Mixxx Macros

Let the machines do the tedious work

There are moves through which a DJ expresses himself - and there are many other, sometimes tedious tasks that need to be done as groundwork. Instead of having to juggle beatmatching, effects, cues and sliders all at once, a DJ should be able to rely on his tools so he can focus on what really matters. For some, beatmatching and keeping all tracks in sync is an art in itself - for others, including me, it is something happily delegated to the software - that is why there is a sync function.
But sometimes, more sophisticated automation can be helpful - maybe you want to skip a breakdown, shuffle around verses, loop an intro a specific way or deliberately repeat certain parts. Handling that while messing with effects and getting the next track ready can be tricky; that is where Mixxx Macros come in. With this feature implemented, it will be possible to record specific moves while playing a track and store them in a rack, to be used when it gets hot.

Timeline

May - Research & Community bonding:

- Get more acquainted with the Mixxx code by investigating small issues
- Look into how other software implements similar features and how that relates to Mixxx Macros
- Gather input from different communities & users of similar features

June - Design Phase:

- Design the structure for saving Macros
- Add the required controls to an existing skin
- Potentially create a practice skin

July - Coding Phase:

- Implement the controls and map them to a controller
- Implement the data structures
- Enable recording & playback of Macros

August - Refinement Phase:

- Refine the UX
- Enable Import from Serato & explore possibilities for sharing
- Integrate the controls into other skins
- If there's more time, consider integration with AutoDJ

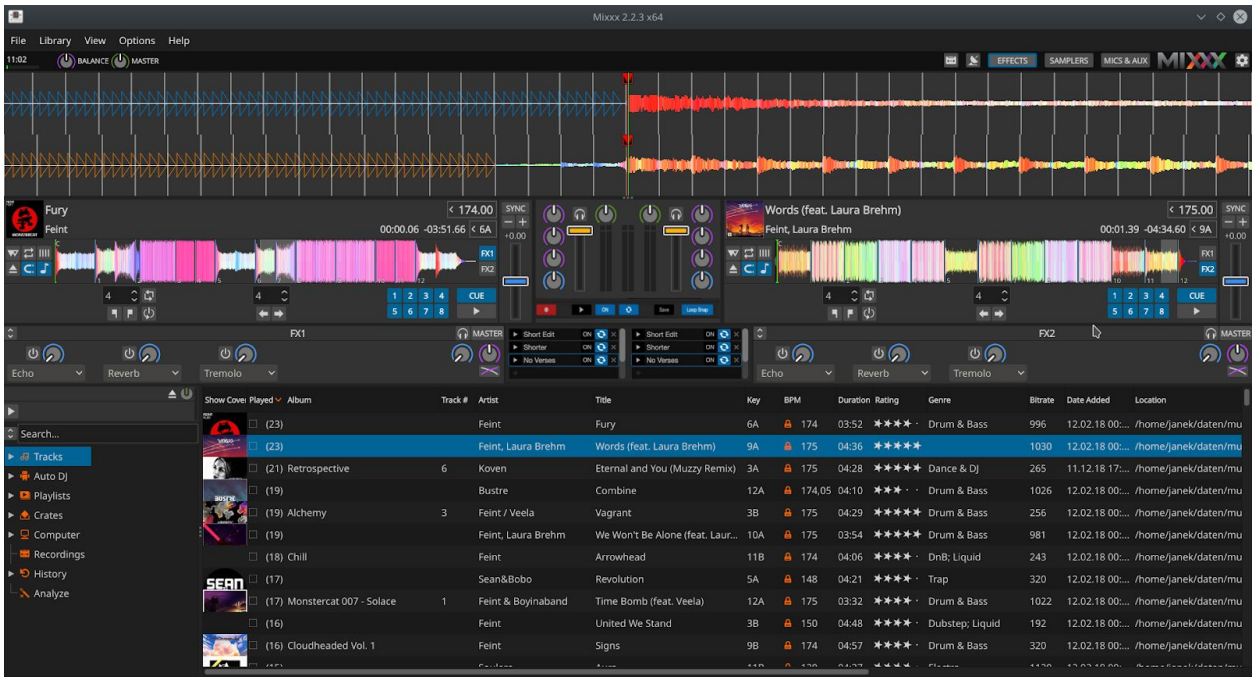
Details

Based on the suggested Saved Hotcue Routines project and the corresponding Launchpad Issue, this feature draws inspiration from Serato Flip. It will enable recording track movements and other controls as Macros - edits or modifications of a track without changing the original.

