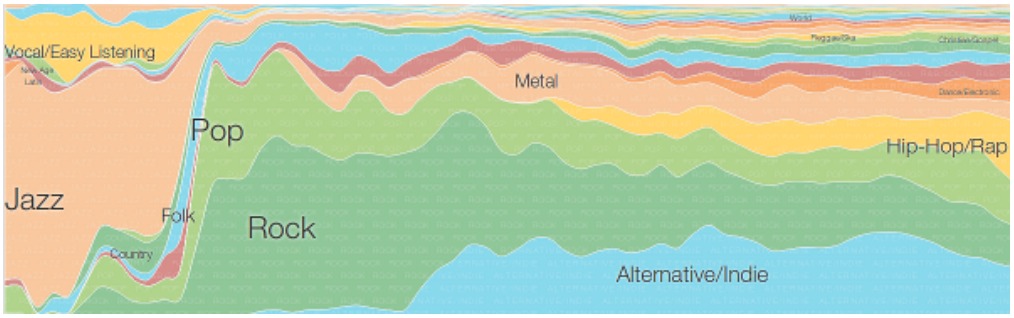


Figure 2.3: Music Timeline, Big Picture and Music Intelligence research groups at Google.

This project is linked to Google Play store: users who find interesting music are redirected to the online store where they can buy the album.

[10] Frederic Vavrille. Liveplasma discovery engine. `http://www.liveplasma.com/`.
[11] Google Research. Music timeline. `https://music-timeline.appspot.com/`.

Music Timeline can show two dimensions (year and genre) on the same bidimensional space. It is a great example of context-based browsing using a map to visualize metadata. This system represents a considerable improvement from the visual feedback point of view. However, it still suffers from the problems of context-based systems and no personalization is allowed.

2.1.2.7 JANAS

Janas [15] is a semantic text-based music search engine. User types a query describing some desired characteristic of the song. This system recognizes natural-language-qualitative adjectives. The system then shows up a set of song which reflects the given semantic query.

This project allows to perform queries on a music library using a subset of (English) natural language. The systems can deal with both emotional descriptors mapped to the Valence-Arousal plan and a set of non-emotional descriptors. It accepts also adjectives qualifiers to modify the weight of the related concept.

Janas represents a bridge between text-based music browsers and content-based music browsers. The interesting part of this project is that a user can query the system using an old-school-text-based approach having in return a content-based result. Its problem remains the textual approach in browsing query results.

2.1.2.8 Musicovery

Musicovery[12] is a music discovery service that shows songs as elements in a 2D map. The position on the map reflects a similarity distance computed on the tracks.

Musicovery uses the Valence-Arousal plan [16] to plot songs; each song is a point on this plan and its color depends on the music-genre classification. This service allows the user to decide the desired degree of novelty to show on the map: each song has a popularity support index. In order to fit different markets, this service has localized versions with different element support.

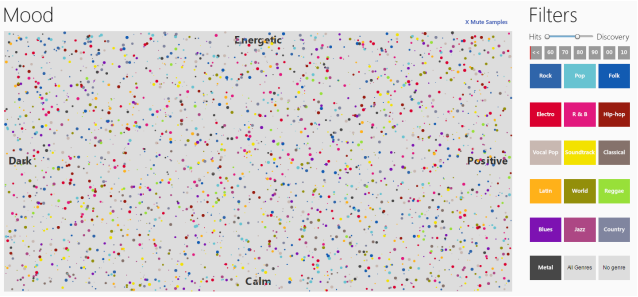


Figure 2.4: Musicovery.com. Musicovery Company. Screenshot authorized by the Musicovery Team

This project introduces a content-based space for browsing songs. It exploits

[12]Vincent Castaignet and Frederic Vavrille. Musicovery. `http://musicovery.com/`. Musicovery company.

different modifiers of the elements on the map (color, scale) to render other context-based metadata. Personalization is limited to a few parameters (localized version, filters on genre and popularity index).

2.1.2.9 MusicBox

MusicBox [17] is an application that visualizes a music collection by mapping songs onto a bidimensional space. MusicBox assigns song locations by analyzing many descriptive features of a song's sound. These features can be quantitative, such as the proportion of high frequency sounds, or qualitative, such as characterizing a song as "happy". MusicBox's organizational algorithm displays songs with similar feature values closer together. MusicBox performs a dimension reduction by Principal Component Analysis (PCA) [18] to map elements in a bidimensional space. This means that there is no direct correlation between axes and features. User can only select which features should be included or not in the PCA.

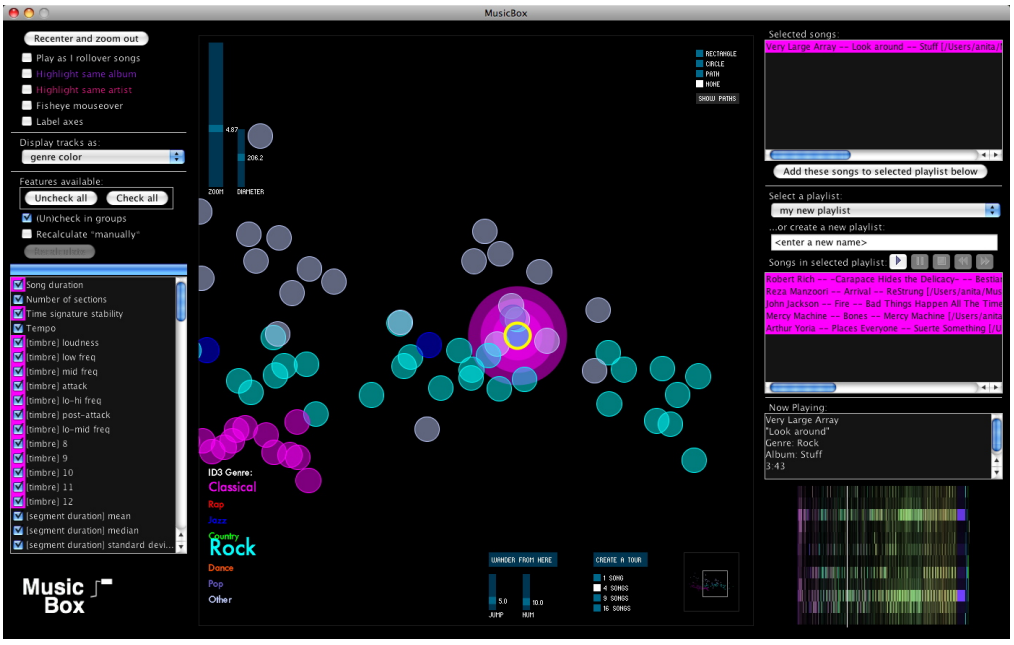


Figure 2.5: Musicbox. Anita Lillie, Media Laboratory, MIT, Cambridge, Massachusetts

Playlist by path This project allows the user to automatically create playlists exploiting the similarity showed on map. MusicBox finds a tour with the songs nearest to a path and queue them in a new playlist by simply drawing that path on the screen.

This approach allows to create a playlist directly from the on-screen bidimensional representation of the library. Users appreciate this feature and this demonstrates

that space-oriented music browsing can replace text-oriented music browsing in the future.

About Self-Organizing Maps MusicBox uses a Self-Organizing Map (SOM) [19] to display content-based and context-based metadata. In this way the user is able to see a distance between elements based on most-significant-available data. Browsing personalization is possible, but it is still complex to achieve (PCA parameters can be enabled or disabled).

SOM is not a recent approach. SOMs have already been used as a fixed map to visualize songs in *Islands of Music* [20] and to browse clusters of songs in PlaySOM [21, 22]. These projects were designed to discover some kinds of song similarity in a large music collection. In these cases, the problem of the SOM approach is that the concept of most-significant data is provided by the software and not by the user: the user may find not interesting what the software defines as relevant. In addition, the use of a bidimensional space with mixed axes does not ensure an immediate understanding of what these axes represent.

2.2 Game engine technology

2.2.1 What a game engine is

Videogame programmers built up year by year various tools to speed up their work. Some of these tools were released for public use. A few big software houses and some community of videogame developers decided to create collections of their tools; this is how game development kits were born. Some of these development kits grew so much that they could act as full development frameworks; with such a framework it is possible do develop a videogame without using external software. Following the needs of the videogame industry, game engines started to be designed and developed as projects per se. These game engines became less and less specific: in the past they were designed to develop an episode of a specific videogame title; later they were generalized to develop an arbitrary title of the same game style; and nowadays they can be used to develop games regardless of the game style. They are also widely employed to build several types of non-game software.

A game engine is a complete development environment and allows programmers and artists to design their project within the game engine and export the product for several target platforms. A game engine creates an abstraction layer that allows code portability through the cross-compilation of the project.

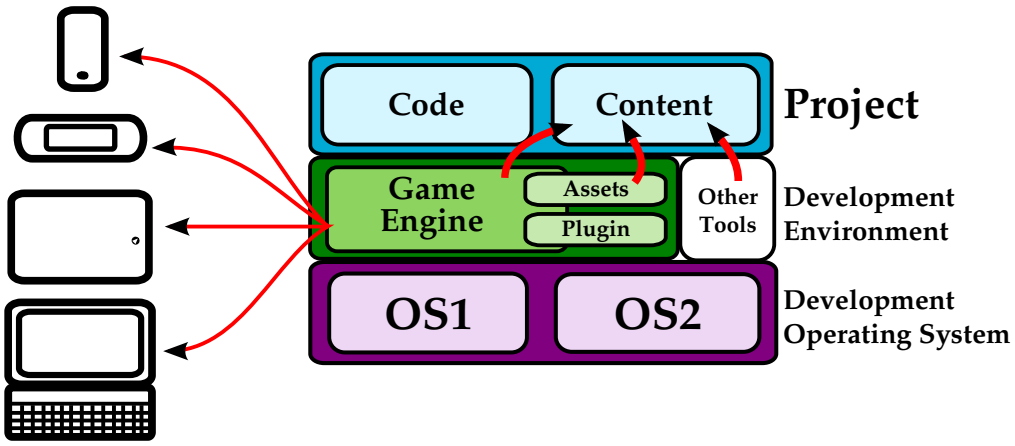


Figure 2.6: Game engine development environment

2.2.2 Why a game engine

2.2.2.1 Success stories

It is not so difficult to find in literature success stories of researchers who exploited a game engine to achieve important results. The authors of [23] use different game engines to produce virtual reconstructions of the same cave with three chambers. In the archaeological field it is possible to visualize building of historical interest as they were supposed to be thousands of years ago. A first review of the potential of computer games technology was written in 2004 by Meister and Boss [24]; more recent works achieve better results, as can be seen in [25, 26]. Authors of these works [27, 28] shows that it is possible use GIS (Geographic Information System) data to reconstruct and visualize a landscape in a game engine. Among the various-non-gaming purposes where game engines are used, there is the area of simulators. A few examples of simulators designed for model elaboration and verification can be found in [29] and [30].

2.2.2.2 Main Features

A game engine simplify the development process. It usually provides:

- Data preset:
  - world components
  - user interface API
  - full-example projects
- Advanced development tool:
  - object physics simulation

  - output preview

- Product flexibility:
  - modular development
  - multiple input control management
  - multi-platform export

This project was chosen to be developed within a game engine in order to achieve a fancy result without an extraordinary effort. It also played an important role the possibility to be easily extended with graphical and animation improvements without changing the underlying structure.

2.2.3 Which game engine

In 2014 a survey[13] over ten thousands of app developers asked which development environment they use. Table 2.1 shows the most used tools with the percentage of interviewed developers who *strictly use* or *also use* a development environment.

The most common game engine is Unity3D[14]. As second preference there are developers who directly develop with native code and without any kind of framework nor game engine. Then, comes developers who prefer to use their own framework or game engine. Cocos2d[15] is the next game engine in this ranking. Not far from it, there come Adobe Air[16] and Unreal Engine[17].

Table 2.1: App development environment popularity

	strictly	also
Unity	29%	47%
Native code	29%	42%
Custom solution	9%	21%
Cocos2d	8%	19%
Adobe Air	6%	15%
Unreal Engine	3%	13%

Cocos2d, as the name suggests, is designed to render bidimensional projects. 3D support for Cocos2d was introduced recently and it is not yet well developed. For the purpose of this project it was decided that the bidimensional restriction was excessive and the analysis was not elaborated beyond this point. Adobe Air is only a framework for code portability and should not be considered a game engine.

In addition to popular game engines found in table 2.1, another product mentioned in several works [23, 27, 29] was taken into account for a deeper analysis:

[13] Vision Mobile, Developer Economics, State of the Developer Nation Q3, 2014 `http://www.visionmobile.com/product/developer-economics-q3-2014/`

[14] Unity 3D. `http://unity3d.com/`

[15] Cocos2d-x.org. `http://www.cocos2d-x.org/`

[16] Adobe Systems Incorporated. Adobe Air. `http://www.adobe.com/products/air.html`

[17] Epic Games Inc. Unreal Engine. `https://www.unrealengine.com/`

CryEngine[18]. A brief analysis of all these products is summarized in table 2.2.

Table 2.2: Game engine comparison

	price	source code	development platform	target platform
Unity3D version 4	free (full version 75$/month or 1500$)	not provided	Windows Mac OS	Mobile phone Consoles Portable Consoles Windows Mac OS Linux Unity Web player
Unreal Engine version 4	free (royalties 5%)	full access	Windows Mac OS Linux	Mobile phone Consoles Windows Mac OS Linux HTML 5 Oculus Rift
CryEngine version 3	9.90€/month	not provided	Windows	Mobile phone Consoles Windows Mac OS Linux

This project is not a videogame and it is possible that during the development process or in the future will be required to interact with low-level engine features and maybe modify the engine itself to adapt it to the project needs. Access to the source code was found to be a very important point in favour of **Unreal Engine** and since there are not great differences with other competitors, the decision was to start the development with this game engine.

[18]Crytek GmbH. CryEngine. `http://www.cryengine.com/`

Chapter 3

Theoretical background

Our system exploits a mixed content-based and context-based approach. The content-based aspect of our project requires that a song to be represented by one or more pieces of information extracted from its content. In order to do this, we need to understand how content-based information can be extracted from an audio file.

In this chapter the theoretical background needed to understand the project will be introduced. First, it will be given an overview of Signal Processing, then it will be discussed topics of Music Information Retrieval and related tools and methods. Finally, in section 3.3 the discussion will focus on the results of a survey about habits and desires of users in the field of music browsing.

3.1 Signal Processing Overview

In this section we briefly introduce basics of digital signal processing. It is not purpose of this thesis to provide a full discussion of this field. A deeper analisys can be found in [31,32].

