Music Analyser

Technische Dokumentation - Jonathan Held und Julian Heinken

Einleitung

Wir haben mit JavaScript und der WebAudioAPI eine WebApplikation entwickelt, mit der sich Songs in ein Rhythmusspiel verwandeln lassen. Zuerst wird das Back-End und dessen Funktion der Transientenerkennung erklärt, darauf folgt die Erklärung des Front-End.

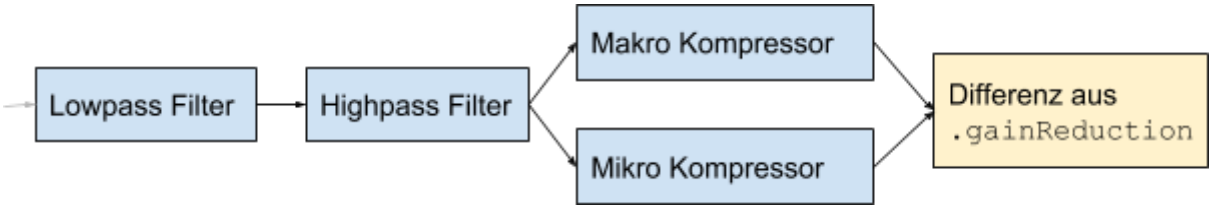
Beat-Erkennung

Einleitung

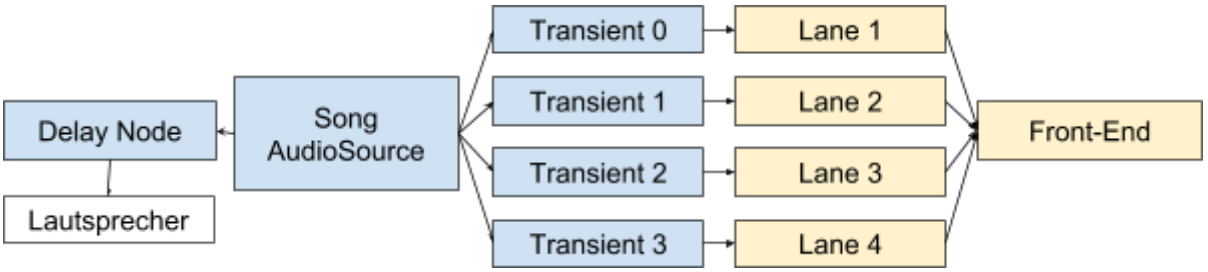
Um bei einem Signal Anschläge zu erkennen, benutzen wir eine Transientenerkennung. (Referenz: https://de.wikipedia.org/wiki/Transientendesigner). Das originale Konzept benutzt sogenannte “Hüllkurven-Folger” also Geräte, die der Lautstärke eines Signals folgen. Da die WebAudio API so etwas nicht bietet, “missbrauchen” wir zwei WebAudio Kompressoren. Diese geben nämlich die Werte ihrer Lautstärke anpassung aus (`.gainRedction`), welche sich als Ersatz eignen. Außerdem erlauben es Kompressoren leisere Signale zu ignorieren. Über den Schwellenwert (`.thresshold`) können wir beispielsweise sagen, dass Signale unter -40dB nicht beachtet werden sollen. Dieses war für unsere Erkennung sehr hilfreich.

Implementierung

Wir haben nun zwei Kompressoren: Makro- und Micro. Einer passt sich umgehend der Lautstärke des Signals an (Micro Kompressor). Der andere ist etwas “träger” (Macro-Kompressor). Ist nun die Lautstärkereduzierung (`.gainReduction`) im Micro Kompressor höher als die im Makro Kompressor, wissen wir das wir einen Anschlag hatten.



Zusätzlich haben wir vor den Kompressoren eine Low- und Highpass Filter eingesetzt um jede Lane grob einem Frequenzbereich zuzuordnen. Zusammengefasst sieht unser WebAudio Graph wie folgt aus:



Anzumerken ist, dass wir eine Delay-Node verwenden, welches das gehörte Signal so verzögert, dass die Noten im richtigen Moment an der Unterseite eintreffen. Mehr dazu im Front-End Teil.

