Umsetzung einfacher DSP Probleme als Spiel

Fabian Schmieder

Lehrstuhl für Systemtheorie und Signalverarbeitung
Uni Stuttgart

Übersicht

Motivation

Grundaufbau

- Framework
- Aufgaben

Baublöcke

- Ausgabe Elemente
- Aktive Elemente
- Formatierungs Elemente

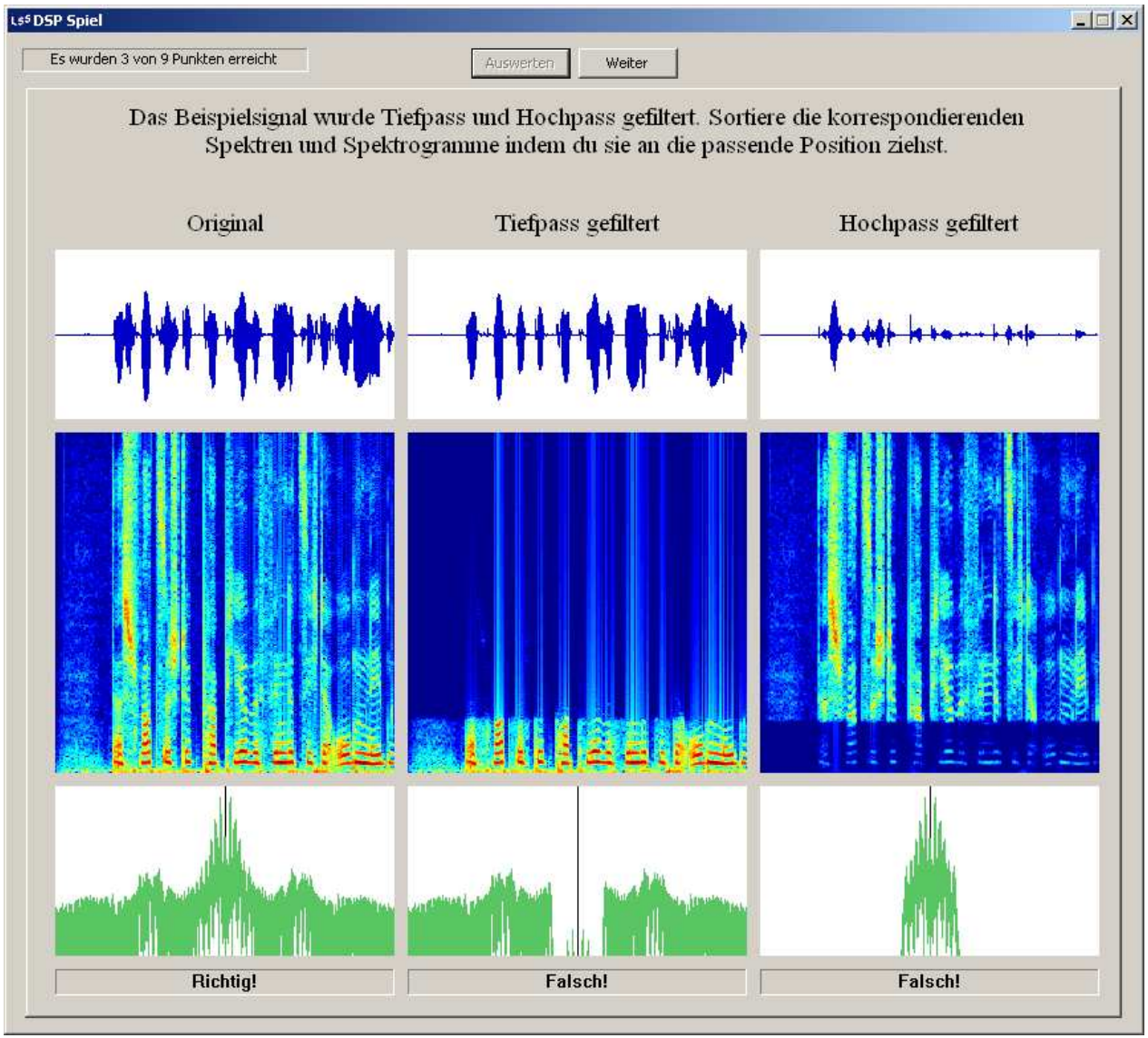
Signalzuordnung

Verwaltung der Aufgaben und Beispiele

Verbleibende Aufgaben

Motivation

- Darstellung von Methoden und Werkzeuge der Signalverarbeitung
- Veranschaulichung von grundlegeneden Zusammenhängen