3.1.1 Fourier Analysis

Before starting the discussion of the fourier analysis it is useful to summarize here some recurrent symbols:

$s(t)$ continuous-signal function, it describes the evolution of a signal over time

s_k discrete-signal function, it describes sample-by-sample the evolution of a signal over time

$S(f)$ continuous-frequency-domain representation of the signal s

S_k discrete-frequency-domain representation of the signal s

$\frac{A_0}{2} = \frac{a_0}{2}$ average value of the signal

θ_n phase of the n-th component of the signal

A_n peak-to-peak amplitude value of the n-th component of the signal

c_n n-th Fourier coefficient

$\overline{\bullet}$ complex conjugate of $\bullet$

3.1.1.1 Fourier Serie

The Fourier serie is one of the foundamentals of the signal processing field.

The Fourier serie S_N expresses a periodic function $f(x)$ of period P as sum of N sin functions. Its mathematical formulation is

$$S_N(x) = \frac{A_0}{2} + \sum_{n=1}^{N} A_n \cdot \sin\left(\frac{2\pi n x}{P} + \theta_n\right)$$

It has been proved that $\lim_{N\to\infty} S_N(x) = f(x)$.

It is possible to apply trigonometric properties to rewrite the formulation as

$$\begin{aligned} S_N(x) &= \frac{a_0}{2} + \sum_{n=1}^{N}\left(a_n \cos\frac{2\pi n x}{P} + b_n \sin\frac{2\pi n x}{P}\right) \\ &= \sum_{n=-N}^{N} c_n \cdot e^{\frac{2\pi i n x}{P}} \\ c_n &= \begin{cases} \frac{A_n}{2i} e^{i\theta_n} & n > 0 \\ \frac{a_0}{2} & n = 0 \\ \overline{c_n} & n < 0 \end{cases} \end{aligned}$$

3.1.1.2 Fourier Transform (FT)

The Fourier transform provides a frequency-domain representation of a signal described in the time domain. Given a signal $s(t)$, the fourier transform of this signal is defined as

$$S(f) = \int_{-\infty}^{+\infty} s(t) \cdot e^{-2\pi i f t} dt$$

3.1.1.3 Discrete-Time Fourier Transform (DTFT)

When it comes to analyze a discrete-time signal, here comes the need of a discrete formulation of the Fourier transform. The Discrete-Time Fourier Transform provides a frequency-domain representation of a discrete-time signal function $s[n]$ originated from an uniform sampling of a continous function $s(t)$. Its mathematical formulation is

$$S(f) = \sum_{n=-\infty}^{\infty} s[n] \cdot e^{-2\pi i f n}$$

with $n \in \mathbb{Z}$.

3.1.1.4 Descrete Fourier Transform (DFT)

The Discrete Fourier Transform (DFT) provides a discrete-frequency representation of a discrete-time signal function. Unlike the DTFT, input and output of the DFT are both finite.

Given N samples of the discrete signal s_k, the DFT is defined as

$$S_k = \sum_{n=0}^{N-1} s_n \cdot e^{-2\pi i k \frac{n}{N}}$$

with $k \in \mathbb{Z}$. The DFT converts a finite number of equally spaced samples of a function to a list of coefficients of complex sinusoids. It is used to represents a sampled function of time in the frequency domain.

3.1.1.5 Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT) is a Fourier related transform. It is similar to the DFT, but, unlike the DFT, DCT allows only real input data with even symmetry [33].

Given N samples of the discrete signal s_k, the DCT is defined as

$$S_k = \sum_{n=0}^{N-1} s_n \cdot \cos\left[\left(n + \frac{1}{2}\right) \cdot \frac{k\pi}{N}\right]$$

with $k \in [0; N-1]$. Using only cosine functions, the DCT has been found to approximate with a lower number of coefficients a typical input signal.

3.1.2 Spectral Analysis

3.1.2.1 Power spectrum

A periodic signal has infinite energy, but it has a finite average power, given from the formulation

$$P_s = \frac{1}{T_p} \cdot \int_{T_p} |s(t)|^2 \, dt = \frac{1}{T_p} \cdot \int_{T_p} s(t) \cdot \overline{s}(t) \, dt$$

where T_p is the period of the signal $s(t)$ and $\overline{s}(t)$ is the complex-conjugate of $s(t)$. It is possible to take the Fourier serie decomposition (3.1.1.1) of $\overline{s}(t)$ and substitute

it in the previous equation.

$$
\begin{aligned}
P_s &= \frac{1}{T_p} \cdot \int_{Tp} s\left(t\right) \cdot \sum_{k=-\infty}^{\infty} \overline{c_k} e^{-2\pi i f_0 t} dt \\
&= \sum_{k=-\infty}^{\infty} \overline{c_k} \left[\frac{1}{T_p} \cdot \int_{T_p} s\left(t\right) e^{-2\pi i f_0 t} dt \right] \\
&= \sum_{k=-\infty}^{\infty} |c_k|^2
\end{aligned}
$$

The diagram of $|c_k|^2$ over $k \cdot f_0$ is known as power density spectrum, or simply *power spectrum*.

An example of power spectrum is shown in figure 3.1.

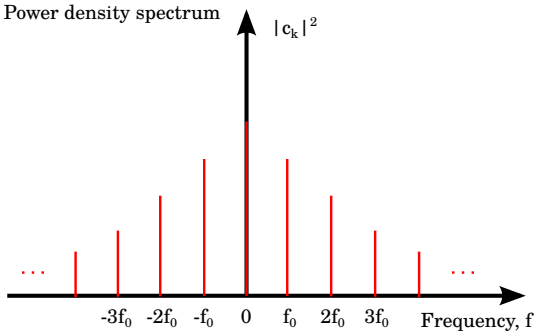


Figure 3.1: Power density spectrum of a continous-time periodic signal

3.1.2.2 Magnitude spectrum

In section 3.1.1.1 we defined Fourier coefficients as complex numbers c_k. We can also write them in the form $c_k = |c_k| e^{i\theta_k}$. It is possible to plot $\{|c_k|\}$ and $\{\angle\theta_k\}$ over $k \cdot f_0$. Respectivly they are known as magnitude voltage spectrum, or simply *magnitude spectrum*, and *phase spectrum*.

3.1.3 Windowing

Sometimes might want to capture the evolution over time of the spectrum of the signal. In order to do this, what can be done is a frame decomposition of the signal: spectral analysis is then applied segment-by-segment instead of to the whole signal.

Therefore, there is a need to analyze only a part of the signal. A window signal (or window function) is a signal zero-valued outside a certain interval. Windowing a signal $s\left(t\right)$ means to multiply the signal to another window signal $window\left(t\right)$. A

special case of window signal is the rectangle defined as

$$rect\,(t) = \begin{cases} 0 & t < -a \\ 1 & -a \leq t < a \\ 0 & t \geq a \end{cases}$$

This signal allow to analyze the original signal $s\,(t)$ only between $-a$ and a. An example of how windowing works is shown in figure 3.2.

However, a window function brings side-effects in the spectral result because of its own spectral component. This effect is known as *Spectral leakage*.

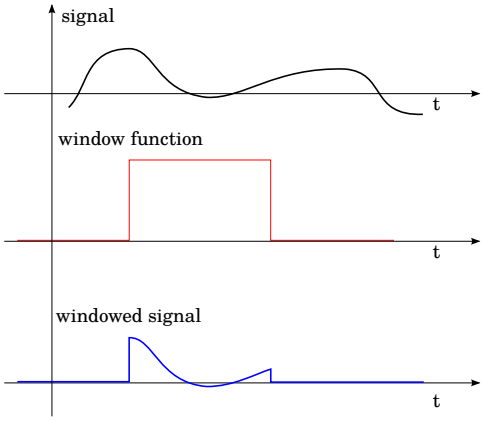
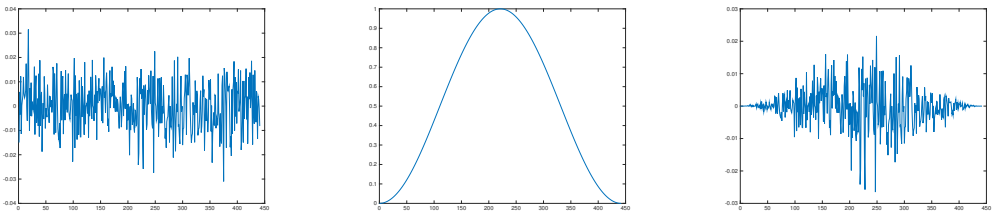


Figure 3.2: Windowing

When it comes to apply windowing to real-life problems, the most critical point is that window functions dirty the signal near their edges. In order to mitigate this effect, it is best-practice to consider overlapping frames when performing the decomposition. Furthermore, different window functions were proposed and a lot of research has been done in this field. There is not a best window function that can always be applied with optimal results. There are, instead, window functions that, depending on the signal, can lead to better performance with respect to others. Most popular window functions are the *rectangular window*, *Hann window* (aka *Hanning window*), *Hamming window*, *Blackman windows*.

In figure 3.3 is shown an example of a frame taken from an audio signal 3.3a, the Hann window function 3.3b and the resulted windowed signal 3.3c.



(a) Original frame (b) Hann window function (c) Windowed frame

Figure 3.3: Windowing applied to a frame of audio signal

3.2 Music Information Retrieval

Music Information Retrieval (MIR) is a multidisciplinary research field. Its goal is to extract meaningful information from music. In this field information are expressed by descriptors. It is possible to recognize different kinds of descriptors: *low-level features*, *mid-level* and *high-level features*. Low-level features are those which are directly connected to the audio signal, but they are meaningless for almost everyone who has not studied MIR. On the other side, high-level features are those which carry greater semantic importance for the public, but they are subjective and there is not a direct connection with the audio signal. This difference is also known as *semantic gap* between low-level features and high-level features. However, it is possible to define as mid-level features those features with an abstraction level between high-level and low-level features.

3.2.1 Low-Level and Mid-Level Features

Low-Level Features (LLFs) are also known as audio features. This is because they are objective and can be computed just applying a mathemathical transformation to the digital-audio signal. Focusing on the aspect that a feature investigate in the signal, it is possible to distinguish between spectral, temporal and rhythmic features.

In table 3.1 we summarize the LLFs and MLFs considered in this work. A more detailed description will follow, most of them are defined in [34–36].

Table 3.1: Low-level and mid-level feature considered in this work

Low-level features	
Spectral	MFCC, Spectral Centroid, Spectral Spread, Spectral Skewness, Spectral Kurtosis, Spectral Inharmonicity, Spectral Flux, Spectral Rolloff, Spectral Slope, Spectral Standard Deviation, Spectral Irregularity, Spectral Smoothness, Spectral Flatness
Temporal	Zero Crossing Rate

Mid-level feature	
Rhythmic	Tempo
Chroma	Chromagram

3.2.1.1 Mel-Frequency Cepstrum Coefficients

Mel-Frequency Cepstrum Coefficients (MFCCs) are spectral LLFs designed for speech recognition, but it is currently used in many aspects of MIR. MFCCs are

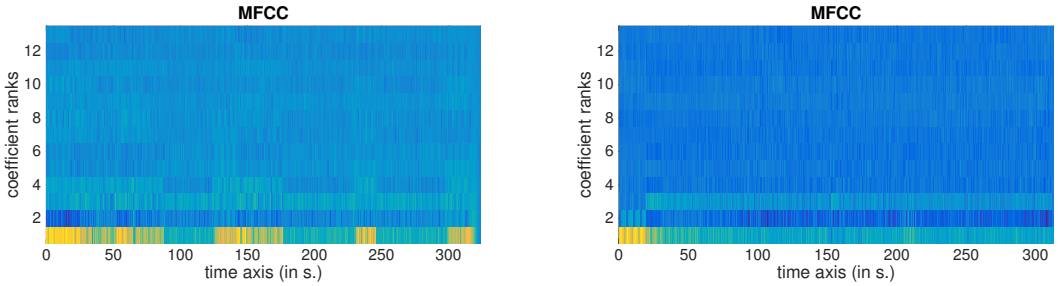
based on the Mel-Frequency scale, a model for the frequency perception of the human auditory system.

They are computed from a sort of nonlinear spectrum of a spectrum. More precisely they are computed from a Discrete Cosine Transform (DCT) applied on a reduced Power Spectrum. This reduced Power Spectrum is obtained from the log-energy of the spectrum pass-band filtered by a mel-filter bank. The mathematical formulation is:

$$c_i = \sum_{k=1}^{K_c} \left\{ \log\left(E_k\right) \cdot \cos\left[i \left(k - \frac{1}{2} \right) \frac{\pi}{K_c} \right] \right\} \qquad \text{with} \qquad 1 \leq i \leq N_C$$

where c_i is the $i-th$ MFCC component, E_k is the spectral energy measured in the critical band of the $i-th$ mel-filter, N_c is the number of mel-filters and K_c is the number of cepstral coefficients c_i extracted from each frame.

MFCC for two songs is shown in figure 3.4. It is possible to clearly recognize in the lower part of the graph the famous intro with tolling of bells of 3.4b while in 3.4a can be observed the difference between the depressive segments and aggressive ones.



(a) Placebo - «Passive Aggressive»

(b) AC/DC - «Hells Bells»

Figure 3.4: MFCC for two songs

3.2.1.2 Zero Crossing Rate

Zero Crossing Rate (ZCR) is, by definition, the normalized frequency at which an audio signal $s\left(n\right)$ crosses the value 0, i.e. changes from positive to negative or viceversa.

Formalized, it is:

$$F_{ZCR} = \frac{1}{2} \left(\sum_{n=1}^{N-1} \left| \operatorname{sgn}\left(s\left(n\right)\right) - \operatorname{sgn}\left(s\left(n-1\right)\right) \right| \right) \frac{F_s}{N}$$

where F_s is the sample rate and N the number of samples in $s\left(n\right)$.

3.2.1.3 Spectral Centroid

Spectral Centroid (SC) is the center of gravity of the magnitude spectrum. It determines the point in the spectrum where most of the energy is concentrated.

Given a frame decomposition of the audio signal, Spectral Centroid is computed as:

$$F_{SC} = \frac{\sum_{k=1}^{K} f(k) S_l(k)}{\sum_{k=1}^{K} S_l(k)}$$

where $S_l(k)$ is the Magnitude Spectrum at the $l-th$ and the $k-th$ frequency bin, $f(k)$ is the frequency corresponding to $k-th$ bin and K is the total number of frequency bins.

This feature is directly correlated with the dominant frequency of the signal. It is usually associated with the perception of brightness and sharpeness of the sound.

A deeper analysis about sound sharpness and its relation with SC can be found in [37].

3.2.1.4 Spectral Spread

Spectral Spread (SSP) measures the spectral shape. It is also called *instantaneous bandwidth* and it is computed, according to [34], as the second central moment of the log-frequency spectrum. It can mathematically be written as

$$F_{SSP} = \sqrt{\frac{\sum_{k=0}^{K} [f(k) - F_{SC}]^2 \cdot S_l(k)}{\sum_{k=0}^{K} S_l(k)}}$$

where $S_l(k)$ is the Magnitude Spectrum at the $l-th$ and the $k-th$ frequency bin, $f(k)$ is the frequency corresponding to $k-th$ bin, K is the total number of frequency bins and F_{SC} is the Spectral Centroid computed as described in 3.2.1.3 for the given frame.