Debugging und Entwicklung

Die Werte wie Frequenz der Filter so wie die exakten Kompressoren Einstellung unserer Transientenerkennung sind nicht beliebig. Wir benutzten die JavaScript Bibliothek [dat.gui] um uns die wichtigsten Werte im Browser anzeigen zu lassen und On-The-Fly zu ändern. dat.gui bietet an, die Werte automatische zu ändern. Da jedoch die Einstellungen von WebAudio-Nodes hinter dem Feld `.value` abstrahiert sind, sind diese für dat.gui nicht zugänglich. Deshalb werden diese Werte zwischengespeichert (`transientDectBands[].settings`).

Frontend

Um wiederholtes Laden von unterschiedlichen Seiten zu vermeiden haben wir alle Bereiche unser Anwendung in unterschiedlichen Divs untergebracht. Von diesen wird immer nur das jeweils relevante angezeigt und die “Display”-Werte der anderen auf “hidden” gesetzt.

Wenn die Seite gestartet wird ist zuerst das “MainMenu” aktiv. Von diesem kann der User die anderen Bereiche aktivieren und über ein Dropdown zwischen unterschiedlichen Liedern wählen.

Wenn der User auf “Play” drückt, lädt die Beat-Erkennung den im Dropdown-Menü ausgewählten Song. Sobald der Song geladen ist, ruft die Beat-Erkennung nach jeder Analyse die “Step”-Funktion auf und übergibt ihr die Werte der Analyse.
Die Daten der Beat-Erkennung werden dabei als Array aus Bool-Werten an das Frontend weitergegeben. Aus diesem werden dann unterschiedliche Noten erstellt.
Wenn zum Beispiel der Array {1,0,0,1} an das Frontend überreicht wird, werden ganz links und ganz rechts Noten erstellt.

Wenn eine Note neu erstellt wird, wird auf diesem Kanal für eine gewisse Zeit nicht überprüft, welcher Wert vom Analyser übergeben wird, und die Note stattdessen nur verlängert. So haben die Noten eine einheitliche Größe, ohne sich zu überlappen.
Mit jedem Update wird jede Note um einen festgelegten Wert weiter nach unten bewegt. Bei Noten die gerade noch erstellt werden wird zusätzlich der “Top:”-Wert auf die Oberkante des

Spielbereiches gelegt. Dadurch verlängert sich die Note. Da die Noten ein als Kreise dargestellt sind, die an die Notengröße angepasst sind entsteht so ein “Wachstums”-Effekt.

Neben der Erstellung von Noten findet im Frontend auch die Verarbeitung des User-Inputs statt.
Hierfür werden im ersten Schritt die Werte eines UserInputArrays über buttonDown-Events auf 1 und buttonUp-Events auf 0 gesetzt.
Nun wird bei jedem Update der UserInputArray mit dem des letzten Updates verglichen. Wenn in dem Array ein Wert von 0 auf 1 gewechselt wurde werden im HTML-Dokument alle Elemente überprüft, die sich unter dem entsprechenden Knopf befinden. Wenn eines davon nur die Klasse “block” hat, wird dieses Element auf “block destroyed” gesetzt und der User erhält Punkte. Hat keines die Klasse “block” verliert der User Punkte.
Wenn der User einen “block” trifft, wird außerdem die “Opacity” eines “HitBlock” divs in der entsprechenden Spalte hoch gesetzt. Dies dient als visuelles Feedback, um dem User besser mitzuteilen, ob eine Note getroffen wurde oder nicht.
Die Opacity-Werte aller “HitBlocks” werden durchgehend reduziert um so einen “Fade-Out”-Effekt zu erzielen.

Wenn das Lied beendet ist, wird die “showHighscores”-Funktion aufgerufen und die Punkte an sie übergeben.
Jetzt wird jeder gespeicherte Punkte-Wert aus den Highscores mit den erzielten Punkten verglichen, bis entweder alle Punkte-Werte verglichen wurden, oder ein kleinerer Wert als die erzielten Punkte gefunden wurde.
Wenn ein kleinerer Wert gefunden wurde wird der neue Punktewert an der Stelle eingefügt und der niedrigste Punktestand entfernt.
Jetzt wird aus den Punkteständen und gespeicherten Namen die Highscore-Liste erstellt.
Wenn der Spieler genug Punkte gemacht hat, wird an der entsprechenden Position ein Input-Feld anstelle eines Textfeldes erstellt. Dieses trägt den Namen in die Highscoreliste ein.