Chapter 1

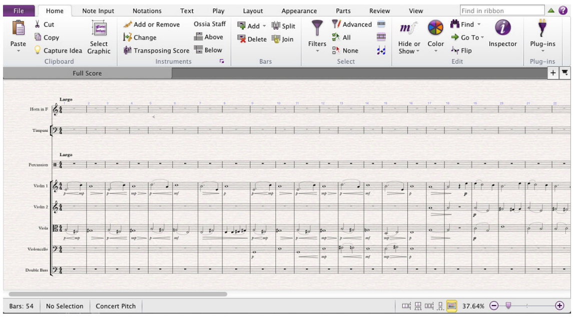
Introduction

1.1 Motivation

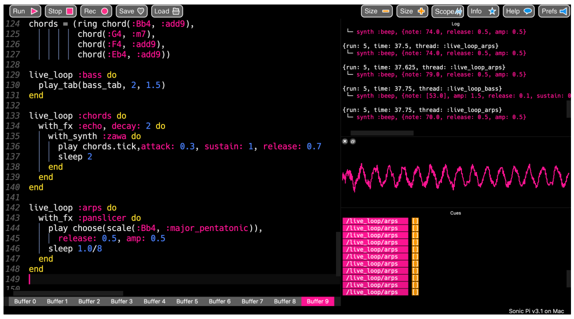
There exist many programs for music notation and composition. In Sibelius (fig 1.1 (a)), users write scores using traditional western music notation, whilst music is produced in the live programming interface Sonic Pi (fig 1.1 (b)) by writing Ruby code [1]. These require users to gain familiarity with a new interface, often with a large threshold to creating simple musical ideas. There are four times more spreadsheet users than programmers [24], it being the preferred programming language for many people [20]. I believe that this ubiquituity, along with the affordances of the spreadsheet, would enable new ways to interact with musical notation that capitalise on existing familiarities with spreadsheets and their data handling capabilities.

The use of grid structures is an established concept in music programs, with most sequencing software using one axis of the screen for time, and the other for pitch or musical parts. Chris Nash's Manhattan [19] (fig 1.1 (c)) uses a grid structure where formulae be define a cell's value, like in a spreadsheet. However, it is limited to columns defining tracks and rows corresponding to time. Advait Sarkar's SheetMusic [23] investigated including formulae with sound output within the spreadsheet paradigm. This also introduced abstracting time away from the grid, in this case using an incrementing global `tick` variable which could be referred to in formulae. Both axes can be used interchangeably for SheetMusic notation or non-musically interpreted markup that the user wishes to include, a concept idiomatic to Excel usage. Simple formulae such as `if(tick%2==0) p('snare') else p('kick')` allow musical structures to be defined without advanced programming knowledge but quickly become unwieldy for larger pieces, especially if they are not highly repetitive. Whilst other spreadsheet music projects exist [12], these simply use the spreadsheet as the medium for conventional sequencing with an auxiliary script to parse the grid and create musical output.

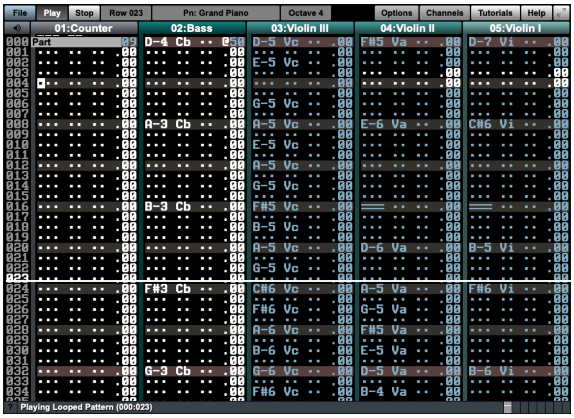
Excello (fig 1.1 (d)) is an Excel add-in for end-user music programming where users define music in the spreadsheet and can play it back from within Excel. It maintains the abstraction of time from the grid to keep the flexibility spreadsheets offer, but was designed so individual cells would not become too complex. Existing Excel functionality can be used, both accelerating the learning curve and increasing the available functionality.



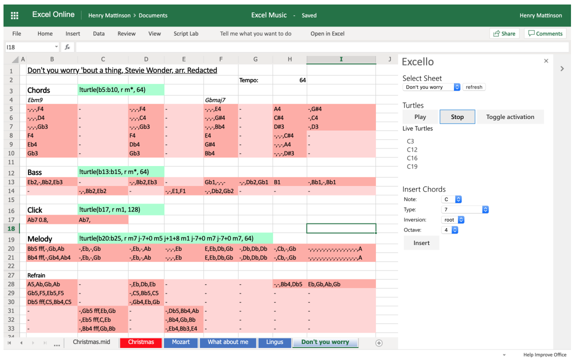
(a) Sibelius - an editor with playback for staff notation © Gorazd Bozic[1]



(b) Sonic Pi - code is written on the left and output information on the right



(c) Manhattan - columns of parts played from top to bottom



(d) Excello - a spreadsheet of music notation with a window for playback on the right

Figure 1.1: The interfaces of (a) Sibelius, (b) Sonic Pi, (c) Manhattan, (d) Excello

1.2 Outline of work

1. I designed and built a prototype for musical expression and playback within Excel satisfying the all the success criteria for the system.

2. Participatory design began using this initial prototype. Following formative evaluation sessions with 21 participants, issues and feature requests were identified. Users continued to give feedback as they used the updated prototypes.

3. A series of extensions were implemented, solving problems identified by participants and adding requested features.

4. A converter from MIDI to the Excello format was built and a large MIDI corpus translated.

5. Summative evaluation was performed with the participants. Evaluating the features implemented during participatory design and Excello's usability using the Cognitive Dimensions of Notation (CDN) framework [10], using Sibelius as a comparison.

Chapter 2

Preparation

This chapter shall first address the refinements made to the project proposal. Then I shall explain the tests that established Excel's suitability for musical development. Next, the initial design decisions for Excello shall be explained. The software engineering tools and techniques employed will then be introduced. Finally, the research conducted to decide to implement the converter from MIDI to Excello shall be summarised.

2.1 Proposal Refinements

The project designed a notation for defining music within a spreadsheet, along with a system for interpreting this to produce audio output. This continued to explore abstracting time away from the grid.

This would be implemented as an Excel add-in subject to successful initial testing. An add-in is a web application displayed within Excel that can read and write to the spreadsheet using the Office Javascript API[1]. Arbitrary additional data and markup can be included in the spreadsheet. Tests verified that audio output can be produced for music end-user music programming within Excel.

A sizeable addition to the project beyond the initial proposal was to perform participatory design [18] to advise on improvements beyond the initial prototype. Participants would be able to identify aspects of the current design that (don't) work well or add cognitive difficulty. This lead to the implementation of new features and improvements. As a result, the proposed extension of live-coding would only be implemented if deemed high priority by the participants. Participants who gained sufficient familiarity with the project were used for more informed summative evaluation.

The converter translated MIDI to a CSV file for use with Excello. Explanation on the choice of MIDI is provided below. It was also motivated by participants who wished to integrate Excello with their use of digital audio workstations such as Logic Pro, Ableton Live, and GarageBand.

[1]https://docs.microsoft.com/en-us/javascript/api/excel?view=office-js

2.2 Feasibility Analysis

The following section outlines the tests carried out to assess the feasibility of synthesising notes with data in a spreadsheet using an Excel add-in. All tests were carried out in Excel Online[2] using Script Lab[3], an add-in that allows users to create and test simple add-ins experimenting with the Office Javascript API. These add-ins can access libraries and data elsewhere online.

If add-ins were run in an older version of Internet Explorer, playing sound or using the Web Audio API would be possible.An add-in that played a WAV file verified that sound could be created. [17].

2.2.1 Note Synthesis Library

The Web Audio API allows audio to be synthesised from the browser using Javascript [17]. Creating a program for users to define and play musical structures requires synthesising arbitrary length, pitch and volume notes. To avoid the lower-level audio components (e.g. oscillators), I researched libraries that would allow me to deal with higher level musical abstractions of the synthesised notes. Sarkar's SheetMusic used the library tones[4], an API where only the pitch and volume envelope[5] of the notes. This also only included simple waveform synthesisers.

Tone.js[6] is a library built on top of the Web Audio API that provides greater functionality. An `Instrument` such as a `Synth` or `Sampler` is defined. The `triggerattackrelease` release method of these instruments allows a note of a given pitch, volume and duration to be triggered at a particular time. Notes are defined using scientific pitch notation (SPN) (e.g. `e.g. F#4`), name (with accidental[7] if required) (e.g. `F#`) combined with octave (`4`). As Script Lab can reference libraries from the Node Package Manager (NPM), I tested playing notes with pitch defined in the add-in Javascript.

2.2.2 Office Javascript API

To produce music within Excel, the musical output must be informed by the data in the spreadsheet. Previous tests defined notes in the add-in Javascript. To test the Excel API, I played a note with the pitch defined in the spreadsheet. This was extended so note playing, not just the pitch, was defined within a cell (`play(F#4)`), detected and executed.

Next, I was able to play a sequence of constant length notes defined in consecutive cells (`play(H1:K1)`). The range of cells was accessed using the Excel API and the values were

[2]https://office.live.com/start/Excel.aspx
[3]https://www.microsoft.com/en-us/garage/profiles/script-lab/
[4]https://github.com/bit101/tones
[5]How the note's volume changes over its duration
[6]https://tonejs.github.io/
[7]Sharp or flat symbol used for black notes on a piano.

played using the `Tone.Sequence` object. These tests confirmed Tone.js combined with the Excel API provided adequate functionality to implement the project as an add-in.

2.3 Excello Design and Language

2.3.1 Abstracting Time

Dave Griffith's Al-Jazari [15] takes place in a three-dimensional world where robotic agents navigate around a two-dimensional grid. The height and colour of the blocks over which the agents traverse determines the sound they produce. The characteristics of the blocks are modified manually by users at run-time whilst the agents are moving. The basic instructions have the agents rotate and move forwards and backwards in the direction that they are facing. There exists a dual formalism in both the agent's instructions and the block state. This design is intended to improve live coding accessibility, both when viewing performances and becoming a live coder.

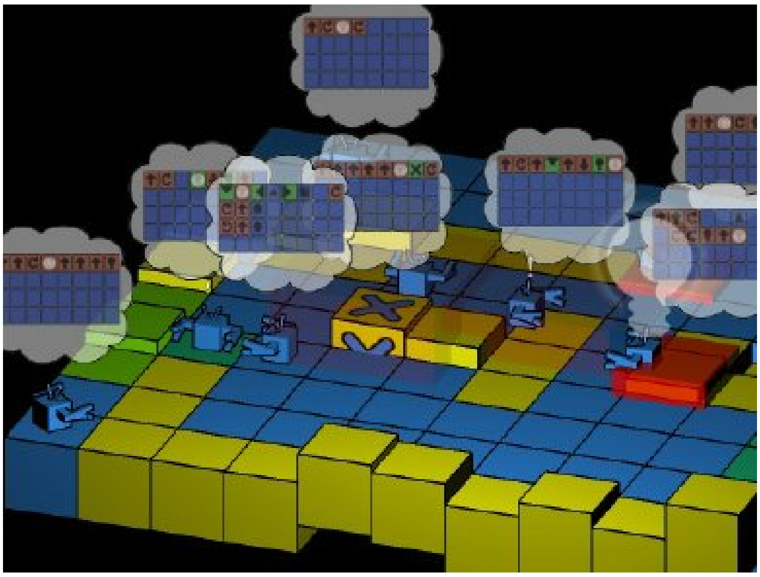


Figure 2.1: Programmed agents moving around a grid in Al-Jazari © Alex Mclean

In Al-Jazari, the agents are programmed with symbols corresponding to different movements in thought bubbles above them. This is not suitable for programming within spreadsheets where all data must exist alphanumerically within cells. If an agent was to continue moving forwards many times in a row, it would become tiresome to keep adding this symbol. This is less of an issue in Al-Jazari where the grid only measures ten cells wide and long.

Having a cursor navigate around a cartesian plane is the method used by turtle graphics. Just as this concept is used in Al-Jazari for agents to play the cell they occupy rather than colour it, it is suitable for spreadsheets. The turtle abstraction is employed by Excello by defining notes in cells and agents, known as turtles, to move through the spreadsheet activating them. To play a chord, multiple turtles must simultaneously pass through multiple cells corresponding to the notes of the chord. This maintains high notational consistency but sacrifices the abstractions for musical structures that are available in languages like Sonic Pi - `chord('F#', 'maj7')`. By implementing methods in the add-in to add the notes of chords to the grid, abstractions can be maintained whilst preserving consistency and cleanness in the spreadsheet itself.

Turtle are the crux of the Logo programming language [9] where turtles are programmed entirely by text to produce graphical output. For example, `repeat 4 [forward 5 right 90]` has a turtle move forwards 50 units and turn 90 degrees to the right four times to draw a square. A similar method is employed in Excello but the language is designed to be less verbose.

2.3.2 Initial Prototype Design

Notes are placed in the cells of the spreadsheet and turtles are defined to move through the grid using a language based on Logo. The notes in cells are played when turtles move through those cell. When the play button is pressed, the melodic lines produced by all turtles defined in the grid are played concurrently. Turtles are defined with a start cell, movement instructions, the speed they move through the grid at (cells per minute) and the number of times they repeat their path. As in Al-Jazari, distance in space maps to time [16]. Excello extends upon this as turtles can move at different speeds. Therefore parts with longer notes can be defined more concisely and phase music[8] can be easily defined. The Excello turtle movement instructions are explained below with examples.

Turtles begin facing north. The move command, `m`, moves the turtle one cell forward. Like Logo, turtles always move in the direction they are facing. The commands `l` and `r` turn the turtle 90 degrees left and right respectively. Logo commands are repeated with `repeat` followed by the number of repeats and the commands [9]. To reduce verbosity, single commands are repeated by placing a number immediately after it. For example, `m4` has the turtle move forwards four cells in the direction that it is facing. The direction a turtle is facing can be defined absolutely with `n`, `e`, `s` and `w` to face the turtle north, east, south and west. These rotate the turtles rather than moving them to maintain the consistency that turtles always move in the direction they face. To change notes' volume, dynamics (`ppp`, `pp`, `p`, `mp`, `mf`, `f`, `ff`, `fff`) can be placed within the turtle instructions. Any notes played after are played at that dynamic. Just as dynamics in western notation are a property of the staff, not individual notes, dynamics were originally defined in the turtle not notes. To repeat multiple instructions, they are placed in brackets followed by the number of repeats. For example, `(r m5)4` defines a clockwise path around a five by five square. This seven character example is equivalent to the above Logo example which requires 29 characters. This repetition of instruction is why relative movements `l` and `r` are included in the language despite being less explicit than the compass based directions. This is demonstrated in figure 2.2.