3.2.1.5 Spectral Skewness

Spectral Skewness (SSK) is the third moment of the distribution. It gives an estimation on the symmetry of the magnitude spectrum values. A positive value of Spectral Skewness represents an asymmetric concentration of the spectrum energy on higher frequency bins, while negative coefficients represent a distribution with a higher concentration on lower frequency bins. The perfect symmetry corresponds to the zero Spectral Skewness value. Its mathematical formulation is:

$$F_{SSK} = \frac{\sum_{k=1}^{K} (S_l(k) - F_{SC})^3}{K \cdot F_{SS}}$$

where $S_l(k)$ is the Magnitude Spectrum at the $l-th$ and the $k-th$ frequency bin, $f(k)$ is the frequency corresponding to $k-th$ bin, K is the total number of

frequency bins, F_{SC} is the Spectral Centroid at the $l-th$ frame (3.2.1.3) and the F_{SS} is the Spectral Spread at the $l-th$ frame (second moment of the distribution).

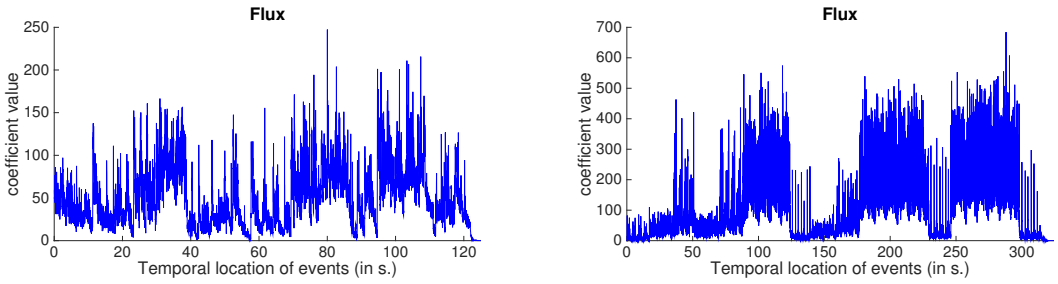
3.2.1.6 Spectral Flux

Spectral Flux (SF) measures how quickly the spectrum of a signal is changing. It is computed from the distances between the amplitudes of the Magnitude Spectrum between two consecutive frames. Using the Euclidean distance, the mathematical formulation is:

$$F_{SF} = \frac{1}{K}\sum_{k=1}^{K}\left[\log\left(|S_l\left(k\right)+\delta|\right)-\log\left(|S_{l+1}\left(k\right)+\delta|\right)\right]^2$$

where $S_l\left(k\right)$ is the Magnitude Spectrum at the $l-th$ and the $k-th$ frequency bin, $f\left(k\right)$ is the frequency corresponding to $k-th$ bin and δ is a small parameter to avoid $\log\left(0\right)$.

Spectral Flux for two songs is shown in figure 3.5. A quick comparison between those two plots and their scale on this feature axis allows to point out that Elvis's song 3.5a has less than half F_{SF} with respect to Placebo's one. Therefore, we can say that 3.5b is more dynamic than Elvis's song.



(a) Elvis Presley - «I Believe»

(b) Placebo - «Passive Aggressive»

Figure 3.5: Spectral Flux for two songs

3.2.1.7 Spectral Rolloff

Spectral Rolloff that characterize the energy distribution of a signal. It is a sort of percentile. It represents the lowest frequency F_{SR} at which the value of the sum of the power spectrum of lower frequencies untill F_{SR} reaches a certain amount of the total sum of the magnitude spectrum. Spectral Rolloff is formalized as:

$$F_{SR} = \min\left\{f_{K_{roll}}|\sum_{k=1}^{K_{roll}}\left(S_l\left(k\right)\right) \geq R\cdot\sum_{k=1}^{K}\left(S_l\left(k\right)\right)\right\}$$

where $S_l(k)$ is the Magnitude Spectrum at the $l-th$ and the $k-th$ frequency bin, $f(k)$ is the frequency corresponding to $k-th$ bin, K is the total number of frequency bins, K_{roll} is the frequency bin index corresponding to the estimated rolloff frequency $f_{K_{roll}}$ and R is the frequency ratio. The frequency ratio is an arbitrary parameter, in this work the value of R is set to 90%.

3.2.1.8 Spectral Flatness

Spectral Flatness (SFlat) is a measure useful to check how much a signal is noisy, estimating the similarity between the magnitude spectrum of the signal frame and the flat shape inside a reference frequency band. It is mathematically computed by the ratio of the geometric mean of the power spectrum over the arithmetic mean of the power spectrum:

$$F_{SFlat} = \frac{\sqrt[k]{\prod_{k=1}^{K-1} S_l(k)}}{\prod_{k=1}^{K} S_l(k)}$$

where $S_l(k)$ is the Magnitude Spectrum at the $l-th$ frame and the $k-th$ frequency bin, K is the total number of frequency bins.

3.2.1.9 Spectral Kurtosis

Spectral Kurtosis (SK) is a statistical parameter that attempts to capture the evolution of the impulsiveness over frequencies.

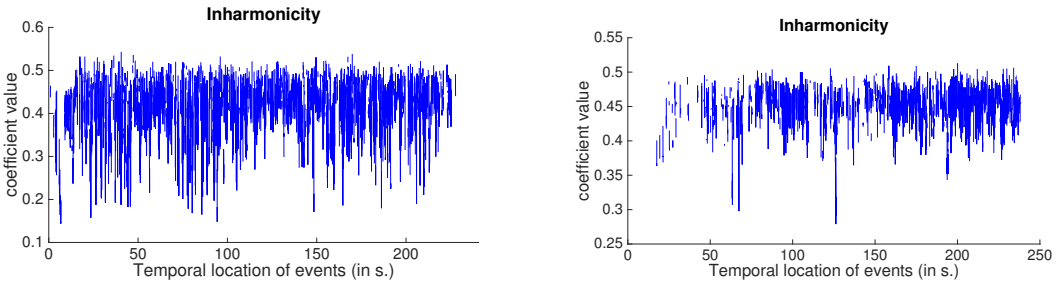
It was initially exploited to detect transients in sonar signals, but it is also useful to detect impulses in other signals. A more detailed discussion about this feature and its applications is available in [38].

3.2.1.10 Spectral Inharmonicity

Spectral Inharmonicity is a features inversely proportional to the spectrum regularity, more precisely it measures how much partial tones depart from the harmonic serie (multiples of the fundamental frequency). Spectral inharmonicity is not related to unpleasantness of a sound, but it is an intrinsic characteristic of some instruments (e.g. string or percussion instruments have complex spectral inharmonicity).

This feature can be useful to detect voice intonation and the presence of some kinds of instruments in a composition. A deeper analysis of this feature can be found in [39].

Spectral Inharmonicity for two songs is shown in figure 3.6. From these plot it can be seen that Bon Jovi's «Livin On a Prayer» 3.6b has a value of Spectral Inharmonicity stable for the whole song while «So Emotional» by Christina Aguilera 3.6a involves several vocalizations and the Spectral Inharmonicity reflects these virtuosos.



(a) Christina Aguilera - «So Emotional» (b) Bon Jovi - «Livin' On a Prayer»

Figure 3.6: Spectral Inharmonicity for two songs

3.2.1.11 Spectral Slope

Spectral Slope deals with the energy of the signal. It measures the decay of the energy of the signal as the frequency increase. The trend of many signal to have less energy at high frequencies is well-known and it depends on the source of the sound.

It is a timbre feature and it is useful for speech detection and voice intonation recognition.

3.2.1.12 Spectral Standard Deviation

This is basically the standard deviation computed on all the digital samples of the magnitude spectrum.

3.2.1.13 Spectral Irregularity

Spectral Irregularity (SpIrr) is a timbre feature that describes the relations between the intensities of adjacent harmonics. Higher is the SpIrr, lower is the similarity between the intensity of adjacent harmonics. A deeper analysis can be found in [40].

This feature is useful to recognize the timbre of some musical instruments within the spectral domain.

3.2.1.14 Spectral Smoothness

Spectral Smoothness captures the irregularity of the shape of the magnitude spectrum. This is another feature that is useful to recognize musical instruments.

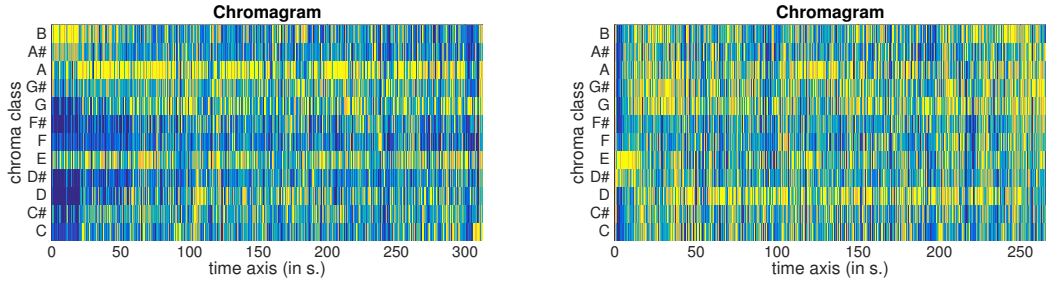
A deeper analysis can be found in [41].

3.2.1.15 Chroma features

Chroma features attemps to capture information about the musical notes from the spectrum of the audio signal. The log-magnitude spectrum is mapped into a log-frequency scale. Given the frequencies of each note in a twelve-tone scale, regardless

od the original octaves, the histogram of notes that can be built is called *chromagram*. Each bin in the chromagram represents a semitone.

Chromagram for two songs is shown in figure 3.7.



(a) AC/DC - «Hells Bells»

(b) Aerosmith - «Back in the Saddle»

Figure 3.7: Chromagram for two songs

3.2.1.16 Tempo

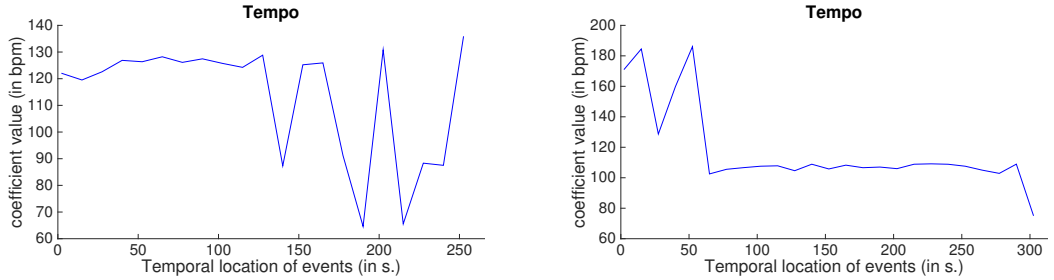
Tempo is a mid-level feature that captures the speed of a given musical composition. A common measure for tempo is Beats-Per-Minutes (BPM), it indicates how many beats must be played in a minute. An abstract definition of *Beat* is given in [42] as

> *the temporal unit of a composition, as indicated by the (real or imaginary) up and down movements of a conductor's hand.*

There exist different approaches to compute this measure. Some simply find peaks in the magnitude spectrum assigning a high/low label to samples; others methods may apply a filter before analyzing the signal. A simple two-state beat tracking method is described in [43] while another more complex version that involves dynamic programming is described in [44]. The tempo estimation method we considered is a hybrid of these two, written by Matthew Davies and Christian Landone[1].

An example of tempo for two songs is shown in figure 3.8. We can identify two different evolution pattern: «Back In The Saddle» 3.8a is stable for half of the song and than has some spikes; 3.8b shows a floating phase as intro, then it keeps its quite-low value until the end.

[1] Matthew Davies and Christian Landone, Tempo and Beat Tracker, QM Vamp Plugin set, Centre of Digital Music at Queen Mary, University of London - `http://vamp-plugins.org/plugin-doc/qm-vamp-plugins.html#qm-tempotracker`



(a) Aerosmith - «Back in the Saddle» (b) AC/DC - «Hells Bells»

Figure 3.8: Tempo for two songs

3.2.2 Machine learning overview

In this section we will give an overview of machine learning methods. This thesis does not mean to give an exhaustive explanation of the machine learning field and will focus only on the topics needed to completely understand what we discuss later. For a deeper analysis of this field, see [45–47].

Machine learning is a research field that deals with systems that are capable of learning from the experience. A system is said to learn from experience E, with respect to a class of tasks T and performance criteria P, if performance in a task in T, measured with P, increase after E.

For the purpose of this thesis it will be considered as learning machine a system where E is made of data and T is a prediction over the same kind of data given as E.

Machine learning algorithms can be divided in *unsupervised learning* and *supervised learning*. Unsupervised learning algorithms take as input only data and without any kind of feedback; their purpose is usually to discover pattern in the data. Supervised learning algorithms are trained on labeled examples to generalize mapping functions between input and output; the expected output is known in advance and provided to the system for a given set of data and then the system is asked to predict the output of future set of data.

In MIR supervised learning techniques are exploited to fill the gap between low-level and high-level descriptors.

3.2.2.1 Regressor

In supervised learning a regressor is defined as a function $r(\cdot)$ that, given a set of input data $(\bar{x}_i, y_i)$ with $i \in \{1, \ldots, N\}$, minimize the mean squared error (MSE) between the predicted output $r(\bar{x}_i)$ and the desired output y_i.

Here it is possible to clearly identify the machine learning experience $E = (\bar{x}_i, y_i)$, the task T that is predicting y_i given $\bar{x}_i$ and the performance measure $P = \min \epsilon$

where

$$\epsilon = \frac{1}{N}\sum_{i=1}^{N}\left(r\left(\bar{x}_i\right) - y_i\right)^2$$

An example of linear regressor function that tries to fit a distribution is shown in figure 3.9.

A regressor is usually estimated by two steps: a training phase and a test phase. During the training phase a subset of all available data (*training set*) are provided to the system and a model is created. The performances of this model are then validated using another subset of available data (called *test set*). The block diagram in figure 3.10 shows the relation between these phases.

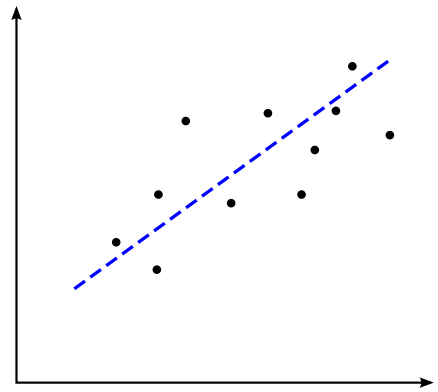


Figure 3.9: Linear regressor

Testing After the testing phase, it is important to evaluate the model obtained in the training phase. In order to compute a performance measure of a model, it must be put into operation. A test set is taken from the available data (known input, known output). This test set should be different from the training set to be a reliable measure.

