ETH Semester Thesis

MusicExplorer Amarok2 Plug-in

- Bringing MusicExplorer to the user -

Michael von Känel

Advisors:
Olga Goussevskaia and Michael Kuhn

Zürich, December 2008

Abstract

Many music lovers nowadays possess too big music collections to be organized manually. MusicExplorer is one way to cope with this problem by organizing songs in a high dimensional space. However there is no way to use this feature in an actual player. This problem is resolved during this work by developing a plugin for Amarok2. This plugin is able to generate dynamic smooth playlists using MusicExplorer coordinates and bring MusicExplorer to the user. The plugin is published on KDE-Apps.org receiving positive user feedback.

Table of Contents

1. Motivation....2
1.1. Media Player....2
2. Script Organization....3
2.1. QtScript....3
2.2. Code Overview....3
2.3. Download....4
2.4. KD-Tree....5
2.5. GUIs....7
3. Playlist generation....8
3.1. Song reordering....8
3.2. Interpolation....8
3.3. Autonomous behavior....9
3.4. Avoid songs....9
4. User feedback....10
5. Discussion and further work....11
5.1. Provide better coordinates....11
5.2. Playlist optimization....12
5.3. Avoidance of songs....13
6. Conclusion....14
7. Appendix: Practical problems and opinion....15
7.1. Coding problems encountered....15
7.1.1. Bugs....15
7.1.2. Changing environment....16
7.2. Personal Conclusion....16

1. Motivation

Many music lovers have big media collections nowadays. It is relatively easy to collect huge databases, but organizing these is quite complex. Just looking at genre tags in mp3 files for example gives us a very fuzzy discretization of music. Besides the fact that many songs are difficult to categorize. It is even more complex since most generated mp3 files do not even have manually set genre tags. Besides meta-data analysis there are many approaches how to organize the music: Some analyze the acoustics itself and others are more subjective and based on the opinion of the user. However, analyzing acoustics often does not reflect the listening experience and leads to poor results in replacing a human DJ for example. Data that well reflects the subjective perception of individual users, however, faces the problem that it might perform badly for the whole set of music listeners.

MusicExplorer[1] is one way to organize the music. It is a project which places all songs in a high dimensional space where similar songs lie near to each other. This database was generated using last.fm, a music portal listing users listening patterns. The problem that it might be too subjective is addressed by using a big number of these patterns. Individual user behavior gets lost in this big amount of data.

At start time of this work there existed a Java applet at musicexplorer.org, which was able to generate text-based playlists given a set of songs. It works as a proof of concept but does not help to bring this database to the real user. One could think of creating a whole new player based on MusicExplorer but most users will still go on using their favorite music player as before. Therefore the idea came up to actually implement this as a plugin for a popular music player, which is done in this work.

1.1. Media Player

There are several players available, which are used to organize big music collections. The most famous ones for Windows are probably the Windows Media Player and Winamp. On Apple computers it is definitely iTunes. Since none of those players actually work natively on Linux systems (and I am a Linux user), I had to look for a more platform independent music player. VideoLanClient (VLC) is a widely used video/audio player, which works on various operating systems, but sadly comes without collection browser. On KDE based distributions Amarok is the widely used choice. It works also for surfaces other than KDE. At the beginning of this work, most KDE users were using Amarok 1.4, which has an easily usable interface to communicate with scripts (DCOP). Scripts could therefore be written in any script language (like Ruby, Python, etc.). However there is no way to port Amarok 1.4 to other operating systems and the even bigger drawback was that the developers stopped the development for Amarok 1.x and started coding the sequel. Amarok2 is based on Qt and therefore portable to any other Qt supporting operating system. Even though the first beta version of Amarok2 was just released, while taking a decision for a player, the scripting interface seemed to be pretty well developed and there were already 2-3 little demo plugins available.

1 “From Web to Map: Exploring the World of Music”, Olga Goussevskaia, Michael Kuhn, Michael Lorenzi, and Roger Wattenhofer, ETH Zürich, 2008

	Amarok 1.4	Amarok 2-beta
+	• Stable • widely distributed • easy to code plugins (Ruby, Python, etc.) • many example plugins/documentation	• future oriented • powerful script language (QtScript) • Win/OSX versions • D-Bus[2] interface • active community
-	• will be replaced soon • only KDE • uses outdated DCOP bus.	• (nearly) no documentation • still under heavy development • early beta → many bugs • difficult debugging

Table 1: Comparison of Amarok1.4 vs. Amarok2-beta at start of this work.

In the end I decided to implement a plugin for Amarok2, mainly because of the portability, the opportunity to create one of the first scripts for this application and because Amarok1.4 will be replaced soon.

2. Script Organization

The main goal was to bring MusicExplorer to the user. The script should therefore be as easy to use as possible. Its purpose was to generate playlists as described in a separate chapter. The script should nicely integrate into the Amarok appearance such that the user immediately feels comfortable.

2.1. QtScript

QtScript[3] is a scripting engine of the Qt toolkit, used as the single scriptable language in Amarok2. The main reason for this is, besides the power of portability to very different computers, that it works without additional packages. The language is based on the ECMAScript standard. ECMA is also the standard for JavaScript, so QtScript has a similar syntax, but without all the DOM functions of JS. QtScript is shipped with a lot of powerful libraries, so most stuff you can do in Qt is somehow also possible in QtScript. Therefore, besides the core functionality, one can create GUIs (even OpenGL), talk to SQL databases, generate XML parsers and write Web servers.

