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 formative evaluation using this initial prototype shall be summarised, and 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

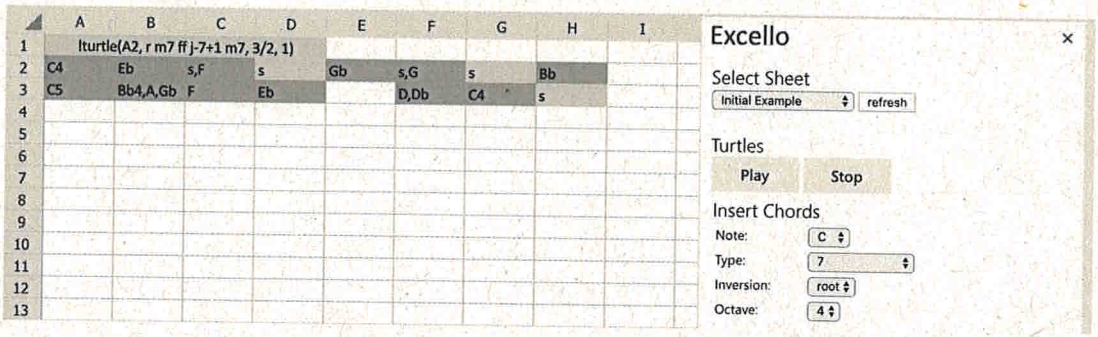

Figure 3.1: A two line motif defined in the initial Excello prototype

Notes and turtles can be defined in any cell. Turtles' interpretation of cells is shown in Table 3.1. When the Excello add-in is opened, a window opens on the right side of Excel as shown in Figure 3.1. Play and stop buttons launch all the turtles defined in the spreadsheet, and initiate playback with a realistic piano sound.

3.1.1 Turtles

The following formula is enetered into a cell in the grid to define a turtle:
`!turtle(<Starting Cell>, <Movement>, <Speed>, <Number of Loops>)`

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.

what is a string?

Table 3.1: Interpretation of cells.

Activation

The prefix "`!`" 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 (`A2` in Figure 3.1), 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.

↳ I've learnt a new word!

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.

Did this not happen? or is this explained later on?

Movement

Turtles start facing north. The language for programming turtle movement is discussed in the Preparation chapter. The instrutions are `r m7 ff j-7+1 m7` in Figure 3.1. Using brackets to repeat movements was not implemented by the start of the participatory design process.

had not been

Speed

What was the 1st/2nd argument? What is an argument?

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. "`3/2`", as in Figure 3.1, would move the turtle at 240 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.

I don't understand

Number of Loops

An optional fourth argument defines the number of repetitions of the turtle's entire path (1 in Figure 3.1). By default, the turtle loops infinitely. Repeating parts (e.g. the cello of Pachelbel's Canon in D) therefore only need defining once.

does this need explaining for people who don't know the piece → e.g. the two-bar (8 note) phrase repeated throughout the piece by the cello in... the same two bar

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[1] and starting octave of the chord are input 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[2] 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. Twenty-one University of Cambridge students, across a range of subjects, took part in the participatory design process. Initially, one-on-one tutorials with the initial prototype were given followed by a short exercise. After 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 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 collected as participants used Excello in their own time. This also ensured the summative evaluation was done with users with sufficient experience of the interface.

[1] Which note of the chord comes first, the other notes ascend from this. This is much like list rotations.
[2] Where the notes of a chord are played in rising or descending order.

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 (e.g. `ppp m p m mf m ff m`) made establishing the turtle's path harder, as not all commands related to movement ("`m4`"). 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.

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 (with "`!`"), 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. As these were often in a straight line, the Excel status bar allowed users to select cells and immediately see how many there are. However, this is still not productive, and is 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 [24] 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[3] melody line required writing out again and could not be derived quicker from the original version.

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[4] 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 [21]. 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

[3] Where the pitch of every note has moved by the same amount.
[4] A type of chord where the seventh note in the scale is added.

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 incorporate 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 relative value used initially. However, values less than 10 are interpreted in the original relative way for backwards compatibility for participants' existing work. To maintain consistency in a production version, this will be removed so speed must be defined absolutely. As speed and dynamics are different orders of magnitude, confusion between them is reduced.

3.3.4 Custom Excel Functions

Two custom Excel functions were implemented to aid composition. One to define turtles and a second transposes notes. This allows Excello to take advantage of the functionality of the existing Excel ecosystem; drag-fill, autocomplete, cell referencing, etc.