It happens that a regressor that tries to approximate too much input data could memorize the data rather than learning from them. This issue is known as the *overfitting problem*.

One of the techniques that can be exploited to recognize and avoid overfitting is the k-fold cross validation. Using the k-fold cross validation, initial available data set is randomly partitioned into k subsets. For each one of these k subset it is trained a different model using that subset as test set and all the other $k-1$ subsets as training set. In the end, k error measures are available. These measures can be averaged to obtain a single error measure.

3.2.2.2 Neural Networks

The model of a regressor we have previously defined is a very powerful tool, but for complex problems it can be useful to perform a progressive model enhancement via a multiple rounds of training and testing instead of a one-time training and testing. This approach is known as multi-stage regression.

A neural network is a machine learning model that can be used for multi-stage regression or classification. We exploited this model to estimate the HLFs.

This model is based on the concept of artificial neuron; this component is composed by a set of weighted input synapses (a_i, w_i), an activation value z and an

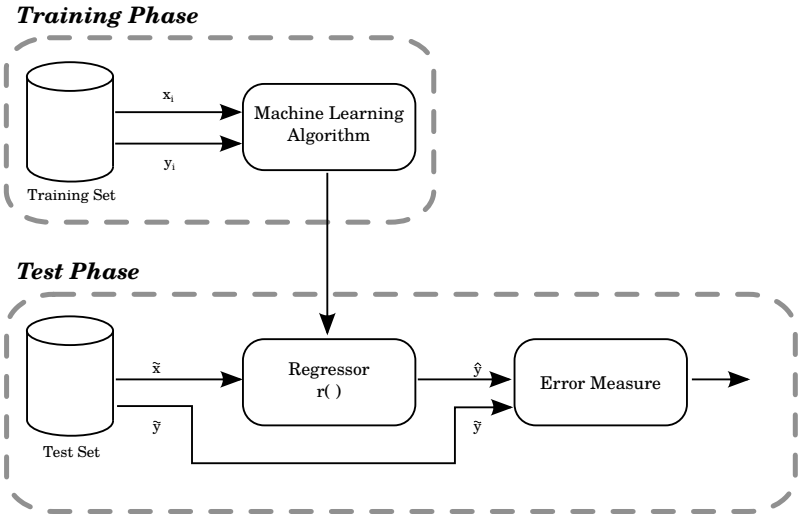


Figure 3.10: Supervised learning training and testing phases

activation function $g(\cdot)$. The output o is given by:

$$o = g\left(\sum_{i=0}^{N} w_i \cdot a_i\right)$$

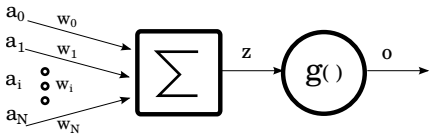


Figure 3.11: Artificial neuron

However, an artificial neuron, as above described, has many limitations. For instance, an artificial neuron can't be trained to compute the XOR function. A more complex model was elaborated, simply putting together many of these neurons.

Multi-layer perceptron A multi-layer perceptron is an artificial neural network, i.e. a set of neurons connected according to a topology. In a multi-layer perceptron it is possible to identify layers of neurons basing on the given topology; a layer is composed by all the neurons at the same distance from the input data.

Input layer layer of neurons that directly receives as input the data to process.

Output layer layer of neurons that provides the output of the network.

Hidden layer(s) all neurons that are not part of the input layer nor the output layer and produce intermediate values.

A multi-layer perceptron can be trained to produce multiple outputs from the same input. Therefore, it is possible to train multiple regressors using different outputs of the same perceptron.

3.2.3 High-Level Features

High-Level Features (HLFs) are descriptors that deals with the human perception of music. All of these features describe common concepts of music, but they still are subjective descriptors. Therefore, there is not a known direct correlation with the audio signal nor a universally-accepted mathematical formulation to describe them.

HLFs of a song are usually manually annotated. The value of a HLF for a given song is widely-accepted to be the average values given by a group of expert people.

It is always possible to extract LLFs from an audio file, but it is not always possible to have a group of experts available to manually annotate an audio file. Starting from the HLFs annotations available for a given set of audio files, it is possible to train a regressor to find an approximated relation between LLFs and a HLF. This regressor can then be used to obtain HLFs from LLFs for any audio signal.

In the next few lines we are giving a brief widely-accepted description of each HLF we are going to consider in our work. The value of each HLF is given by the answer in the $[0; 1]$ domain to the question asked next to each description.

3.2.3.1 Emotional Descriptors

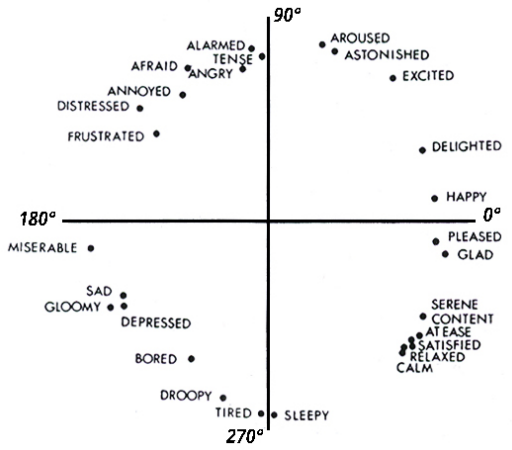


Figure 3.12: Original Circumplex Model of Affect by Russell [1]

There are two main approaches to model emotions. Emotion classification can be done by regression to emotion categories which consider every possible emotion as a different label or with a dimensional approach that classifies emotions along axes. The latter is the most common approach in music information retrieval and many other fields [48].

A consolidated model is the bidimensional *Circumplex Model of Affect* provided in 1980 by Russell [1]. Although other models has been proposed, exploiting other dimensions, like Thayer's energy-stress plan [49], the work of Russell is the most widely used in MIR. Therefore, our analysis was incentrated on the Russell's model. It defines two orthogonal concepts, here summarized.

Arousal Emotional activity quantifier. It ranges from soothing, calm to exciting, agitated. In figure 3.12 it goes from 270° to 90°.

Valence Emotional pleasantness quantifier. It ranges from highly negative to highly positive. In figure 3.12 it goes from 180° to 0°.

The original model is shown in figure 3.12.

3.2.3.2 Non-Emotional Descriptors

Non-Emotional descriptors are those HLFs that are not designed to capture the emotion of a song. These features describe other characteristics of the audio. We exploited part of the work made in [15] on human perception of couples of adjectives in table 3.2. In that work, users were asked to manually annotate audio sample in a scale from 1 to 9 where 1 means that the adjective on the left side best describes the sample and 9 means that the user found the right adjective to properly describe the audio sample.

Table 3.2: High-Level Features

Heavy	Soft
Compact	Void
Difficult Listening	Easy Listening
Dynamic	Static
Roughness	Armonic
Stuttering	Flowing
Clear	Dull
Not Groovy	Groovy

3.3 Music browsing: a survey

To address the music browsing problem it is important to set the focus on the user. Users were interviewed and they were asked few questions about their habits and their desires when they browse music. In this section, we introduce a survey that we conducted before the development of our project.

3.3.1 Structure of the survey

The survey is designed into three sections. The first section contains five questions and asks some personal information and something about their habits. The second section asks two questions about the needs and desires of the users. The third section investigates the interest of users on the aspects which project is focused on. Each section is showed up in a different web page to reduce the influence that the proposed solutions could have on the previous questions. All questions are closed-answers questions with, in some case, the ability to add an open feedback; here follows a list of all question asked to users; available options and answers will be discussed later in this chapter.

3.3.1.1 User's habits

- How old are you?
- How familiar you are with
  - computers
  - tablets
  - smartphones
- From which kind of device you listen to music?
- How do you organize your own music library?
- In which order you like to listen to music?

3.3.1.2 User's needs and preferences

- Would you like to browse your own music library in a map?
- How much do you believe it is useful / important
  - to have the ability to visualize the emotional representation of a song
  - to have the ability to visualize the speed of a song (BPM)
  - to have the ability to visualize the evolution over the time of the above characteristics
  - the aspect of video effects when you listen to music
  - the graphical interface when you chose what to listent to

3.3.1.3 User's reaction to a new proposal

- Imagine having the ability to view the songs as elements placed on a map. By what criteria would you place them?
- Being able to view the songs on a plane or in a space instead of a list, what would you prefer?
- With respect to the previous question, how you would like that the songs were displayed on the map?

3.3.2 Answers

Result of this survey are available online[2].

[2] `https://it.surveymonkey.com/results/SM-NNC7RLP/`

Public reached This survey was spread across social networks to reach up to a hundred of Italian people. The limit regarding the number of answers is imposed by the platform exploited to organize the survey and harvest results. With this survey we reached 95 people.

Percentage Results are normlized in percentage over the number of answers for each question, but since most of the questions allow multiple answers, the sum of the percentages can be higher than 100%.

3.3.2.1 User's habits

From table 3.3 and 3.4 it is possible to see that the public reaches is almost composed by young people with very good knowledge of computers and average familiarity with other kinds of device. Results shown in table 3.5 states that the personal computer is the most-widely-used device for listening to music and the traditional devices designed only to play music follows by many percentage points. Although almost everyone use PC as device for listening to music, table 3.6 denotes that there is no large trust in software when it comes to organize the music library. Some of them probably chose to do not organize at all their music library, but there is a good part of them that choose to organize the music library on their own; for those people who use personal computer but does not use any software to manage their music library, there is a need for better softwares. Results in table 3.7 shows that very few people make use of tags other than album or artist name, the percentage of people who relies (also) on automatically-generated playlists is less than one over five; there is high trust in context-based descriptors and just a little use of recommendation systems. Question 5 has also a field of free text to let users write other answers, more then one person declare to choose what to listen to according to the mood.

Table 3.3: Survey - question 1

How old are you?	< 18	$18 - 25$	$26 - 35$	$35 - 50$	> 50
%	5.26	71.58	17.89	5.26	0

Table 3.4: Survey - question 2

How familiar you are with	**computers**	**smartphones**	**tablets**
from 1 (none at all) to 5 (expert)	3.55	3.11	2.46

Table 3.5: Survey - question 3

From which kind of device you listen to music?	%
Personal Computer	90.53
MP3 player	68.42
Smartphone	55.79
CD player / Stereo	43.16
Tablet	9.47
Interactive TV / Videogame Console	5.26
Other	11.58

Table 3.6: Survey - question 4

How do you organize your music library?	%
Check artist, album tags are correct	64.13
Manage files in subfolders	59.78
Use a software to manage the music library	23.91
Divide songs in playlists	19.57
Check genre tag is correct according to my opinion	15.22

3.3.2.2 User's needs and preferences

In question 6 it is asked to the users if they would appreciate a map instead of list as browsing approach. The feedback is globally positive: more than two third of interviewed people declare to appreciate the idea and would like to experiment. The next question is more general and asks about the needs of the users. It is possible to see in table 3.9 that although the average results are quite balances, there is high variance among the answers. In particular, it is important to note that users globally recognized as very important the graphical interface when they are choosing what to listen to.

3.3.2.3 User's reaction to a new proposal

In this section is given a concrete proposal to users and it is asked them what would they do with such a software. In table 3.10 context based data such as genre and year are in high demand, but the content based proposal reach an important percentage of preferences. In this environment the concept of playlist is not so in demand as in a fully context-based environment. From open proposals has to be noticed requests of the artist's hometown, personalization and to combine different criteria.

Question 9 asks if users would prefer a simple environment or a colorful and enriched environment. Table 3.11 shows that they surely appreciate colors and details, but both the bidimension and the tridimensional solution are almost equally in demand.

Table 3.7: Survey - question 5

In which order you like to listen to music?	%
One album at time	43.48
I search songs by artist, album tags	36.96
Random over the whole music library	34.78
Random over a part of the music library	25.00
I created one (or more) playlist for each task I usually do	17.39
I let automatically-generated playlists choose for me	7.61
I search songs by genre tag	7.61
I search songs by year tag	1.09

Table 3.8: Survey - question 6

Would you like to browse your own music library in a map?	%
Yes, using a personal computer	62.11
Yes, using a smartphone	25.26
Yes, using a tablet	16.84
No, in any case I would prefer ordered lists	32.63

The last question investigates preferences concerning which kind of placeholder is most suitable for a song. From the provided options, users choose the simplest ones. Results in table 3.12 show that preferences for spheres and cubes are much higher than all other options. Open proposals contain more than one demand for customized placeholders.

It can be said that users strongly prefer simple placeholders and in an enriched environment.

Table 3.9: Survey - question 7

How much do you believe it is useful / important	−2	−1	0	+1	+2	**avg**
to have the ability to visualize the emotional representation of a song	22.11%	13.68%	24.21%	28.42%	11.58%	−0.06
to have the ability to visualize the speed of a song (BPM)	17.89%	11.58%	31.58%	25.26%	13.68%	+0.05
to have the ability to visualize the evolution over the time of the above characteristics	21.74%	7.61%	41.30%	22.83%	6.52%	−0.15
the video effects when you listen to music	29.47%	22.21%	25.26%	20.00%	3.16%	−0.55
the graphical interface when you choose what to listen to	16.84%	5.26%	30.53%	33.68%	13.68%	+0.22

Table 3.10: Survey - question 8

Imagine having the ability to view the songs as elements placed on a map. By what criteria would you place them?	%
Genre	76.92%
Mood associated to the song	43.96%
Year	25.27%
Custom playlists	18.68%
Speed (bpm)	17.58%

Table 3.11: Survey - question 9

Being able to view the songs on a plane or in a space instead of a list, what would you prefer?	%
enriched 2D plan, with colored details	45.74
enriched 3D space, with colored details	39.36
monochromatic 2D plan, without details	7.45
monochromatic 3D space, without details	7.45

Table 3.12: Survey - question 10

With respect to the previous question, how you would like that the songs were displayed on the map?	%
Spheres	44.94
Cubes	17.94
Little charachters	16.85
Face of a character	15.73
Little animals	4.49

Chapter 4

Methodology

We introduce here methods and techniques exploited in the development of this project.

We propose here AMBIF. AMBIF stands for Advanced Music Browsing Interactive Framework. This project is a framework that give the chance to the user to highly personalize his/her own music browser. In this chapter, we first introduce the framework structure and details; after that, in section 4.2 we discuss the process to obtain the music features that we use to represent music content.

4.1 Music Browsing Framework

In this section will be described our AMBIF project from its requirements, through its design, to its implementation structure.

4.1.1 Requirements

In this project we wanted to develop a framework that allows each user to create his/her own personalized music browser. **Personalization** is the key of this project. Our system should be easy to personalize at runtime. This system should allow the user to see every available descriptor, or a subset of them, for each audio file and, whenever it is available, their evolution over time.