2.2. Code Overview

Amarok uses an internal MySQLe database. Every song in the collection is first parsed and its meta data is stored to this database. Therefore it is sufficient to keep track of this database in order to fetch the MusicExplorer coordinates of every song the user has. Additionally it is also possible to add its own tables to the same database. So one has fast access to the coordinates and all Amarok data is stored at the same location. All communication has to run over an internal function of Amarok, because there are no drivers to access the database from outside Amarok or from the script directly.

2 D-Bus specification and source, http://www.freedesktop.org/wiki/Software/dbus
3 http://doc.trolltech.com/4.4/qtscript.html

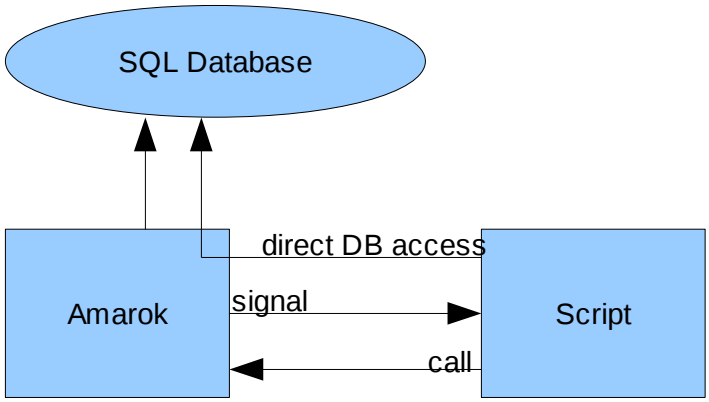

Illustration 1: General overview of communication between Amarok, the Script and the DB.

The first time the script is started or when the user adds new songs to his collection, the script will automatically fetch all missing coordinates. While the script is running it holds all songs with coordinates in memory and keeps track of changes made to the playlist by connecting to several signal handlers, provided by Amarok. Communication with the user is mainly done over a separate window.

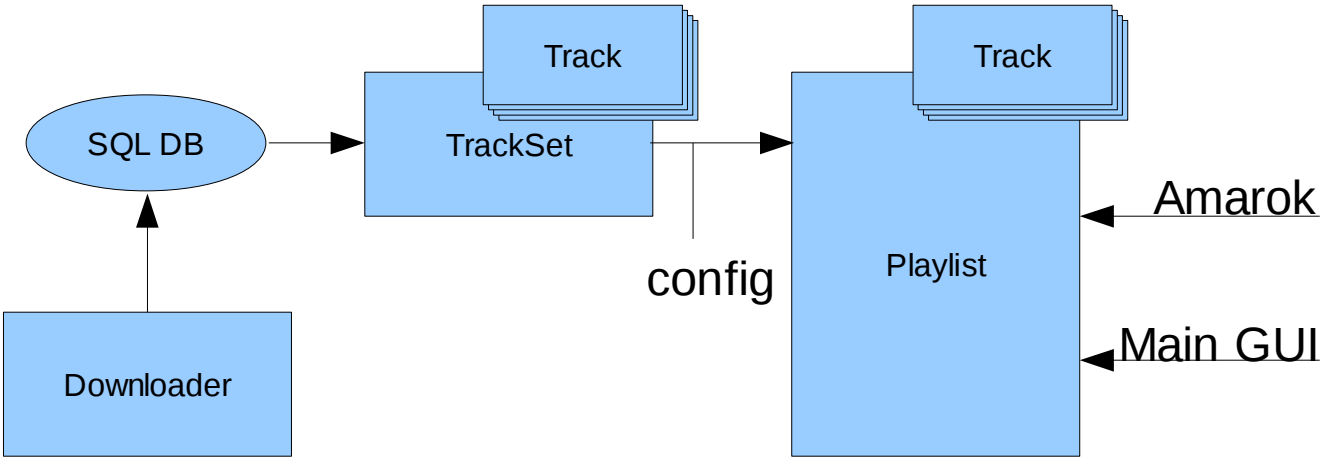

Illustration 2: Script internal class overview.

A TrackSet provides functions to get the desired Track. Implementation can be a KD-Tree (as discussed in chapter 2.4) or a simple list. The user can configure several aspects over a separate window in order to generate different playlists.

2.3. Download

The Downloader is triggered every time the script starts. It determines the songs for which no coordinates were looked up online. For every track, the server returns an XML file, containing the coordinates or an error message. To download, a function which was designed to download lyrics is abused. The advantage is that caching proxy settings and alike is already implemented. The drawback on the other hand is that we have to download the song coordinates sequentially, because this function calls back to a single global function with only the response as argument.

2.4. KD-Tree

During playlist generation one often has to find the nearest neighbor to a given song or point. The standard way to find near neighbors in a high dimensional space is to use a KD-Tree[4]. It acts as a binary tree in space. Inserting and finding nodes has complexity O(log(n)), where n is the number of elements in the tree . There were some libraries available for the most common languages like Java and C. Unfortunately there were no libraries for QtScript or even JavaScript to find. So a new KD-Tree library was coded heavily based on an existing Java library[5]. The functionality of the own implementation was verified against the mentioned Java implementation.

In the following scenario two data structures are tested under real conditions. The first one is just a greedy approach which loops through the whole list and takes the nearest neighbor. This is compared to the mentioned KD-Tree. In this test case, randomly generated track-objects located at uniformly distributed random coordinates are inserted.

