Methodology

Introduction

This chapter outlines the approach that will be taken in the realisation of the SonicSketch application. An assessment of SonicPainter will be given to identify the features that will be incorporated, improved on or omitted. The major technologies being used in the project will then be introduced along with some justifications for their usage. These include the *Web Audio API*, *Tone.js*, *Paper.js*, *ClojureScript* and Facebook's *React* framework.

Assessment of SonicPainter

As has been mentioned previously, the SonicPainter application developed by William Coleman will form the basis of the work on SonicSketch. The basic design of SonicPainter works very well and is aligned with the goals of SonicSketch in the following ways:

- Provides a simple immersive score space that represents time on the x-axis and frequency on the y-axis.
- Allows multiple voices or timbres to be represented in this space so that all notes can be seen at a glance without having to click into separate tracks.
- Visualisation of vibrato effect using overlaid sine wave is suitable in theory but perhaps could be implemented better.
- Uses FM synthesis as the synthesis engine. This versatile and efficient synthesis technique is a good fit particularly for use in the web browser where performance can be an issue.

Some issues that have been identified with SonicPainter will now be addressed. It should be noted that as the software is only at a prototype stage some of the shortcomings outlined here may not be design decisions and instead may be implementation issues.

Usability issues

While the application is conceptually straightforward and beginner friendly, there are some issues that would make it difficult for a new user to begin using it. Starting

a note is obvious and involves simply clicking on the screen to anchor the start point. A line now becomes attached to from this point to the mouse pointer location, indicating that another press will finish the note. However, this instead adds a point to the note, indicated by the fact that a new line appears between this new point and the pointer position. It is unclear how to finalise the note. The solution is to press the **shift** modifier key and click. Some simple labeling could alleviate this issue, by perhaps using a small toggleable information box. Unfortunately, this issue repeats itself in a good deal of the functionality with additional features hidden behind keyboard shortcuts that would be difficult or impossible to discover.

Issues with audio visual connection

Some deficiencies were found with the mapping between visuals and audio. It is possible to draw multiple concurrent notes that share the same timbre suggesting that the audio would polyphonically sound the notes together. This is not the case however and only one note can be played back at the same from the same timbre. To achieve this, a separate voice must be used by clicking the up arrow. To remedy this issue, it could either prevent the creation of concurrent notes visually or allow multiple notes to sound together by making each voice polyphonic.

Invalid states

A serious technical issue with SonicPainter is the presence of invalid application states that are not properly handled in the code. These arise when the user performs a series of actions that are not properly handled either by catering for them or by disallowing them. An example of this is when the user starts a note and then changes mode to, for instance, vibrato. When this happens, the application allows the mode to change and carry out the vibrato change all the while keeping a line connected to the pointer position. From here it is not possible to complete the note without returning to the original mode and clicking multiple times.

This unexpected behaviour could be avoided by either disallowing mode changes during note creation or by ending the note when the mode has changed. These type of issues are common even in commercial applications as is illustrated in Horrocks [1999]. An example he gives is that of a calculator built by Microsoft that enters an invalid state in a number of different situations. To counter these issues a disciplined

approach will be taken to managing state in SonicSketch by performing *validation* operations and the use of *finite-state Machines* (FSM). Validation is a technique used to ensure that data is in the correct format and that any required information is present. A finite-state machine is a computational model that can be in only one of a finite number of states. These states may change due to external input which triggers transitions from one state to the next. They can prove useful in avoiding invalid states and to reduce the amount of conditional logic in the program.

Linear representation of frequency

SonicPainter represents a frequency range of 20-500 Hertz (Hz) on its Y-axis. This is mapped linearly from the on-screen coordinates to frequency values. It is not made clear why this range and mapping was chosen and may have been for aesthetic reasons. Psychoacoustically it may be suitable to map this linearly as this corresponds to the linear perception of frequency at the lower frequencies. This phenomenon is represented in the Bark Scale, which is a psychoacoustic frequency scale on which equal distances correspond to perceptually equal distances. However, the range of 20-500 Hz frequency is quite limiting given that the highest note of a piano can reach to 4186 Hz. In addition, there is a strong cultural expectation of logarithmic mapping due to the fact that the semitone is based on a logarithmic scale.

Technical approach

The technical approach taken in the realisation of SonicSketch will now be given. A number of specialist environments exist to realise a project such as this, including that offered by /Processing/[1] (www.proccessing.org) used by SonicPainter. However, it was decided, after some early experimentation (to be discussed), that the web browser would offer the key advantage of being easily accessible and, therefore, was the chosen platform. The technology that makes this viable is the Web Audio API. This will now be discussed alongside the libraries used in the project. Where applicable, the reasons for choosing the particular libraries will also be given.