EXCELLO.TURTLE

When writing a formula, a prompt informs users the position of arguments and whether they are optional. This outputs the turtle definition as text. 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 3.2.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Speed:	200															
2																	
3	Melody:	C4	D	E	F	G	D	E	B	A	B	D	E	F	G	C	-
4	Bass:	C2	G	A	F												
5		lturtle(B3, r m*, 200)															
6		=EXCELLO.TURTLE("B4","r m*", 0.25 * B1)															
7		EXCELLO.TURTLE (start cell, instructions, **[speed]**, [loops])															
8																	

Figure 3.2: 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 dynamics. A section can be modulated by calling this function on the first note with a provided interval and then using drag-fill. 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 3.3.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Intervals:		1m	4m	1m	2M	1m	8M	-9M	1m	-2m	1m	-2M	1m	-2M	8M
2	Notes:	C4	C4	F4	F4	G4	G4	G5	F4	=EXCELLO.MODULATE(I2,J1)						C5

Figure 3.3: 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.[5] 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

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

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.

3.3.10 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 turtle system for playing the grid contents, the chord input tool, and the custom Excel functions.

When the play button is pressed, turtle definitions in the grid are identified. For each, the starting cell and movement instructions are used to establish the contents of the cells 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 3.4.

The `Sampler` is an extension of the `Tone.Instrument` class. This interpolates between pitched samples to create arbitrary notes. A sampler is loaded using the Salamander grand piano samples [6] 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

A drop-down is populated with names loaded using the Office API. Having pressed play, the cell values from the selected sheet are loaded then analysed for highlighting and

[6] https://freepats.zenvoid.org/Piano/acoustic-grand-piano.html.