Aufbau des Framework

Vorraussetzungen:

- Das Spiel sollte möglichst unabhängig von anderer Software sein
- Es sollte möglichst einfach sein neue Aufgaben hinzuzufügen
- Die Beispiele sollten einfach austauschbar sein
- Die Entwicklung sollte mit möglichst wenig Aufwand verbunden sein

Aufbau des Framework

Vorraussetzungen:

- Das Spiel sollte möglichst unabhängig von anderer Software sein
- Es sollte möglichst einfach sein neue Aufgaben hinzuzufügen
- Die Beispiele sollten einfach austauschbar sein
- Die Entwicklung sollte mit möglichst wenig Aufwand verbunden sein

Aufgrund dieser Randbedingungen wurden folgende Entwurfsentscheidungen getroffen.

- Das funktionelle Framework wurde in C++ mithilfe der MFC entworfen
- Die Aufgaben werden zur Laufzeit eingelesen und durch eine XML Datei beschrieben
- Das Framework übernimmt die gesamte Signalverabeitung, daher müssen keine Ergebnisse vorberechnet werden
- Wenn möglich werden externe Bibliotheken verwendet
  - Die FFT wird mithilfe von fftw berechnet
  - Die XML Dateien werden mit TinyXML eingelesen

Aufbau der Aufgaben

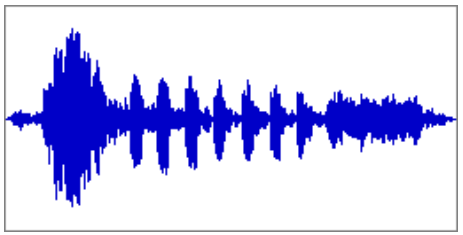
- Die Aufgaben sind durch ein Baukastenprinzip aufgebaut
- Es gibt bis jetzt 9 Baublöcke
- Eine XML Datei beschreibt den Aufbau der Aufgabe, z.B.

```
<?xml version="1.0" ?>
<GameSlide>
  <opt width="256" height="128" vspace="10" hspace="10" />
  <static width="700" height="40">
  Ordne die Spectrogramme den Zeitsignalen zu indem du sie an die
      passende Position ziehst.
  </static>
  <table rows="2" cols="2">
    <row>
      <wave id="1" example="*" />
      <wave id="2" example="*" />
    </row>
    <row shuffle="1">
      <spectrogram id="4" example_from="1" switch="1" height="257" />
      <spectrogram id="5" example_from="2" switch="1" height="257" />
    </row>
  </table>
</GameSlide>
```

Ausgabe Elemente

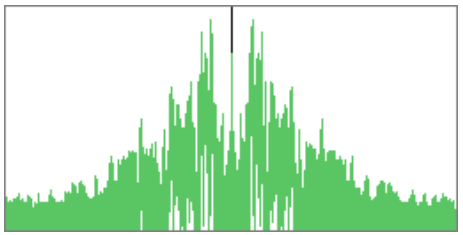
- Statischer Text: `<static ...>Text</static>`
- Zeitverlauf eines Audiosignals: `<wave .../>`

```
id="#", example="path", example_from="#",
points="#"
```



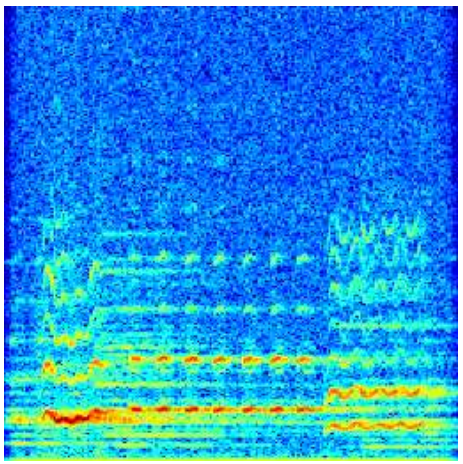
- Spektrum eines Audiosignals: `<spectrum .../>`

```
id="#", example="path", example_from="#",
points="#", scale="log|linear"
```



- STFT eines Audiosignals: `<spectrogram .../>`

```
id="#", example="path", example_from="#",
points="#", fmax="dargestellte frequenz"
```

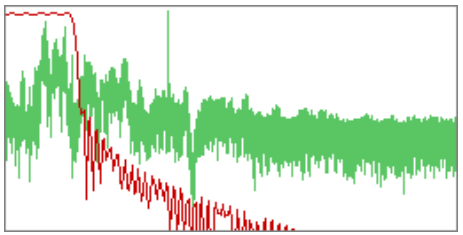


Aktive Elemente

Die folgenden Elemente erzeugen ein veränderetes Ausgangssignal, welches wieder von anderen Ausgabe oder aktiven Elementen aufgegriffen werden kann.

