Projectomschrijving CSD2d

Door: Harold Groenenboom

Green AudioSearcher

In dit verslag beschrijf ik het concept van mijn project, het einddoel en de toekomst van dit project en ik beschrijf wat ik aan het eind van dit project lever. Ik zal langslopen hoe ik te werk ben gegaan, hoe ik heb ontworpen en hoe het resultaat er uiteindelijk uit ziet. Vervolgens zal ik kort reflecteren over mijn bezigheden.

Inhoud

1. Concept

Als componist of producer ervaar je vaak dat je een sample vindt en dat die net niet is wat je wilt. Toch is de toonhoogte precies perfect. Of juist de timing. Of de klankkleur. Dit is iets wat ik zelf vaak ervaar. Hiervoor probeer ik in dit project een tool te maken die de gebruiker met veel gemak naar vergelijkbare samples laat zoeken. De *Green AudioSearcher* is een audioapplicatie waarin de gebruiker audiosamples kan opzoeken die op een aangegeven audiosample lijkt. Hierbij staat user centered design centraal. Het is namelijk belangrijk dat deze tool een toegevoegde waarde kan hebben voor het creatieve proces, en dat deze tool het proces in ieder geval niet verstoord. Hierom heb ik allereerst nagedacht wat ik wil en heb ik op papier een design uitgetekend.

Voor de Green Audiosearcher heb dus ik allereerst gewerkt aan een volledig design. Zo wil ik van een volledig design terugwerken naar een prototype design waarin ik kan uittesten hoe de workflow bij de gebruiker aanslaat, in dit geval ik. Hierom focus ik in mijn prototype op de essentiële elementen om mijn concept uit te voeren en probeer ik bewust zo min mogelijk extra's toe te voegen. Ik als componist ben de voornaamste doelgroep, omdat ik dit zelf als tool wil hebben, maar ik wil het ook zo designen dat het ook voor andere componisten en producers in ieder geval logisch werkt. Zo kan ik uiteindelijk een iteratie ontwikkelen die bij de buitenwereld aanslaat.

Een bestaande applicatie die zo een analyse doet is Sononym. Deze heb ik kort uitgeprobeerd en de rede dat ik mijn applicatie nog steeds wil ontwikkelen is omdat ik de in Sononym gebruikte technieken wel workflow verbeterend vindt, maar het design niet. Omdat zo een tool een kleine stap is in het compositieproces, denk ik dat je als gebruiker niet te veel opties en niet te veel visuele informatie wilt. Voor mij is dus de zoektocht om een goede interactie en interface te ontwikkelen voor een creatief bemiddelend product.