Research

To get further insight into useful features, requirements and pain points with existing implementations, some user research will be conducted before the start of the coding phase. For that I will reach out to different DJ communities via r/DJs, forums and personal contact, furthermore looking into tutorial videos to see how it is used and where they hack around limitations.

Interface



Mockup of how Macros could be integrated into Mixxx' Deere Skin, using the controls of Serato Flip

The main controls to consider are the Macro rack and the controls needed for recording. Since a Macro is always only about a single track, the recording controls could replace

the crossfader in the prototyping phase, while the racks could be put alongside the effects racks. But while the effect racks can be rebound to other decks, the Macro racks are always track-specific, so this is not ideal. A solution has to be found that integrates these racks for each track without cluttering the interface.

Creating a practice/Macro recording skin that has only one deck is probably a good idea. This could then also be useful for other tasks such as library management and track preparation.

Recording

After hitting record, the first activated hotcue will start the actual recording. A recording is configurable - having quantize active will automatically quantize all actions, which is desirable for loops or jumps. Now, there are a few open questions here: Should the Macro be bound to the hotcue or its timestamp? Should the Macros record the timestamps of jumps, as Serato does, or the actual control actions? Which controls should be recorded? Where should the Macros be saved? These will be answered within the design phase.
The second question is particularly relevant, since recording hotcue invocations would allow the Macro to be changed by moving the hotcue, but that can also be problematic. If the hotcue is moved for a different purpose or deleted, the Macro breaks. That's why the proposed format below shows a format that allows both.

An interesting idea is to assign Macros to hotcues - that way the DJ has full freedom. The Macro can trigger automatically once that hotcue is reached or be manually invoked through the hotcue. This could reduce the number of additional controls needed. Of course, if this is to be implemented, a way has to be found that does not sacrifice flexibility.

Implementation

Controls

For the controls, we only need Buttons for the beginning - instances of the PushButton control with a PushButton widget for the skins. These are the main controls:

- Start/Stop the recording of a Macro - the infrastructure behind it could draw from the RecordingManager used for recording the whole set
- Controls & Widgets for triggering as well as enabling looping and other configuration for the currently recorded and the Macros in the rack
- The structure for the Rack itself, which could draw from the structure and visuals of the samplers

The option for assigning and potentially reassigning hotcues to Macros as well as deletion of Macros will at first only be available through the interface, since these are very infrequently needed and not during performances. Drag-and-drop support as well as a modification to the colors of a hotcue widget that has been assigned to a Macro are something to be considered.

Data

Right now, Mixxx stores almost all information about the library in its database, the main exceptions are the waveforms. Sidecar files are not really an option, as they quickly pollute the file system. Writing such information into ID3 tags like Serato is a consideration, but it carries multiple problems:

- If we move Macros into there, hotcues, cues and other information should also be saved in there, which would be a big effort and could lead to the file being bloated unnecessarily
- Editing tags requires modifying the actual file, which can't be done while the track is playing and like all I/O operations carries some risk
- Macros wouldn't be easily accessible for modification, since the format should be concise

Another option is adding them into the database; this again makes them inaccessible and additionally seriously hampers sharing possibilities, but makes the implementation much easier.

The most promising way seems to be saving the Macros in a separate folder under *.mixxx*, like controller mappings and analysis results. That way there can be a theoretically infinite amount of Macros, they can be saved in a human-readable file

format and shared. As a bonus, migration to aoide won't be an issue. However, sharing wouldn't be as easy as copying the file over, since an entry in the database linking a Macro to its track is still needed. Thus a separate interface for sharing and importing Macros which could include converting/extracting Serato Flips is likely necessary.

Format

The database would store the name of the Macro file along with the id of the track as well as its position in the rack in a table similar to *track_analysis*. Versioning the file format might also be useful, but since the Macro files are useless without the link to their track, that information can be in the database as well, along with a checksum of the file to prevent accidental manipulation. The database can then also store a Macro's state, i.e. if it's enabled, bound to a hot cue or whether it should be looped, since this information neither needs to be shared nor open for manual editing. This way Mixxx doesn't need to modify a Macro after it has been recorded.