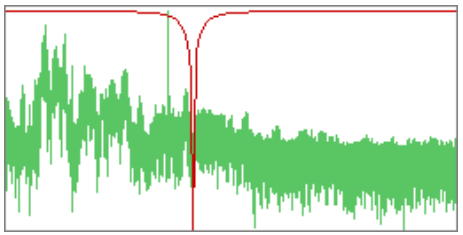
- Konstantes digitales Filter: `<afilter .../>`

```
<a>a0 a1 ... aN</a><b>b0 b1 ... bM</b>
id="#", example="path", example_from="#",
points="#", scale="log|linear", spec_in="0|1",
transfer="0|1"
```



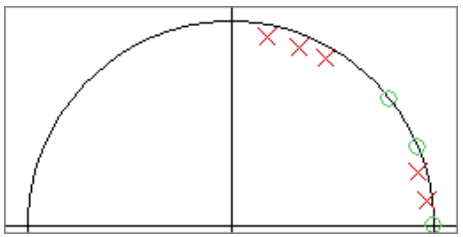
- Interaktives Kerbfilter: `<notch .../>`

```
id="#", example="path", example_from="#",
points="#", scale="log|linear", spec_in="0|1",
transfer="0|1"
```



- Interaktives Pol-Nullstellen Diagram:

```
<polezero .../>
id="#", example="path", example_from="#",
points="#"
```



Formatierungs Elemente

Zusätzlich zu den eingefürten Elementen gibt es noch weitere Element, welche genutzt werden um die sichtbaren zu gruppieren. Diese Elemente sind:

- Ein Optionselement: `<opt ... />`
  Dieses Element kann benutzt werden um Basis-Höhe, -Breite und -Abstände zu definieren.
- Eine Tabelle: `<table rows="#"cols="#"/>`

  ```
  <row shuffle="1|0">passende Anzahl an Elementen</row>
  ```

  - verwaltet die position der enthaltenen Elemente
  - Falls `shuffle="1"` werden die Elemente der Reihe zufällig vertauscht
- Ein verstecktes Element: `<hidden/>`
  - Kann eine belibige Anzahl an Elementen enthalten
  - Alle enthalten Elemente sind nicht sichtbar
  - Kann benutzt werden um Filter im Hintergrund zu definieren

Signalzuordnung

- Es gibt drei Wege einem Element ein Signal zuzuordnen
  - Mit `example="path_to_wav"` wird die angegebene WAV-Datei benutzt
  - Mit `example="*"` wird zufällige eine WAV-Datei ausgewählt
  - Mit `example_from="n"` wird das Signal des Elements mit `id="n"` genutzt
- Es werden nur WAV-Dateien mit 8 bit oder 16 bit Genauigkeit unterstützt
- Falls die angegebene WAV-Datei mehr als einen Kanal besitzt wird nur der erste Verwendet
- Die zufällige Auswahl ist eindeutig und wiederholt sich erst wenn alle Beispiele einmal ausgewählt wurden
- Es können nur Signale von Elementen genutzt werden, welche dies mit `example="."` eingelesen haben oder selber erzeugen, wie die aktiven Elemente

Verwaltung der Aufgaben und Beispiele

Die vorhandenen Beispiele und Aufgaben werden durch eine weitere XML Datei angegeben.

```
<?xml version="1.0" ?>
<game>
  <font name="Times" size="140" />
  <audioexamples>audio</audioexamples>
  <slides>aufgaben</slides>
  <course titel="Intro">
    <slide>aufgaben\notch.XML</slide>
    <slide>aufgaben\spec_sort.XML</slide>
  </course>
</game>
```

- Eine Liste von `<audioexamples />` definiert die WAV Ordner
- Eine Liste von `<slides />` definiert die Aufgaben Ordern
- Die `<course />` Elemente defineren einen vordefinerten Ablauf von Aufgaben.
- Zum Spielstart werden alle Kurse eingelesen und zur Auswahl dargestellt

Verbleibende Aufgaben

Verbleibende Aufgaben und mögliche Erweiterungen sind z.B.

- Entwicklung von zusätzlichen Baublöcken
- Beschreibung von weitern Aufgaben
- Erstellen der Einführung für die Elemente
- Eine Verbesserung der zufälligen Auswahl
- Entwicklung einer Ausgabe für das Ergebniss durch Bestenliste