2. Ontwerp

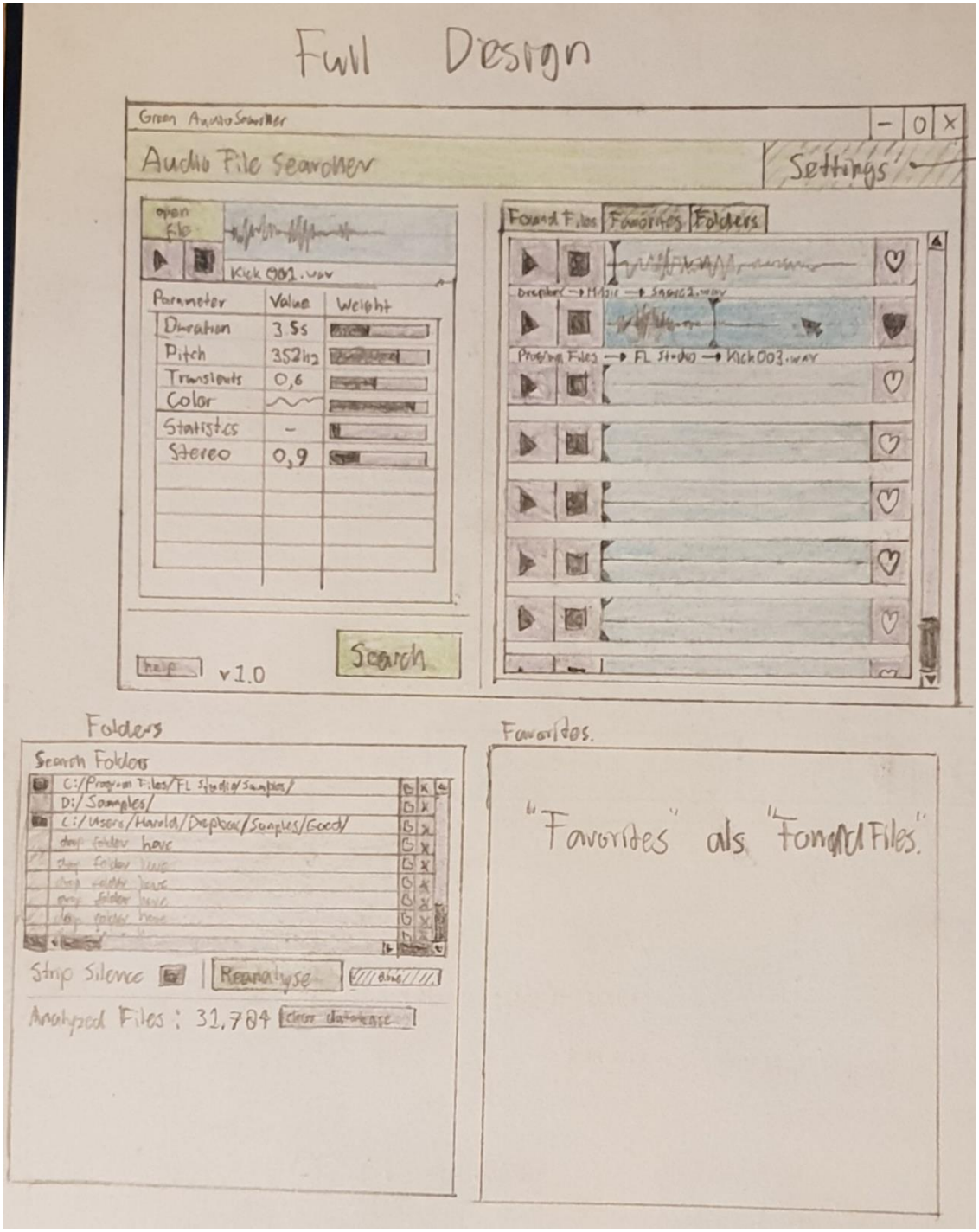
Volledig ontwerp

Het volledige ontwerp heb ik allereerst op papier uitgewerkt. Ik ben gaan nadenken over een workflow die mij handig lijkt en ik ben gaan nadenken over technieken waarmee samples kan zoeken die muziekaal gezien verwant zijn. Op de volgende bladzijde is het volledige design te vinden. Er is direct een belangrijke onderverdeling te onderscheiden. Zo bevindt zich links de input audiofile en rechts de gevonden vergelijkbare audiofiles. Linksonderin is de search button. De gebruiker werkt zo van linksboven naar linksonder, en vervolgens van rechtsboven naar rechtsonder. Ook heb ik de window expres klein gelaten, zodat het niet snel voor de gebruiker in een DAW in de weg zal zitten. Ik zal een paar belangrijke keuzes in het design langslopen.

Euclidean distance en weights.

Om de audiofiles te kunnen vergelijken analyseer ik eerst elke audiofile op verschillende features. Zoals piekfrequenties, lengte, dynamiek en klankkleur. Deze features normalizeer ik naar een range van 0-1 met een normalizatiefunctie, bijvoorbeeld een *Sigmoid function*. Het programma krijgt een lijst van mappen met audiofiles om te analyseren en creëert een database van geanalyseerde FeatureSet objecten. Om de gebruiker een specifiek zeggenschap te laten hebben op de features van de audiofile die hij of zij belangrijk acht, kan de gebruiker met weights aangeven welke waardes meer in de *similarity* vergelijking worden meegenomen. Het is dan wel belangrijk dat de features zo genoemd worden dat ze heel logisch en intuïtief zijn voor de gebruiker. De uiteindelijke *similarity* tussen twee samples s en s_2 wordt met e*uclidean distance* als volgt berekend:

$$Sim(s, s_2) = \sqrt{(w['dur'] * (s['dur'] - s_2['dur'])^2 + (w['pitch'] * (s['pitch'] - s_2['pitch'])^2 + \dots}$$