[1]Processing is a creative coding environment that runs in the Java virtual machine

Audio in the modern Web Browser

The W3C Web Audio API specification allows audio processing to take place in the web browser. Audio generation and processing is defined using an `AudioContext` graph of connected `AudioNode` objects. While JavaScript processing is supported (by using a `ScriptProcessorNode` object), most processing takes place in optimized lower level languages such as C or C++. Advanced synthesis techniques are possible by connecting audio generating nodes to processing nodes. Audio generating nodes include the `OscillatorNode` to generate a periodic waveform and the `AudioBuffer` to playback audio waveforms. Processing nodes include the `GainNode` to adjust the amplitude of a signal and `BiquadFilterNode` filter the signal. Parameters of these nodes may be adjusted smoothly over time using the `AudioParam` interface to, for instance, slowly fade a synth sound in.

Tone.js

Tone.js is a Web Audio framework that provides several helpful abstractions and libraries to help interaction with the Web Audio API. A central aim of Tone.js is to enable some of the conveniences of DAWs and is formed on three tenets: musicality, modularity, and synchronization. An example of this is the flexibility it allows to express time values, eg. "`4n`" for a quarter note in metrical notation, `1` for a second and "`100hz`" to express 100 hertz, etc. These are all converted to seconds before scheduling them with the Web Audio API. A system called "just in time scheduling" ensures that tempo-relative values are not scheduled until the latest possible moment, thus ensuring that they reflect the latest tempo value.

The primary reason that Tone.js was chosen was for the signal system that it uses to make working with parameter modulation easier and more efficient than working directly with Web Audio API. This uses a constant signal generator running at audio rate connected to a `GainNode`. In addition, this value can be multiplied and summed using `GainNodes` native capabilities enabling performant signal processing operations on all parameters. For instance, *FM synthesis* generally requires that a relative relationship is maintained between the two oscillators, the carrier, and the modulator. When directly using the Web Audio API Param, a `ScriptProcessorNode` would need to be setup to calculate this, which is not as efficient or as straightforward as carrying out the calculations using Tone.js signals.

Paper.js

Paper.js is a descendant of Scriptographer, a scripting tool for Adobe Illustrator, a vector graphics program. It runs in the web browser canvas element, running in its 2d mode. Paper.js adds a number of features to the browser's native canvas element including a *scene graph*, geometry and vector abstractions, as well as tools to draw and animate shapes on-screen. The central abstraction in Paper.js and any vector system is the *path*. This allows for the specification of any shape by describing start points and endpoints for a series of paths. Curves can be added to these paths by adjusting an additional set of points associated with a path, the `handleIn` and `handleOut` points. These define control handles that alter the curvature of the line using Bezier mathematics.

A primary reason that Paper.js was chosen for SonicSketch is the path simplification algorithm that allows the data captured with freehand input to be simplified and smoothed. Instead of just plotting every point captured, an optimized subset of these points is used to reduce memory usage and speed up drawing. It is based on an algorithm by Philip J. Schneider published in Graphics Gems (1990). In addition, these simplified paths are more suited to mapping to control data for the audio system.

React framework

React is a web framework built by Facebook that aids the developer in updating the document object model (DOM), a process that is required when the state of the application changes. This was a role traditionally carried out by the web server and delivered to users as a static page. This saw a significant change however with the rise of single page applications (SPA) around 2010. The advantage of the SPA is increased interactivity and responsiveness to user input, allowing the look and contents of the page to update dynamically, as the user interacts with the webpage. To aid in the construction of these SPA's, a number of frameworks to help the process were introduced by the open-source community. Some popular early examples include Backbone.js and Angular.js. A technique that saw some popularity was a system called two-way binding which created a two-way link between the current state in the model and the visual appearance of the view. This, however, has a number of issues including some serious performance issues, in addition to some conceptual

problems [Whelpley, 2014].

React offers a simpler one-way binding system using what is termed the *virtual DOM*. This works by maintaining a virtual version of the DOM in a JavaScript. When the virtual DOM changes, the parts of the real DOM that require changing can be pinpointed and efficiently updated. This system has proven to be particularly beneficial when paired with *functional programming* techniques, a style of programming that encourages the use of pure functions as the primary building block of programs. In the case of working with the DOM, it can lead to not only an increase in efficiency in the rendering of the applications but also a simplification of the programming model as a secondary benefit. A number of projects have emerged that attempt to bring this secondary benefit beyond the realm of the DOM. This includes writing command-line programs [Demedes, 2017], writing web audio applications [FormidableLabs, 2017] and even for embedded electronics [Kasten, 2017].