It should also be designed as a framework and not a closed stand-alone project. Therefore, **extensibility** is the second most important requirement for this framework. In order to improve extensibility, open source technologies and open formats are preferred in every design choice.

We discussed in chapter 1 the benefits and malus of content-based and context-based browsing approaches. In chapter 2 we analyzed the state of the art and we saw how all mainstream music browser exploit a context-based approach; we also discussed the problems that content-based browsing systems have. This system should

exploit a **hybrid content-based and context-based** browsing approach keeping the good parts of both systems.

We also wanted to meet the preferences and needs of the users. As described in section 3.3, users prefer to browse a music library in an **enriched environment**. Acording to these people, attractiveness should improve with a **colorful interface**. Therefore, our system should be enriched with colorful details.

Finally, this system should also allow to reproduce an arbitrary audio file from the music library.

4.1.2 Design

In order to satisfy the previously described requirements, we decided to implement our system within a well-supported game engine. This choice will allow extensibility and portability of our framework. Indeed, game engine are designed to deal with modular software. Furthermore, they allow to develop a project within their environment and export it for many supported platforms. A game engine can also make easy to add amazing graphic effects. We discussed in the state of the art which game engine are more suitable and we decided to exploit Unreal Engine.

In this section we are going to investigate different aspects of the design process.

4.1.2.1 Data management

In this project we deal with audio files and related metadata. The music library composed by audio files will be stored in the local hard drive. All song-related metadata are stored in a separate file.

We have already discussed the importance of dealing with both content-based and context-based music descriptors. From a design and technological point of view, it makes no longer sense to discriminate between them. Content-based descriptors and context-based descriptors will be threaten in the same way. Furthermore, we are going to use the term *song dimension* to refer a song descriptor without the need to specify whether it is content-based or context-based. Discrimination can be done on the type of song dimension, e.g. whether it should be a numerical or a string value, but the origin of the song dimension itself does not make any difference.

Metadata information are split into different tables to preserve clarity and improve access performance. We predict a number of access reading these metadata greater than the writing access because the writing operations (browser configuration) will happen rarely while reading operations (metadata inspection) will happen more than once per program execution. Therefore, for the principle of spatial locality, access to metadata stored in the hard drive will occur by metadata class to facilitate metadata inspection. Headers of these tables are described in table 4.1.

During runtime it is most likely that access will occur element by element. In

Table 4.1: Stored metadata

(a) SongMetadata

Field	Description
ElementID	Song unique identifier
SongName	Title of the song
FilePath	Relative or absolute path to the audio file
Artist	Author
Length	total playtime (seconds)

(b) DimensionList

Field	Description
ID	Dimension unique identifier
FullName	Dimension complete name
Description	Text that shown to the user for this dimension
LowerBound	Lower bound of the dimension domain
UpperBound	Upper bound of the dimension domain
IsOrdinal	Data type
Replace MissingValues	Replace or do not replace missing values
DefaultValue	Default value used when replacing missing values

(c) ViewList

Field	Description
ViewID	View unique identifier
XDimension	Dimension to be shown on x axis
YDimension	Dimension to be shown on y axis
ZDimension	Dimension to be shown on z axis
HueDimension	Dimension to be shown as Color Hue
SatDimension	Dimension to be shown as Color Saturation

(d) Properties

Field	Description
ElementID	Identifier of the song
IdDimension	Identifier of the dimension
SummaryValue	Average value to be shown for this property
{time, value}*	Optional values to describe the evolution over time. *time* represents the initial time (in seconds) of the segment where this property has value *value*. More than one segment accepted.

order to improve the performance, some metadata are replicated in RAM with a different structure, according to the principle of spatial locality. Differences from the previously described structures are summarized in table 4.2.

A deeper analysis of these tables is given in the Main implementation details section 4.1.3.

Table 4.2: Runtime metadata

Field	Description
ElementID	Song unique identifier
SongName	Title of the song
FilePath	Relative or absolute path to the audio file
Artist	Author
Length	total playtime (seconds)
Properties	List of properties for each property there is a SummaryValue and an hashmap of values indexed on the timestamp of segment start

4.1.2.2 Game Engine paradigm

Developing a game with a game engine is like shooting a movie. There is one or more cameras and there are actors to put into one or more scenes. Only one scene at time can be active and usually no more than one camera is shown to the user. A user in this system is represented by a subclass of actor, named character. With an object-oriented programming language it is possible to make these actors interact between them; communication between objects is usually event-based.

For the purpose of our project, we will use only one camera, one character and many actors. The camera is fixed, attached to the character and therefore, its shot depends on the character position and rotation. The character will determine the movement of the user in the music library space.

All actors in the scene have visibility and interaction settings. It is possible configure each actor to be visible or invisible and which types of component it should interact with. Our character will be kept not visible and will interact only with other special actors that will act as map boundaries.

In our project we are going to have other actors: main other actors are those who concern the music library and those who contain the framework logic. Each element of the music library will be represented by a visible actor in the scene. In order to preserve the clarity of the interface, the framework logic will be implemented as invisible custom-defined actors placed in the scene.

Unreal Engine (UE) allows actor definition and specialization with the C++ class hierarchy system. Component prototyping programming style is strongly en-

couraged. UE provides also a visual-scripting language called Blueprint (BP); BP is also an object-oriented language designed to facilitate creation and edits of prototypes. Within UE it is possible to specialize a C++ class with a BP script. Although BP scripts do not reach the performance of C++ code, they are very useful during the development process because it is very easy to see the activation chain of the events in the system. In addition, BP scripts can reference actor instances and content elements directly from the scene or the content browser in the Unreal Editor. C++ is usually exploited for the core of the critical components and BP is most suitable to manage actor references and connect events to their handlers.

Content management Unreal Engine, like many other game engine development environments, discriminate between the artistic component and the programming component of a project. The artistic component includes textures, 3D models, audio components and it is managed through the Unreal Editor with import / export API. The programming component is managed with an Integrated Development Environment (e.g. MonoDevelop, Microsoft Visual Studio).

A game engine is optimized to build a self-contained package and every component is expected to be available at compile time. There is no reason to support dynamic content loading at runtime. Indeed, there is not any official support for this kind of operation. However, our project needs to load artistic and data content (audio files and metadata) at runtime. In section 4.1.3 will be discussed how we addressed this issue.

4.1.2.3 Unreal Engine project structure

The source code of this project is split into different packages. An overview of the usage relation between them is shown in figure 4.1.

A detailed analysis can be found in 4.1.3, here an overview of the available packages follows.

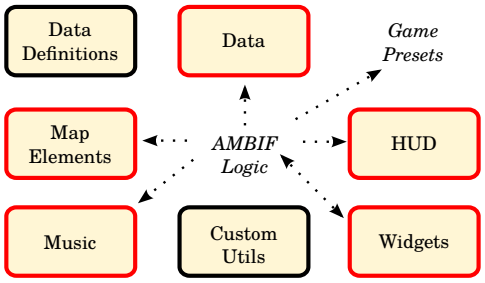


Figure 4.1: Project packages

In figure 4.1 we marked with a black contour the packages with **data definitions** and **custom utility functions**. These packages are used by almost all other packages and will not be discussed any longer. In the upper-right corner we positioned the **Game Presets**: with this term we meant to classify all those components that can be found in any project developed with a game engine; they are usually presets auto-generated from the engine and require very little customization. They are not located inside any package and this is the reason there is not any contour around this term. In the middle of

this schema there is the **AMBIF Logic**. Like the game presets, this part of the framework is not located inside any package, but it is left with global scope. AMBIF Logic contains some components needed to manage music browsing capabilities in this project; these components need to interact with all other packages to coordinate them. The rest of them are red-contoured to remark that they are packages. The **Data** package manages internal data buffers and provides input and output API. The **Map Elements** package is the package that contains and provides API to interact with the song-placeholder elements on the map. The **Music** package provides a basic music player able to load and play an audio file from the local hard-drive. This package does not contain any audio codec, it relies on the UE internal-audio decoder that supports only OGG/Vorbis audio files[1]. The **HUD** package contains the Head-Up Display graphical user interface class(es). This is the standard technology that was originally used in UE until version 4.5 to implement the graphical user interface; in this project we still exploit this system, but only to show tooltip messages when the mouse comes over elements on the map. In the end, the **widget** package contains adaptors to interact with the new widget technology we exploited to implement the GUI.

4.1.3 Main implementation details

The source code is available online[2]. Here we describe the structure of the framework class by class. We are going to discuss only the main classes; a simplified class dependency diagram is shown in figure 4.2. In this class diagram we used different colors to discriminate different class roles. Interfaces are colored with yellow, custom defined actors are contoured in blue, standard C++ classes are green and special game engine classes are violet. Data definitions are kept grey.

4.1.3.1 Logic

The AMBIF logic of the framework is composed by two main classes: *Logic Controller* and *Presentation Layer*. The logic controller provides API to handle all the framework events.

The Presesentation Layer is an actor that manages the logic of the visible elements in the scene. It also defines and manages which are the **Plottable Dimensions** available in the framework. A plottable dimension is a way to show render audio metadata (song dimensions) in the scene. At the moment there are five plottable dimensions:

Position X position of the song placeholder along the x axis.

[1] under Windows, Linux. Other target platforms may have different default-supported-audio encoding.

[2] Stefano Cherubin, ISPG - Image and Sound Processing Group, Politecnico di Milano, `https://github.com/skeru/ambif`

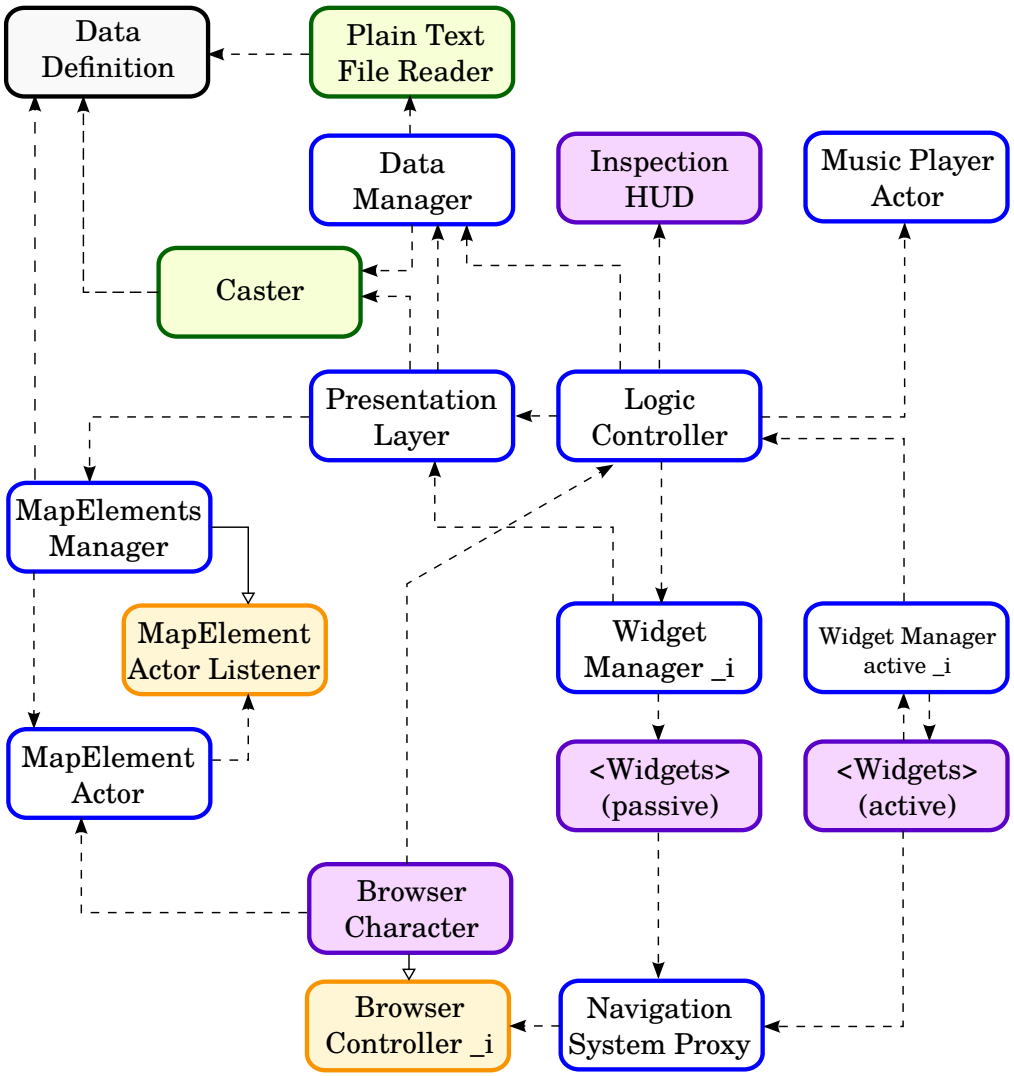


Figure 4.2: Simplified class dependency diagram

Position Y position of the song placeholder along the y axis.

Position Z position of the song placeholder along the z axis.

Color Hue hue parameter of the color of the song placeholder.

Color Saturation saturation parameter of the color of the song placeholder.

4.1.3.2 Data

The data package contains the *Data Manager* and some auxiliary classes. The main purpose of the data manager is to keep a buffer of available metadata and to interact with the database system to keep its buffer up-to-date. This actor also provide API to allow the logic to query the database.

The database needs to be kept as much flexible as possible. We chose to implement it with a simple set of plain-text-Coma-Separated-Values (CSV) files. In particular, there are four CSV files that our framework wants to read. Their headers are described in table 4.1.

Song Metadata table defines what is the content of the music library: each entry in this file represent a different song of the library. Here are also defined all the context-based metadata that will be available in the browser.

The *Dimension List* table contains the list of the song dimensions that are available for at least one song. Here are defined, for each song dimension, all the details needed to manage this dimension. In particular, it should be noticed that there is a field that specifies whether this dimension provides ordinal data or not; we reserved this field to specify what kind of data will be provided in this field. Our framework provides support for ordinal values as numbers (integer, fixed-point and floating-point numbers). Although it is not fully tested feature, we also support nominal genre values based on the Google Music Timeline[3] genre and subgenre hierarchy. For numerical and nominal values we specify which are the lower and the upper bound of the domain of this song dimension. This will allow us to cast every value from this domain to the proper plottable dimension domain. This cast function is provided by the *Caster* utility class and allows to cast different value types thanks to C++ function templates. In addition, once the song dimension bounds are set, this system allows to avoid problems with outlier values in metadata casting them to the bounds and no beyond them.

The *View List* table is designed for future extensions and contains the list of available view presets that could be loaded in the browser. As view we mean a function that for each plottable dimension determines which of the available song dimensions should be rendered on that plottable dimension.

[3]Google Research. Music timeline. `https://music-timeline.appspot.com/`.