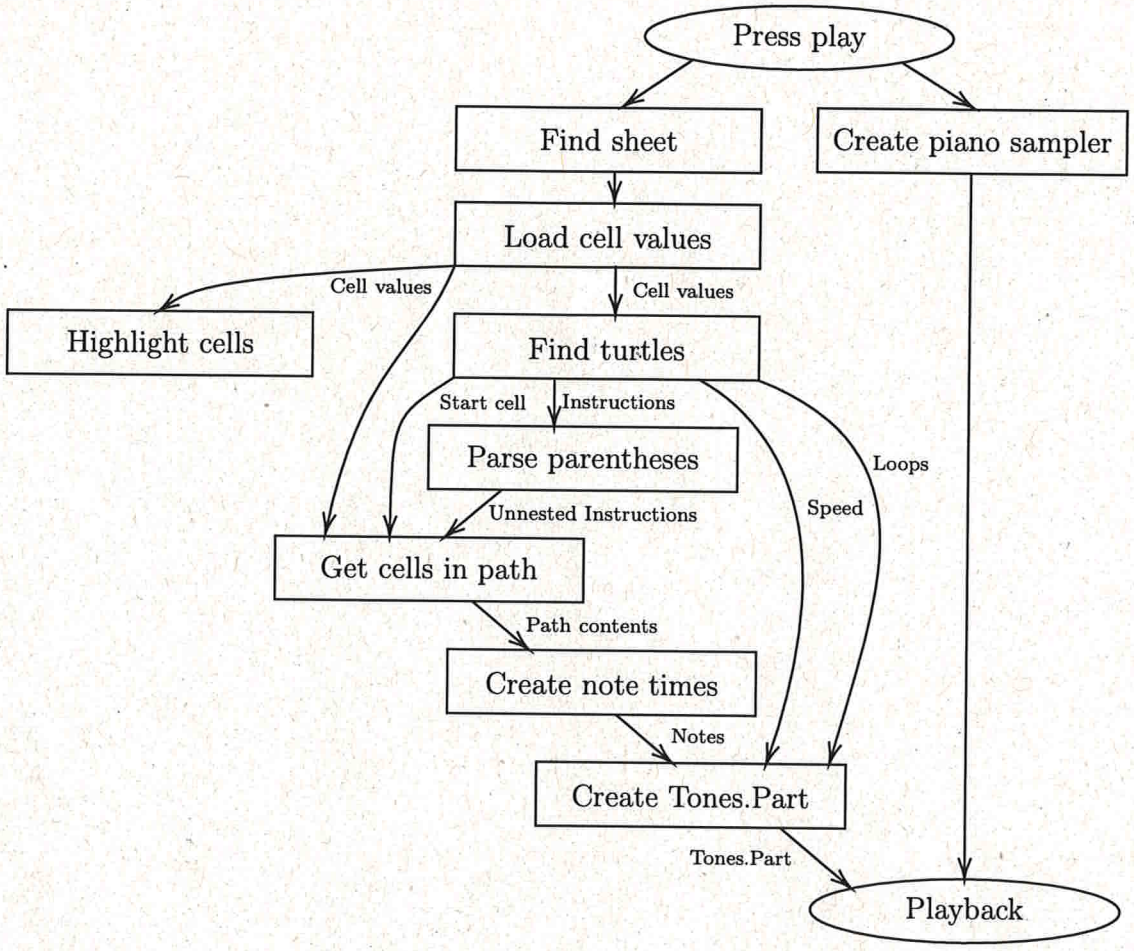

Figure 3.4: An overview of the playback algorithm and dataflow of Excello

calculating the musical output. Cells containing at least one note definition are highlighted red. A note must contain a note name, an optional accidental, optional octave number, and optional volume instruction following a space. This is 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 (e.g. "`C4 ff,-, ,D`", "`,,G,F#`") are split using commas. The resulting strings are trimmed of starting and ending whitespace and then must either be a note, a sustain ("`-`" or "`s`"), an explicit rest ("`.`") or an empty string (created from trimming a rest). Cells matching "`-`","`.`" or "`s`" are highlighted a lighter red. Turtle definitions (e.g. "`!turtle(B2:B4, r m*, 200, 1)`") are identified using:

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

and these cells are highlighted green. The address of cells containing a turtle definition are added to the live turtle section of the add-in window.

3.4.2 Parsing Movement Instructions

Movement instructions are converted to a single unnested list of commands (e.g. "`(r m2)2`" becomes "`[r, m2, r, m2]`") so the turtle's path can be established. The `parse` method of the Parenthesis[7] library seemed suitable for aiding this string manipulation. This parses strings containing brackets into a nested array. For example, `parse('a(b[c{d}])')` gives `['a(', ['b[', ['c{', ['d'], '}'], ']'], ')']`.

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

This has two main steps. First, the deepest bracketed expression is stored in an array with the brackets removed. This expression is replaced in the original string with the string `'___`x`___'` where x is the expression's index in the array. This is repeated until the string contains no brackets. Then a recursive function uses the values of x to reconstruct the string in the nested 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 turtle movement instructions, I implemented a fix to the Parenthesis library. The library previously performed replacements with the string `'___`x`'`. x and following numbers would concatenate forming a single number, causing the library to fail. My method of having an identifier before and after x fixed the issue. I also added additional tests to Parenthesis to verify my method and ensure that previous tests all passed before submitting a pull request. I have made an open source contribution as this has been merged into the library and published.

I wrote an additional recursive method to unnest the array into a single stream of commands. An empty string, `s`, is initialised. For each item in the array, if it is an array, unnest the contents recursively and add the result to s. If not, it will be one or more single movement commands. If the single movement commands start with a number, the result 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.2.

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

Algorithm 1 Parsing bracketed expression. `str.replace(regex,f)` (line 13) performs `f(s)` on the first substring, `s`, of `str` matching the regular expression `regex`.

```
1: procedure PARSEBRACKETS(str)
2:     idPadding ← '___'
3:     unnestedStr ← []
4:     deepestLevelBracketsRE ← RegExp('\\([^\\(\\)]*\\)')
5:     replacementIDRE ← RegExp('\\' + idPadding + '([0-9]+)' + idPadding)
6:
7:     procedure REPLACEDEEPESTBRACKET(x)
8:         unnestedStr.push(x.substring(1, x.length-1))
9:         return idPadding + (unnestedStr.length - 1) + idPadding
10:    end procedure
11:
12:    while deepestLevelBracketsRE.test(str) do
13:        str = str.replace(deepestLevelBracketsRE,replaceDeepestBracket)
14:    end while
15:    unnestedStr[0] = str
16:
17:    procedure RENEST(outerStr)
18:        renestingStr ← []
19:        while There is a match of replacementIDRE in outerStr do
20:            matchIndex ← index of the match in outerStr
21:            matchID ← ID of the match (number between padding)
22:            matchString ← matched string
23:
24:            if matchIndex > 0 then
25:                renestingStr.push(outestStr.substring(0, matchIndex))
26:            end if
27:            renestingStr.push(reNest(unnestedStr[firstMatchID]))
28:            outestStr = outestStr.substring(matchIndex + matchString.length)
29:        end while
30:        renestingStr.push(outestStr)
31:        return renestingStr
32:    end procedure
33:
34:    return RENEST(unnestedStr[0])
35: end procedure
```

can't help here - sorry!