[8]Music where identical parts are played at different speeds

	A	B	C	D
1	Start: A2	Instructions:	e m3 (r m2)2 m	
2	C4	D4	E4	F4
3				G4
4	C4	C5	B4	A4

Figure 2.2: The instructions for a turtle following the path of notes from A2 to A4

It may not be convenient to constrain each melodic line to a single path of adjacent cells. Just as conventional score notation often spans across multiple lines, splitting up parts is a useful form of secondary notation. This requires the turtle to move to non-adjacent cells. For graphic drawing in Logo, lifting the pen allows the turtle to move without colouring the space beneath it. However, here the number of steps the turtle takes doesn't affect the output, only the cells it colours does. However, the musical output of Excello is dependent on the turtle's spacio-temporal information, so this would introduce large rests. Analogous to lifting the pen for graphical turtles, the turtle could enter a mode where it passes through cells immediately without playing them until it is placed back in playing mode. Here the actual path of the turtle is insignificant, only the cell it ends up in. I have therefore added jumps to the language. This can be defined with `j` in absolute terms with a destination cell (e.g. `jA5`), or relatively (e.g. `j-7+1`), with the number of rows and columns jumped. An absolute jump may be more explicit for human readers, but relative jumps facilitate repeats jumping to different cells each time. For example, `r (m7 j-7+1)9 m7` plays 10 rows of 8 cells from top to bottom playing each row left to right. A jump is demonstrated in figure 2.3.

	A	B	C	D
1	Start: A2	Instructions:	r m3 jA4 m3	
2	C4	D4	E4	F4
3				
4	G4	D4	E4	C4

Figure 2.3: The instructions for the path from A2 to D2 then A4 to D4

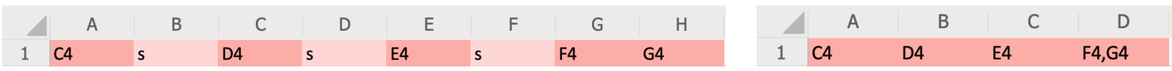
The language for turtle movement instructions is summarised by the following context-free grammar, $(N, \Sigma, S, \mathcal{P})$. Non-terminal symbols N = $(\langle\text{Full Instructions}\rangle, \langle\text{Instruction Series}\rangle, \langle\text{Single Block}\rangle, \langle\text{Command}\rangle, \langle\text{Relative}\rangle, \langle\text{Absolute}\rangle, \langle\text{Sign}\rangle, \langle\text{Cell}\rangle, \langle\text{Dynamic }\rangle)$, terminal symbols $\Sigma = (z{\in}\mathbb{Z}, n{\in}\mathbb{N}, c{\in}\,\texttt{[A-Za-z]}^+, \texttt{m}, \texttt{j}, \texttt{l}, \texttt{r}, \texttt{n}, \texttt{e}, \texttt{s}, \texttt{w}, \texttt{+}, \texttt{-}, \texttt{ppp}, \texttt{pp}, \texttt{p}, \texttt{mp}, \texttt{mf}, \texttt{f}, \texttt{ff}, \texttt{fff})$ and starting symbol $S = \langle\text{Full Instructions}\rangle$. The set of grammar rules $\mathcal{P}$ are shown in figure 2.4:

Notes are defined in the cells using SPN. Empty cells are interpreted as rests. To create notes longer than a single cell, the character `s` sustains the note that came before it as in figure 2.5 (a). A cell can be subdivided time-wise into multiple notes with multiple comma

$$
\begin{array}{rcl}
\langle\text{Full Instructions}\rangle & ::= & \langle\text{Instruction Series}\rangle \\
\langle\text{Instruction Series}\rangle & ::= & \langle\text{Single Block}\rangle \mid \langle\text{Single Block}\rangle\langle\text{Instruction Series}\rangle \\
\langle\text{Single Block}\rangle & ::= & \texttt{(}\langle\text{Instruction Series}\rangle\texttt{)}n \mid \langle\text{Command}\rangle \\
\langle\text{Command}\rangle & ::= & \texttt{m}z \mid \langle\text{Relative}\rangle \mid \langle\text{Relative}\rangle z \mid \langle\text{Absolute}\rangle \mid \langle\text{Dynamic}\rangle \\
 & & \mid \texttt{j}\langle\text{Cell}\rangle \mid \texttt{j}\langle\text{Sign}\rangle n\langle\text{Sign}\rangle n \\
\langle\text{Relative}\rangle & ::= & \texttt{l} \mid \texttt{r} \\
\langle\text{Movement}\rangle & ::= & \texttt{n} \mid \texttt{e} \mid \texttt{s} \mid \texttt{w} \\
\langle\text{Sign}\rangle & ::= & \texttt{+} \mid \texttt{-} \\
\langle\text{Cell}\rangle & ::= & cz \\
\langle\text{Dynamic}\rangle & ::= & \texttt{ppp} \mid \texttt{pp} \mid \texttt{p} \mid \texttt{mp} \mid \texttt{mf} \mid \texttt{f} \mid \texttt{ff} \mid \texttt{fff}
\end{array}
$$

Figure 2.4: The grammar rules, $\mathcal{P}$, for turtle movement instructions in Backus-Nour form

separated notes as in figure 2.5 (b). This was designed so the length a cell corresponds to is not bound by the length of the smallest note in the piece. For example, a piece defined primarily with crotchets (one unit) but with occasional quavers (half a unit) does not necessitate double the number of cells and introduce many additional `s` cells in the entire piece.



(a) Using sustain cells (b) Using a multi-note cell

Figure 2.5: Two identical motifs defined by using sustains or with multi-note cells

2.4 Software Engineering

2.4.1 Requirements

The success criteria of the project are:

1. Implementation of an system for music playback within a spreadsheet where users can:
   - Play individual notes and chords with defind durations.
   - Define multiple parts.
   - Play loops.
   - Define sequences of notes and chords and be able to call these for playback.
   - Define the tempo of playback.
2. Participatory design with formative evaluation sessions.

3. Summative evaluation with participants who have gained familiarity with the system.

4. Implementation of a converter from MIDI to the spreadsheet representation.

5. In addition to these, the following extension work was completed:

   - Implement additional features that arise from participatory design.
   - Explore a compressive conversion from MIDI to the Excel system.

2.4.2 Tools and Technologies Used

Table 2.1 outlines the tools, languages and libraries used.

Software	**Type**	**Usage**
Scriptlab	Add-in	Writing initial Excel add-in tests.
Typescript	Language	Writing Excello. Used for static type checking and Javascript libraries.
Yeoman[9]	Add-in Generator	Creating the blank Excel add-in project.
NodeJS	Javascript Environment	Manage library dependecies and run local web servers.
Surge[10]	CDN	Hosting Excello online for participants' use.
Jupyter Notebook	Python Environment	Implementing the MIDI to Excello converter.
Tone.js	Library	Synthesising and scheduling sound via the Web Audio API.
tonal	Library	Generating the notes of chords.
Mido[11]	Library	Reading MIDI files in Python.

Table 2.1: Tools used during the project

2.4.3 Starting Point

Having used the Yeoman generator to create an empty Excel add-in, all the code to produce Excello and the MIDI converter was produced from scratch using the libraries described above.

I had written simple Javascript for small websites, but had no experience using Node, libraries or building a larger project. Having never used any of the libraries before, reviewing the documentation was required before and during development. I had gained experience with Python and Jupyter Notebooks from a summer internship.

[9] A generator for scaffolding Node.js web applications, https://github.com/OfficeDev/generator-office
[10] Static webpage publishing tool and hosting https://surge.sh/
[11] https://mido.readthedocs.io/en/latest/

2.4.4 Evaluation Practices

To best tune Excello's to the needs of potential users, formative evaluation sessions were carried out with participants. A spiral development methodology [6] was used. This iterates the following steps: determining objectives, identify problems and solutions, develop and test, deploy and prepare next iteration. This was more suitable than sequential document-driven process methods such as the waterfall method. Due to the number of participants and the project timeframe, there were only two major development iterations with additional incremental updates. The first prototype, and the second fixing issues and implementing requests from participants.

Summative evaluation was carried out with users involved in participatory design. Therefore, experienced users of Excello could be used for evaluation despite the product not yet being released in the public domain.

2.5 MIDI files

MIDI is a communications protocol to connect electronic musical instruments with devices for playing, editing and recording music. A MIDI file consists of event messages describing on/off triggerings to control audio [14]. MIDI files were designed to be produced by MIDI controllers such as electric keyboards. As such, MIDI files contain controller specific information that is not necessary for the creation of an Excello file. Musical formats such as MusicXML specify the musical notation and could be suitable for conversion to Excello.

Many programs support importing and exporting MIDI files. By converting MIDI files to the Excello notation, Excello is more integrated into the environment of programs for playing, editing and composing music. Furthermore, there exist many datasets available for MIDI [13] which can immediately be played back for comparison.

Chapter 3

Implementation

This chapter shall explain how turtles are defined and cover the remaining features of the initial prototype. The format and results of the formative evaluation using this initial prototype shall be summarised. Next, the design decisions and changes that were made to Excello during the participatory design process will be discussed. Then, the technical details of Excello and the MIDI to Excello converter will be explained. Concluding with an overview of the project repository.

3.1 Initial Prototype

Notes and turtles can be defined in any cell. Turtles' interpretation of cells is shown in table ??. When the Excello add-in is opened, a window opens on the right side of Excel. Play and stop buttons can be used to launch all the turtles defined in the spreadsheet and initiate playback. Playback is a realistic piano sound.

Interpretation	**Format**
Note	Name (A-G), optional accidental and octave number e.g. `F#4`
Sustain	`s`
Multiple notes subdivided in time	Notes, rests or sustains separated by a comma. Rests must be a space or an empty string e.g. `E4,,C4,s`
Rest	Any cell not interpreted as a note, sustain or multi-note.

Table 3.1: Interpretation of cells.

3.1.1 Turtles

Turtles are defined by
`!turtle(<Starting Cell>, <Movement>, <Speed>, <Number of Loops>)`

Activation

"`!`" indicates the turtle will be activated when the play button is pressed. Just as digital audio workstations allow track muting and soloing, this can be used to modify which turtles play without losing their definitions.

Starting Cell

The turtle's starting cell, which is also played, is a cell reference. As with Excel formulae, this is a concatenation of letters for the column and numbers for the row.

As each turtle only plays one note at a time, multiple turtles must be defined to play polyphonic music such as chords. It was believed that users would define turtles following identical paths but in adjacent rows or columns. Multiple turtles following identical paths but starting from adjacent cells are defined using the existing Excel range notation for the starting cells. "`A2:A5`" would define four turtles in the cells A2,A3,A4,A5. This prevents writing multiple turtle definitions differing in only the start cell row.

Movement

Turtles start facing north. The language for programming turtle movement is discussed in the preparation chapter. Using brackets to repeat movements was not implemented by the start of the participatory design process.

Speed

An optional third argument is the speed of the turtle relative to 160 cells per minute. The default, 1, corresponds to 160 cells per minute. "2" would move the turtle at 320 cells per minute. Relative speed was used so it would be easier to tell the speed relation between turtles. This particularly suits phase music. Arbitrary maths can be provided, allowing turtles' speeds to be irrational multiples of each other.

Number of Loops

An optional fourth argument defines the number of repetitions of the turtle's entire path. By default, the turtle loops infinitely. Repeating parts (e.g. the cello of Pachabel's Canon in D) therefore only need defining once.

3.1.2 Highlighting

To assist recognising notes and turtles, when the play button is pressed, cells are highlighted. Activated or deactivated turtle definitions are highlighted green. Cells containing definitions of notes, or multiple notes, are highlighted red. Sustain cells are highlighted a lighter red, showing correspondence to notes whilst maintaining differentiation.

3.1.3 Chord input

To use the musical abstractions of chords whilst keeping the paradigm that a turtle is responsible for up to one note at any time, a tool to add chords is included. The note, type, inversion and starting octave of the chord are inputted in four drop-downs. The insert button enters the notes of the chord into the grid. If a single cell or range taller than it is wide is highlighted in the spreadsheet, the notes are inserted vertically starting at the top-left of the range. Otherwise, the notes will be inserted horizontally. Whether the turtles are moving horizontally or vertically both chords and arpeggios[1] can be easily defined. Thus, helpful musical abstractions are still available whilst keeping the cleanness of the turtle system.

3.2 Formative Evaluation

To guide development to best suit users, participants were involved in formative evaluation. 21 musical University of Cambridge students, across a range of subjects took place in the participatory design process. Initially, one-on-one tutorials with the initial prototype were given followed by a short exercise. After both of these, users were interviewed on how they found Excello, drawing particular attention to actions that they found particularly unintuitive or requiring notable mental effort. Comparisons were made to musical interfaces that participants were already familiar with. The sessions were audio recorded to prevent jotting down notes causing delays. These were later typed up. The ethical and data handling procedures are discussed in the evaluation chapter.

To realistically simulate how Excello would be used, participants carried out an exercise of their choice. Often this was transcribing a piece from memory or traditional notation into the Excello notation. Two exercises were provided if participants had no immediate inspiration; transcribing a piece from western notation or changing existing Excello notation.

These sessions were carried out at in January 2019. Participants were asked to continue using Excello until the summative evaluation sessions in March. Additional feedback was given as participants used Excello in their own time. This also ensured the summative evaluation was done with users with sufficient experience of the interface.

3.2.1 Issues and Suggestions

The issues and suggestions from the participatory design process are summarised below.

Turtle Notation

Dynamics in the turtle instructions made establishing the turtle's path harder as not all commands related movement. As the dynamics weren't next to the notes they corresponded to, knowing the volume of a note or where to place the dynamics within the turtle to apply to notes in the spreadsheet was challenging. The initial prototype had no way to assign a dynamic to notes in the first cell. The starting cell could be empty but this was inconvenient for looping parts as it would be included in the loop. Users not familiar with western notation dynamics found them unintuitive. Furthermore, these discrete markings do not make available a continuous volume scale.

[1] Where the notes of a chord are played in rising or descending order

When transcribing a piece, dividing its tempo by 160 for the relative speed caused unnecessary work. Users also forgot whether relative speed referred to time spent in each cell or how quickly the turtle moved. Following the tutorial, users often had to check the position and meaning of turtle arguments.

As the number of dynamics and movement commands grew, instructions became long and establishing turtle behaviour became cognitively challenging. Some users confused the "`s`" within the turtle instructions to mean sustain (as it does in cells) and not south.

Feedback

It was often unknown if pressing play registered, especially if the workbook saving delayed Excello's access to the spreadsheet. If a turtle had accidentally been left activated, the entire grid required searching to locate it. Users requested a summary of active turtle locations in addition to the highlighting.

MIDI conversions

Users of production software said importing and exporting MIDI files would be helpful. If working with an existing MIDI file, converting that into the Excello notation would be convenient. Exportation would let Excello be used to create chord sequences, bass lines and the piece structure, before adding additional effects and recording in digital audio work stations.

Sources of effort when writing

After inputting notes in the grid, the number of cells the turtle had to move required counting. This is often in a straight line. Whilst the Excel status bar allows users select cells and immediately see how many there are, this is unproductive. This was particularly inconvenient when users were writing notes and periodically testing what they had written so far. Some users instructed turtles to move forward significantly more steps than required to prevent counting. This is not feasible for looping parts. It was suggested that turtles could figure out how far they should move.

Instructions with repeated movements such as moving to the end of a line and jumping down to the beginning of a line below, required a lot of repetition.

Many of the notes in melodic lines are in the same octave. As such, repeatedly writing out the octave number was tiresome. One user made a comparison to LilyPond [22] where if the note length is not defined, the previous length would be used.

Some users find it more intuitive to consider a melodic line by the intervals between notes rather than note names. A modulated[2] melody line required writing out again and could not be derived quicker from the original version.