The *Properties* table contains all the available content-based metadata. Each entry of this file represents a song dimension description for a specified song. Our framework requires that for each entry there must be at least one summary value for each song dimension; it is also possible to add, after the summary value, a list of one or more$< timestamp, value >$ tuples to specify that at the second *timestamp* a new segment is identified within the song and within that segment, this song dimension should be considered *value*.

The dynamic load of these files at runtime is provided by the *Plain Text File Reader* utility class. Here lies the part of the code that allows to perform direct access to the file system to solve the game engine limitation issue. It also provides a conversion to the appropriate data structures.

4.1.3.3 Map Elements

We define as *Map Element Actor* a song placeholder positioned in the scene. In our framework we derived a Blueprint script that specifies some graphic parameters for the placeholder, but it can be said that almost all the logic of this component is defined in the C++ class. We need to spawn (i.e. to instantiate and put in the scene) a different actor for each song of the music library. This actor holds in its internal state the song identifier and parameters that express the song dimensions. Every Map Element Actor checks whether it is interacting with other components or not. This actor can detect whether the cursor is over it, there was a mouse click on it or there is a collision with the character.

The *Map Elements Manager* references all the map element actors in the scene and provides API to spawn and to access them. It also subscribes itself to listen all the events detected from the song placeholders using a publish-subscribe model. Periodically the framework logic harvests this information and decide how to deal with these events.

4.1.3.4 Character

The *Browser Character* is a very complex class. This is a game engine preset that is linked to user controls. We decided to link here also the navigation system. Here are defined functions that allow the user to move across the space and change the camera position and rotation.

We also provided an interface called *Browser Controller_ i* to allow the Character to be used only as a control system and we encapsulated this system in an easy-to-reference actor named *Navigation System Proxy*. This proxy actor is directly used by the graphical interface component that manage the zoom command.

4.1.3.5 Music

This package contains the *Music Player Actor* that allows to play an audio file. This actor also provides a workaround to partially solve the dynamic content load issue that we previously described. It can dynamically load an audio file from the hard-drive to the memory space of the browser. If the load has gone fine, it can then exploit the internal decoder to decode and play the audio file. This is done through the instantiation and management of a special component provided by Unreal Engine.

This actor provides the following API:

- Load file from path
- Play
- Pause
- Stop
- Get playtime

This package still lacks of decoding capabilities to support multiple audio encoding formats.

4.1.3.6 HUD

The Heads-up Display system represented the state of the art in creating user interfaces with Unreal Engine until the end of 2014 when interface widgets were introduced as presets. HUD has two main limitations: first, only one HUD class can be active at time; second, the mouse events need to be manually detected.

For the purposes of this project we decided to use the HUD system to show tooltip labeles when the cursor is found to be over a given song placeholder. This is done in the *Inspection HUD* class: the logic controller provides the text to be shown and this class simply converts it into a graphic element to draw on screen.

In this class are also defined API to show arbitrary GUI components, for possible future uses.

4.1.3.7 GUI Widgets

Here there are the Graphical User Interface components. In this package we coded two classes: one abstract class (named passive) for receiving update orders from the logic controller and one class (named active) for translating events from widgets to the logic controller API. The passive abstract class is extended via a Blueprint script that implements all the events defined in this class.

In order to avoid possible infinite loops, we made every interface update completely passive; this means that an interface update will not trigger any other logic controller API call.

The actual GUI is implemented through Blueprint interface widget scripts, but all of those visual scripts refers to these C++ classes to interact with the framework logic.

4.1.4 How to create a music browser with AMBIF

AMBIF is a music browsing framework, that means it is possible to instantiate different music browsers with using framework. There is a difference between instanciating a music browser and personalizing a music browser. The personalization aspect is discussed in section 4.1.5, here we are going to introduce browser instanciation. This is also what we refer to when we say browser creation.

Once it has been understood the AMBIF data system, it is very easy to create a music browser. It is simply needed to provide a consistent database to the framework. The database is a set of plain-text-CSV files and the structure was given previously in this chapter. In particular, according to data specified in the *Dimension List* CSV table described in Stored metadata, our framework will provide to the user a different set of personalization options and therefore a different music browser.

Furthermore, once it has been understood how the AMBIF logic and the map element package work, it should be very easy to change the source code to implement more plottable dimensions to add more personalization possibilities.

4.1.5 Browsing experience

The browsing experience is inspired by a third-person videogame. User is fully concentrated on the space he/she is exploring. This space is represented by a flat landscape world surrounded by a sky at sunset.

As the program run, the user is assisted with a board that described the user commands, figure 4.3.

The user can move around the space using the keyboard and rotate the camera view using the mouse. The *mouse over a song placeholder* event will show a tooltip description of the song, the *mouse click on a song placeholder* event will show its details in the bottom panel, as shown in figure 4.4.

The audio will start playing by clicking the *Play* button in the upper-right corner of the screen. As the song starts, all other song placeholders will reduce their size

Figure 4.3: AMBIF help board

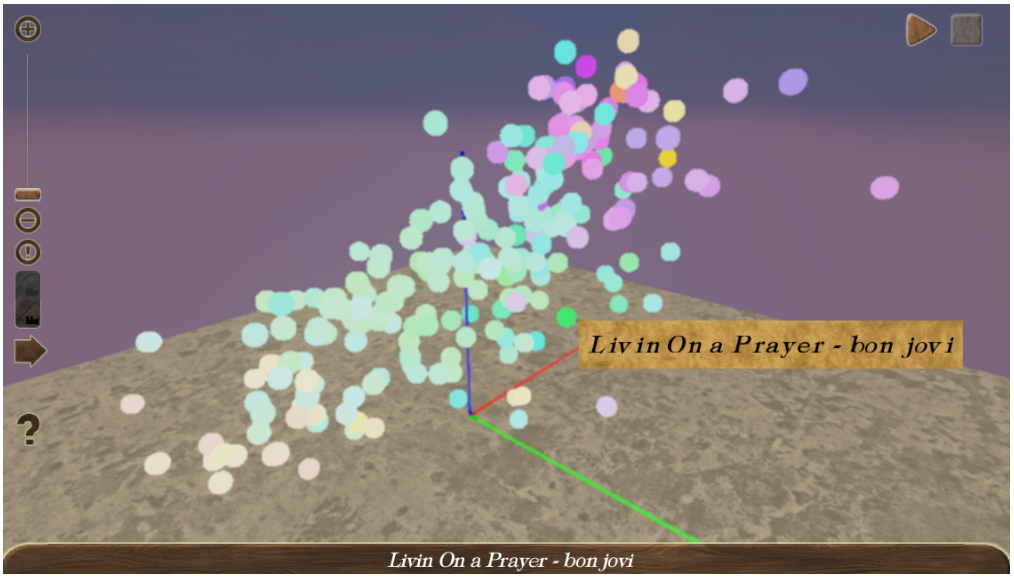


Figure 4.4: AMBIF mouse click

and fade their color to let the user identify the evolution of the song over the time. An example of this phase is shown in figure 4.5.

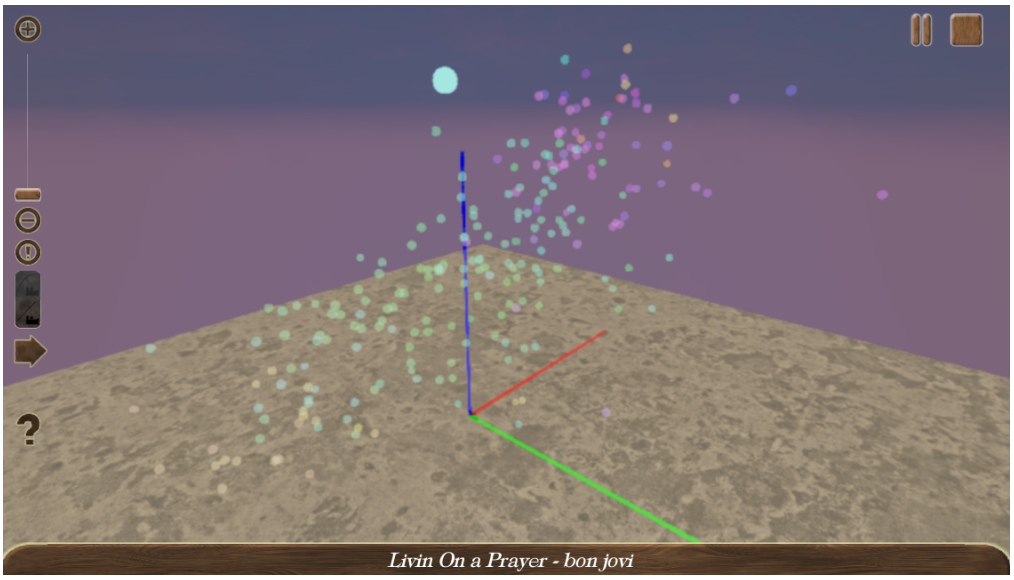


Figure 4.5: AMBIF song evolution

Personalization Given a music browser instance, it is possible to personalize the browsing experience at runtime. In the upper-left corner of the screen can be shown a *Dimension Menu* (hidden by default). Through this dimension menu, it is possible to understand the meaning of the plottable dimensions. For each one of these plottable dimensions, there is a drop-down menu that the user can change to select which song dimension he/she prefers to render on that plottable dimension. An example is shown in figure 4.6.

This menu allows has also other personalization capabilities. We are going to introduce some of them in the next few lines.

The user can choose to reduce the number of enabled plottable dimensions in the browsing space. He/she can select *Do Not Use This Dimension* from the available drop down menu for each plottable dimension. In this way, the user removes one plottable dimension (and therefore one song dimension) from the browsing space.

A special case of dimension reduction is the 2D mode. When the user changes to 2D mode, mouse controls change to reflect a top-down view. Furthermore, the Z axis dimension is removed from the browsing space. It is possible to change between 2D and 3D mode an unlimited number of times. It is also possible to do-not-use the Z axis dimension within the 3D mode.

The user can also choose to strengthen one song dimension by choosing to plot it on more than one plottable dimension. In this way he/she can browse the music library receiving a greater feedback from that song dimension.

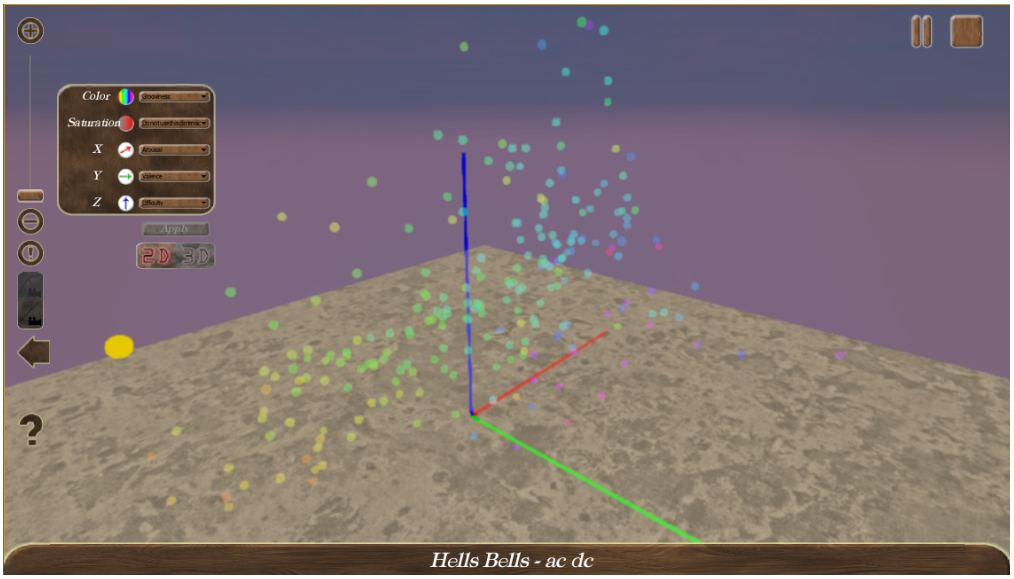


Figure 4.6: AMBIF Dimension change

4.2 Feature extraction

In this section we discuss how we extracted audio descriptors we exploited for this project. We worked on the MsLite dataset proposed in [50]. The MsLite dataset contains 240 full-length songs, a fifteen seconds excerpt for each full-length song and emotional descriptors annotations.

4.2.1 Low-Level Features extraction

LLFs were extracted using Vamp Plugins[4] and sonic-annotator[5]. In particular, we used two plugins libraries: Queen Mary[6] and libxtract[7]. From those plugins we chose a set of LLF to extract for each song. These feature are described in chapter 3 and here we report the respective plugin identifiers.

- vamp-libxtract:mfcc:mfcc
- vamp-libxtract:spectral_centroid:spectral_centroid
- vamp-libxtract:spectral_skewness:spectral_skewness
- vamp-libxtract:spectral_kurtosis:spectral_kurtosis
- vamp-libxtract:spectral_inharmonicity:spectral_inharmonicity

[4] Vamp Plugins, Centre of Digital Music at Queen Mary, University of London - `http://www.vamp-plugins.org/`

[5] `http://www.vamp-plugins.org/sonic-annotator/`

[6] QM Vamp Plugin set, Centre of Digital Music at Queen Mary, University of London - `http://vamp-plugins.org/plugin-doc/qm-vamp-plugins.html`

[7] `https://code.soundsoftware.ac.uk/projects/vamp-libxtract-plugins`

- vamp-libxtract:spectral_slope:spectral_slope
- vamp-libxtract:spectral_standard_deviation:spectral_standard_deviation
- vamp-libxtract:irregularity_k:irregularity_k
- vamp-libxtract:irregularity_j:irregularity_j
- vamp-libxtract:noisiness:noisiness
- vamp-libxtract:spread:spread
- vamp-libxtract:flatness:flatness
- vamp-libxtract:rolloff:rolloff
- vamp-libxtract:sharpness:sharpness
- vamp-libxtract:smoothness:smoothness
- vamp-libxtract:zcr:zcr
- qm-vamp-plugins:qm-chromagram:chromagram
- qm-vamp-plugins:qm-tempotracker:tempo

4.2.2 Hig-Level Features Regression

In order to obtain HLFs for the MsLite dataset we exploited a neural network regression system. This kind of supervised machine learning system is described in chapter 3. We built a training set using both emotional and non-emotional features. The list of HLFs we want to obtain is shown in table 3.2.

For training purposes we considered the emotional descriptors (Arousal and Valence) annotations provided by the MsLite dataset for the fifteen seconds excerpts and the non-emotional descriptors for the same excerpts annotated for the JANAS project [15]. Both emotional and non-emotional descriptors are annotated in the $[0; 1]$ domain.

We then used the model obtained in this way to annotate the full-length songs in the dataset. With respect to the figure 4.7, it is meaningful to point out that the *LLFs Extraction* block provides low-level features annotations for each second of the input audio file both in case of MsLite excerpts and MsLite full-length songs. Therefore, it is possible for the *Predictor* to provide estimated high-level features for each second of the full-length songs.

