HAW Hamburg

Audio-Video-Programmierung

WS18/19

Vokalo

Technische Dokumentation

https://github.com/vincefried/vokalo

https://www.neoxapps.de/vokalo/

Authoren:
Fynn Theiss
2322408
Vincent Friedrich
2326998

Betreuer:
Prof. Dr. Andreas Plaß
Jakob Sudau

13. Januar 2019

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Zusammenfassung

Diese Dokumentation beschäftigt sich mit der technischen Umsetzung eines *Audio Samplers* angelehnt an gängige DAWs[1]. Dabei können zunächst Samples für vorgegeben Spuren eines *Sequencers* aufgenommen werden. Anschließend ist es möglich diese Samples entsprechend der Belegung im Sequencer abzuspielen.

1 Technische Voraussetzungen

1.1 Benutzeroberfläche

Abbildung 1: Vokalo UI

Für dieses Projekt wurde entschieden, dass es, um für möglichst viele Benutzer zugänglich zu sein, als Applikation im *Browser* realisiert wer-

[1] *Digital Audio Workstation*

den sollte. Dazu wird für die Benutzeroberfläche *HTML5* sowie *CSS3* genutzt. Weiterhin wird das CSS-Framerwork *Bootstrap*[2] genutzt. Während der Konzeption der Benutzeroberfläche wurde als Gestaltungsstil *Flat Design*[3] gewählt, um eine möglichst simple Struktur und ein positives Nutzererlebnis zu ermöglichen.
Des Weiteren wurde *Javascript (ECMAScript 2018)*[4] mit der Bibliothek *jQuery*[5] genutzt um Animationen und visuelles Feedback für den Benutzer zu realisieren.
Für die Fortschrittsanzeige während des Aufnahmeprozesses wurde das Projekt *progressbar.js*[6] genutzt.

1.2 Logik

Die Logik um Aufnahme und Wiedergabe der Samples zu implementieren basiert ausschließlich auf *Javascript*. Dabei wurden insbesondere die *Web Audio API*[7] sowie die *MediaStream Recording API*[8] genutzt.

2 Implementierung

2.1 Aufnahme

Kernstück der Aufnahme ist das *MediaRecorder* Objekt der *MediaStream Recording API*. Es wird der *Audiostream* des Input-Gerätes (i.d.R. Mikrofon) dem *MediaRecorder* übergeben. Neue Daten werden während der Aufnahme kontinuierlich in einem Array abgelegt. Wird die Aufnahme gestoppt, erzeugt das Skript aus diesen *Audio Chunks* ein *Blob*[9], erstellt für dieses eine Objekt-URL und erstellt anschließend mit selbiger ein neues

[2] `https://getbootstrap.com`
[3] `https://en.wikipedia.org/wiki/Flat_design`
[4] `https://www.ecma-international.org/ecma-262/9.0/index.html`
[5] `https://jquery.com/`
[6] `https://kimmobrunfeldt.github.io/progressbar.js/`
[7] `https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API`
[8] `https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API`
[9] **B**inary **L**arge **Ob**ject

HTML Audio Objekt.
Jedes so aufgenommene Sample wird nun für die Verwendung in der vorher ausgewählten Spur **16-fach** dupliziert und in einem Array abgespeichert. Dies ist daher nötig, da ein Sample in der Lage sein muss zeitlich überlappend abspielbar zu sein.

2.2 Rhythmus

Um einen Rhythmus mit den verschiedenen, eingespielten Samples einzuspielen wird pro Spur ein Array mit **16** Werten initialisiert. Wird einer der 16 Buttons in der UI einer Spur betätigt, so wird an die entsprechende Stelle des Arrays eine 1 geschrieben, beim "deaktivieren"des Buttons wird dieser Wert wieder auf 0 gesetzt.

2.3 Wiedergabe

Zunächst wird ein Intervall abhängig der BPM[10] gestartet. Dieses ruft regelmäßig die Funktion *playOneBeat* auf. Bei jedem Funktionsaufruf iteriert das Skript einen Schritt durch die o.g. Rhythmus-Arrays". Befindet sich eine 1 an der derzeitigen Position wird der Sound der entsprechenden Spur in dem bei der Aufnahme initialisierten SSample-Array"der selben Position abgespielt. Ist das Ende des Arrays erreicht oder das Limit der maximalen *Beats* erreicht so iteriert die Funktion erneut vom Beginn der Arrays.

2.4 Andere Interaktionsmöglichkeiten

Es ist außerdem möglich

- die Lautstärke zu ändern. Dies passiert, indem mittels des *AudioContext* der *Web Audio API* eine *Gain Node* erstellt wird. Diese wird nun mit dem *Output* des *AudioContext* verbunden. Somit kann man die Lautstärke des **gesamten** Outputs steuern.

[10] **B**eats **p**er **m**inute

- die BPM zu steuern, indem die Abstände des o.g. Intervalls verändert werden. Das respektive UI Element in- oder dekrementiert eine globale Variable auf welche das Intervall während der initialisierung zugreift. Die BPM kann den Wert **40** nicht unter- und den Wert **220** nicht überschreiten. Da die Javascript Funktion *setInterval* mit Millisekunden rechnet und von 4 Takten à 4 Beats ausgegangen wird, wird mit der Formel $60 * 1000/BPM/4$ der richtige Wert ermittelt.

- die Anzahl der Beats zu ändern. Prinzipiell unterstützt die Applikation 4 Takte mit jeweils 4 Beats, also insgesamt 16 Beats. Man kann mit dem entsprechenden UI Element jedoch auch weniger, sprich zwischen 1 und 16 Beats wiedergeben lassen. Dabei beginnt die Wiedergabe erneut an der ersten Position wenn die Anzahl der abzuspielenden Beats erreicht ist. Dies ist wie oben genannt durch einen simplen Zähler während des Abspielens realisiert.

3 Schwierigkeiten

Die implementierte Realisierung des Projektes warf einige Schwierigkeiten auf.

- Die *MediaRecorder API* bietet nur begrenzt Unterstützung in den gängigen Browsern.

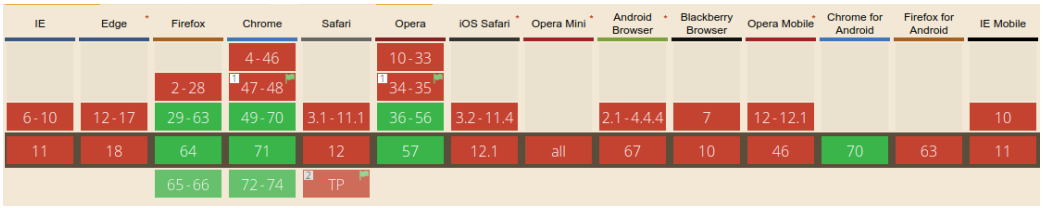

Abbildung 2: Unterstützung der MediaRecorder API in verschiedenen Browsern

- Ursprünglich war das Transponieren der aufgenommenen Samples geplant. Dies erwies sich als kompliziert, da man außer der Veränderung des Abspieltempos der Samples von der *Web Audio API* keine günstigen Alternativen angeboten bekommt.
- Die aufgenommenen Samples werden »as is« abgespielt. Beginnen diese mit einer Pause, so wird im abgespielten Rhythmus eine Pause für den jeweiligen Sound hörbar sein.