[2]Where the pitch of every note has moved by the same amount.

Chords

Most users used a small subset of the available chord types but had to find these in a large list. Separation of the more common chords was requested. Initially, notes inserted vertically had the lowest note at the top with pitch increasing down the column. Because higher pitch notes appear higher up the staff, it was suggested that inverting the order would be more intuitive. Initially, it was unclear what the different drop-downs corresponded to, with some users selecting the 7 from the octave number to try and insert a Maj7 chord.

Activation of turtles

When toggling turtles' activations, entering the edit mode for each to add or remove the exclamation mark was very tedious.

3.3 Second Prototype

Following the formative evaluation sessions and feedback, additions and modifications were made to solve the problems and opportunities that arose.

3.3.1 Dynamics

To help extract a turtle;s path and establish notes' volumes, dynamics are instead inserted in the cells after the note, separated by a space as in Manhattan [19]. As before, this will persist for all following notes until the volume is redefined. A single turtle definition with multiple start cells can now play parts of different volume. However, notes in the grid can be limited to only playing at their given volume. To play the same notes at a different volume, a new path must be made. Overall, the new system was believed to be preferable.

To use a continuous volume scale, in addition to existing dynamic symbols, a number between 0 and 1 can instead be provided where 0 is silent and 1 is equivalent to fff.

3.3.2 Nested Instructions

Nested instructions with repeats reduce turtle instructions and more easily encorporate repeated sections or movements. Multiple commands placed within parentheses followed by a number are repeated that number of times. Whilst the fourth argument of the turtle repeats the turtle's entire musical output, repetitions within the movement instructions allow paths to be defined more concisely.

3.3.3 Absolute Tempo

The turtle's speed is defined by cells per minute, rather than the initial relative value used initially. However, values less than 10 were interpreted in the original relative way for backwards compatibility with participants' existing work. To maintain consistency in a production version, this will be removed so speed must be defined absolutely. Speed and dynamics will be different orders of magnitude and hence reduce the confusion between them.

3.3.4 Custom Excel Functions

Two custom Excel functions were implemented to aid composition. One to defines turtles and a second transposes notes.

Excello.Turtle

The existing formulae writing tools provided within Excel are utilised. When writing a formula, a prompt informs users the position of arguments and whether they are optional. This outputs the turtle definition as text. Other cells can be referenced. For example, all turtles could reference a single cell for their speeds. Relative tempi can be implemented by the speed argument of each turtle being a multiple of this global speed as shown in figure 3.1.

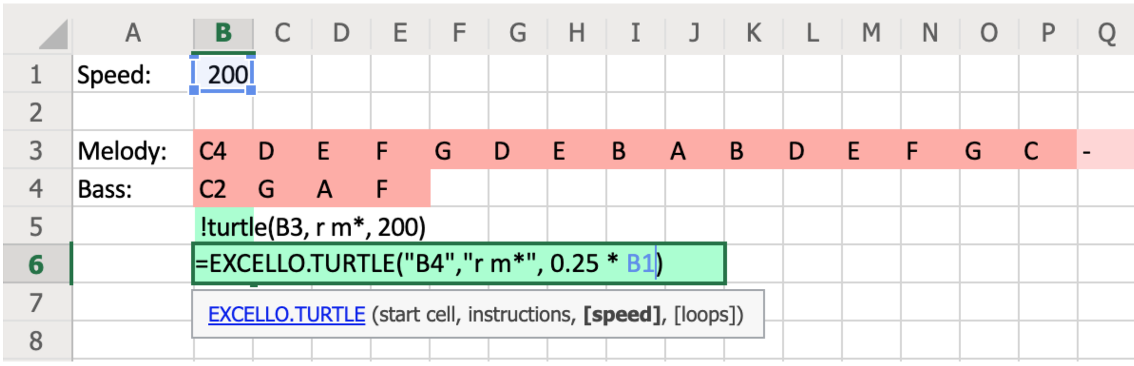


Figure 3.1: Defining a turtle using the EXCELLO.TURTLE function.

Excello.Modulate

A modulating function lets melodic lines be defined by the intervals between notes and provides easy modulation of existing sections of a piece. The function takes a cell and an interval and outputs the cell with any notes transposed by the interval, maintaining any dynamic. A section can be modulated by calling this function on the first note with a provided interval and using the existing drag-fill functionality of Excel. By using the previous note and one of a series of intervals as the arguments, a melodic line can quickly be produced from a starting note and a series of intervals as shown in figure 3.2.

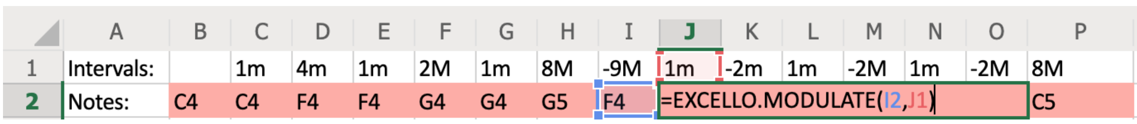


Figure 3.2: Transposing notes using the EXCELLO.MODULATE function.

3.3.5 Sustain

To prevent confusion between the south instruction and sustains. The symbol "-" sustains a note. This was chosen because it is light and also has some similarity to a tie[3]. Again, to maintain backwards compatability, "s" in a cell is still interpreted as a sustain.

3.3.6 Active Turtles

To show that active turtle definitions have been recognised, a list of locations of the active turtles is given below the play button. This also helps find spurious turtles not intended to be activated.

3.3.7 Automatic Movement

To prevent counting the cells in a line, `m*` instructs a turtle to move as far as there are notes or sustains defined in the direction it is facing. After adding more notes, the turtle instructions do not need editing before pressing play. A part may be meant to finish with a number of rests. As rests are notated with blank cells, a method to extend the path to include these rests was required. A cell can be explicitly defined as a rest with ".". This is required if multiple turtles were playing a repeating section where one does not have its final cell as a note, sustain or multi-note cell. Without an explicit rest, the turtle would repeat too soon and the parts would be out of phase.

3.3.8 Inferred Octave

Octave numbers are inferred if omitted. Two methods were considered. Firstly, as most intervals within melodic lines are small, the nearest note could be played. Whilst this may require the fewest explicit statements of octave numbers, it would be hard to find the octave of a given note. The last defined octave in the path would need finding and then all subsequent notes walked through keeping track of the octave. The second consideration was to always use the last defined octave. Whilst this may require many octave definitions around the boundary between octaves, it is easier to find the octave of a note by backtracking. The second option was implemented.

3.3.9 Chords

To aid entering common chords, common types are repeated in a section at the top of the type drop-down. The chord drop-downs layout was improved with labels to make it clearer what the values refer to. If the notes were entered vertically, the order was reversed, increasing correspondence with traditional staff notation.

[3]A line to increase the length of a note by joining to another.

3.3.10 Activation of turtles

A “Toggle Activation” button was added to the add-in window. When a cell or range is highlighted in the spreadsheet, the activation of any turtle definitions in this range will be toggled when the button is pressed. This significantly decreases the time to toggle activations as only two clicks are required rather than entering the cell edit mode to add or remove an exclamation mark.

3.4 Final Prototype Implementation

This section discusses the underlying implementation of the final prototype, following the participatory design. Excello consists of three main parts. The first, and largest, is the turtle system for playing the grid contents. The second is the method for inserting the notes of chords into the grid. Thirdly, turtle input and modulation is made available through the custom Excel functions.

When the play button in the add-in window is pressed, the turtle definitions within the grid are identified. For each, the starting cell and movement instructions are used to establish the contents of the cells which it passes through. This is converted to a series of note definitions - pitch, start time, duration, volume. The speed and loop parameter are used to create the structure interpreted by the Tone.js library to schedule and initiate playback. An overview of the data flow and subtasks required to create the musical playback is shown in figure 3.3.

An extension of the `Tone.Instrument` class is a `Sampler`. This interpolates between a set of pitched samples to create notes of arbitrary pitch and length. A sampler is loaded using the Salamander grand piano samples which includes four pitches (out of a possible 12) per octave. This accurately interpolates notes whilst reducing loading times and storage requirements.

3.4.1 Identifying Cells

Using the office API the sheet names are loaded and a drop-down menu for sheet selection is populated. When a user presses play, the cell values from the selected sheet are loaded. Then, the cell contents can be analysed for highlighting and calculating the musical output. Cells containing at least one note definition are highlighted red. A cell defining a note must contain a note name, an optional accidental, optional octave number, and optional volume instruction following a space in the form of a dynamic marking or number between 0 and 1. Notes are identified using the following regular expression:

```
'^[A-G](#|b|)?[1-9]?( (0(\.\[0-9]+)?|1(\.0)?|ppp|pp|p|mp|mf|f|ff|fff))?$'
```

Cells containing multiple definitions are split using commas. The resulting strings are trimmed of starting and ending whitespace and then must either be a note, a sustain

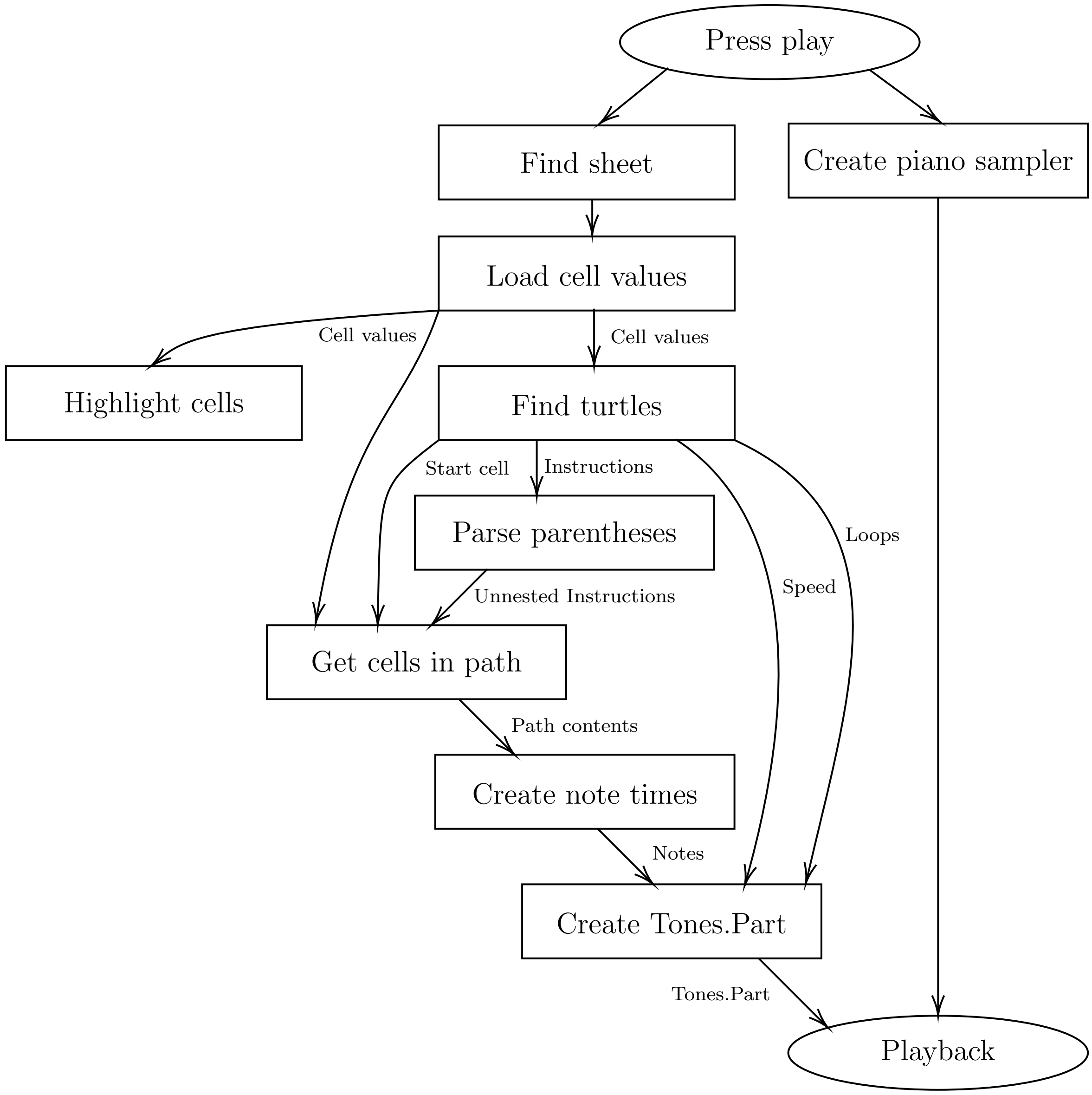


Figure 3.3: An overview of the playback algorithm and dataflow of Excello

("`-`" or "`s`"), explicit rest ("`.`") or an empty string (created from trimming a rest). Cells matching "`-`","`.`" or "`s`" are highlighted a lighter red. Turtle definitions are tested using:

```
'^(!turtle\().*(\))$'
```

and cells containing turtle definitions are highlighted green. The same regex is used to identify definitions of turtles. The address of cells containing a turtle definition are added as text nodes to the live turtle section of the add-in window.

3.4.2 Parsing Movement Instructions

First, the movement instructions are converted to a single unnested list of movements (e.g. "`(r m2)2`" becomes "`[r, m2, r, m2]`") so the turtle's path can be established. Initially the `parse` method of the Parenthesis[4] library seemed suitable for aiding in this string manipulation. This parses strings containing parenthesis into a nested array. For example, `parse('a(b[c{d}])')` gives `['a(', ['b[', ['c{', ['d'], '}'], ']'], ')']`.

This suggests that a string like "`(r m2)2`" would become `['(', ['r m2'], ')2']`. By removing the brackets from the strings within the array, a simple recursive method could be built to output `'r m2 r m2'` from `[['r m2'], '2']`. However, upon testing this, an undefined array was outputted. From investigating the source code, I established that strings with a number following a closing parenthesis would all produce an error. Substituting characters for the numbers or placing a symbol before all numbers and then later reverting this change would allow the library to be used. Instead, using the method employed by Parenthesis as inspiration, I implemented my own parsing function tailored for the needs of Excello.

This has two main steps. Firstly, the deepest bracketed expression is identified and stored in an array with the brackets removed. This expression is replaced in the original string with the string '___x___' where x is the index of this expression stored in the array. This is repeated until the original string contains no brackets. Secondly, a recursive function uses the indices placed between the '___' to reconstruct the string in the desired array format. This method is outlined in algorithm 1. The Typescript implementation is in Appendix A.1.

Having submitted a bug report on the Parenthesis Github, and implemented my own method for parsing the turtle movement instructions, I implemented a fix to the Parenthesis library. The existing function performed the initial replacement with the string '___x'. Therefore the x and following numbers would concatenate forming a single number, causing the library to fail. Utilising my method of having an identifier before and after the index number fixed the issue. I added this fix and additional tests to the Parenthesis to verify my method and ensured that previous tests all passed before submitting a pull request to the developers. This has since been merged into the library and published.

I wrote an additional recursive method to unnest the outputted array of this function into a single stream of instructions. An empty string, s, is initialised. For each item in the array, if it is an array, unnest the contents recursively. If not, it will be one or more single movement instructions. If the last item was an array the result of that array being unnested is added to s. If the first item in the single movement instructions is a number, the result of the array being unnested is added to s that number of times. The remaining instructions are added to s. This is outlined in algorithm 2. The implementation is shown in Appendix A.1.