Algorithm 2 Unnesting parsed bracketed expressions.

```
1: procedure PROCESSPARSEDBRACKETS(arr)
2:     s ← ''
3:     previousArr
4:     for v in s do
5:         if v is an array then
6:             previousArr ← processParsedBrackets(v)
7:         else
8:             if previous instruction was an array then
9:                 s ← s + previousArr
10:                if next instruction is a number then
11:                    s ← s + previousArr, that number of times minus one
12:                end if
13:            end if
14:            s ← s+ remaining instruction in v
15:        end if
16:    end for
17:    return s
18: end procedure
```

3.4.3 Getting Cells in Turtles' paths

If the turtle's first argument defines a range of starting cells, the cell addresses of this range are calculated. For each starting cell, the unnested instructions and sheet values are used to find the contents of the cells the turtle passes through. This process models the movement of the turtle within the grid, keeping track of its position and where it is facing. For each instruction, the position and direction are updated as required, and the contents of any new cells entered are added to a list of notes.

For the "`m*`" instruction, the number of steps the turtle takes must be computed. Given the turtle's current position and direction, 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 explicit rest. The number of steps is the array's length minus the index of the first element in the reversed array satisfying this criterion.

3.4.4 Creating Note Times

The cells a turtle moves through are used to create a data structure containing the information for playback 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>]]`. Volume and octave are added to each note if they were omitted from a cell.

The Tone Transport is a timeline along which events can be scheduled. This supports many different representations of time. I used Transport Time for all note onsets and durations. This is of the form `'BARS:QUARTERS:SIXTEENTHS'` where the three values are

number and can be non-integer. With `QUARTERS` representing the number of cells, the calculation of exact times, or arbitrary musical can be avoided. This allowed the contents of cells to easily be converted to times to be interpreted by the Tones library.

The note sequence array is initiated by counting the notes that are defined in the cell contents using 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 track how many cells and notes through the process the algorithm is, and whether it is currently 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 also keep track of these. Table 3.2 outlines the actions performed when a cell is read. Notes are added to the note sequence in the form `[currentStart, [currentNote, '0:' + noteLength + ':0', currentVolume]]`.

Cell	State	Action		
Note	Note	Note, octave and volume established from cell contents and previous values	Previous note added to note sequence	currentStart = '0:'+beatCount+':0' currentNote = value noteLength = 1 currentVolume = volume
	Rest		inRest = false	
Sustain	Note	noteLength++		
	Rest	Nothing (has no semantic value)		
Rest	Note	Previous note added to note sequence inRest = true		
	Rest	Nothing		

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.

The same method is used for multi-note cells, except the note length and cell count are incremented by the appropriate fraction for each item in the cell. If after the final cell, the state is a note, it is ended and added to the note sequence.

The values in the note sequence are sufficient for the piano sampler to play a note using the `triggerAttackRelease` function. The `Tone.Part` class allows multiple calls to this method be defined which can be started, stopped and looped as a single unit. This is defined with the note sequence and its speed set with the evaluated turtle speed argument. The number of cells, calculated when creating the note times, number of repeats, and the evaluated speed argument are used to control when looping ends.

3.4.5 Chord Input

When the insert button in the add-in window is pressed, the note, type, inversion and octave of the chord are extracted from their HTML elements. The chord interface within

the add-in window is shown in Figure 3.5. The tonal library is used to generate the chord notes:

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

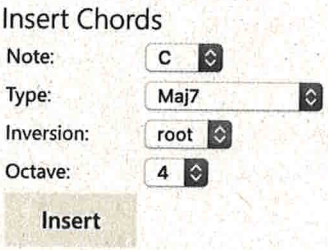

Figure 3.5: The interface within the add-in window for inserting the notes of chord.

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

The given octave number is appended to the first note. A dictionary matches note names, accounting for enharmonics,[8] to position in the chromatic scale starting at C (the first note of the octave in SPN). 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 selected range is acquired with the Office API. The chord notes are entered starting at the top-left corner of this range. If the range's height is greater or equal to its width, the notes are entered vertically going down. Otherwise, they are entered horizontally going right. Using the Office API, the range is set to the 2D array where the chord is entered.

3.4.6 Custom Excel Functions

Custom functions are implemented using another add-in. Rather than a separate window like the main Excello add-in, additional functions can be used in cells with the prefix "`=EXCELLO.`". The file structure was generated with the Yeoman generator. The name, description, result type, and parameter names and types are stored in a JSON schema. This is used by Excel to provide argument prompts and autofill when writing formulae. Functions are defined in Typescript.

The turtle function concatenates the arguments into the Excello turtle format. Other cells can be referenced, for example, the speed variable can reference a global tempo variable as in Figure 3.2.

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

The modulation function separates the note and volume for every note defined in a cell. The note is modulated using the tonal `Distance.transpose` function and recombined with the volume. The drag fill feature of Excel can be employed by the user to transpose sections or define melodic lines using the interval between notes as in Figure 3.3.