Full Design
Green AudioSearcher
Audio File Searcher
Settings
open File
Kick 001.wav
Parameter
Value
Weight
Duration
3.5s
Pitch
352Hz
Transients
0,6
Color
Statistics
-
Stereo
0,9
Found Files
Favorites
Folders
Search
help
v1.0
Folders
Search Folders
C:/Program Files/FL Studio/Samples/
D:/Samples/
C:/Users/Harold/Dropbox/Samples/Good/
drop folder here
Strip Silence
Reanalyse
Analyzed Files: 31,784
Favorites
"Favorites" als "Found Files"

Kleurgebruik

Om de interactie zo duidelijk mogelijk te maken onderzoek ik de waarde van kleurgebruik. Zo geef ik de belangrijkste knoppen de kleur lichtgroen. Deze knoppen worden door de gebruiker als het goed is het meest gebruikt. De minder belangrijke knoppen, waar de gebruiker niet per se aan hoeft te komen om het programma te gebruiken, geef ik de kleur paars. Audio geef ik de kleur blauw. Zo weet de gebruiker altijd meteen waar audiofiles zijn. Een andere belangrijke reden hiervan is dat ik alle Audio, weergegeven als blauwe waveforms, een drag-and-drop functie wil geven. Zo wordt de workflow voor de gebruiker optimaal. Elk gekleurd veld wordt ook subtiel gehighlight als de muis zich boven het veld bevindt om de gebruiker te laten weten dat er een interactie mogelijk is.

Workflow:

1. *Selecteer een sample uit de DAW.*
2. *Sleep de sample naar de waveform van de AudioFileAnalyser en wacht op voltooien van analyse.*
3. *Adjust de weights naar behoefte*
4. *Druk op search en wacht tot de search compleet is.*
5. *Luister naar de audiofiles en sleep de gewenste audiofile naar de DAW of weer opnieuw in de waveform van de AudioAnalyser.*

Favorites

Een gebruiker kan een gevonden audiofile als favorite markeren. Zo kan de gebruiker uit gevonden samples zijn of haar favorieten selecteren. Hiermee kan de gebruiker vrijuit spelen met het kiezen van nieuwe input audio files, zonder bang te zijn eerder gevonden audiofiles die zijn bevallen kwijt te raken.

Folders

De gebruiker kan in de *folders* tab aangeven welke folders in de search geinclude moeten worden. Zodra de gebruiker deze folders aan heeft gegeven, moeten in een keer alle folders geanalyseerd worden en vervolgens nooit meer. De gebruiker krijgt hierbij de controle om zelf het analyse proces te starten, zodat het voor de gebruiker voelt alsof hij of zij volledige controle heeft over het programma. Ik vind het zelf namelijk storend als een programma een analyseproces start waar ik zelf controle over wil hebben. Deze soort interactie waarbij de gebruiker eigenlijk juist meer doet dan normaal wil ik hier uitproberen.

Save / Load

Tot slot kan de gebruiker settings laden en saven. Zo opent het programma automatisch de laatst geopende setting. De preset bevat de folderpaths, de favorites en de database. Zo kan de gebruiker makkelijk wisselen tussen verschillende ‘zoekopdrachten’ op een intuïtieve wijze.