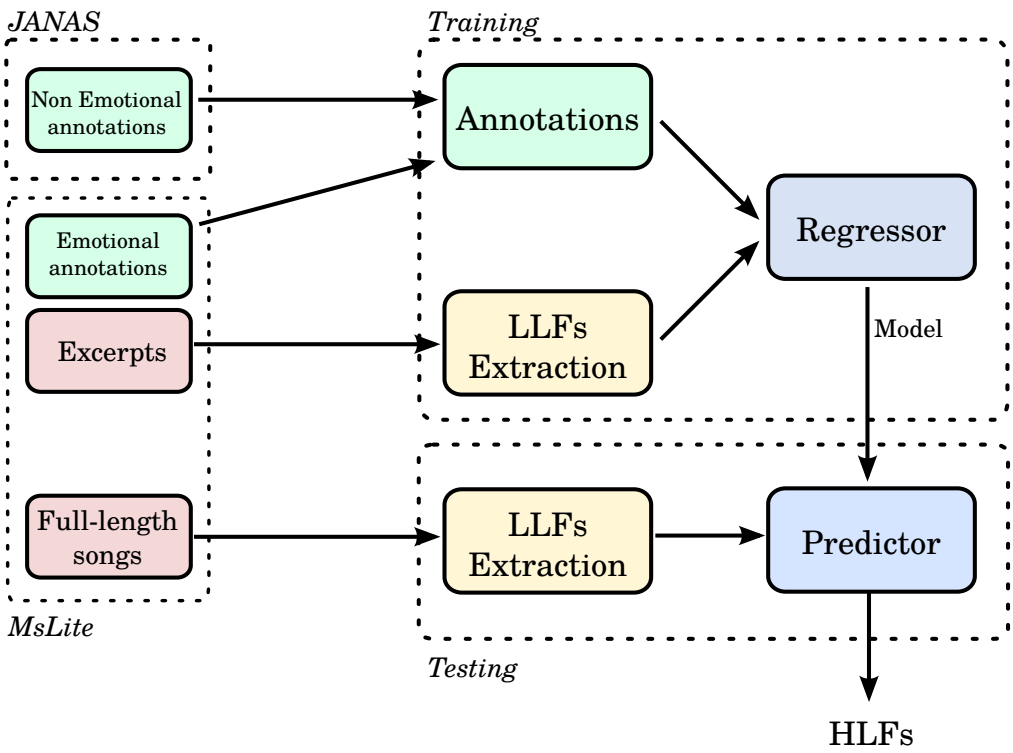


Figure 4.7: High-level features regression block diagram

Chapter 5

Experimental results

In this chapter will be discussed the evaluation process of this system. Our project is a music browsing framework. In order to evaluate our framework we created a music browser software with this framework and we asked different people to evaluate it. In the Demo section describe the music browser we created for evaluation purposes; in section 5.2 we discuss the evaluation method and results.

5.1 Demo

We created a music browser instance using AMBIF framework and we asked to a set of testers to use it.

5.1.1 Preparation

We used the MsLite [50] dataset and we analyzed those audio file to extract the Low-Level and Mid-Level features in table 3.1. We did this for each audio file, for each second. With the regression method described in 4.2, we then obtained High-Level Features for each song, for each second.

In order create a music browser instance, we created an entry in the Dimension List file (see table 4.1) for each HLF we considered.

We wanted to provide to our testers a clean visualization of the evolution over time of each descriptor therefore, we decided to keep the update frequency under a certain bound. Consequently, we increased the granularity of the segmentation; for each audio file we divided the song into segments of five seconds of length and we took the average value of each descriptor within that segment. In order to smooth this segmentation, segments are taken with 40% of overlap ratio (we start to compute the average from the last two seconds of the previous segment) to reduce the error at the boundaries of each segment.

These data were converted into AMBIF CSV format and loaded in the music browser.

5.1.2 Test phases

Each tester who took part of the test had to deal with the same situation.

> *Dear tester, I'm introducing you a novel music browser. Try it for a couple of minutes without any kind of help.*

This phase was meant to analyze how much AMBIF interface and approach is intuitive and self-explanatory. After five minutes, we ensured that each tester learned how to personalize the music browser by changing some dimensions of the browsing space. They were finally asked to complete a task within that editor.

> *Dear tester, please change the dimensions of the browser as you like them more and then find in this music library a song that fits well with your current mood.*

At the end of this task we asked each tester to fill out an online survey.

5.2 Final survey structure

After the test phase we asked our testers to fill out an online survey. In this section will be described the survey structure. Survey results will be discussed in section 5.3.

We chose a set of critera from the QUIM model [51] that we found meaningful to analyze and we asked our testers to give us a feedback by rating their experience with respect to those critera in a five-grade scale from -2 to +2. In this scale, the lower bound means *totally negative experience* and the upper bound stands for *very positive experience*.

5.2.1 Criteria

We here report the list of criteria and their description as they were exposed to our testers.

Time Behavior capability to consume appropriate task time when performing its function.

Resource utilization capability to consume appropriate amounts and types of resources when the software performs its function.

Attractiveness Capability of the software product to be attractive to the user.

Likeability User's perceptions, feelings, and opinion of the product.

Flexibility Whether the user interface of the software of the product can be tailored to suit users' personal preferences.

Minimal action Capability of the software product to help users achieve their task in a minimum number of steps.

Minimal memory load Whether a user is required to keep a minimal amount of information in mind in order to achieve a specified task.

Operability Amount of effort necessary to operate and control a software product.

User guidance Whether the user interface provides context-sensitive help and meaningful feedback when error occur.

Consistency Degree of uniformity among elements of user interface and whether they offer meaningful metaphors to users.

Self-descriptiveness Capability of the software product to convey its purpose and give clear user assistance in its operation.

Feedback Responsiveness of the software product to user inputs or events in a meaningful way.

Accuracy Capability to provide correct results or effects.

Completeness Whether a user can complete a specified task.

Fault tolerance Capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.

Readability Ease with which visual content can be understood.

Controllability Whether users feel they are in control of the software product.

Navigability Whether the user can move around in the application in an efficient way.

Simplicity Whether extraneous elements are eliminated from the user interface without significant information loss.

Familiarity Whether the user interface offers recognizable elements and interaction that can be understood by the user.

Load time How fast it responds to user inputs.

Appropriateness Whether visual metaphors in the user interface are meaningful.

5.2.2 Other questions

In addition to these parameters, we asked our testers to rate their experience with respect to their music browsing habits through these set of further questions:

1. How would you rate the usefulness to visualize evolution over time of the dimensions of a song?

2. Would you prefer a system like AMBIF instead of a list-based one?

We made available an optional text-field to let the testers provide a brief-text feedback.

5.3 Results evaluation

In this section we are going to discuss the experimental results obtained from our tests.

Before going deeper in the result analysis, we need to define evaluation metrics. For the purpose of our project, we decided to evaluate the usability of the music browser we instantiated with AMBIF and described the section 5.1.

Authors of [51] propose a set of usability factors for a software system and criteria to describe them. Criteria are specific parameters that can be measured. Factors are high-level descriptors of the usability of the system and summarize several criteria within them.

We rely our evaluation on a subset of those criteria that we found meaningful to analyze. In particular, we did not focus our evaluation analysis on criteria that can not be applied to a music browser, e.g. privacy and safety of the users. We also avoided to focus on resource utilization because we believe that this kind of measure requires a targeted test that we chose to leave for future analysis. According to [51], criteria contibute to define factors as shown in the sparse matrix in table 5.1.

We computed for each considered factor a value as the average value of the criteria it is affected by. It is possible to define a contribution weigth for any criteria q and usability factor m as $h_{q,m} = 0$ if q does not contribute to m and $h_{q,m} = 1$ if q contributes to m. The usability value $U_m \in [-2; +2]$ for each usability factor m can be written as

$$U_m = \frac{\sum_{q=1}^{Q} v_q \cdot h_{q,m}}{\sum_{q=1}^{Q} h_{q,m}}$$

with Q number of available criteria, v_q value that testers have given to critera q.

Results of the survey are available online[1] and are summarized in table 5.2 and 5.4.

[1] `https://it.surveymonkey.com/results/SM-Z8YW3867/`

Criteria	Factors								
	Efficiency	Effectivneness	Satisfaction	Productivity	Learnability	Trustfulness	Accessibility	Universality	Usefulness
Time behaviour	●			●					
Attractiveness			●					●	
Likeability			●						
Flexibility		●	●				●	●	●
Minimal action	●		●		●		●		
Minimal memory load	●		●		●		●	●	●
Operability	●		●			●	●		●
User guidance			●		●		●	●	
Consistency		●			●		●	●	
Self-descriptiveness					●	●	●	●	
Feedback	●	●						●	●
Accuracy		●							●
Completeness		●							
Fault tolerance						●			●
Readability							●	●	
Controllability						●	●	●	●
Navigability	●	●				●	●	●	
Simplicity					●		●	●	
Familiarity					●	●			
Load time	●			●				●	●
Appropriateness						●	●	●	●

Table 5.1: Usability factors and criteria

Usability criteria evaluation shows that testers gave us a very-high rating on *Likeability*, *Attractiveness*, *Accuracy* and *Completeness*. Therefore, we can say that our testers like our system and they believe it provides complete and exact information about the content. Although it is still positive, we have a low value on load time; this is probably given by the fact that our tests were performed on an obsolete machine[2]. We believe that further development can also enhance this criterion by exploiting parallelization techniques.

From those values of criteria we then computed the usability factors, as previously described in this section. As we can see from a quick overview of table 5.3, every usability factor is above the avarage value of the scale and some of them, more than others, are encouraging. In particular, the high *Satisfaction* value, together with a considerable result in *Learnability*, gives us a reason to believe that our system worth further developments.

[2]ASUS G73JH

5.3.1 Usability factors

In this section we explain the meaning of each usability factor as described in [51] and we discuss the values we have computed for them with respect to our project.

Values are shown in table 5.3.

Usability criteria	v_q
Time behaviour	+0.52
Attractiveness	+1.09
Likeability	+1.12
Flexibility	+0.70
Minimal action	+0.52
Minimal memory load	+0.30
Operability	+0.33
User guidance	+0.33
Consistency	+0.73
Self-descriptiveness	+0.76
Feedback	+0.33
Accuracy	+1.00
Completeness	+1.00
Fault tolerance	+0.42
Readability	+0.76
Controllability	+0.76
Navigability	+0.58
Simplicity	+0.48
Familiarity	+0.39
Load time	+0.21
Appropriateness	+0.76

Table 5.2: Criteria $v_q \in [-2; +2]$ obtained from the evaluation survey

5.3.1.1 Efficiency

Efficiency is defined as the capability of the software to consume a reasonable amount of resources with respect to the achieved effectiveness in given context.

This factor, with respect to our demo, is slightly positive. This result means that our testers are happy to see that it is possible to perform with the given resources.

5.3.1.2 Effectiveness

Effectiveness is the capability of the software to enable its users to achieve a specified task with accuracy and completeness.

Our testers led us to a high value of effectiveness. Therefore, we can say that our demo made with AMBIF was able to provide an accurate and complete content-based music browsing experience.

5.3.1.3 Satisfaction

This factor is purely subjective. Here are summarized the opinions of users about how they feel when using the software.

Our testers provided a cheering feedback with a high values in the corresponding questions in the survey. We are glad to say that our testers had a nice experience with our demo.

5.3.1.4 Productivity

Productivity captures the level of effectiveness achieved with a given amount of resources.

We asked our tester to complete just one task with our system therefore, the value we computed could be less reliable than values computed for other factors. However, considering results from the survey, we can say that the experience of our testers with respect to this factor is globally positive.

Usability factors	U_m
Efficiency	+0.349
Effectiveness	+0.723
Satisfaction	+0.627
Productiviy	+0.365
Learnability	+0.501
Trustfulness	+0.571
Accessibility	+0.584
Universality	+0.599
Usefulness	+0.498

Table 5.3: AMBIF usability factors
$U_m \in [-2; +2]$
computed as weighted average of criteria

5.3.1.5 Learnability

Learnability is the ease with which users can master the functionalities of the software.

This factor also captures the feel that users have when discovering features of the software. Our demo made with AMBIF provides a help board with basic-browsing controls. The rest of the interface was kept minimal and our testers intuitively went to discover their meaning. Their feedback is globally positive.

5.3.1.6 Trustfulness

This factor was originally designed to describe e-commerce systems. However, it can be applied also to other kinds of software. We computed this factor to find out whether our testers feel comfortable and would return using our music browser rather than other systems. The result we got is encouraging.

5.3.1.7 Accessibility

Accessibility is the capability of a software to be used by people with some types of disability. None of our testers had particular disability that compromised the use of the software. Therefore, it can be said that this value is an estimation that should be validated by targeted tests.

5.3.1.8 Universality

Universality is a factor that captures the capability of the software to be used by users with different cultural backgrounds. Our demo made with AMBIF was appreciated by English-speaking people who did already know the meaning of the song dimensions we proposed them while it was needed to explain the meaning of these song dimensions to some other testers. We believe that this problem can be

solved by building localized versions of the demo and it does not depend on the framework itself. Indeed, the global feedback is one of the highest in the table.

5.3.1.9 Usefulness

Usefulness is defined as the capability of the software to enable users to solve real problems in an acceptable way.

In our demo we asked our testers to solve a real problem by completing the task we gave them (*find a song that fits well with your current mood*). Most of our testers found easy to achieve the given task and the value for this factor is positive, as we did expect.

5.3.2 Other questions

In addition to the criteria and factors measurement, we asked our testers more direct questions.

First, we wanted to know how interesting our testers have found the capability of the software to show the evolution of the song dimensions over time.

Then, we asked whether they would use a system like those we introduced them with our demo or they still prefer a list-based music browser.

Answers to both quesitons were graded in a scale of five integer values from -2 to $+2$. Results are shown in table 5.4.

Question	average
How do you rate the usefulness of the capability to visualize the evolution over time of the dimensions of a song?	+0.61
Would you prefer a system like AMBIF instead of a list-based one?	+0.42

Table 5.4: User habits quesitons

Results are positive and we are positivly surprised to see that the average tester would like to use a music browser made with AMBIF.

5.3.3 Open comments

Comments leaved by our testers are globally cheering. There are some feature requests and a few complaints.

Complaints concern the lack of textual information within the browsing space and the low reactivity to operation like dimension change or the lack of some features. We have deepened this problems and we propose some solutions in chapter 6.

Requested features are:

Click-to-play testers suggested to play a song at first click in its placeholder and not waiting a click on the *Play* button. Classified as: *important*.

Labels for every ax and other plottable dimension, testers asked for a textual label on the screen or some kind of legenda. Classified as: *intermediate*.

Mixer some testers asked for a mixer to manage volumes inside the music browser. Classified as: *low priority - possible extension*.