The format for Macros could then be a JSON file containing the name and a list of the actions taken, which would be an ordered list of triggers together with the action to perform (i.e. where to jump to or which control to invoke). An ordered list is important, because a Macro might hit a trigger multiple times. The main problem I see with that format is that it does not support recording continuous controls, but it is questionable whether that is necessary. It could even be added by marking different points in time with their values and then interpolating based on some predefined function.

```
{
	"name": "Double Chorus",
	"actions": [
		{
			"start": { "type": "timestamp", "value": 187626 },
			"action": "jump",
			"value": { "type": "timestamp", "value": 157896 }
		}
	]
}

{
	"name": "Double breakdown as Intro",
	"actions": [
		{
			"action": "jump",
			"value": { "type": "hotcue", "value": 2 }
		},
		{
			"trigger": { "type": "hotcue", "value": 3 },
			"action": "jump",
			"value": { "type": "hotcue", "value": 2 }
		},
		{
			"trigger": { "type": "hotcue", "value": 3 },
			"action": "control",
			"value": { "key": "cue_gotoandplay", "value": 1 }
		},
		{
			"action": "control",
			"value": { "key": "cue_gotoandplay", "value": 0 }
		}
	]
}
```

Example Macros in JSON format

Above are two exemplary Macros that illustrate some details. The first will simply jump 30 seconds back when a specific part of the song is hit. Since all enabled Macros will run from the beginning of the track if they are enabled and looping won't be on for this Macro, it essentially functions as a one-time loop.
For the second example, since the first action has no trigger defined (same when the trigger is *null*), it will be executed immediately when the track is started, unless it is disabled. It will jump to hotcue 2, repeating the part between hotcue 2 and 3 twice and then jumping to the cue of the track.

The value field is merely a JSON object that will be utilised by the implementation of the action. In contrast to Serato Flips, the Macros don't have an initial starting point - the trigger of the first action will be the starting point, this eliminates the need for an additional field and makes Macros more flexible, because as long as they are enabled either from the start of the track or by manually starting them, they have only one routine: Wait until the trigger of the next action is met and then execute it. By making triggers optional, an arbitrary amount of actions can be chained without needing nested structures.

Refining it further, even the name could become part of the database, actions might be broken down to control invocations only and could also be stored in a binary format etc. - but the ironing out of the details will be part of the project, after gathering the needs of users. Based on that, the way of storage and file format might change a lot, but these examples already provide a proof of concept and a rough idea.

Code

Within Mixxx, the Macros need to be able to listen for updates on a track regarding seeking and the current playback time. When the user skips over the trigger of an active Macro by triggering another hotcue or simply seeking forward, it might be sensible to disable that Macro - user research has to be conducted on the expected behaviour.

To be able to do that, Macros need to be tied in with the main audio/engine thread, but without adding any blocking behaviour. That is an argument that speaks against using hotcue positions as triggers, since then the position of the hotcues need to be kept in sync within the part of the code listening for the triggers, potentially updating them.

Additional features

It was also considered that this project might require multiple loop storage to be implemented first. While that feature indeed has merit, it is not required for Macros, since the stored format should be independent of the loops saved in the track.

If there is additional time, integrating Macros with Auto DJ by adding an extra control that denominates whether a Macro should also be invoked when the Auto DJ plays a track could be a handy addition.

About Me

Name: Janek Fischer
Email: janek.fischer@code.berlin
Zulip: Xerus

Involvement in Mixxx

Hey, my name is Janek and I have been using Mixxx for 2 years, contributing towards controller scripts last year as well as being active in the community and reporting some issues. I have also written an external tool in Kotlin which works with the mixxxdb.

I own versatile DJ gear by Native Instruments which I have mapped myself, so I can easily test new controls properly. Using Mixxx on Linux, I have performed some Mixes which are public on Mixcloud as well as a few small gigs.

My Work

I have many years of development experience in web development (PHP, HTML, CSS, JS) as well as desktop development with Java/Kotlin, mainly using the JavaFX library - this is my biggest project in that realm: https://github.com/Xerus2000/monsterutilities

Studying Software Engineering in the 4th semester, I dove into C++ last semester and am currently looking deeper into open source as a whole - as part of these efforts I have also created a collaboration convention open for anyone, which I am adopting in the open source projects I lead. You can find more of my activities on my GitHub.