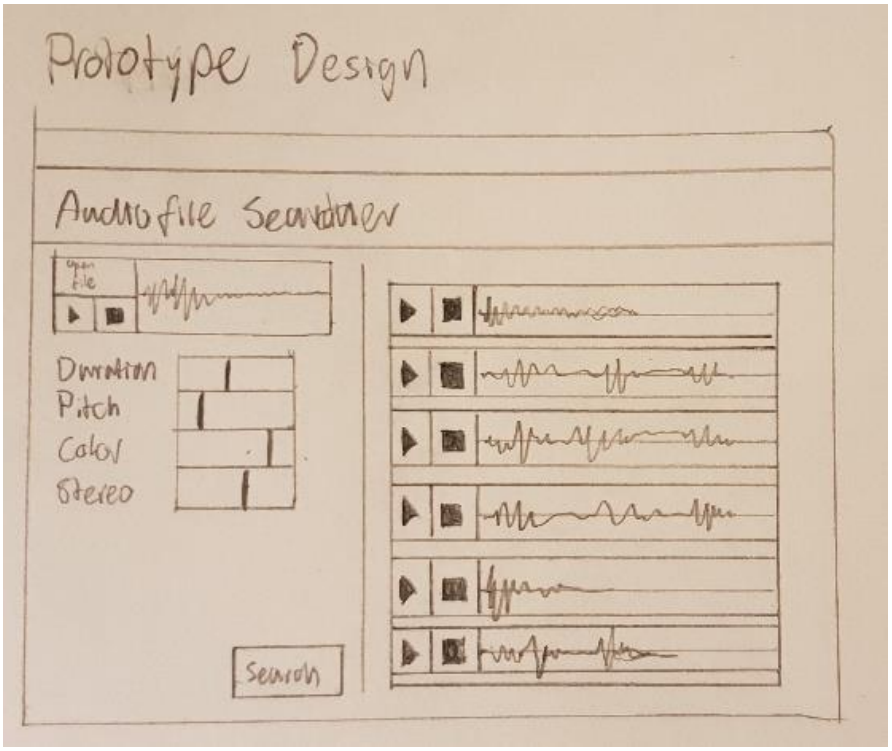
Samengevat

Het volledige ontwerp bevat dus al aardig wat opties. Een aantal opties hiervan zijn gebaseerd op aannames en het ontwerp is volledig op papier gedaan aan de hand van mijn smaak. De allereerste aanname, namelijk of de zoekmethode de gebruiker helpt, zal ik uittesten in dit project. Hiervoor heb ik geanalyseerd wat essentieel is uit het volledige concept en dit heb ik gebruikt om mijn prototype concept te vormen. Door het volledige concept achter de hand te hebben, hou ik in mijn systeemontwerp rekening met te verwachten vormen van uitbreiding.

Prototype Concept

In het prototype in dit project wil ik een werkende audioapplicatie bouwen die een audiofile analyseert en een reeks vergelijkbare audiofiles teruggeeft. Hiervoor kan de gebruiker voor nu één folder instellen. Met het prototype wil ik het kleurgebruik en de workflow onderzoeken. Zo hoop ik na het prototype op een zinnige manier het volledige concept te kunnen uitwerken.

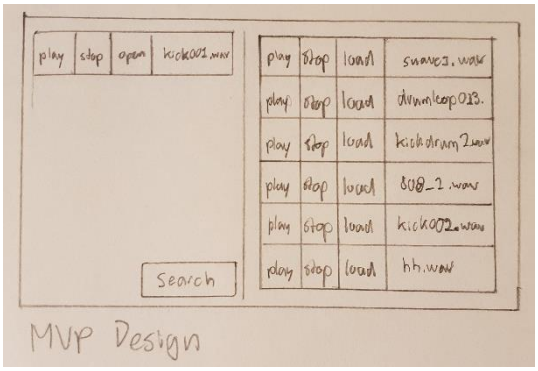
Hiervoor heb ik een prototype ontwerp gemaakt. De layout is hierbij bewust hetzelfde. Zo werk je nog steeds van linksboven naar linksonder en vervolgens naar rechtsboven naar rechtsonder. Alle interacties hier zijn interacties die iedere DAW gebruiker al kent.