ClojureScript

ClojureScript is a compile-to-javascript programming language that is based on Clojure, a modern Lisp that runs on the Java Virtual Machine (JVM). Lisp is a programming language that was invented by John McCartney in the 1960s and is known for its minimal syntax consisting primarily of parens. The word Lisp is derived from the term "List Processor" as Lisp source code and data structures are built around lists. Clojure and ClojureScript promote a functional programming style. ClojureScript and other functional programming languages such as Elm have seen an increase in usage in the past number of years as this paradigm has proved useful in managing complex stateful UIs. Some annotated examples of ClojureScript follow that show the basic building blocks of the language and will help in understanding the code walkthrough in the next chapter.

```
;; A list. It is quoted as otherwise it would treat it as a function call
'(1 2 3 4)

;; Call a function. In this case calling the multiply function.
(* 2 2)

;; Define a variable a
(def a 1)
```

```
;; Define a function b
(defn b [n] (* n n))

;; A vector is the most common way to represent sequential data in clojurescript
[1 2 3 4]

;; A hashmap is used for associative data and more often than not uses keywords
;; as keys. Keywords are used similarly to constant strings in other languages.
{:hello "world"}

;; Vectors and hashmaps are immutable and can't be changed after they've been created
;; An atom is used for data that needs to change (to model state changes for instance).
(def c (atom 1))

;; To change it's value use the reset! function.
;; Here it changes the c atom to 2.
(reset! c 2)

;; Clojurescript can interface with javascript in a straightforward manner
;; but uses a slightly different syntax. The term that is normally
;; used to describe this is js interop.
;; Calling a function:
(js/console.log "Hello world")
;; Calling a method on an object:
(.reload js/location)
```

This covers some basics that will be particularly helpful for readers that are familiar with C style languages. A more in depth discussion is beyond the scope of this document and interested readers should consult the many resources available online. The most commonly used build tool that is used with ClojureScript (and Clojure) is *Leiningen*, which takes care of managing the code dependencies and converting the code from ClojureScript to javascript. These dependencies are defined in the "project.clj" file. The primary dependencies used by SonicSketch, *Reagent* and *re-frame* will now be discussed.

Reagent & Re-frame

Reagent is a library that provides an idiomatic ClojureScript interface to React, allowing ClojureScript to harness the DOM manipulation facilities provided by React. This delegates the *side-effects* of rendering and manipulating DOM to

React's reconciler algorithm. Side-effects is a functional programming term to denote anything that is not related to the supplied arguments or return value of a function. This is normally object mutation (to change the state of the program) or input/output (I/O) operations, e.g. writing a file to disk, displaying graphics or playing a sound [Sylwester, 2015]. In addition to the interface to React, it provides a special reactive atom[2] that efficiently re-renders React components when the state changes.

Re-frame is a framework that uses Reagent's interface to React to manage views and it's reactive atom to manage state. It proposes a program architecture consisting of the following 6 elements:

1. Event dispatch
2. Event handling
3. Effect handling
4. Query
5. View
6. DOM

The majority of events that are dispatched are due to user interactions with the system (for instance on a mouse click). Event handling is the code that is run in response to these events. Re-frame submits that these event handlers should supply data to describe the side-effects rather than carrying them out in the handlers. Re-frame carries out this work which is typically to update the application state. This is stored in a single reactive atom and managed by the framework. A subscription system allows the view system to update when the state that it depends on changes. Finally, React updates the DOM to complete the cycle.

Conclusion

SonicPainter was discussed in some depth with a focus on issues that will be improved on in SonicSketch. The technologies being used in the SonicSketch app were described in detail including some brief justification behind these technical choices. This covered the Web Audio API technology that underlies the audio aspect of the project, as well as the libraries and frameworks being used to manage the views and state of the app.

Demedes, V. (2017) *Vadimdemedes/ink: React for interactive command-line apps.*

[2] An atom is the construct used in ClojureScript to contain data that needs to change over time.

Available at: https://github.com/vadimdemedes/ink (Accessed: 23 August 2017).

FormidableLabs (2017) *React-music: Make beats with React!* Formidable. Available at: https://github.com/FormidableLabs/react-music.

Horrocks, I. (1999) *Constructing the user interface with statecharts.* Harlow, England ; Reading, Mass: Addison-Wesley.

Kasten, D. (2017) *React-hardware: A React renderer for Hardware.* Available at: https://github.com/iamdustan/react-hardware.

Sylwester (2015) *The meaning of side-effect in Clojure - Stack Overflow.* Available at: https://stackoverflow.com/questions/31899630/the-meaning-of-side-effect-in-clojure (Accessed: 21 August 2017).

Whelpley, J. (2014) *Is AngularJS Fast Enough?, Jeff Whelpley.* Available at: https://medium.com/@jeffwhelpley/is-angularjs-fast-enough-98dcf96406c8 (Accessed: 23 August 2017).