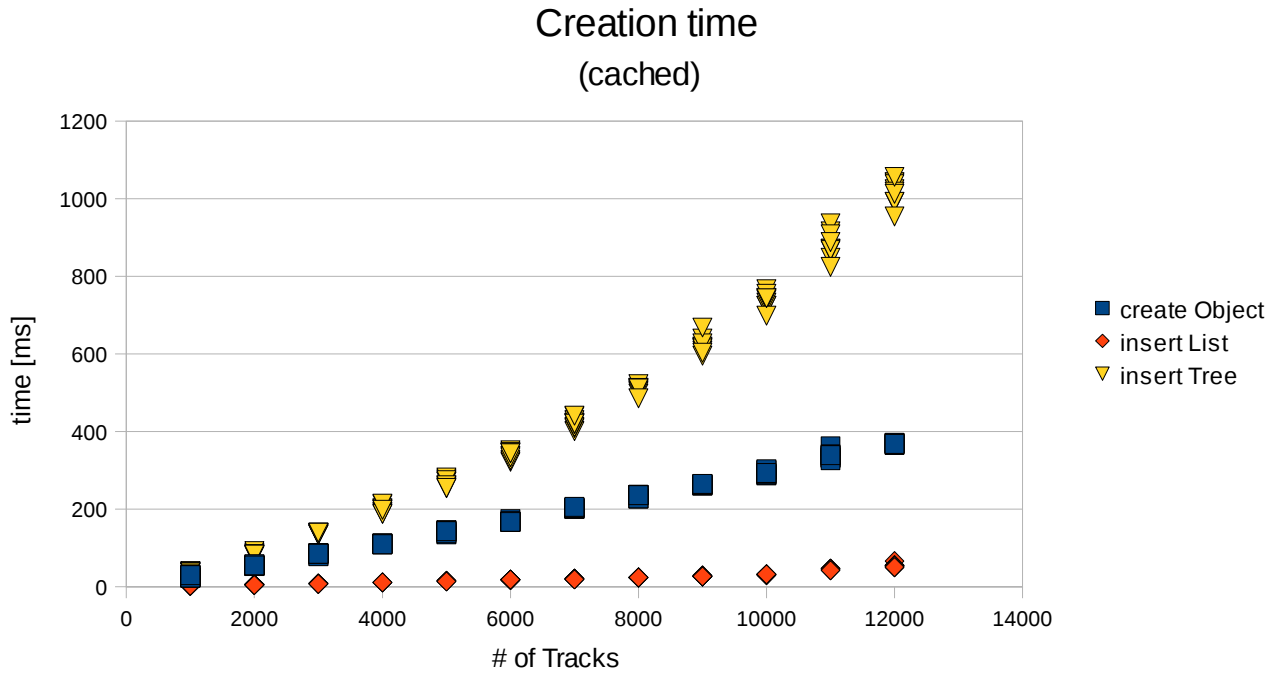

Illustration 3: Time to create a new track object, insert it in a KD-Tree and push it to a list.

Inserting new tracks to the playlist goes with O(n log(n)) as expected. The time it takes to generate the tree at startup is feasible even for large collections. Pushing an object to an array is negligibly fast.

4 http://en.wikipedia.org/wiki/Kd-tree, kd-tree – Wikipedia (visited 8.12.08)
5 http://www.cs.wlu.edu/~levy/software/kd/, KD-Tree implementation of Simon Levy, Bjoern Heckel in Java.