3. Planning

Om het project te structureren heb ik een MVP design gemaakt. Vanuit dit design wordt heel duidelijk wat ik precies wil gaan proberen. Zo is het echt duidelijk dat het vooral gaat om gebruikersgemak en workflow.

Om goed te werk te kunnen gaan heb ik als eerst een MoSCoW opgesteld.



MoSCoW

Must (MVP)

- Audio analyse
  - Duration
  - RMS
- Audiofile players
  - Drag and drop
  - Play, pause en stop
- Search button
- Folder selection
- Euclidean distance

Should

- Audio analyse
  - Pitch
  - Timbre
- FeatureSet database
- Kleurgebruik
- Weights in GUI en euclidean distance

Coulds

- Waveform viewer
- Play-pause en stop knop ontwerp

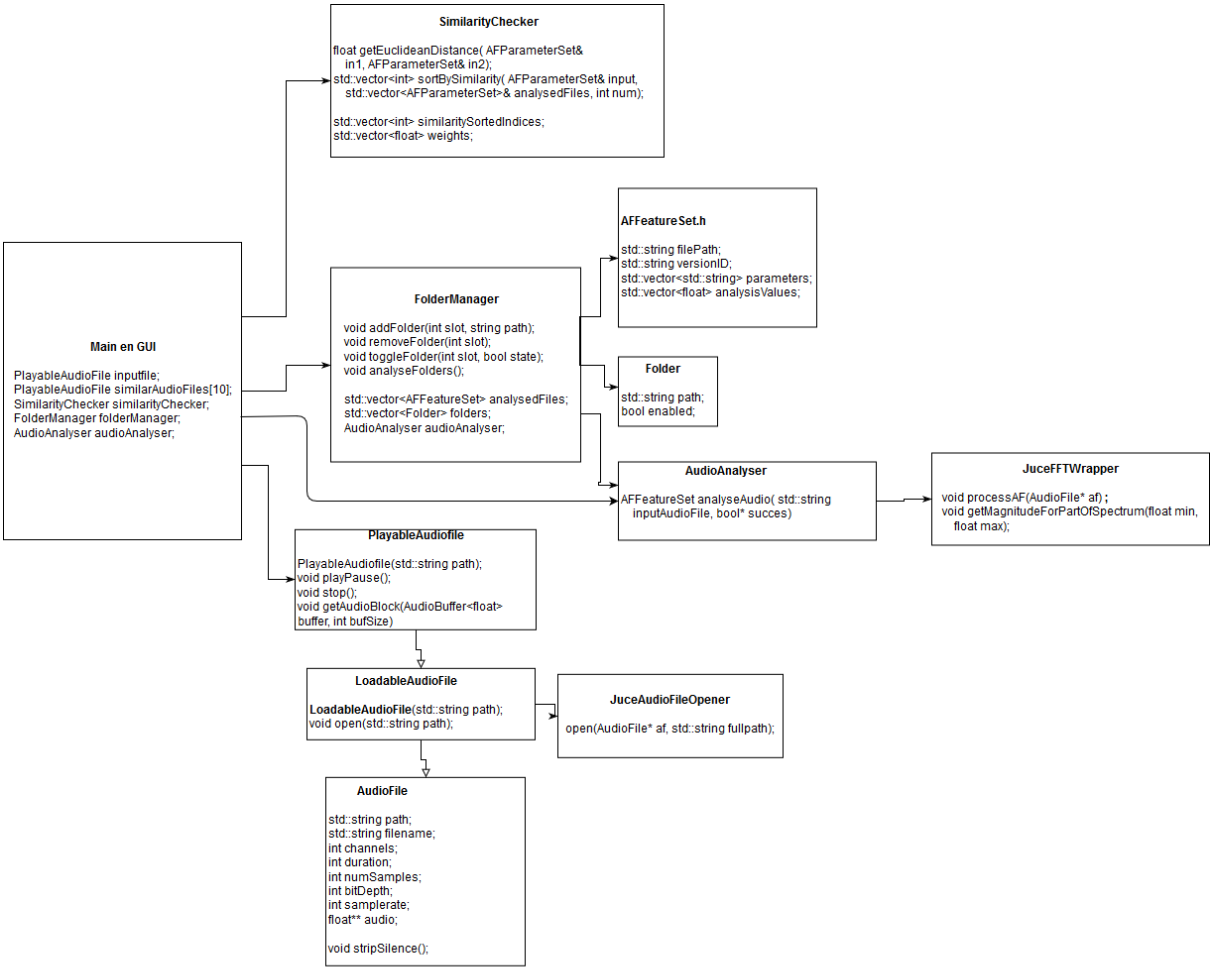
Woulds

- Toevoegen van een folder tab met:
  - Meerdere folders
  - Status bar voor folderanalyse
- Toevoegen van een viewport aan ‘Found Files’ om zo veel vergelijkbare audiofiles aan de user aan te kunnen bieden

4. Systeemontwerp

Voor het ontwikkelen van de applicatie heb ik in JUCE gewerkt en heb ik een niet JUCE-specifiek class diagram ontwikkeld. Zo kan ik duidelijker zien hoe mijn design op zichzelf staat en kan ik mijn ontwerp in andere frameworks ook makkelijk implementeren. JUCE is een framework voor audioapplicaties in C++. Dit framework biedt een combinatie van elegante GUI functionaliteiten, DSP functionaliteiten en het uitlezen van AudioFiles. Ik ben met JUCE gaan werken vanwege het gemak van het framework, vanwege mijn de focus op snelle ontwikkeling en omdat ik het framework al goed ken.

Het systeem bestaat uit 5 verschillende belangrijke elementen: De *SimilarityChecker*, *AFFeatureSet*, *FolderManager*, *AudioAnalyser* en de *AudioFile* class. Hierbij is de FolderManager volledig verantwoordelijk voor het creëren van de database en het bijhouden van de benodigde folders. De SimilarityChecker bevat de weights en kan