[4]https://www.npmjs.com/package/parenthesis

Algorithm 1 Parsing bracketed expression. `str.replace(regex,f)` performs `f(s)` on the first substring, `s`, of `str` matching the regular expression `regex`.

```
procedure parseBrackets(str)
    idPadding ← '___'
    unnestedStr ← []
    deepestLevelBracketsRE ← RegExp('\\([^\\(\\)]*\\)')
    replacementIDRE ← RegExp('\\' + idPadding + '([0-9]+)' + idPadding)

    procedure replaceDeepestBracket(x)
        unnestedStr.push(x.substring(1, x.length-1))
        return idPadding + (unnestedStr.length - 1) + idPadding
    end procedure

    while deepestLevelBracketsRE.test(str) do
        str = str.replace(deepestLevelBracketsRE,replaceDeepestBracket)
    end while
    unnestedStr[0] = str

    procedure reNest(outerStr)
        renestingStr ← []
        while There is a match of replacementIDRE in outerStr do
            matchIndex ← index of the match in outerStr
            matchID ← ID of the match (number between padding)
            matchString ← matched string

            if matchIndex > 0 then
                renestingStr.push(outestStr.substring(0, matchIndex))
            end if
            renestingStr.push(reNest(unnestedStr[firstMatchID]))
            outestStr = outestStr.substring(matchIndex + matchString.length)
        end while
        renestingStr.push(outestStr)
        return renestingStr
    end procedure

    return reNest(unnestedStr[0])
end procedure
```

Algorithm 2 Unnesting a parsed bracketed expression.

```
procedure processParsedBrackets(arr)
    s ← ''
    previousArr
    for v in s do
        if v is an array then
            previousArr ← processParsedBrackets(v)
        else
            if previous instruction was an array then
                s ← s + previousArr
                if next instruction is a number then
                    s ← s + previousArr, that number of times minus one
                end if
            end if
            s ← s+ remaining instruction in v
        end if
    end for
    return s
end procedure
```

3.4.3 Getting Cells in Turtle's path

If the first argument in the turtle is defining a range of starting cells the cell addresses within this range are calculated. For each starting cell, the unnested instructions and sheet values are used to determine the contents of the cells the turtle passes through. Volume was also returned when dynamics were defined within the turtle. This is now handled in the next step. This process models the movement of the turtle within the grid. Keeping track of where it is positioned and which was it is facing. For each instruction, the position and direction are updated as required and the contents of any new cells entered added to a list of notes.

Additional computation is required for the "m*" instruction, as the number of steps the turtle should take must be computed. Given the current position of the turtle and direction it is facing, the one-dimensional array of cells in front of it is taken. The turtle should step to the last cell that defines a note, sustain or explicitly defines a rest. The number of steps is the length of the array minus the index of the first element satisfying this criterion in the reversed array.

3.4.4 Creating Note Times

For each turtle, the cells moved through are calculated. This is used to create a data structure containing the information for playback to be initiated using the Tone library. For each turtle, the following array is produced: [[<Note 1>,...,<Note N>], <number of cells>] (note sequence array). Each note is as follows: [<start time>, [<pitch>, <duration>, <volume>]]. Dynamics and the Octave are also added to each note if they had been omitted from a cell.

The Tone library has many different representations of time. I opted to use Transport Time for all time measurements - start times and durations. This is in the form `'BARS:QUARTERS:SIXTEENTHS'` where the numbers can be non-integer.By QUARTERS representing the number of cells, exact times, ticks, or arbitrary musical notes need not be considered.

The note sequence array is initiated by counting the number of notes that are defined in the cell contents using the regular expressions for identifying notes and multi-note cells. The cells are iterated through keeping track of the active note and adding it to the note sequence when it ends. Outside of this loop, variables are defined to keep count of how many cells and notes through the process the algorithm is and whether the current value is a rest or note. Variables keep track of the note currently being played - when it started (`currentStart`), the pitch (`currentNote`) and volume (`currentVolume`). As volume and octave number may be omitted, variables are also required to keep track of these.

Table 3.2 outlines the actions carried out when a cell is read. When a note is added to the note sequence, it is added in the form `[currentStart, [currentNote, '0:' + noteLength + ':0', currentVolume]]`.

Table 3.2: The actions taken when processing each cell to create note times. The beat count corresponds to the cell number being processed and is incremented each time.

<table>
<tr><th>Cell</th><th>State</th><th colspan="3">Action</th></tr>
<tr><td rowspan="2">Note</td><td>Note</td><td rowspan="2">Note, octave and volume established from cell contents and previous values</td><td>Previous note added to note sequence</td><td rowspan="2">currentStart = '0:'+beatCount+':0'
currentNote = value
noteLength = 1
currentVolume = volume</td></tr>
<tr><td>Rest</td><td>inRest = false</td></tr>
<tr><td rowspan="2">Sustain</td><td>Row</td><td colspan="3">noteLength++</td></tr>
<tr><td>Rest</td><td colspan="3">Nothing (has no semantic value)</td></tr>
<tr><td rowspan="2">Rest</td><td>Note</td><td colspan="3">Previous note added to note sequence
inRest = true</td></tr>
<tr><td>Rest</td><td colspan="3">Nothing</td></tr>
</table>

The same method is used for multi-note cells, except the note length and cell count must be incremented by the appropriate fraction for each item in the cell. If at the end of the final cell, the state is in a note, this is ended and added to the note sequence.

The values in the note sequence are sufficient to play a note with the piano sampler using the `triggerAttackRelease` function. The `Tone.Part` class allows a set of calls to this method to be defined which can be started, stopped and looped as a single unit. Using the note sequence ("`noteTimes`"), number of cells ("`beatsLength`") from creating the note times, number of repeats ("`repeats`") and the evaluated speed argument ("`speedFactor`"), playback is scheduled with the following code (types have been omitted for brevity):

```
var turtlePart = new Tone.Part(function(time, note){
  piano.triggerAttackRelease(note[0], note[1], time, note[2]);
}, noteTimes).start();
if (repeats>0){
  turtlePart = turtlePart.stop("0:" + (repeats*beatsLength/speedFactor) + ":0");
}
turtlePart.loop = true;
turtlePart.loopEnd = "0:" + beatsLength + ":0";
turtlePart.playbackRate = speedFactor;
```

3.4.5 Chord Input

When the insert button is pressed, the note, type, inversion[5] and octave of the chord are extracted from their HTML elements. The tonal library can then be used to generate the notes of the scale:

```
var chordNotes = Chord.notes(chordNote, chordType).map(x => Note.simplify(x));
```

The tonal `simplify` function reduces note definition involving multiple accidentals to contain at most one, as required by Excello. This provides a list of notes in ascending order without octaves or taking into account the inversion. In order to reach the correct inversion of the chord, the array of notes is rotated by the inversion number.

Octave numbers are added by iterating through the notes. A dictionary matches note names to position in the chromatic scale starting at C (the first note of the octave in SPN). This accounts for enharmonic notes[6]. The given octave number is appended to the first note in the chord. For each preceding note, if it appears in an equal or lower position in the scale than its predecessor, the octave number is incremented before appending. Otherwise, it is in the same octave so the octave number is appended without modification.

The range selected by the user is acquired with the Office API. The notes of the chord are entered starting at the top-left corner of this range. If the height of the range is greater or equal to its width, the notes are entered vertically going down from the starting cell. Otherwise, they are entered horizontally going right. This is done by building the 2D array where the chord will be entered and setting that range using the Office API.

3.4.6 Custom Excel Functions

Custom functions are implemented using another add-in. As opposed to offering a separate window as the main Excello add-in does, this allows additional functions to be used in cells by using the prefix "=EXCELLO.". The file structure was generated with the Yeomann generator. The name, description, result type, and parameter names and

[5]which note of the chord is the lowest, the chord ascends from this.

[6]Notes that are the same pitch but different names, such as Ab and G#

types are stored in a JSON schema. This is used by Excel to provide argument prompts and autofill for the user when editing the formula. Functions are given an identifier to link them to a typescript file where they are defined.

The turtle function concatenates the arguments into the correct format for Excello to recognise as a turtle. This allows other cells to be referenced, for example, the speed variable can reference a global tempo variable as shown in figure 3.1.

For every note defined in a cell, if there is a volume defined, the note is separated, modulated using the tonal `Distance.transpose` function and then combined back with the volume. This allows the drag fill feature of Excel to be employed by the user for transposing sections or to define melodic lines using the interval between notes as shown in figure 3.2.

3.5 MIDI Converter

The following section documents how the Python converter from MIDI to CSV suitable for Excello playback works. A MIDI file is divided into up to 16 parallel tracks [2]. Each track contains a series of messages defined using predefined status and data bytes. I used the Mido library[7] to read MIDI files and abstract away from the underlying byte representations and view the messages. Note onsets and offsets are two separate events with two separate messages [2]. A note onset or offset message includes the note pitch and velocity, channel (not relevant) and time in ticks since the last message [4]. The times for messages defining information not relevant for the conversion (e.g. piano pedalling, meta messages) must still be taken into account.

First, the list of messages is converted to a list of notes defined by onset and offset time, pitch and velocity. For each track, the messages are iterated through, using the time value in every message to update a variable tracking time. If the message defines a note onset, this is added to a dictionary mapping pitches to a list of currently active note times. Lists are used because a pitch can be active multiple times at once. For note offset messages, or onset messages with zero velocity, the note popped from the active notes at that pitch, its end time added, and then it is added to the list of all notes defined in the file.

As each turtle can only define one note at a time, the notes are split into lists so no list contains two notes which are playing at the same time. Provided the main list of notes is non-empty, a new list is created. The first remaining note is moved to the new list. The next remaining note starting after the previous note ends is moved to this new list. All remaining notes are iterated over. The number of iterations required is the number of turtles required, n.

[7]https://mido.readthedocs.io/en/latest/index.html

If every tick corresponded to a cell, any combination of note onsets and offsets in a MIDI file could be accurately represented in Excello. To achieve smaller representations, the start and end times are converted to the cell number within the path of the turtle. For many MIDI files, the duration of a note is different from the time it occupies in notation. For example, a note immediately followed by another note in notation may have an end time significantly less than the start time of the next note in MIDI. A method is required to account for this. For all notes, before separation into the streams for different turtles, the length of the notes in ticks and differences between consecutive start times are found. The minimum value greater than 1 or modal value for these times are calculated depending on the compression level giving the $lengthStat$ and $differenceStat$.

$$ratio_{int} = \lfloor \max(lengthStat, differenceStat)/\min(lengthStat, differenceStat) \rfloor$$

For each note, the times are adjusted as follows:

$$length \leftarrow (start - end)/lengthStat \text{ (rounded to the nearest 0.1)}$$

$$start \leftarrow start/differenceStat \times ratio_{int} \text{ (rounded to the nearest 0.1)}$$

$$end \leftarrow start + length$$

Next, the streams, with note start and end times corresponding to cells, are converted to a CSV file to be run with Excello. The path for each turtle is initialised as an array of empty strings. The length of these arrays is the maximum end time for a note in any turtle, L. Each note the turtle plays is entered into the array. MIDI defines pitch using the integers. As there are 12 notes in an octave, taking the modulus and dividing by 12 gives the note name and octave for SPN. If the note velocity is different to the previous note played by the turtle (or the note is the first note played), the eight-bit velocity as defined by MIDI is mapped to the range [0,1] as used by Excello. If the note length is greater than one, sustains are placed in the following cells. These paths go right starting in column A, with the first in row 2.

Finally, the definition of the turtle must be placed in the spreadsheet. The start cell range is "A2:A($n + 1$)". The movement instruction is "r mL". The MIDI file contains meta data for the `tempo` (milliseconds per beat) and `ticks_per_beat`. Cells per minute is calculated as follows:

$$\textit{cells per tick} \times \textit{ticks per beat} \times \textit{beat per minute}$$

$$= \frac{ratio_{int}}{mode\ difference} \times \texttt{ticks\_per\_beat} \times \frac{60 \times 10^6}{\texttt{tempo}}$$

Using one as the number of repeats, the turtle definition is placed in cell A1 and the CSV exported.

3.6 Repository Overview

Figure 3.4 shows a reduced project file structure including all original source code. The directory Excel Music contains all the code for the add-in that parses the notation and produces music. Both this and CustomFunctions were generator using the Yeomann generator. The manifest.xml files are added to Excel and point to the resources to run the add-in. node_modules contains all libraries required to run the add-ins and are managed using npm.

The index.html file defines the window that appears on the right of the spreadsheet. assets contains the piano samples. index.ts defines what happens when the different buttons of the window are pressed and imports from the remaining Typescript files. turtle.ts contains all the code required to produce musical output from the spreadsheet of turtle definitions and notes in cells, with helper functions in regex.ts, conversions.ts and bracketsParse.ts. bracketsParse.ts was based on Parenthesis which was initially incompatible for Excello's needs. chords.ts is for inserting chord notes into the grid with the chord input tool.

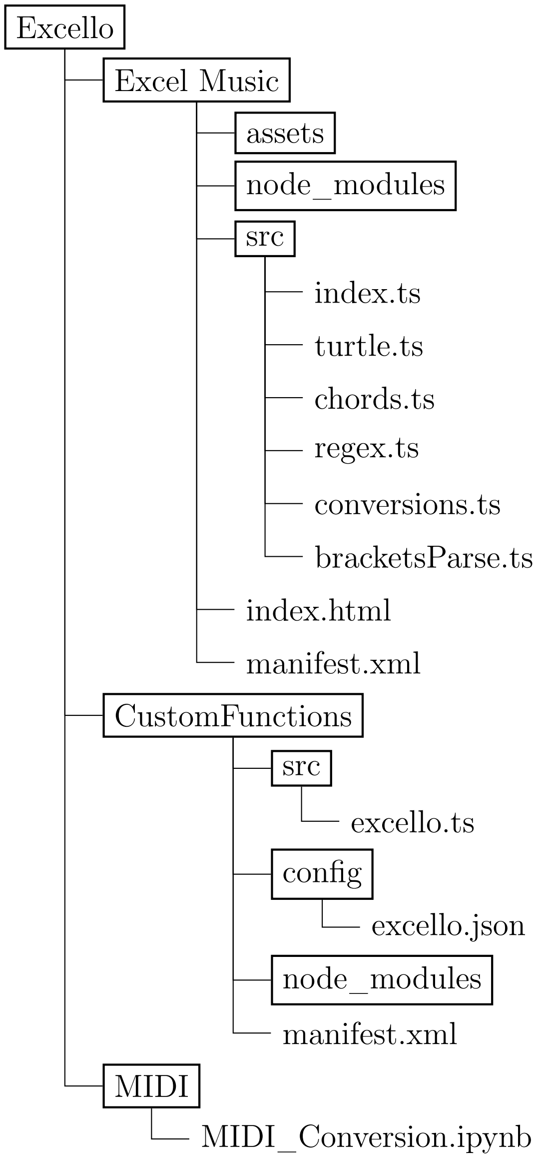


Figure 3.4: File structure overview showing original files

customFunctions.ts contains the implementation of the EXCELLO.TURTLE and EXCELLO.MODULATE functions. The index.html file created when generating this add-in is not seen by the user so was not re-written.