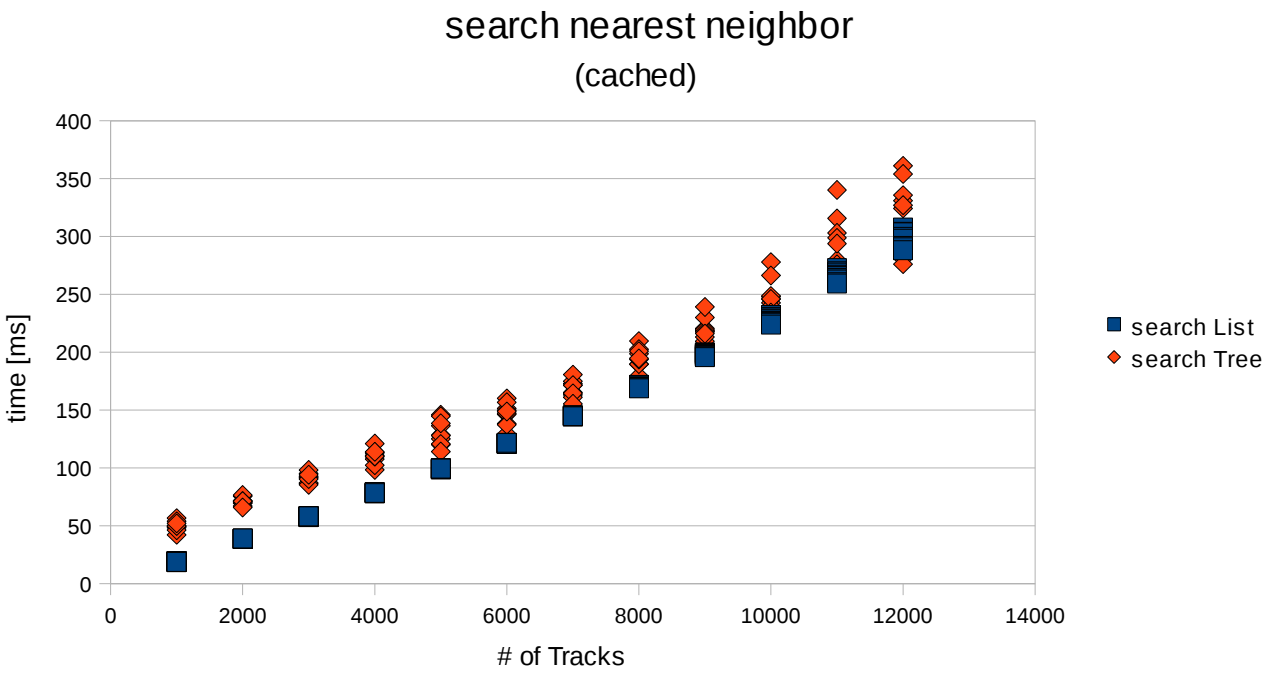

Illustration 4: Finding the nearest neighbor in a KD-Tree and with the greedy approach. Every dot is the average over 10 measurements, so the variance is higher for a single event in the KD-Tree.

In finding the nearest neighbor, a big advantage over the greedy approach is expected. However the result is very surprising, as one can see in Illustration 4. For small collections the KD-Tree had a bigger overhead and it took longer to find the nearest neighbor as expected. However, even with an increasing number of nodes the search time increased linearly like the greedy approach.

The functionality is verified by assuring that it takes log(n) recursions in order to find the correct nearest neighbor. Thus I assume that the strange behavior exists because QtScript is interpreted and not executed directly. So every line of code is an additional overhead. For the dummy approach, caching seems to be very effective, since we only loop through very few lines of code and do not have any recursion. Additionally very basic operations such as cloning of an array are natively missing in QtScript. Also the increasing stack size might cause this problem as it might need to be extended for every recursion step.

Due to these problems we have decided to use the dummy approach, because its easier to implement additional constraints. Constraints could be to only return a near neighbor if it is not in a given list, is from another artist, has at least a given distance and so on.

The topic of the KD-Tree described in this chapter could also be understood as an example problem encountered during this work. If you are interested in even more practically oriented problems you might take a look at the appendix.

2.5. GUIs

Currently the script contains two GUIs. One of them for configuration purposes and the control GUI.

In general the behavior of an existent Amarok feature called Dynamic Playlist is often imitated. Therefore the user feels immediately comfortable in the environment of the script. Dynamic Playlists is the single implemented feature to generate playlists according to certain biases (like prefer similar artist or genre). While the user is listening, Dynamic Playlist puts in new tracks to the playlist such that Amarok never runs out of songs. Songs which are played earlier than five songs ago are removed automatically from the playlist. Hence the playlist keeps a little history and the size remains always the same.

Illustration 5: Main GUI like Dynamic Playlists.

The style is of the main GUI is exactly as the built-in Dynamic Playlist feature of Amarok2. It actually should be integrated directly into Amarok as another option to generate playlists, but it has to stay separated as long as the correct API to Amarok is missing.

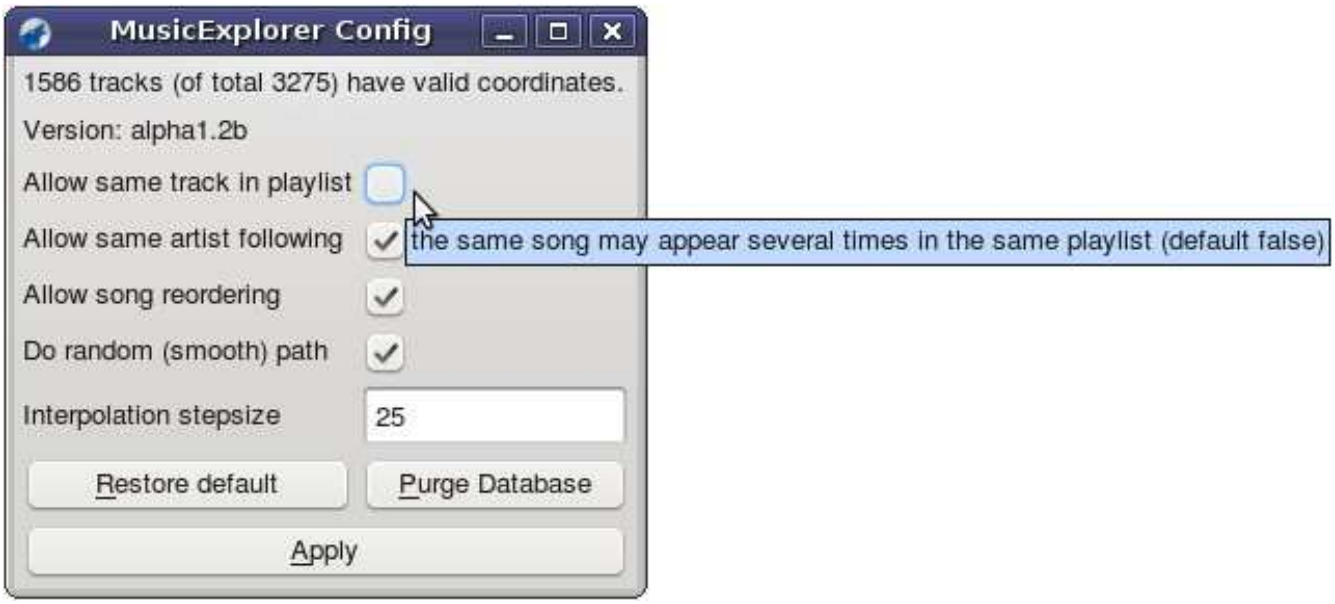

Illustration 6: Configuration window. Only the impact on the playlist generation is visible to the user.

