Technische Dokumentation MusicController

Moritz Stoll - 2288654
Simon Ruisinger - 2298552

MusicController ist eine Webanwendung in der der Nutzer ein Piano mit einem herkömmlichen Gamingcontroller bedienen und dabei von einem KI-Generierten Beat begleitet werden kann.

Eingebundene Libraries

Die Anwendung verwendet die Libraries Tone.js, teoria.js und Magenta.js und die Gamepad API, sowie die Web Audio API.

Web Audio API
https://developer.mozilla.org/de/docs/Web/API/Web_Audio_API

Die Web Audio API stellt Funktionen bereit, die es ermöglichen Sound zu erzeugen und diesen mit einer Reihe verschiedener Audio Nodes zu beeinflussen.

Tone.js
https://tonejs.github.io/docs/

Tone.js baut auf der Web Audio API auf und stellt diverse Objekte wie Player und Synthesizer bereit, die eine harmonisch klingende Tonausgabe vereinfachen. Wie haben dieses Framework verwendet, da es die Zusammenarbeit mit teoria.js enorm vereinfacht. An die Tone.js Player lassen sich neben Frequenzen auch Noten- und Akkordnamen übergeben. So war es übersichtlicher eine musiktheoretische Unterstützung für den User in die Software mit einzubauen.

Teoria.js
https://github.com/saebekassebil/teoria

Teoria.js ist eine Musiktheorie-Framework. Es erlaubt es Noten, Akkord und Tonleiter Objekte zu erstellen. Diese auf Basis des jeweiligen anderen zu erstellen und sowohl die Notennamen als auch die Frequenzen und MIDI-Werte der jeweiligen Noten und Akkord zu erhalten.

Magenta.js
https://tensorflow.github.io/magenta-js/music/

Magenta.js ist ein Machine Learing-Framework von Google, über das sich bereits trainierte ML-Modelle einbinden lassen. Es stellt diverse Objekte und Funktionen wie zum Beispiel NoteSequences bereit. In diesen sind nun alle für die Player nötigen Informationen enthalten um eine Notenfolge zu spielen. Es basiert auf Tone.js, was auch einer der Gründe ist, weshalb MusicController Tone.js benutzt.

Gamepad API
https://developer.mozilla.org/en-US/docs/Web/API/Gamepad_API/Using_the_Gamepad_API

Die Gamepad API stellt eine Schnittstelle bereit, die es uns ermöglicht im Browser die Eingaben eines über Bluetooth oder USB angeschlossenen Controllers auszulesen.

Aufbau

Hier sind die Aufgaben und Funktionen der einzelnen JS-Dateien aus Platzgründen nur allgemein beschrieben. Im Code selbst ist jede Funktion ausführlich in den Kommentaren erklärt.

index.html

Die index.html stellt das Grundgerüst für die Benutzeroberfläche dar. Hier werden die Kontrollelemente für den Klang der Drums und des Pianos dargestellt, so wie ein stilistischer Controller als SVG Datei. So können über die gamepad.js auf dem echten Controller gedrückte Buttons auch auf dem SVG angezeigt werden.

drums.js

Die drums.js ist für die Erzeugung und das Abspielen des KI-Generierten Drumbeats zuständig. Hierbei kann der User auf der Website ein Seed-Pattern erstellen. Dieses wird in der drums.js mit der Funktion createSeedPattern() ausgelesen und auf dessen Basis erzeugt die KI nun einen Beatmithilfe der Funktionen generatePattern(), toNoteSequenze(), fromNoteSequence(), setSeedPattern() und createSeedPattern(). Um einen rhythmischen Beat zu erzeugen, verwenden wir das Clock Objekt von Tone.js. Dieses ruft 4 mal pro Sekunde die Funktion tick() auf. In dieser werden dann die Player angesprochen, die die Drums abspielen. Die Funktionen playDrums() und stopDrms() werden von der gamepad.js aufgerufen

Globale Variablen

state, Tone, sampleBaseUrl, reverb, context, gainNodeDrums, gainSlider, compressor, compressorSlider, distortionDrums, distortionSlider, filterDrums, filterDrumsSlider, drumCompSwitch, drumEqSwitch, drumMenu, compActive, eqActive, ready, midiDrums, reverseMidiMapping, snarePanner, outputs, drumKit, temperature, rnn, pattern, clock, stepCounter, oneEighth