The Python notebook MIDI_Conversion.ipynb converts MIDI files to the Excello notation. This includes conversion of the MIDI corpora included in the MIDI directory.

I shall release Excel Music and CustomFunctions components as an open source project under the MIT license. This is compatible with the MIT licenses of Tone.js and tonal. The Salamander piano samples come under a creative commons license so credit shall be given in the Excel add-in window.

Chapter 4

Evaluation

This chapter discusses the Excello implementation effectiveness with regards to the success criteria and by showcasing examples. The conversion of MIDI corpora to the Excello notation using the converter demonstrates the expressiveness of the notation. Next, the summative evaluation is explained and its data used to assess the features implemented in the participatory design process and to reason about Excello using the CDN framework [10]. Finally, ethics and data handling procedures shall be covered.

4.1 Excello Success

Both a musical notation and corresponding program integrated into Excel for playback have been implemented. As required by the success criteria, users can play multiple notes and chords of different durations. These can be combined into looped sequences and have defined tempo. In the participatory design process, additional features were added as extensions; defining multiple successive notes in a cell, turtles calculating how far they should move and nested instructions with repeats are additional features facilitating more efficient notation. Custom Excel functions, a chord adding tool and faster turtle toggling allows users to work more efficiently. Figure 4.1 shows Excello in use with a participant's arrangement.

The first section of Reich's Piano Phase is two equal piano melodies, one played slightly faster than the other. The two parts move out of phase periodically aligning at different offsets. This is included as an example for many end-user programming tools. This is implemented in Manhattan using three rows of 24 columns [19]. Sonic Pi requires one line for the notes and eight for playback. Piano Phase cannot be concisely notated by western staff notation. Excello only requires two cells to define two turtles of different speeds in addition to the notes. All three implementations are shown in figure 4.2.

4.2 MIDI Corpus Conversion

Whilst being able to concisely notate music western notation and other end-user programming systems cannot, Excello can exactly express piece defined in MIDI. If tempo is

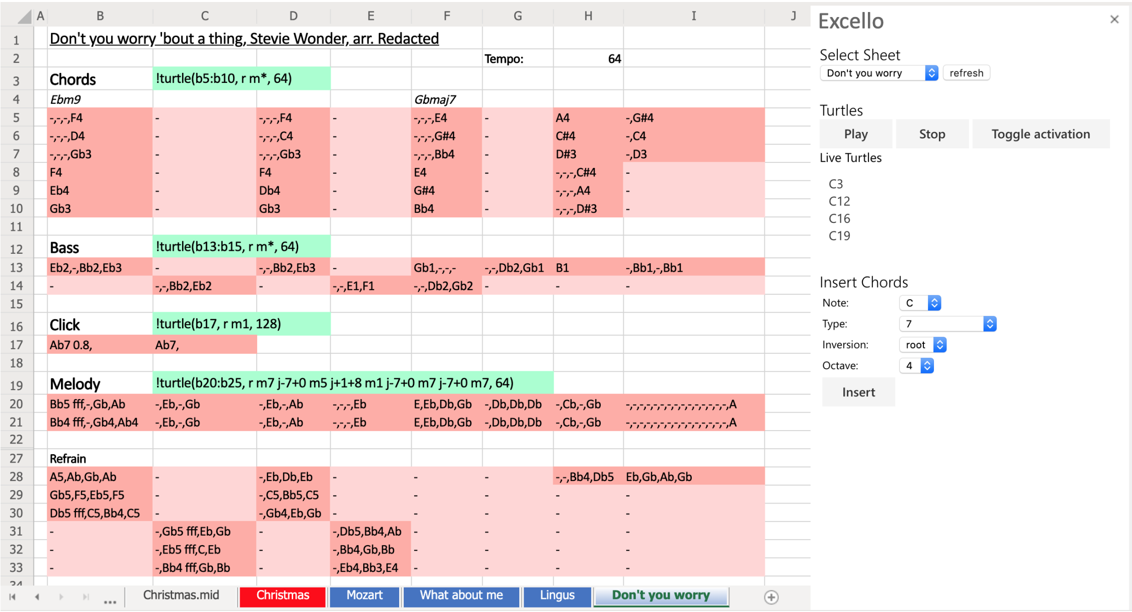
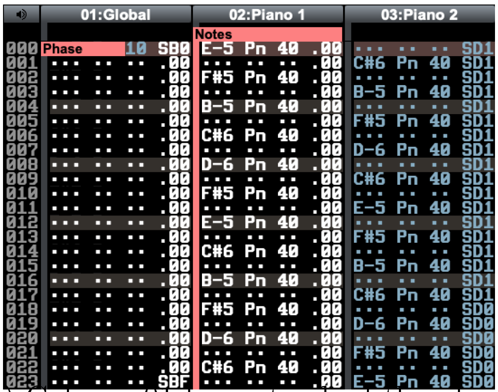


Figure 4.1: An arrangement with separated and labelled parts per instrument. Turtles refer to a global tempo at the top of the spreadsheet.

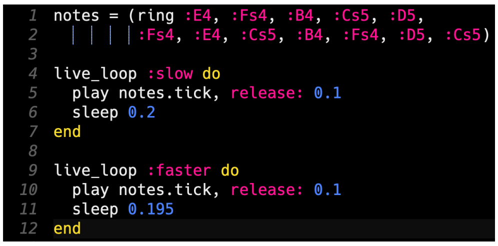
redefined within a track, this is not be accounted for. If instead the time between messages is adjusted, the uncompressed file will account for this but the compressing algorithms will produce erroneous results as the difference between notes deviate too far from non-integer multiples of the minimum. Instrument-specific effects such as piano pedals are not supported. Provided the difference between any two notes is a multiple of the minimum difference, the compression method that divides by this amount accurately reproduces the music, whilst resulting in spreadsheets orders of magnitude smaller. This method would not accurately convert quavers against triplets (three notes played in the same time as two) provided these notes were not multiples of a smaller note. Given the lengths of MIDI notes can be different from the space the note occupies in standard notation, an assumption on the ratio of note lengths was required for a more compressive conversion. The modal compressive conversion is lossy if the minimum note distance is not the modal distance. This is useful if there are ornaments or notes within a piece that dramatically decrease the minimum distance but occur infrequently. Therefore this loss may be useful for more efficient representations.

I have converted three MIDI corpora. A collection of 497 Bach chorales[1] made by Margaret Greentree, 280 piano pieces[2] held by Bernd Krueger under a creative commons license, and 194 Bach pieces made available from "A Johann Sebastian Bach Midi Page"[3]. This is not all the files available from this site as some were not readable by the python MIDI reader. All 971 MIDI files were converted using all three methods.

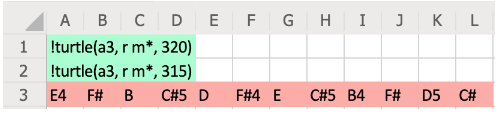
[1]Accessed from https://github.com/jamesrobertlloyd/infinite-bach/tree/master/data/chorales/midi
[2]http://piano-midi.de/midis.htm
[3]http://www.bachcentral.com/midiindex.html



(a) Column 01 keeps track of the phase, 02 defines the notes and 03 is the phased notes - defined with formulae that update depending on the phase and defined notes.



(b) The defined notes are played by two concurrent loops with different gaps between each note.



(c) Two turtles play the same notes at different speeds.

Figure 4.2: Implementations of Steve Reich's Piano Phase in a) Manhattan, b) Sonic Pi, c) Excello

The language of Excello is expressive enough to represent MIDI files and can do so concisely provided the condition of minimum note onset differences is maintained.

4.3 Summative Evaluation Sessions

19 Of the 21 users who participated in formative evaluation continued using Excello so could answer a summative evaluation questionnaire. First, the features that had been added since the initial sessions were recapped. To ensure users had a sufficient understanding of the interface before giving feedback, a short transcription task also requiring some authoring was given.

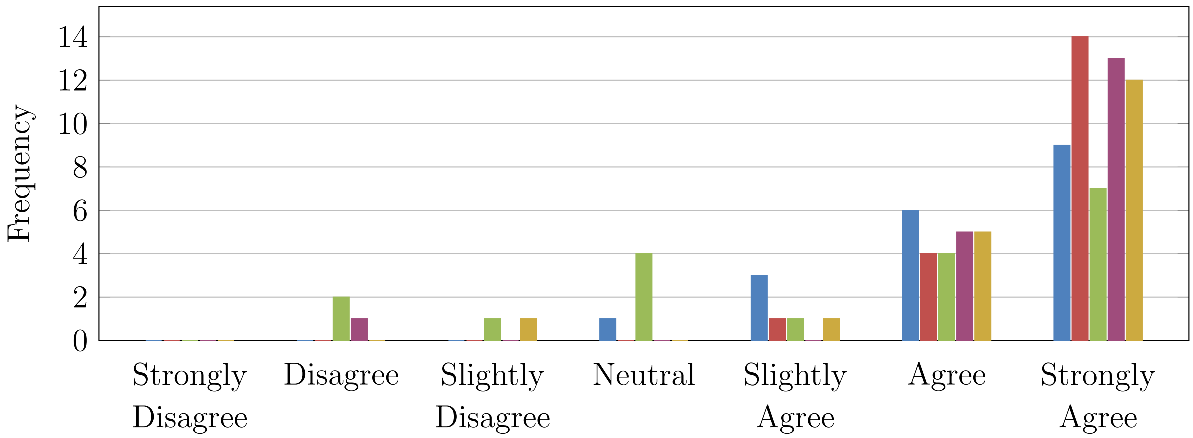
The questionnaire first evaluated the features added during the participatory design process by comparing the interface before and after a feature had been added. Questions tested if the issues had been solved and if overall the change rendered the system more preferable. These were answered using a seven-point Likert scale. The remaining questions were based on Blackwell and Green's CDN questionnaire [5]. CDN can be used to analyse musical notation [8] in addition to software systems [11], therefore it is suitable for the discussion of the Excello notation and interface. Dimensions significance for different activities varies [10], so users identified the percentage of time they spent carrying out these activities (searching for information, translating, incrementation, modification and exploratory design). Likert scale questions focusing on closeness of mapping, consistency, secondary notation, viscosity and visibility were used as planned in the proposal. It was suspected that reasoning about cognitive dimensions would be more challenging for participants, so to reduce the expected variance, only a five-point Likert scale was used. In addition, the two have been shown to produce similar results [7]. CDN results were also collected for the user's preferred music composition interface. 12 users chose Sibelius, which shall be used for comparison.

4.4 Success of Participatory Design

For each feature added, Excello with (system 2) and without this feature were compared. The following charts show the frequency of likert scale responses for each question. I considered the mode of the Likert scale [3]. Chi-squared goodness-of-fit tests confirm the distributions are significantly different to uniform. As all expected values must be greater than 1 and 80% greater than or equal to five [21] and the expected frequency for one result is $19/7 \approx 2.7$, I combine Strongly Disagree with Disagree, Strongly Agree with Agree and the remaining three options into a third group. The p-value from a chi-squared test with these three categories is given.

4.4.1 Dynamics in the Cell

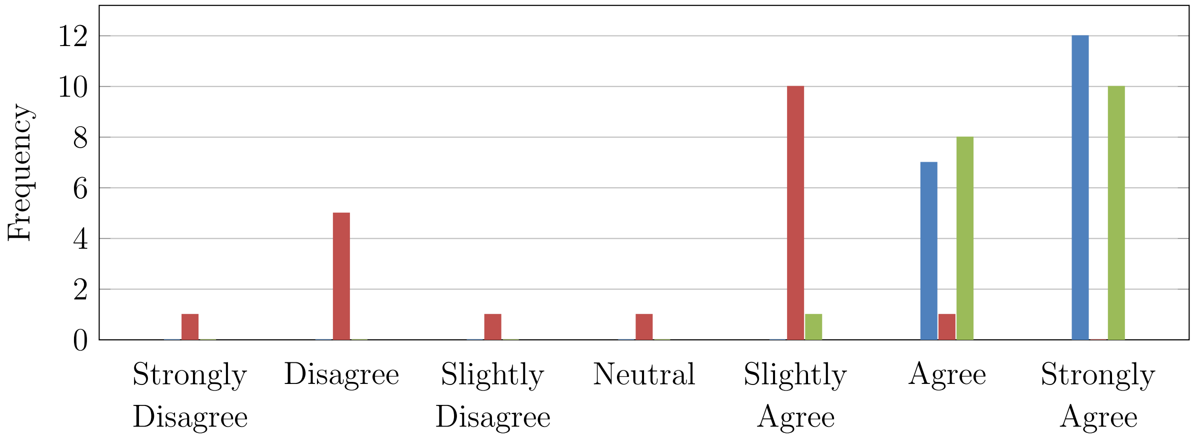
Statement	**Mode**	**p-value**
It is easier to figure out the turtles path.	Strongly Agree	0.0000
It is easier to figure out what dynamics different notes are played at.	Strongly Agree	0.0146
It is easier to tell the order in which dynamics are applied.	Strongly Agree	0.0000
It is easier to write dynamics in the correct place.	Strongly Agree	0.0000
Overall system 2 is preferable.	Strongly Agree	0.0000



There is strong evidence to suggest this change improved some of the issues identified during participatory design, resulting in an improved system.

4.4.2 Inferred Octave

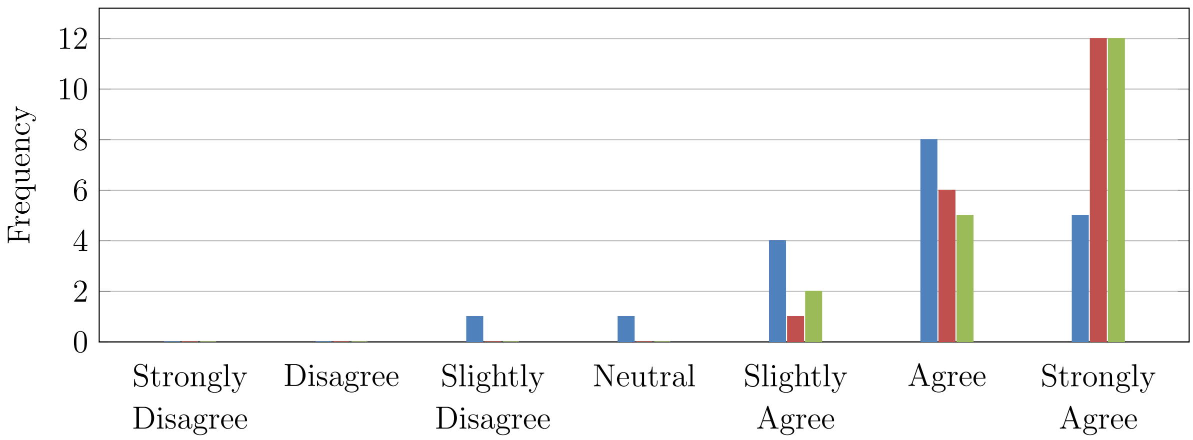
Statement	Mode	p-value
■ Less effort is required to write a part.	Strongly Agree	0.0000
■ It is harder to figure out what octave a note will be played in.	Slightly Agree	0.0639
■ Overall, system 2 is preferable.	Strongly Agree	0.0000



Depending on its use, the inferred octave notation makes octaves harder to infer. However, distribution of responses for this question is not significantly different to uniform. Overall this addition was significantly preferable.