The second GUI is for easy configuration. Some users might play all similar songs and some might use the “random smooth path”. These features will be further discussed in the chapter about autonomous behavior.

3. Playlist generation

Imagine a big party with lots of people who want to listen to completely different music. The DJ receives requests for some electro, oldies and black metal. So he has to somehow bring all these songs into one playlist without shocking all party people with a song of Slayer after Mozart. The MusicExplorer script can do this automatically for you in several ways that will be explained next. In this chapter I will mostly describe the implementation. Problems and some ideas for improvements can be found in the discussion section (see chapter 5).

3.1. Song reordering

Imagine the DJ, if first a guy from the party wants to listen to a pop song, then another one wants metal and in the end a second pop song is requested. Maybe it makes more sense to reorder the three songs such that first the pop songs are played before it is tried to reach the metal song. The script does this by finding a short path from the currently playing song to all requested songs. To find such a short path a greedy algorithms is used (it always takes the nearest song next). For traveling salesman problems it is known that this might lead to a very bad result. However in this problem one does not have to go back to the starting point after "visiting" all songs. Therefore the biggest problem, that often the longest path element is the last one, falls away.

Surely, the algorithm can be improved in several different ways, but it seems to perform well enough for song reordering. There is some fuzziness in the coordinates as well, so one only has to do the most obvious reordering. However if a user does not like this feature he might turn it off and force the script to follow a (possibly long) path.

3.2. Interpolation

In order to get a smooth playlist the DJ might want to add some additional songs. Smooth playlist here means that there are no "big jumps" in the style of two consecutive songs. Even though we want to assure that two consecutive songs are quite similar, one can reach a very distant song after several steps.

I define "interpolation" in this context as finding songs between two given ones such that we gain a smooth playlist. Interpolation is one of the biggest advantages MusicExplorer provides compared to other services that mostly focus on playing similar songs. The easiest way to find songs which might lie between two given ones is a straight line between them. Songs near this line are possible candidates.

One could try to find a fixed amount of songs between two given ones in order to interpolate. This will result in a smooth playlist but the disadvantage is that two similar songs will be interpolated with the same amount of songs like two distant ones. So if two guys at the party requested two Abba songs, we might hear Abba for a long time, which could become annoying. Therefore I decided to use a fixed step size. This step size should be less than the distance for which two songs are not considered to be similar anymore. On the other hand if this step size is too small, it gets annoying to listen to similar songs all the time. If we have for example two songs with distance 30 [MusicExplorer embedding distance unit] (which is relatively similar[6]) and we have a step size of 25, we try to find one song which lies nearest to the point exactly in the middle between the two given songs.

6 "Pancho – The Mobile Music Explorer ", Lukas Bossard, Semester Thesis, ETH Zürich, 2008

If we find the same song being near to two different interpolation points, we will just take it once. Illustration 7 shows an example playlist in two dimensions.

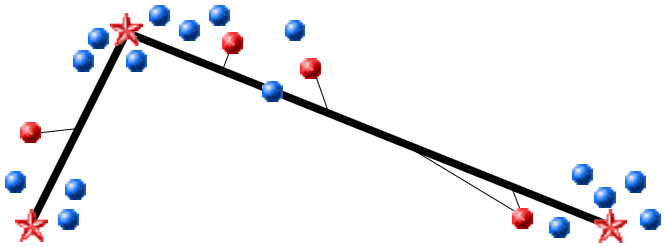

Illustration 7: The red stars are the given songs. All dots represent songs and the red ones will be taken to interpolate.

3.3. Autonomous behavior

After having played all requested songs, the question remains what to play next. The DJ script might just stick to the last song played and play similar songs as long as no other request arrives or it might just continue the smooth path to a random destination. These two options are available in the script.

Especially the random smooth path is interesting. Most users that will use the script will not have parties all the time but somehow want to listen to all songs in their collection in an intelligent way. So they just have to install this script and press play and will gain an infinite smooth playlist. Some songs are kept in history and when we are near to run out of songs a new song is randomly chosen and a smooth path will be found towards there (such that it behaves like the well known Dynamic Playlist feature).

If on the other hand a user wants a playlist without diversity he can choose to play only songs near the last one added. This often will result in a playlist of a single artist (which might be too similar). Therefore it works better in combination with the option where two consecutive songs are forced to be from different artists.

3.4. Avoid songs

By just randomly walking around in this space of songs it might happen that we play a song the user definitely does not want to listen to at that moment. If he just skips this song, the script will still play a very similar one the user is likely to dislike, too. Therefore we also need an option to skip and avoid the currently playing track. Nevertheless, maybe some time later when the user is in another mood, the script might consider playing this track again.

This was implemented by defining regions on the map, where we want to avoid songs. The probability of taking a song near to the one the user did not like is around zero. The further away we try to pick a song the bigger the likelihood of taking it becomes. The idea is as follows: if we do not like a particular song, we will neither like a song very similar to it, and possibly we currently do not want to listen to this whole genre anyways.