First-Person a few testers asked to change the navigation system to reflect a first-person-browsing experience. This will be an interesting feature for some kinds of device, like Virtual-Reality devices. However, we do not consider this feature as an important improvement in the browsing experience. Classified as: *won't fix*.

Auto-update some testers asked to automatically apply a dimension change and not waiting a click on the *Apply* button. Classified as: *important*.

Smooth evolution some testers asked to smooth the animation of the evolution of the dimensions of the song while the song is playing. Classified as: *intermediate*.

Textures some testers asked to provide different graphical textures for the system. Classified as: *intermediate*.

Chapter 6

Conclusions

In this chapter we will review the work presented in this thesis and we will introduce some possible evolutions of this system.

6.1 Conclusions

In this thesis we proposed a novel music browsing framework. We designed it to be personal, interactive, flexible and attractive from a user point of view. It allows users to browse up to five song dimensions (X axis, Y axis, Z axis, Color Hue, Color Saturation) at the same time in a mixed content-based and context-based space.

Our work is defined in the music browsing field. It also involves aspects of music information retrieval, data visualization and software engineering. We wanted to address the problem of browsing a musical library where context-based metadata do not provide enough information to allow a proficient content browsing. We studied the concept of liquid music and we learned that it is not possible to statically model the content of the music. Therefore, also the music browsing software should adapt to the evolving structures and dimensions of music. We proposed a music browsing framework because we believe that a music browsing software with a built-in model of music could not be shaped to plenty satisfy all the possible preferences of users. With this framework it is possible to personalize the music browsing experience by leaving to users the decision of what song dimension is better to visualize and how.

We proposed a music browser instance made with AMBIF. We computed content-based metadata considering time-variant high-level descriptors for the whole music library, we asked to try it to 33 users and we evaluated its usability. The feedback was great and results are promising. We consider this a starting point for further development.

6.2 Future development

In this section we are going to discuss some possible evolutions of the AMBIF project as it was described in this thesis.

6.2.1 Reactiveness enhancement

Some of our testers asked for a more reactive interface and more controls to manage the audio output. Here we discuss what can be done with little or more effort.

6.2.1.1 Auto-update

Some of our testers suggested to decrease the number of click needed to perform some actions.

In particular, they suggested letting the music play at the first click on its placeholder. This feature can be easily implemented and will probably be present on all future versions of our framework.

Another suggested feature was the auto-update of the browsing space on every change of the dimension menu before clicking on the *Apply* button. This feature can also be easily implemented and will probably be present on all future versions of our framework.

6.2.1.2 Smooth movement

Some users asked to somehow hide the discontinuity caused by the movement of the song placeholder every time a new segment is identified for that song descriptor. This feature can be easily implemented, but it requires a fine analysis and related tuning of the update granularity parameters to avoid accuracy and consistency loss with short segments.

6.2.1.3 Audio management

There are various available presets for filtering and managing audio streams withing the Unreal Engine. It is possible to include some of them in AMBIF and let the user apply them to songs from the library.

Future versions of AMBIF may include an equalizer and more audio management options.

6.2.2 Audio codecs

We developed a music browsing framework that allows users to listen to songs while they are browsing the music library. The capability to load and play an audio

file is a very important feature in this system. However, as we described in section 4.1, technological limitations allowed us to dynamically load only audio files encoded as OGG/Vorbis. It is possible to extend the framework by adding decoding components that will interact between AMBIF source code and the Unreal Engine audio system code. This will allow to play also other audio file formats.

6.2.3 Queries

Although our system allows an immersive third-person browsing experience, it can be interesting to integrate in this framework a search engine.

This feature will allow users to apply filters to reduce the number of the song placeholders in the visible space for the sake of browsing clearness.

Possible extension in this direction have no limits. AMBIF provides a browsing space, but what it is shown in this space can be whatever result from any kind of query. It would be nice to integrate semantic queries, e.g. JANAS [15], as long as context-based queries, e.g. Nightingale[1], Quodlibet[2].

6.2.4 Playlist generation

Our system allows users to browse a semantic multidimensional space keeping a third-person perspective of the music library space. A possible extension can be an input tracking system that allows users to define an area or a path where lie placeholders of the songs he/she is interested into.

This feature was already implemented in other systems mentioned in the state of the art. However, none of them deals with space with more than two dimension. Indeed, they only work with bidimensional plans. Bringing this feature to a multidimensional space will be an interesting challenge.

6.2.5 Increase of the available dimensions

AMBIF allows to browse a music library within an up-to-five dimensions space. These dimensions are (position x, position y, position z, color hue, color saturation).

A few of our testers found not very intuitive color saturation or color hue space dimensions. There are also other dimensions in that space that can be explored, e.g. dimension of the placeholder, shape of the placeholder.

Our system can also relies on the power of one of the most advances videogame engine, that is Unreal Engine. This means that graphical enhancemens can rely on the underlying engine, with a large community support and free access to the source code.

[1] Nightingale Community. Nightingale web player. `http://getnightingale.com`.
[2] quodlibet. `https://code.google.com/p/quodlibet/`.

6.2.6 Support for other kind of devices

AMBIF is a framework developed within Unreal Engine and this allows our project to be exported to different kinds of devices. It is possible to tune the navigation control system to allow an easy browsing with:

- Virtual Reality devices (Oculus Rift)
- HTML5
- Mobile phone

Bibliography

[1] James A Russell. A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161, 1980.

[2] Jean Baptiste Joseph Fourier. Théorie analytique de la chaleur. *Firmin Didot, père et fils, Paris*, 1822.

[3] Zygmunt Bauman. *Liquid times: Living in an age of uncertainty*. John Wiley & Sons, 2013.

[4] George Tzanetakis, Manjinder Singh Benning, Steven R. Ness, Darren Minifie, and Nigel Livingston. Assistive music browsing using self-organizing maps. In *Proceedings of the 2Nd International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '09, pages 3:1–3:7, New York, NY, USA, 2009. ACM.

[5] Zhouyu Fu, Guojun Lu, Kai Ming Ting, and Dengsheng Zhang. A survey of audio-based music classification and annotation. *Multimedia, IEEE Transactions on*, 13(2):303–319, April 2011.

[6] François Pachet, Daniel Cazaly, et al. A taxonomy of musical genres. In *RIAO*, pages 1238–1245, 2000.

[7] Gerwin de Haan, Michal Koutek, and Frits H. Post. Towards intuitive exploration tools for data visualization in vr. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '02, pages 105–112, New York, NY, USA, 2002. ACM.

[8] Anne Morgan Spalter, Philip Andrew Stone, Barbara J Meier, Timothy S Miller, and Rosemary Michelle Simpson. Interaction in an ivr museum of color: Constructivism meets virtual reality. *Leonardo*, 35(1):87–90, 2002.

[9] A. Van Dam, A. Forsberg, D.H. Laidlaw, J.J. LaViola, and R.M. Simpson. Immersive vr for scientific visualization: a progress report. *Computer Graphics and Applications, IEEE*, 20(6):26–52, Nov 2000.

[10] Bireswar Laha and Doug A Bowman. Identifying the benefits of immersion in virtual reality for volume data visualization. March 2012.

[11] Apple Inc. iPod + iTunes Timeline. `https://www.apple.com/pr/products/ipodhistory/`. [Online; accessed 27-February-2015] iTunes and iPod are trademarks of Apple Inc.

[12] Elias Pampalk and Masataka Goto. Musicrainbow: A new user interface to discover artists using audio-based similarity and web-based labeling. In *ISMIR*, pages 367–370. Citeseer, 2006.

[13] Elias Pampalk and Masataka Goto. Musicsun: A new approach to artist recommendation. In *ISMIR*, pages 101–104, 2007.

[14] Jim Wolfe. Ipad/tablet application: Discovr apps. *Mathematics and Computer Education*, 47(2):152, 2013.

[15] Michele Buccoli, Massimiliano Zanoni, Augusto Sarti, and Stefano Tubaro. A music search engine based on semantic text-based query. In *Multimedia Signal Processing (MMSP), 2013 IEEE 15th International Workshop on*, pages 254–259. IEEE, 2013.

[16] Yi-Hsuan Yang and Homer H. Chen. *Music Emotion Recognition*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2011.

[17] Anita Shen Lillie. *MusicBox: Navigating the space of your music*. PhD thesis, Massachusetts Institute of Technology, 2008.

[18] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1-3):37 – 52, 1987. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

[19] Yeuvo Jphonen. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1995.

[20] Elias Pampalk, Andreas Rauber, and Wolfdieter Merkl. Content-based organization and visualization of music archives. In *Proceedings of the ACM Multimedia Conference (ACM MM02)*, pages 570–579, 2002.

[21] Michael Dittenbach, Robert Neumayer, and Andreas Rauber. Playsom: An alternative approach to track selection and playlist generation in large music collections. In *In Proc. 1st Intl. Workshop on Audio-Visual Content and Information Visualization in Digital Libraries (AVIVDiLib 2005)*, 2005.

[22] Robert Neumayer, Michael Dittenbach, and Andreas Rauber. Playsom and pocketsomplayer, alternative interfaces to large music collections. In *ISMIR*, pages 618–623, 2005.

[23] Karl-Ingo Friese, Marc Herrlich, and Franz-Erich Wolter. Using game engines for visualization in scientific applications. In Paolo Ciancarini, Ryohei Nakatsu, Matthias Rauterberg, and Marco Roccetti, editors, *New Frontiers for Entertainment Computing*, volume 279 of *IFIP International Federation for Information Processing*, pages 11–22. Springer US, 2008.

[24] Martin Meister and Martin Boss. On using state of the art computer game engines to visualize archaeological structures in interactive teaching and research. *BAR INTERNATIONAL SERIES*, 1227:505–509, 2004.

[25] Qing Xie, Lin Ge, and Zhenyi Wang. Design of the virtual reality roam system for ancient city. In *Systems and Informatics (ICSAI), 2014 2nd International Conference on*, pages 119–123, Nov 2014.

[26] ANTHONY Rigby, Ken Rigby, and Mark Melaney. Virtual archaeological environments generated in avayalive engage. *Archeomatica*, 5(3), 2014.

[27] M. Herrlich. A tool for landscape architecture based on computer game technology. In *Artificial Reality and Telexistence, 17th International Conference on*, pages 264–268, Nov 2007.

[28] Ruzinoor Che Mat, Abdul Rashid Mohammed Shariff, Abdul Nasir Zulkifli, Mohd Shafry Mohd Rahim, and Mohd Hafiz Mahayudin. Using game engine for 3d terrain visualisation of gis data: A review. *IOP Conference Series: Earth and Environmental Science*, 20(1):012037, 2014.

[29] Madeleine Manyoky, Ulrike Wissen Hayek, Kurt Heutschi, Reto Pieren, and Adrienne Grêt-Regamey. Developing a gis-based visual-acoustic 3d simulation for wind farm assessment. *ISPRS International Journal of Geo-Information*, 3(1):29–48, 2014.

[30] Mitchell J Bott and Mikel D Petty. implementing a physics-based model of crowd movement using the unreal development kit. *Journal of Gaming & Virtual Worlds*, 6(3):275–296, 2014.

[31] C.S. S. Burrus and Thomas W. Parks. *DFT/FFT and Convolution Algorithms: Theory and Implementation.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1991.

[32] Dimitris G. Proakis, John G; Manolakis. *Digital signal processing: principles, algorithms, and applications.* Prentice-Hall International, Inc., 3rd edition, 1996.

[33] N. Ahmed, T. Natarajan, and K.R. Rao. Discrete cosine transform. *Computers, IEEE Transactions on*, C-23(1):90–93, Jan 1974.

[34] Hyoung-Gook Kim, Nicolas Moreau, and Thomas Sikora. *MPEG-7 audio and beyond: Audio content indexing and retrieval*. John Wiley & Sons, 2006.

[35] Geoffroy Peeters. A large set of audio features for sound description (similarity and classification) in the cuidado project. Technical report, Ircam, Analysis/Synthesis Team, 1 pl. Igor Stravinsky, 75004 Paris, France, apr 2004. Online, accessed 19 mar 2015 `http://recherche.ircam.fr/equipes/analyse-synthese/peeters/ARTICLES/Peeters_2003_cuidadoaudiofeatures.pdf`.

[36] Geoffroy Peeters, Bruno L. Giordano, Patrick Susini, Nicolas Misdariis, and Stephen McAdams. The timbre toolbox: Extracting audio descriptors from musical signals. *The Journal of the Acoustical Society of America*, 130(5):2902–2916, 2011.

[37] G. von Bismarck. Sharpness as an attribute of the timbre of steady sounds. *Acta Acustica united with Acustica*, 30(3):159–172, Mar 1974.

[38] Nader Sawalhi and Robert B Randall. The application of spectral kurtosis to bearing diagnostics. In *Proceedings of ACOUSTICS*, pages 393–398, 2004.

[39] Anders Askenfelt and A. S. Galembo. Study of the spectral inharmonicity of musical sound by the algorithms of pitch extraction. *Acoustical Physics*, 46:2, s. 121-132, 2000.

[40] Hartmut Meister, Markus Landwehr, Ruth Lang-Roth, Barbara Streicher, and Martin Walger. Examination of spectral timbre cues and musical instrument identification in cochlear implant recipients. *Cochlear Implants International*, 15(2):78–86, 2014. PMID: 24597635.

[41] Stephen McAdams, James W. Beauchamp, and Suzanna Meneguzzi. Discrimination of musical instrument sounds resynthesized with simplified spectrotemporal parameters. *The Journal of the Acoustical Society of America*, 105(2):882–897, 1999.

[42] Don Michael Randel. *The Harvard dictionary of music*, volume 16. Harvard University Press, 2003.

[43] Matthew EP Davies and Mark D Plumbley. Context-dependent beat tracking of musical audio. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15(3):1009–1020, 2007.

[44] Daniel PW Ellis. Beat tracking by dynamic programming. *Journal of New Music Research*, 36(1):51–60, 2007.

[45] Tom M Mitchell. Machine learning. *Burr Ridge, IL: McGraw Hill*, 45, 1997.

[46] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[47] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1 edition, January 1996.

[48] JungHyun Kim, Seungjae Lee, SungMin Kim, and Won Young Yoo. Music mood classification model based on arousal-valence values. In *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, pages 292–295, Feb 2011.

[49] Robert E Thayer, J Robert Newman, and Tracey M McClain. Self-regulation of mood: strategies for changing a bad mood, raising energy, and reducing tension. *Journal of personality and social psychology*, 67(5):910, 1994.

[50] Youngmoo E Kim, Erik M Schmidt, and Lloyd Emelle. Moodswings: A collaborative game for music mood label collection. In *ISMIR*, volume 8, pages 231–236, 2008.

[51] Ahmed Seffah, Mohammad Donyaee, RexB. Kline, and HarkiratK. Padda. Usability measurement and metrics: A consolidated model. *Software Quality Journal*, 14(2):159–178, 2006.