AFFeatureSets vergelijken. AFFeaturesSets zijn simpele containers voor de Features van een AudioFile en de AudioAnalyser is de class waarin wordt bepaald wat er wordt geanalyseerd, hoe de analysesfeatures heten en wat de ID van de specifieke analysesetting is. Elke class heeft een duidelijke functie en de dataflow van de classes is ook makkelijk te volgen. De AudioFile class is een simpele container voor een audiofile. Deze kan ik later gaan uitbreiden met functionaliteiten als opslaan en normalizatie e.d..



5. Resultaat

De belangrijkste features die uiteindelijk in de *Green Audiosearcher* zijn geimplementeerd zijn:

- FolderManager
  - Folder management: std::vector<Folder>, addFolder(), removeFolder(), toggleFolder()
  - Featuresetlijst van alle audiofiles in een folder: std::vector<AFFeatureSet> sets, analyseFolders()
- AudioFile / PlayableAudiofile
  - Dragndrop van OS naar applicatie: JUCE::FileDragAndDropTarget
  - Verwijderen van stilte: stripSilence();
  - Play/Stop en Open
  - Audio playback: getAudioBlock()
- AudioAnalyser
  - Feature analyse: analyseAudio()
  - FFT analyse: FFTWrapper
- AFFeatureSet
  - Van std::string en naar std::string functie voor het opslaan in een database
- SimilarityChecker
  - Euclidean Distance: getEuclideanDistance()
  - FeatureWeights bestuurbaar met sliders: std::vector<float>, std::vector<LabelSlider*>
  - Vinden van vergelijkbare AFFeatureSets: sortBySimilarity()

De interface is functioneel en werkt in ieder geval op 64bit windows. Hieronder is een screenshot van de interface te zien.

Nog niet uitgevoerde taken uit de MUSTs en SHOULDs zijn: RMS, Pitch, Database en Kleurgebruik. Hier ben ik net niet aan toe gekomen. In plaats daarvan heb ik erop gefocussed dat de meest fundamentele classes uit mijn ontwerp goed werken. Namelijk de AudioFile, FolderManager en de AFFeatureSet.