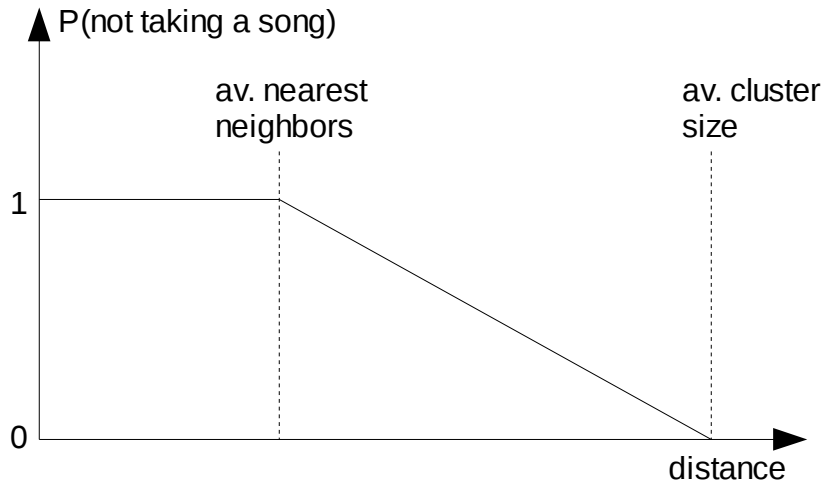

Illustration 8: Initial probability of not taking a song with respect to the distance to a song we avoid.

The probability drops after every successfully played song. Thus, after some time the script “forgets” about a previous made decisions. The reason to decay the avoid-probability over time is to remain adaptive to changes in the user's mood. Every time a random song is picked, the script will make sure, according to probability, that we are not too near to a song we do not want. The probability of taking a song, when there are multiple songs to avoid, is therefore the multiplication of each probability of taking it.

This for itself gave sometimes poor results because often the user does not want to listen to a song because he does not like the whole genre at that time. One way to make sure that we do not end up within the same genre again is forcing the script to take a song which is very distant.

4. User feedback

An alpha version of the script was published on KDE-Apps.org, where all Amarok related scripts are collected[7]. In about three weeks the script was downloaded more than a hundred times and was rated best of all in this category. I received a couple of comments which indicated that the users generally like the plugin:

by davewantsmoore on: Dec 4 2008
I just registered specifically to say YOU ROCK! .... Genius playlists (from iTunes) have been my "missing feature"
I'd never heard of musicexplorer.org.
This is awesome. Your work is much appreciated!

One of the developers wrote me some mails that he would like to use MusicExplorer coordinates in his company. The company is selling licenses for mostly less known artists and if he would have coordinates for all songs of these, he could provide the users similar songs to the ones they are listening to (nearest neighbor in the space of music).

7 http://kde-apps.org/content/show.php/MusicExplorer?content=93830, MusicExplorer KDE-Apps.org (visited 8.12.08)

by sturzi on: Dec 14 2008

The script is simple to use, works as promised and without errors. Only for my collection it's difficult to verify the functionality of the smooth playlist since most of my songs are not recognized. Therefore playlists are often generated out of 2-3 artists which sometime seem very different to me.

As this user complained, the rate of coordinates found for alternative collections is rather low. And for only a hand full of songs, MusicExplorer performs worse than for a big collection of mainstream music.

Even though the topic of how playlists are generated out of MusicExplorer coordinates is rather complex for the average users, some clearly understood the main advantage the plugin provides:

by Hermi on: Dec 14 2008

This is pretty neat. I mostly like the idea that you can actually try to find a smooth transition from something like Metallica to Beethoven, which sounds pretty weird at first. But still, this plugin manages to find playlists with no big jumps in type, but still jumps across artists and albums wildly. This makes it pretty cool to run in the background while doing something else, without having to manually switch back to your playlist and skip this one song that just doesn't fit.

5. Discussion and further work

The first thing that has to be done is to integrate the script into Amarok as it is intended to. This means there should not be a separate control GUI and drag and drop of songs into the playlist should always work. However this can currently not be done better as long as the API does not support such things.

5.1. Provide better coordinates

For mine, the big demo and some of my friends collections, the script recognized around half of the tracks' coordinates, even if tags are available and set correctly. This is rather poor, because tracks without coordinates will currently not be played at all. The best way to improve this on the server side would be adding more tracks to the MusicExplorer database. Unfortunately this will currently lead to feasibility problems. Another approach would be guessing non available coordinates according to other tracks of the same artist or by using something like the similar-tracks API provided by last.fm. This can also be done in the script. Even better would be an approach to randomly throw in unrecognized songs such that the user can decide if this song does fit into the playlist. This could also be reported back to the server in order to improve the embedding.

5.2. Playlist optimization

There are many things one could optimize for the interpolation. Currently trying to do discrete steps often fails in finding a song.

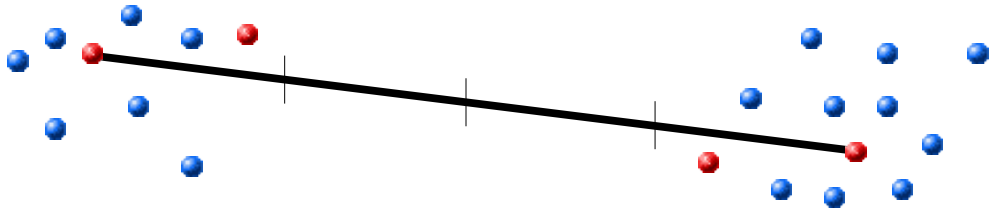

Illustration 9: Blue are available songs and red are the songs actually taken for the playlist.

In the example from Illustration 9, we wanted to interpolate with three songs, but where only able to find two (the red ones). This often happened because the song coordinates are not uniformly distributed over the space of songs but form clusters. This becomes even worse because the few songs lying between these clusters will be picked for almost every path from one to the other cluster, which is shown in Illustration 10.