Funktionen

initDrums(), startDrumMachine(), humanizeTime(), getStepVelocity(), tick(), playDrums(), stopDrums(), generatePattern(), toNoteSequenze(), fromNoteSequence(), setSeedPattern(), createSeedPattern(), drumSwitches()

piano.js

Die piano.js ist für das Abspielen und den Klang des Pianos, das der Nutzer mit seinem Controller steuert, zuständig. Bei einem Tastendruck auf dem Controller werden die

Funktionen playNote() oder playChord(), je nach Tastenbelegun, von der gamepad.js aufgerufen.

Globale Variablen

pianoContext, piano, gainNode, gainSlider, compressor, compressorSlider, distortion, distortionSlider, filter, filterSlider, pianoCompSwitch, pianoEqSwitch, pianoCompSlider, pianoEqslider, compActive, eqActive, notesWhite, notesBlack, sameNotes

Funktionen

initPiano(), playNote(), playChord(), makeDistortionCurve(), pianoSwitches()

gamepad.js

Die gamepad.js benutzt die Web Gamepad API, mit der es möglich ist über den Webbrowser auf ein angeschlossenes Gamepad zuzugreifen. Gamepad.js lädt sich die aktuelle Controllerbelegung aus der gamepadMappingPS4.json, in der alle Buttons mit Funktionen und benötigten Informationen gespeichert werden. Dieses wird im local storage des Users gespeichert, so dass es beim nächsten starten der Anwendung immer noch vorhanden ist. Auch regelt gamepad.js das Auslesen der Buttons und führt die entsprechenden Aktionen durch. Dadurch hat sie Zugriff auf drums.js und piano.js. Hier läuft durchgehend ein loop, der die Eingaben des Controllers checkt. Die Funktion newPress() ist dafür zuständig zu ermitteln, ob es sich dabei um eine eben erst gedrückt Taste handelt oder die Taste gedrückt gehalten wird.

Globale Variablen

mapping, update, pressedbuttons, storage, gamepad, beatPlays

Funktionen

initGamepad(), loadMapping(), startGamepad(), stopGamepad(), loop(), startAction(), stopAction(), newPress(), addListener(), changeButtonColor(), setSound()

pianoConfig.js

Die pianoConfig.js regelt die Belegung der Controllertasten mit Noten oder Akkorden. Sie erzeugt die entsprechenden Menüs auf der Benutzeroberfläche und gibt die ausgewählten Noten oder Akkorde an gamepad.js, wo diese dann auf das Mapping gelegt werden können. Über die Funktion getPianoDOMObjects() werden die dafür notwendigen DOM-Objekte angesprochen. Die Funktionen openDropDown() und createList() öffnen die entsprechende Auswahl und clearModal() und closeModal() schließen sie wieder und setzen das DOM wieder auf den neutralen Zustand zurück.

Globale Variablen

configStates

Funktionen

getPianoConfigDOMObjects(), getChordBuilder(), createEventListener(), openDropDown(), createList(), clearModal(), selectItem(), closeModal()

viewController.js

Die viewController.js regelt sämtliche Steuerungselemente der Anwendung die keinen direkten Einfluss auf den erzeugten Sound haben. So zum Beispiel sich öffnende oder schließende Menüs.
Die Funktion stopLoading() wird von der drums.js aufgerufen, sobald das Modell zur Beatgenerierung geladen wurde.

Globale Variablen

loading, mainScreen, pianoMenu, pianoMenuOpenBtn, pianoMenuCloseBtn, pianoMenuState, info, drumMenu, drumMenuOpenBtn, drumMenuCloseBtn, drumMenuState

Funktionen

initView(), stopLoading()

Diagramm

Wir haben zwar nicht object orientiert programmiert, aber trotzdem eine Art UML-Diagramm erstellt. Dieses soll den Aufbau der Software etwas anschaulicher darstellen und die Zugriffe der JS-Dateien aufeinander zeigen.