Met het testen van mijn programma lijkt het erop dat deze combinatie van features nog niet geschikt os voor wat ik wil bereiken of dat de similarityCheck nog niet goed werkt. Wel voelt de interactie logisch en fijn. De hoop is dus dat bij het itereren van het analyse algoritme, de gevonden audiofiles logischer worden. Zo zou het toevoegen van veel gebruikte analyseparameters goed kunnen werken en zou ik de zinnigheid per analyse parameter zorgvuldig kunnen uittesten.

Instructies

De ‘./Green AudioSearcher.jucer’ file is een *Projucer* file. Deze file wordt geopend met de *Projucer* applicatie van JUCE. Dit project maakt gebruik van de *Projucer v5.4.3* en dit is dan ook een vereiste om het project te builden. De *Projucer* is in staat multiplatform developeromgeving projectfiles te genereren. Ik heb in dit geval voor VisualStudio2017 en voor een Linux Makefile gegenereerd. Daarnaast is er in ‘./Tests/’ tot nu toe een test te vinden voor het AFFeatureSet object. De source code van de hele applicatie is te vinden in ‘./Source/’

Windows

Het meest recente VisualStudio2017 project is te vinden in ‘./Builds/VisualStudio2017/’. Je kan ook dit project opnieuw genereren met de JUCE Projucer v5.4.3. Het project vereist wel de JUCE framework. Eventuele builds zijn ook al te vinden in ‘./Builds/VisualStudio2017/x64/’

Linux

De meest recente LinuxMakeFile is te vinden in ‘./Builds/LinuxMakeFile/’. Deze is eveneens gegenereerd door de JUCE Project v5.4.3 en is hiermee dus ook opnieuw te genereren. Het project vereist wel de JUCE framework. De build is **niet** getest op een linux device en kan dus fout gaan.

6. Conclusie

De uiteindelijk ontwikkelde applicatie voldoet nog niet aan de verwachtingen. Wel werkt alles technisch gezien en is de interactie al uitvoerig te testen. Achteraf denk ik deze MVP heel slim was om naar toe te werken omdat ik nu goed op de interactie en het analyse-algoritme kan itereren. De interactie werkt fijn en ik denk dan ook dat dit een goed moment in het ontwikkelingsproces zou zijn om de interactie te verbeteren. Daarnaast denk ik dat het zaak is om de audioanalyse uit te breiden om betere auditieve vergelijkingen te kunnen doen. Daarbij wordt het ook heel belangrijk om meer te testen of alles goed werkt. Hierbij is mijn plan dus: Iteratie (interactie en audioanalyse) -> Testen -> Iteratie -> etc...

Technisch gezien merkte ik dat het werken met een framework, die voor standard library definities vaak eigen classes ontwerpt, soms erg verwarrend kan zijn. Hierdoor ben ik veel gaan nadenken over wanneer ik iets van de standard library wil gebruiken en wanneer van een dergelijk framework. Dit heeft me kritischer gemaakt over de syntax die ik gebruik. Daarnaast merk ik dat ik soms moeite heb met het implementeren van externe libraries of frameworks. Ook merkte ik dat ik deze keer in de war raakte bij het maken van een class diagram. Dit zijn dingen waar ik graag mee wil oefenen omdat het basale en belangrijke skills zijn.

Ik ben uiteindelijk tevreden over wat ik heb ontwikkeld. Ik betwijfelde op het begin sowieso of wat ik zou maken meteen aan mijn verwachtingen zou voldoen, hierom heb ik een Full en MVP design gemaakt. Het feit dat ik mijn applicatie zelf creatief gezien interessant vind om te gebruiken is voor mij een goede reden om het project al als geslaagd te zien. Mijn vervolg zoals ik eerder al noemde zal allereerst bestaan uit een cyclus van tests en iteraties. Zo wil ik wat ik nu heb staan perfectioneren voordat ik aan het *Full Design* ga werken.