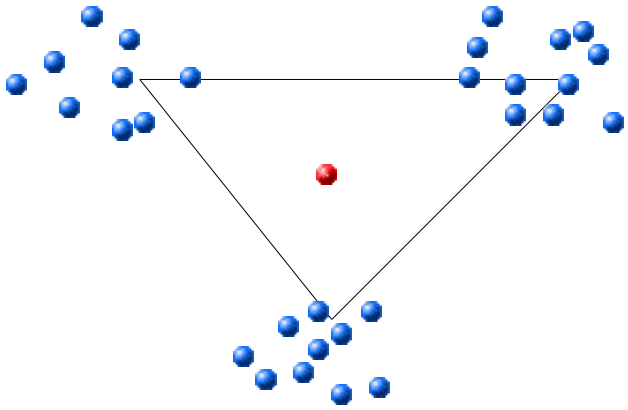

Illustration 10: The blues songs form clusters. The red is taken for every playlist between these clusters.

Hearing a single song again and again becomes annoying to the user. If we stick to the model of MusicExplorer it makes sense to pick this song because it seems to be forming a connection between two clusters, but in practice these are often somehow special songs that will not fit in any playlist. Or it might be an outlier. A good example is the Penis song of Monty Python. It seems that people listening to various different genres all however similarly listen to this song. Therefore this song is correctly embedded between these three clusters. However, listening to it too often might still get annoying. An easy way to avoid this problem is through user feedback. Since we have access to the Amarok built in rating functionality we could just avoid taking songs which are rated too low or played too often in the recent past. But these are all workarounds and will stay as long as the embedding tends to build clusters.

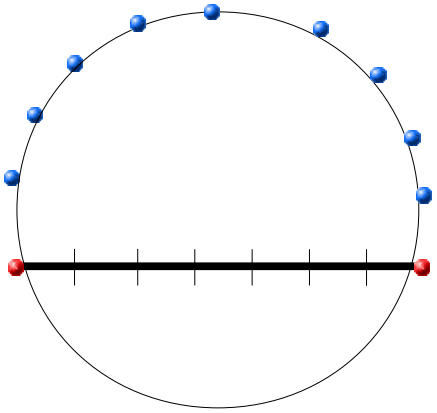

Illustration 11: Worst case example for the current implementation: No interpolating song will be found.

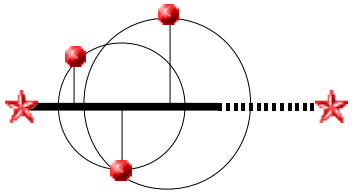

Illustration 12: Worst case example, where interpolated path becomes significantly larger than the direct link.

In a worst case scenario a playlist could become very bad. We can get void playlists (as illustrated in 11) or playlists, which are significantly longer than the direct link. If we only care that an interpolation song is the nearest to a point we want to interpolate, we can get very bad results (Illustration 12). Therefore this distance has to be limited to some value (eg. the step size), which has also the positive side effect that orphaned songs (between the clusters) get picked less frequently.

A way to completely avoid such behavior would be to go away from the step based approach to a more intelligent solution, such as looking some steps ahead in order to find the smoothest playlist. One could try for example to create complete playlists using a traveling salesman algorithm.[8] This might however, lead to computational problems. In the worst case we need to compute all possible paths between our songs which is infeasible to do online for large collections.

5.3. Avoidance of songs

A current weakness of the implementation is that regions of avoidance are only considered while taking the random points. If they happen to be on two different sides of such a region, the songs in between are taken anyway.

8 "Generating similarity-based playlists using traveling salesman algorithms", T. Pohle, E. Pampalk and G. Widmer, 2005

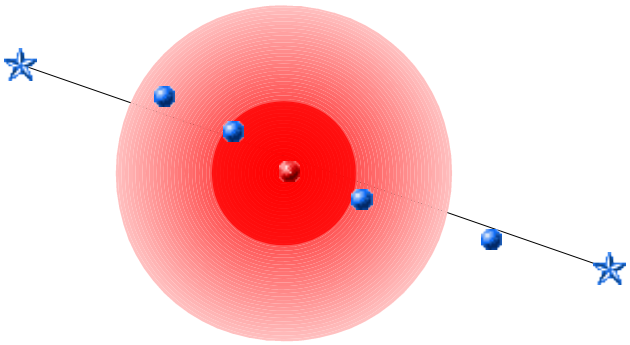

Illustration 13: Current interpolation ignores regions we we want to avoid.

On the other side it seemed that in practice the described effect does not happen often since we explicitly jump away from this region first. In the worst case, if the script tries to enter such an undesired region again the user can also jump away another time and make the avoidance region even larger.

An alternative way to define avoidance regions would be with flycasting[9]. Flycasting uses a collaborative filtering strategy to generate playlists. This technique could also be used to define regions of interest.

6. Conclusion

For the first time MusicExplorer is practically available to public. A plugin for Amarok2 which can generate dynamic playlists on the fly was developed . For this to work it is sufficient for the users to define artist and title tags of their songs, such that the script can download coordinates online.

A playlist is represented as a path in the high dimensional space MusicExplorer provides. The users are able to push various songs they want to listen to their playlist. The script recognizes these and will reorder them such that the distance traveled gets minimized. Additionally the script tries to find new songs along this path. These interpolating songs are chosen in a way that two consecutive songs are still considered similar. The result of this process is a smooth playlist without rapid changes in style.