4.4.3 Nested Instructions

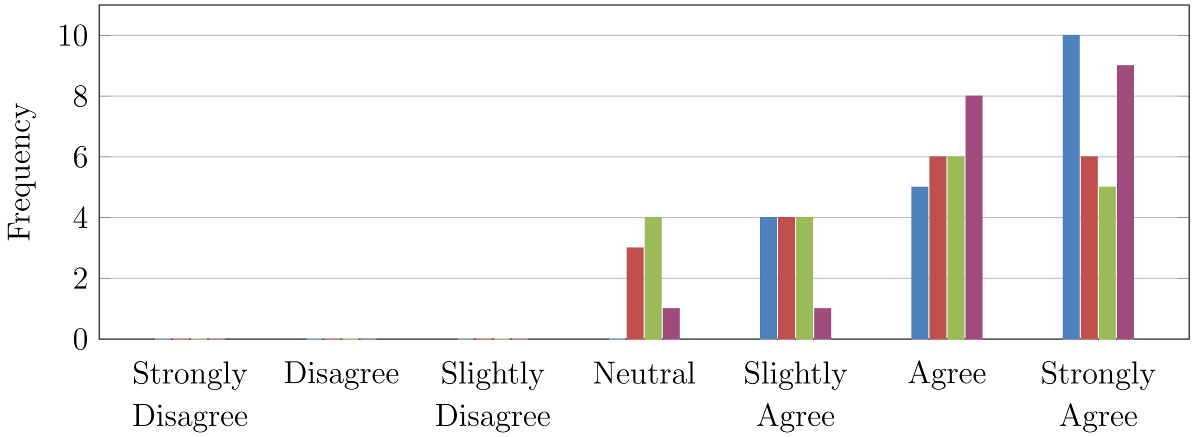
Statement	Mode	p-value
■ It is easier to parse the turtle instruction and tell what it will do.	Agree	0.0003
■ It is easier to repeat sections of notes.	Strongly Agree	0.0000
■ Overall, system 2 is preferable.	Strongly Agree	0.0000



All participants found the addition of nested instructions with repeats preferable, with the majority strongly agreeing.

4.4.4 Active Turtles List

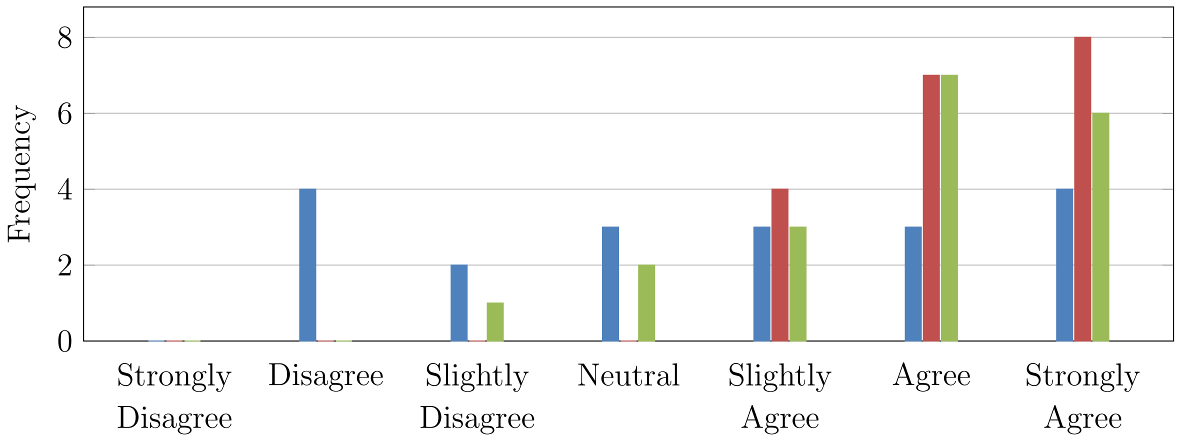
Statement	Mode	p-value
It is easier to tell if a certain turtle has been registered.	Strongly Agree	0.0000
It is easier to see where the active turtles are.	(Strongly) Agree	0.0011
It is easier to toggle the activation of turtles.	Agree	0.0038
Overall, system 2 is preferable.	Strongly Agree	0.0000



The addition of a list of active turtles was found by all users to have a neutral or positive effect on Excello.

4.4.5 Continuous Volume

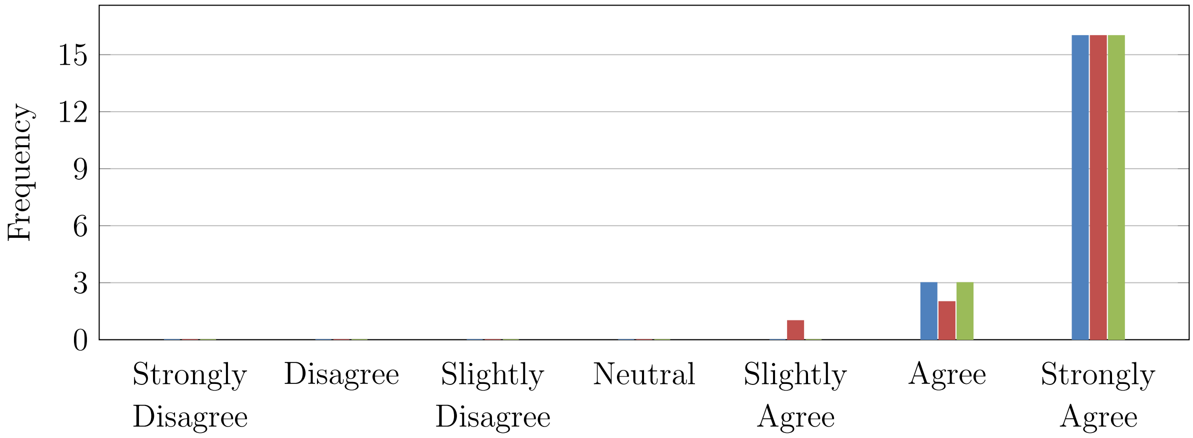
Statement	Mode	p-value
It is more intuitive how loud a note will be played.	Disagree Strongly Agree	0.6592
The volumes available are less limited.	Strongly Agree	0.0000
Overall, system 2 is preferable.	Agree	0.0003



There is no significant result for whether the ability to define volume in the range [0,1] is more intuitive. All users agreed that the volumes were less confined. However, only one user did not find this change preferable. Given that, it can be omitted and the previous conventional dynamic markings used, this supports the addition being successful.

4.4.6 Automatic Stepping

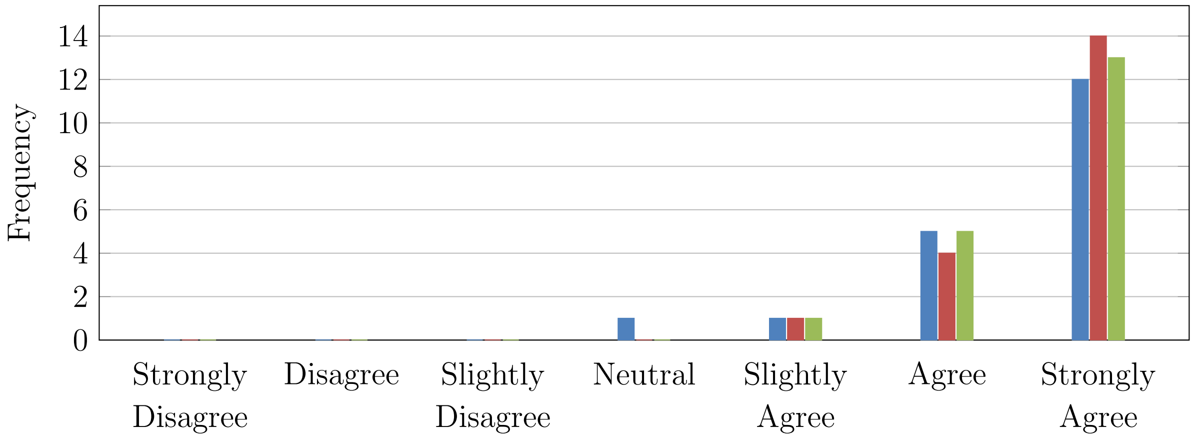
Statement	Mode	p-value
Less mental work is required to write the turtle. instructions.	Strongly Agree	0.0000
Less work is required when more notes wish to be added.	Strongly Agree	0.0000
Overall, system 2 is preferable.	Strongly Agree	0.0000



The addition of this feature was particularly successful with 16 of the 19 users strongly agreeing the system was more preferable with automatic stepping available in the turtle instructions.

4.4.7 Absolute Tempo

Statement	Mode	p-value
It is easier to tell what the speed instruction corresponds to.	Strongly Agree	0.0000
Giving an exact tempo (e.g. when transcribing sheet music) is easier.	Strongly Agree	0.0000
Overall, system 2 is preferable.	Strongly Agree	0.0000



The initial design was changed so that turtle speed was defined in absolute cells per minute. All users found this to be an improvement.

Overall, the participatory design process was very successful. Multiple features were added to Excello to solve problems identified through formative evaluation sessions and longer-term feedback from users. There is significant evidence to suggest all added features were found to improve Excello.

4.5 Cognitive Dimensions of Notation

Figure 4.3 shows the time users identified carrying out the different cognitive activities [10] in Excello and in Sibelius. There are 19 users for Excello and 12 for Sibelius. This shows Translation is a very important activity for both interfaces. There is more exploratory design for Excello but as users become more familiar with the system and the amount of existing Excello notation increases, modification and incrementation may become more important. Little time is spent searching for information in the notation in either.

A series of statements from [5] were selected to assess the CDN of Excello. Users responded with a five-point (Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree) Likert scale. The significance of the results was verified with a chi-squared test. First, the data was combined into negative and non-negative categories. The dimension being assessed, p-value from the chi-squared test and modal response for each statement are shown in table 4.1. The distribution of responses is shown in figure 4.5.

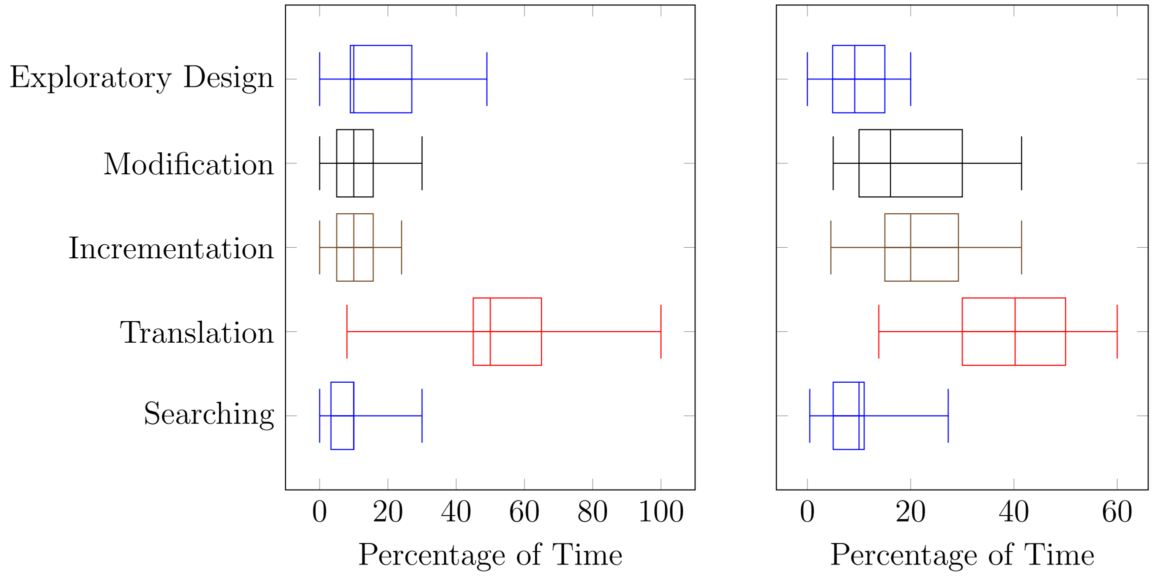
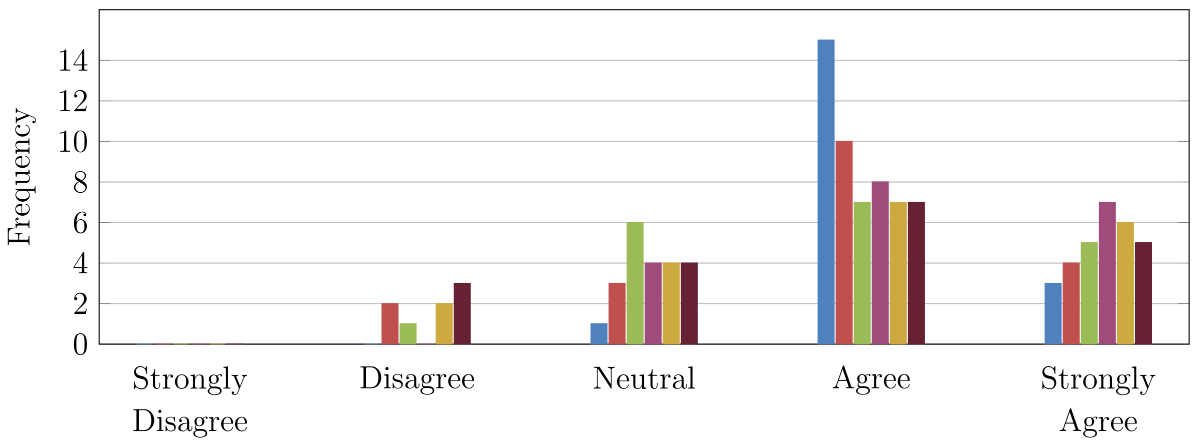


Figure 4.3: The percentage of time spent performing the different cognitive activities in Excello (left) and Sibelius (right).

Statement	**CDN**	**Mode**	**p-value**
(a) The notation used (In Excello: notes/ dynamics in cells and the definition of turtles) is related to the result you are describing (In Excello: Musical output)	Closeness of Mapping	Agree	0.0004
(b) Where there are different parts of the notation that mean similar things, the similarity is clear from the way they appear.	Consistency	Agree	0.0087
(c) You can add extra marks (or colours or format choices) to clarify, emphasise or repeat what is there already.	Secondary Notation	Agree	0.0020
(d) When you need to make changes to previous, work it is easy to make the change.	Viscosity	Agree	0.0004
(e) It is easy to see or find the various parts of the notation while it is being created or changed.	Visibility/ Juxtaposition	Agree	0.0087
(f) If you need to compare or combine different parts, you can see them at the same time.	Visibility/ Juxtaposition	Agree	0.0312

Table 4.1: Questions and results for testing the CDN of Excello



As these questions were also answered for the user's interface of choice, a comparison to Sibelius is made. As the data does not meet the assumptions of the t-test [3], I performed a Wilcoxon matched pairs signed-ranked test on the 12 pairs by encoding the five possible responses as -2,-1,0,1,2. For all six questions, there is no indication that the answers for the two interfaces come from populations with different means.