A second mode can provide dynamic playlist on the fly without the need of user interaction. Most other autonomous playlist generators available are able to play similar music to a given track. This functionality is also possible in the developed script but the big advantage, which is not seen anywhere else in this form, is the power to create random smooth paths. These will cover the whole collection the user has in a nice way without sticking to a specific style of music.

Should it happen that the style of a played song does not please the users, they can temporally avoid this region. The plugin will then instantly change to a very different song and avoid the undesired region in the nearer future.

Many users tried this plugin in the weeks after the first release and the feedback is generally good. Nonetheless there is space for optimization, some of which have been outlined.

9 “Flycasting: On the Fly Broadcasting”, J. French and D. Hauver, University of Virginia, 2008

7. Appendix: Practical problems and opinion

Since the main focus for this work was practically oriented I would like to point out also some lessons learned and conclusions out of the perspective of a script developer working with Amarok2 beta.

7.1. Coding problems encountered

7.1.1. Bugs

Bugs were the main problem during this work. Thereby not things I did wrong, but things that were obviously coded erroneously either in Amarok or even in the underlying Qt toolkit are meant. A first problem is that there is no real debugging environment for QtScript. So the best way to actually find bugs is by the use of console output, which makes it much harder to understand what is really going on. Further there is no documentation for all functions provided by Amarok. Most information I got out of the source code itself and wrote this to the Amarok wiki afterwards.

Some Amarok bugs were really obvious and easy to detect. Such bug reports were fixed (or hacked) by the community relatively fast. This were things like Amarok was crashing when a script reported back that it finished[10], or when stopping a running script (out of a while(true)-loop)[11].

Some other bugs were harder to detect. For example the “countChanged-signal handler” was the only way to actually determine whether a user added a song by himself. Unfortunately this was corrupted (and still is at the time of writing). If the user removed a song from the playlist the signal handler did report back to the script, but way too early, such that the playlist did not change at all at this time[12]. So there was no way to detect a removal of a song. It came out that this is a hack, done in the core playlist generation of Amarok. So the bug propagated all the way on to the scripting engine.

While testing the KD-Tree I encountered, besides the strange timing already discussed in a previous section, an even stranger timing problem. Sometimes the easy operation of just passing a reference of an object to a given array took a long time for large arrays. Not just twice the time but up to hundred times the normal passing of the reference. This made it unfeasible to use the script for more than about 2000 songs. By writing a demo code, it came out that this problem is caused by Qt itself. The reason is still unknown. An assumption is, that often the object is cloned where just the reference should be passed. Gladly for me I found an ugly workaround[13]. Sometimes QtScript even injures the JS standard. An example is the array.splice() function used to remove and replace array elements, which is corrupted, and returns wrong results[14]. Thus it was not possible to remove array elements in this quick way.

10 http://bugs.kde.org/show_bug.cgi?id=175049, Amarok.end() crashes amarok2 (visited 8.12.08)
11 http://bugs.kde.org/show_bug.cgi?id=175050, stopping a running script crashes amarok (visited 8.12.08)
12 http://bugs.kde.org/show_bug.cgi?id=175654, CountChanged signal wrong if song removed (visited 8.12.08)
13 http://trolltech.com/developer/task-tracker/index_html?id=236220&method=entry, 236220 - QtScript: Array.push() takes a long time (visited 9.12.08)
14 http://trolltech.com/developer/task-tracker/index_html?id=233853&method=entry, 233853 - QtScript: Array.splice, (visited 9.12.08)

7.1.2. Changing environment

At the start of this project Amarok2 was in beta state and therefore some changes were expected to happen. However the changes were often much bigger than assumed. This often lead to a complete malfunction of the script. I will point out the main problems here.

Fist the SQL database changed from SQLlite to MySQLembedded. Since my script was natively communicating with the database over sockets this made it impossible to use the database at all, because there are no drivers to communicate to MySQLe over sockets. This caused a lot of disturbance in the community[15]. At least for my part, the solution was a built in function of Amarok to communicate to the DB. The query execution is much faster there, but all results were returned as QStrings, such that parsing all for the floating number coordinates results in a higher overhead.

Then some very important signal handlers that reported when songs were added to the playlist were thrown out because of changes made to the core of the playlist. So I was left with the countChanged signal handler to determine whether new songs have been added to the playlist (and this was even buggy).

Finally the latest code version from SVN was often not able to even play any music so the new versions had always to be tested on a separate system first.

7.2. Personal Conclusion

It is a great chance for the MusicExplorer database to get some attention because it is the first plugin of this kind for Amarok2. At the time of this report the script had to stay in alpha phase because some needed API to Amarok was still missing but they are easy to implement now. Nevertheless the user feedback is promising and MusicExplorer made one step out of research towards usability.

However, I would not try to implement such a script in a system under development, like Amarok2 was, again. The changing of core parts of the code and the absence of a script documentation was a huge drawback and often frustrating to work with. On the other hand I had the opportunity to help this open source project with bug reports and wiki pages. Therefore I will also continue working on this project in order to release a finalized version free of bugs. One should also focus on a bigger database of coordinates or at least a better recognition rate on the server side.

The algorithms used were mostly straightforward as the focus was on usability and less on research. So there is space for improvements.

15 http://amarok.kde.org/blog/archives/842-Avast,-We-Be-Getting-Slandered,-Yar.html, Avast, We Be Getting Slandered, Yar – Amarok Blog (visited 9.12.08)