4.5.1 Closeness of Mapping

As there is no significant evidence that the population means for Excello and Sibelius were different, this suggests Excello's notation with spreadsheets has not compromised that closeness of mapping of traditional notation. This is helped by using an existing notation (SPN) for defining the individual notes, the turtle instructions mapping to a movement through the grid, and by adjusting the speed argument to be an absolute, not relative, parameter. Being less familiar with staff notation, user 4 found Sibelius's notation unintuitive.

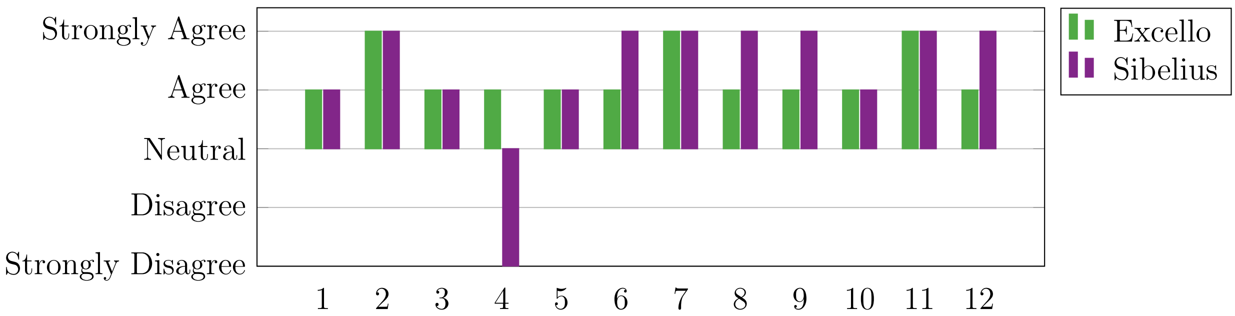


Figure 4.4: User responses for *closeness of mapping* for Excello and Sibelius from (a)

4.5.2 Consistency

As each cell and turtle only causes one note to play at a time, consistency is maintained by building pieces from these elements. Excello keeps consistency with Excel by sharing notations (e.g. A1:A5 for ranges) and using the existing formula editor. Within the turtle instructions, using a number after instructions to repeat holds for both individual instructions and sequences. This all contributes to no significant result from the Wilcoxon test.

4.5.3 Secondary Notation

Given the translation time, secondary notation is particularly important [8]. Given that Excello abstracts time from the axes of the grid the distribution of parts is up to the user and cells can be used for arbitrary marks. Therefore, existing Excel features for formatting and grouping cells remain available. This is utilised by the automatic highlighting of notes and turtles. That there is no significant difference in population means, this suggests that the spreadsheet paradigm can provide equal secondary notation abilities to Sibelius, software already equipped with numerous ways to customise a score.

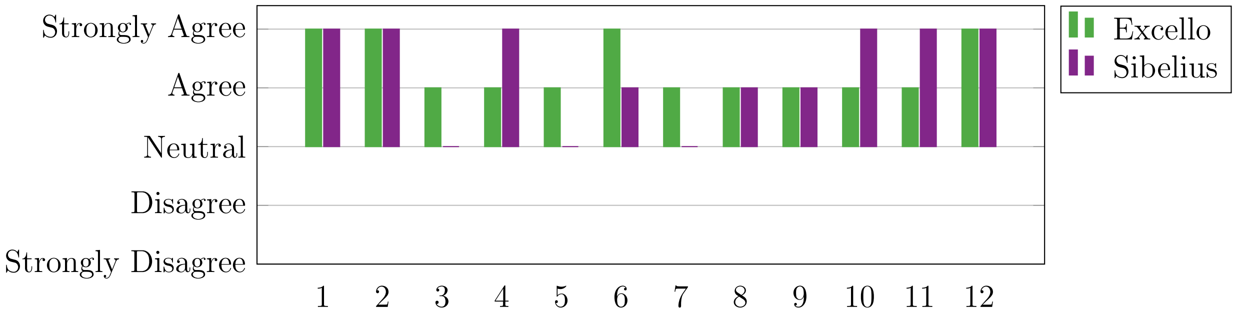


Figure 4.5: User responses for *consistency* for Excello and Sibelius from (b)

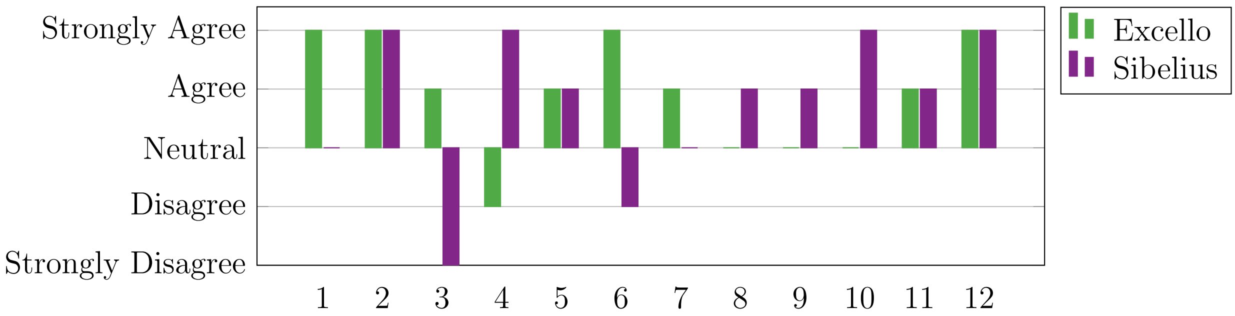


Figure 4.6: User responses for *secondary notation* for Excello and Sibelius from (c)

4.5.4 Viscosity

By allowing dynamics and octave marking to be omitted and the ability for turtles to step forward automatically, there is low resistance to making additions and changes to the music. The toggling of turtle activations dramatically reduces the actions required to turn turtles on and off. Furthermore, Excel allows for the easy editing and movement of cells. No significant result in the Wilcoxon test suggests the interfaces have comparable viscosity.

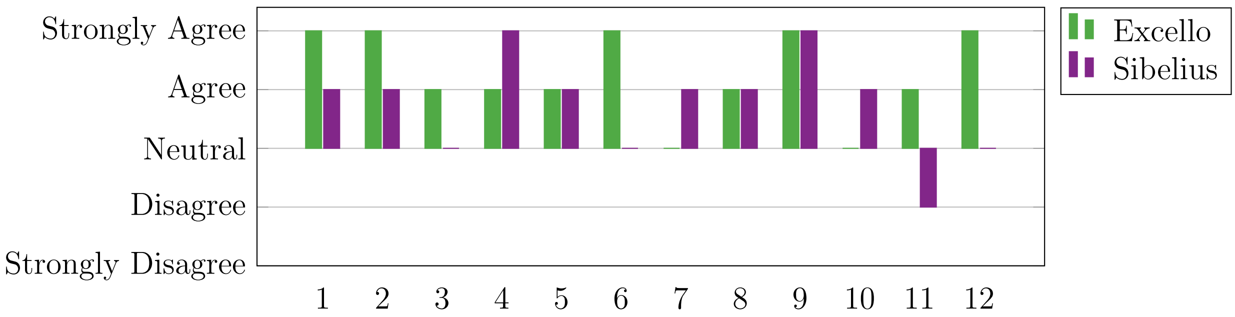


Figure 4.7: User responses for *viscosity* for Excello and Sibelius from (d)

4.5.5 Visibility / Juxtaposition

For both questions, there was no significant difference in population mean. This suggests that the spreadsheet interface can provide a similar ability to view components than Sibelius.

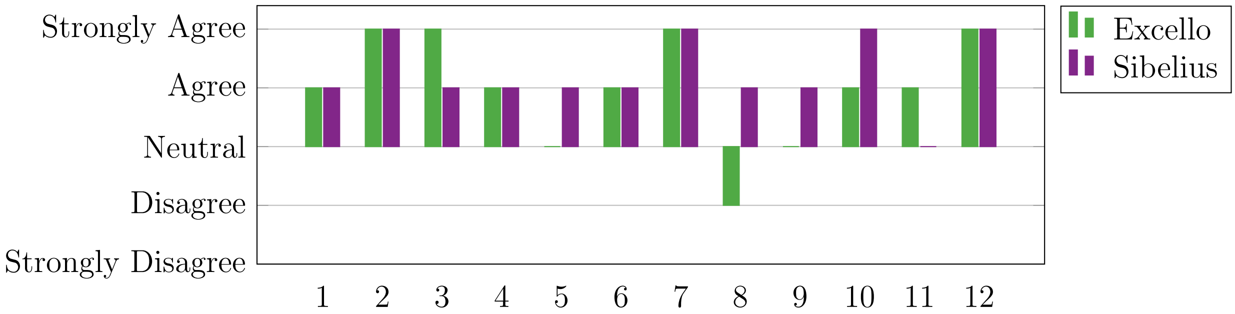


Figure 4.8: User responses for *visibility/juxtaposition* for Excello and Sibelius from (e)

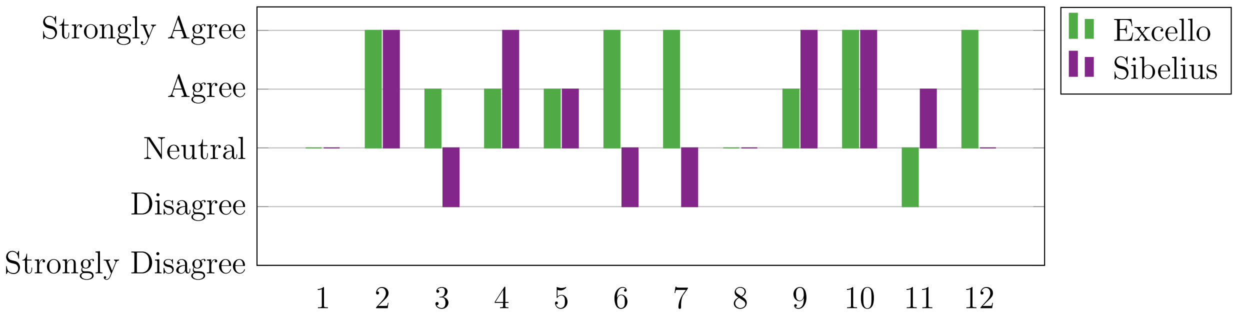


Figure 4.9: User responses for *visibility/juxtaposition* for Excello and Sibelius from (f)

Sibelius uses established music notation as part of large professional software. However, there was no significant evidence to suggest Sibelius outperformed Excello across these CDN. This suggests that despite being built for a general purpose spreadsheet environment, Excello is a successful interface for writing music.

4.5.6 Other Dimensions

If users are unfamiliar with the turtle paradigm, this may reduce the *role-expressiveness*. Turtles and notes are the only musical spreadsheet components and are identified by highlighting. Whilst notes and turtles can be added in any order, adding an additional part may require many line insertions which would increase the *premature commitment*. The dual-formalism of the turtles and notes could create high *diffuseness* but the user's freedom to lay these out allows this to be minimized as in figure 4.1. This also shows how representations can have strong *synopsie*, as the notes or turtles don't need to be examined to understand what is happening. This may come at the expense of *hidden dependencies* if it is not immediately clear which notes are triggered by which turtles. Volume is also dependent on the notes turtles play before it. But as a single cell could be played at multiple volumes, this is a tradeoff of this design decision.

As well as the "`m*`" notation decreasing *viscosity*, it also improves the *progressive evaluation* as turtles can be played before a whole part has been transcribed. The highlighting of cells also helps users receive more feedback. The ability to define a turtle and fill in the notes later also improves the *provisionality*. "`m*`" also reduces *hard mental operations* and the chord input tool removes manual calculation of the notes of chords.

Spreadsheets are "an abstraction-hating system" [10], therefore little *abstraction* is provided by Excel, but the grouping of turtles in one definition and nested bracketing in turtle movement instructions improve this. These features also provide good *legibility*.

4.6 Ethics and Data Handling

After ethics approval, the pilot session for the formative evaluation session was designed. After the pilot session (also performed for summative evaluation), the session was revised before continuing with the remaining sessions. Participants were provided with a consent form explaining the project and the format of the session. Participants had the choice to remain anonymous so they would not appear in acknowledgements. All participants' data was labelled with a unique ID for participants to be able to use to request removal or anonymising of their data. All participant data was only seen by myself. Formative evaluation sessions were audio recorded, typed up after the session, and then deleted. All participant data was also backed up on GitHub with the rest of the project but in an encrypted folder. All physical backups were also encrypted.

Chapter 5

Conclusion

The project set out to explore the hypothesis that spreadsheets would provide a productive medium for musical expression. Excello is a notation and corresponding program for musical playback integrated within Microsoft's Excel. By abstracting time away from the axes of the grid, the existing functionality of Excel remains highly useful. Having satisfied the initial success criteria for the program, development continued, carrying out participatory design with 21 users. As a result of this, many additional features, beyond the initial scope of the project, were implemented all of which have been shown to significantly improve the interface. With respect to CDN, reasonable closeness of mapping, high consistency, high secondary notation, low viscosity and high visibility were all achieved as desired. Quantitatively, Excello is able to express any MIDI music, and a converter was built to translate existing corpora of MIDI files to CSV files for Excello. The converter included two additional compression mechanisms, which still represent all musical information under certain, but common, conditions.

During development, I submitted part of my code as an improvement to the open-source library Parenthesis. This was merged and has been published. The package has over 20,000 weekly npm downloads.

Excello freely provides a simple, but powerful program for musical composition to the hundreds of millions of users already familiar with the spreadsheet interface.

Bibliography

[1] Samuel Aaron, Alan F. Blackwell, and Pamela Burnard. The development of sonic pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming. *Journal of Music, Technology and Education*, 9:75–94, 05 2016.

[2] MIDI Manufacturers Association. *The complete MIDI 1.0 detailed specification*, 1997.

[3] Dwight Barry. Do not use averages with likert scale data, 01 2017.

[4] Ole Martin Bjrndalen. Midi files. `https://mido.readthedocs.io/en/latest/midi_files.html`, 2016. Accessed: 2019-04-18).

[5] Alan F. Blackwell and Thomas R. G. Green. A cognitive dimensions questionnaire optimised for users. In *PPIG*, 2000.

[6] B Boehm. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4):14–24, August 1986.

[7] Dr. John Dawes. Do data characteristics change according to the number of scale points used? an experiment using 5-point, 7-point and 10-point scales. *International Journal of Market Research*, 50(1):61–104, 2008.

[8] Alan F. Blackwell, Thomas Green, and Dje Nunn. Cognitive dimensions and musical notation systems. *Workshop on Notation and Music Information Retrieval*, 11 2000.

[9] Ron Goldman, Scott Schaefer, and Tao Ju. Turtle geometry in computer graphics and computer-aided design. *Computer-Aided Design*, 36:1471–1482, 2004.

[10] Thomas Green and Alan Blackwell. Cognitive dimensions of information artefacts: a tutorial. Technical Report Version 1.2, BCS HCI Conference, 1998.

[11] Thomas Green and Marian Petre. Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages*, 7:131–, 06 1996.

[12] Sven Gregori. Never mind the sheet music, heres spreadsheet music, 2019.

[13] Allen Huang and Raymond Wu. Deep learning for music. *CoRR*, abs/1606.04930, 2016.

[14] D.M. Huber. *The MIDI Manual: A Practical Guide to MIDI in the Project Studio*. Taylor & Francis, 2012.

[15] Alex Mclean, Dave Griffiths, Foam Vzw, Dave@fo Am, Nick Collins, and Geraint Wiggins. Visualisation of live code. 01 2010.

[16] Alex Mclean and Geraint Wiggins. Texture: Visual notation for live coding of pattern. 01 2011.

[17] Mozilla. Web audio api. `https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API`, 03 2019. Accessed: 2019-04-02).

[18] Michael Muller and Sarah Kuhn. Participatory design. *Communications of the ACM*, 36:24–28, 06 1993.

[19] Chris Nash. Manhattan: End-user programming for music. In *NIME*, 2014.

[20] Simon Peyton Jones, Margaret Burnett, and Alan Blackwell. A user-centred approach to functions in excel. June 2003.

[21] S.M. Ross. *Introductory Statistics*. Elsevier Science, 2010.

[22] Erik Sandberg, Examensarbete Nv, Reviewer Arne Andersson, and Examiner Anders Jansson. Separating input language and formatter in gnu lilypond, 2006.

[23] Advait Sarkar. Towards spreadsheet tools for end-user music programming. In *PPIG*, 2016.

[24] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users and end user programmers. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 207–214, Sep. 2005.

Appendix A

Excello Implementation

A.1 bracketsParse.ts

```
// inspiration taken from:
// https://github.com/dy/parenthesis/blob/master/index.js

/**
 * Given a turtle instruction sequence this unwraps any brackets to create exact instrutions
 * @param str Turtle movement instructions e.g. "(r m3)4"
 * @return explicit unwrapped instructions e.g. "r m3 r m3 r m3 r m3"
 */
export function parseBrackets(str: string) {

var unnestedStr = ['will become highest level'];
var idPadding = '__';

var deepestLevelBracketsRE = new RegExp('\\([^\\(\\)]*\\)'); // finds bracket with no brackets inside

// store contents of bracket it unnestedStr and replace contents in str with ID
while (deepestLevelBracketsRE.test(str)) {
str = str.replace(deepestLevelBracketsRE,function(x) {
unnestedStr.push(x.substring(1, x.length-1)); // add the token without the brackets
return idPadding + (unnestedStr.length - 1) + idPadding;
});
}
unnestedStr[0] = str; // make first element in array the highest level of the string

var replacementIDRE = new RegExp('\\' + idPadding + '([0-9]+)' + idPadding);

// transform references to tree
function reNest (outestStr: string) {
var renestingStr = [];
var match;

while (match = replacementIDRE.exec(outestStr)) {

var matchIndex = match.index;
var firstMatchID = match[1];
var fullStringMatched = match[0];

// push what was before
if (matchIndex > 0) {
renestingStr.push(outestStr.substring(0, matchIndex))
}
//perform recursively
renestingStr.push(reNest(unnestedStr[firstMatchID]))
// remove the string that has been processed
outestStr = outestStr.substring(matchIndex + fullStringMatched.length)
```

```
}
renestingStr.push(outestStr)
return renestingStr
}

return reNest(unnestedStr[0])
}

export function processParsedBrackets(arr) {
var s = "";
var wasPrevArray = false;
var prevArray = "";
for (let val of arr) {
if (val.constructor === Array) {
prevArray = processParsedBrackets(val)
wasPrevArray = true;
}
else {
var singleInstructions = val.trim().split(" ");
if (wasPrevArray) {
s = s + prevArray;
if (!isNaN(singleInstructions[0])) {
for (var i=1; i<singleInstructions[0]; i++) {
s = s + prevArray;
}
singleInstructions = singleInstructions.slice(1);
}
}
for (let instruction of singleInstructions) {
s = s + instruction + " ";
}
wasPrevArray = false;
}
}
if (wasPrevArray) {
s = s + prevArray;
}
return s;
}
```

Appendix B

Project Proposal

Henry Mattinson
Christ's
HCM50

Computer Science Part II Project Proposal

Music Generation in Microsoft Excel

16/10/2018

Introduction

Excel and other spreadsheet tools have become universally popular, both in businesses and individually, for storing, processing and visualising data. However, at present there is not the functionality for the playback of music. Many existing music production packages utilise a grid like format with time passing along the x-axis and parts down the y-axis. Therefore, spreadsheets seem like a promising environment from which to be able to carry out basic music composition in this format and others.

Many people are already familiar with representing concepts in spreadsheet form. This project will explore the use of Excel for musical expression and, as an extension, as a live music coding environment.

Starting Point

No existing work or further knowledge than part Ia and Ib courses. I am a seasoned musician and musical theory enthusiast so possess all the required musical theory knowledge.

I will be building on top of existing spreadsheet service. I would aim to use the Microsoft Office API Office.JS (a public API) and use Add-ins for adding my own functionality, if there is sufficient support for sound. This is publicly available. If not, I would either be able to use a Typescript prototype of Excel from Microsoft or an existing open source JavaScript implementation of a spreadsheet.

Substance and Structure

By using a TypeScript or JavaScript version of Excel run in browser, playback functionality can be built on top of the web audio API. Functionality for note and sequence synthesis functions will be required. A converter from an existing formal music specification to the spreadsheet representation will be implemented. As an extension, live coding can be implemented.

Firstly I will have to establish what notation is used within the cells. Within a given cell, I would like to be able to play single notes and chords. Beyond this defining scales and arpeggios would be useful for reducing the size of grid required to define pieces. This notation must then be interpreted with a resulting call to the browser audio API. It would also be desirable to be able to define sequences of notes (e.g. baselines, repeating melodies) elsewhere in the grid and then be able to call these elsewhere in the playback. This means that users do not have to copy and paste repeating sections and it would also be clearer where sections are repeated.

Next, the representation that is supported between cells must be decided and implemented. The flexibility of spreadsheets allows users to define their own secondary notation in the way that sections within the grid are laid out. As a result, allowing for the relative positions of different sections within a sheet and their orientation to vary would allow familiar Excel users to continue defining their own layout. The definition and re-use of phrases and parts would allow for fast prototyping of musical ideas. The representation will likely be that of a main playback loop (which can be split into multiple parts), with definitions of sequences outside of this main loop section.

After establishing my notation and supported layouts, the program must compile this representation into audio output for playback. Firstly, defined variables (e.g. Tempo) and regions where melodic lines are defined out of the main playback loop must be detected. Next, the main playback loop region and its orientation must be detected. After this, the information can be processed and converted into calls to the audio API.

As an extension, I can add support for live music coding. To facilitate live music coding, it should be possible to change notes within the grid and recompile whilst playback is occurring. Live music coding encourages a loop based approach to music so run/compiled changes to the grid should become apparent in the playback whilst not requiring a restart of the output. The program would be able to parse the data within the spreadsheet and identify different regions and declarations. From this it would convert the main playback loop with the output being calls to the growers audio API. This would include integrating sequences that are defined out of the main loop but called with in.

Once the representation of musical structure has been decided and the playback of this representation implemented, I will implement a conversion from some form of formal music notation (e.g. MusicXML or MIDI) to the spreadsheet representation. Existing pieces can then be immediately transformed into the spreadsheet layout and played back using the spreadsheet music API.

As an extension I could explore reducing the size of the representation within the spreadsheet. For example, a repeated chord sequence could only be shown once in the spreadsheet whilst keeping an understandable representation. Whilst this is not conventional compression, similar lossless or lossy algorithms for eliminating statistical redundancy can be employed.

The project has the following main sections:

1. Facilitating audio playback from a spreadsheet, run from the browser.
2. Execution and playback of musical definition code in the grid cells.
3. Playback of multiple cells where time is represented in an axis or the code within cells.
4. Implementation of a converter from a formal music notation to the spreadsheet representation.
5. Evaluation and the preparation of examples to demonstrate the success of the implementation.
6. User testing.
7. Writing the dissertation.

Evaluation and Success Criteria

A successful implementation should allow a user to carry out the following:

- Play individual notes and chords and define their durations.
- Defining multiple parts.
- Play loops.
- Define sequences of notes and chords and be able to call these for playback.
- Define the tempo of playback.

Qualitatively, use of the music playback API can be analysed using a friction analysis approach as in [3] and a cognitive dimensions profile strategy.

With some basic explanation, users can be measured carrying out simple tasks or free composition. From this we can measure Time To Hello World (TTHW) (e.g. playing a note). Friction diagrams generated based on observation of a user working with the program in a usability study can be used to evaluate the productivity of users of the tool.

We define the following desirable features in a cognitive dimensions profile. This defines the desirable structural usability properties of the API and interaction UI.

- Reasonable **closeness of mapping** (use of the grid structure should allow for much higher closeness of mapping than e.g. Sonic Pi where there is only one file of code).
- High **consistency** for the definition of notes and chords within phrases.
- Layout within the grid should allow for high **secondary notation**
- Low **viscosity**
- High **visibility**

We shall then use a Cognitive Dimensions questionnaire to empirically categorise users' response to it. Evaluation can be carried out by comparing that response to this desired profile.

Quantitatively, the expressiveness of the API can be verified by a translation of a musical corpus from the formal notation to the spreadsheet representation.

The compression rates achieved in the compression of the representation can also be measured and compared to a benchmark of a naive conversion.

Success criteria

For the project to be deemed a success the following must be completed:

- Implementation of an API for music playback within a spreadsheet using the above implementation features.
- Implementation of a converter from formal music notation to the spreadsheet representation.
- Usability testing for music generation implementation.

Plan of work

Below I outline the plan for successful completion of a successful project. I have outlined above various extensions, some of which I hope to be able to also implement. I am aiming to finish coding in good time to allow for user testing, evaluation and the dissertation writeup to be completed in time for me to carry out ample revision before my finals.

Before Proposal Submission: - 19/10/18

Submit the final project proposal before Friday 19th October, 12:00.

Section 1: 19/10/18 - 9/11/18

3 weeks
Weeks 3-5 of Michaelmas term

Gain familiarity with the system which I will be building on. Work on facilitating basic music playback so that basic notes can be played from cells within the Excel grid.

This time can also be used to consider the layouts of musical representation that will be supported by the API.

<u>Milestone: Ability to create sound from within Excel grid</u>

Section 2: 9/11/18 - 30/11/18

3 weeks
Weeks 6-8 of Michaelmas term

Begin implementation of spreadsheet API for music generation and implement tempo/tick so that timing can be specified. Implement playing of arbitrary notes at arbitrary times. At this point sequences can be defined and played back.

<u>Milestone: Ability to play through arbitrary notes at arbitrary timings</u>

Section 3: 10/12/18 - 24/1/19

2 weeks
Out of Cambridge

Make it possible to define note/chord sequences outside of the main playback loop and have this integrated into playback. The sections where these are defined must be found within the spreadsheet and their definitions matches to names in the main playback section.

Increase API for music performance so that chords, scales and precision can be specified.

<u>Milestone: Completion of spreadsheet API for music generation (not live coding)</u>

Section 4: 24/12/18 - 4/1/19

1.5 weeks
Out of Cambridge

History would suggest that the presence of Christmas and the end of the year will require a reasonable amount of my attention. My family will most likely appreciate this period being a little less demanding.

This period can be used to neaten the existing codebase. It may be useful to reimplement certain functions to help with the following stages for implementing live coding and conversion. This time can also be used to research and consider the method for implementing the conversion and live coding. At this point I should be familiar with the audio API and have more sensible ideas for doing this.

This would also be a good time to write a first draft of the introduction section to ensure adequate time can be given to the implementation and evaluation sections at the end of the project.

<u>Milestone: Introduction section draft</u>

Section 5: 4/1/19 - 17/1/19

2 weeks
In Cambridge before term starts including Lent term week 0

Build converter from formal music format to the spreadsheet representation. Demonstrate success with the conversion of a corpus to the spreadsheet format.

<u>Milestone: Conversion of formal music format to spreadsheet representation</u>

Section 6: 17/1/19 - 7/2/19

2 weeks
Weeks 1-3 of Lent Term

Prepare Presentation

This time can be used to catch up if any of the previous milestones have not been adequately reached. Then this time can be used to work on extension tasks.

<u>Milestone: Submission of Project Report and Presentation</u>

Progress Report Deadline: Fri 1 Feb 2019 (12 noon)
Progress Report Presentations: Thu 7, Fri 8, Mon 11 or Tue 12 Feb 2019 (2:00 pm)

Section 7: 7/2/19 - 28/2/19

2 weeks
Weeks 4-5 of Lent Term

Prepare examples and methods for evaluation. For human evaluation, the interface and tasks to perform must be planned and prepared.

<u>Milestone: Prepare examples and methods for evaluation</u>

Section 8: 28/2/19 - 14/3/19

3 weeks
Weeks 6-8 of Lent term

Perform write up of results of user testing and analysis. This time can be used to perform small changes for potential improvements that may arise during testing and evaluation.

Milestone: Complete coding and evaluation for dissertation write up

Section 9: 14/3/19 - 18/4/19

5 weeks
Easter vacation

Full time Dissertation write up. As marks are awarded on the final dissertation I would like to be able to allocate a lot of dedicated time for writing this up. I would also like to be almost complete by the time I return to university for Easter term as I would like to spend this time onwards mostly on revision. By submitting a draft before the end of the vacation I will be able to go over it with by supervisor when I return to Cambridge and have time to go over any changes.

Milestone: Submit Dissertation First Draft

Section 10: 18/4/19 - 9/5/19

3 weeks
Start of Easter term

I will have returned to Cambridge by this time and hope to be spending the majority of my time revising for my final exams. This, however, allows time between a draft submission and final deadline to make any final changes.

Milestone: Submit Final Dissertation

Dissertation Deadline (electronic): Fri 17 May 2019 (12 noon) Source Code Deadline (electronic copies): Fri 17 May 2019 (5:00 pm)

Resources Required

Development Machine I shall use my personal laptop for most development work for this project. It is an *Apple MacBook Pro* 13" (2015), 2.9 GHz *Intel* i5 CPU with 16GB RAM. I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure. I can use MCS machines to do any lighter, more portable work. These shall certainly be used if my machine become unavailable.

Software Access to a suitable spreadsheet tool will be required. This will depend on the audio capabilities of the implementations outlined above. If OfficeJS if unsuccessful, this will be facilitated by my supervisor (Advait Sarkar, advait@microsoft.com) who works at Microsoft Research. *Git* will be employed for version control of both implementation source code and documentation. The Dissertation shall be written in *LaTeX*.

Backups I shall use *Github* to remotely back up source code and documentation. These can then be pulled to an MCS machine in the case of personal machine failure. I shall periodically pull this repository to the MCS anyway so that a recent snapshot is always stored on the University system.

References

[1] A. Sarkar, A.D. Gordon, S. Peyton Jones and N. Toronto, "Calculation View: multiple-representation editing in spreadsheets" in *Visual Languages and Human-Centric Computing (VL/HCC), 2018 IEEE Symposium on.* IEEE, Oct 2015 pp. 85-94

[2] A. Sarkar, "Towards spreadsheet tools for end-user music programming", Computer Laboratory University or Cambridge

[3] Macvean. A, Church. L, Daughtry. J, Citro. C, "API Usability at Scale" in *Psychology of Programming Interest Group (PPIG), 2016 - 27th Annual Conference.*
Accessed (15/10/2018): http://www.ppig.org/sites/default/files/2016-PPIG-27th-Macvean.pdf

TC:endignore