3.5 MIDI Converter

This section explains the Python converter from MIDI to CSV for Excello playback. 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[9] to read MIDI files and abstract away from the underlying byte representations to view the messages. Note onsets and offsets are two separate events with two separate messages [2]. These messages include the note pitch, velocity, channel (not relevant) and time in ticks since the last message [4].

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 (including control and meta messages) to update a variable tracking time. For note onset messages, this is added to a dictionary mapping pitches to a list of currently active note start 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 with end time added is added to the list of all notes defined in the file.

As each turtle only plays one note at a time, the notes are split into lists so no list contains concurrently played notes. Provided the initial list of notes is non-empty, a new list is created. The first remaining note is moved to the new list. Then iterating over the remaining notes, the next note starting after the previous note ends is moved to this new list. The number of iterations required is the number of turtles, n.

If every tick corresponds to a cell, any combination of note onsets and offsets in a MIDI file can be accurately represented in Excello. To achieve smaller representations, the start and end times are converted to smaller cell numbers 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 creating the different turtle streams, 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*.

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

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

For each note, the times are adjusted as follows:

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

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

$$end \leftarrow start + length$$

The streams, with note start and end times corresponding to cells, are converted to a CSV file for Excello. Each turtle's path is initialised as an array of empty strings with length equal to 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, modulus and division with 12 gives the note name and octave for SPN. If velocity is different to the previous note played by the turtle (or the note is the first note), the eight-bit MIDI velocity is mapped to Excello's [0,1] range. 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.

first what?

Finally, the turtle definition is 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:

$$cells\ per\ tick \times ticks\ per\ beat \times beat\ per\ minute$$

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

With a value of 1 for repeats, the turtle definition is put in cell A1 and the CSV exported.

3.6 Repository Overview

Figure 3.6 shows a reduced project file structure including all original source code. Excel Music is the add-in that parses the notation and produces music. Both this and CustomFunctions were generated using the Yeomann generator. The manifest.xml files are added to Excel and point to the resources to run the add-in. Users were given a different manifest pointing to a distribution hosted online with Surge. node_modules contains all libraries required to run the add-ins and is 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 buttons of the window are pressed and imports from the remaining Typescript files. turtle.ts contains all the code to produce musical output from the spreadsheet, 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.

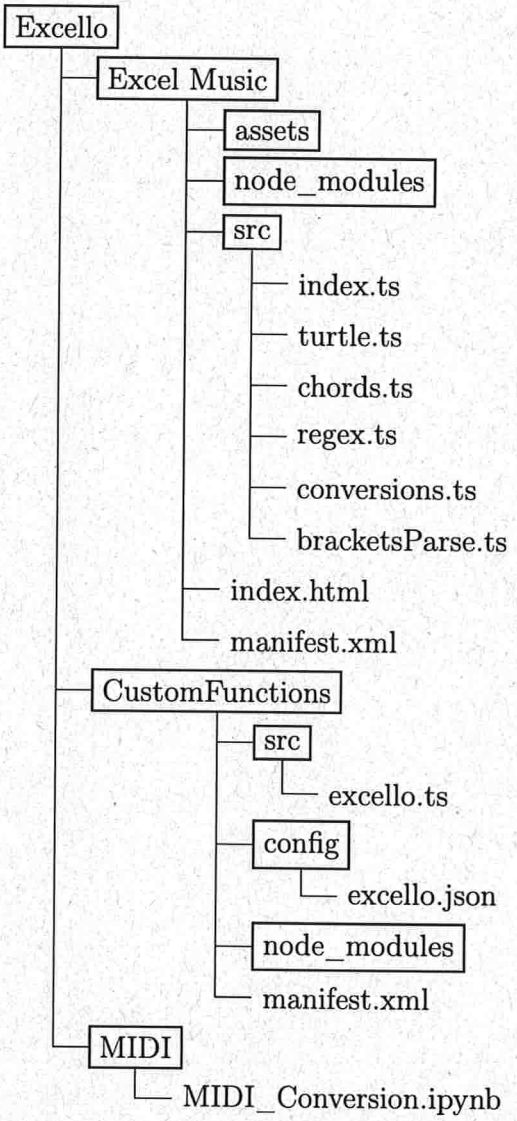

Figure 3.6: File structure overview showing original files

customFunctions.ts contains the implementation of EXCELLO.TURTLE and EXCELLO.MODULATE. 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. Conversions of MIDI corpora are included in the MIDI directory.

I shall release Excello as an open source project under the MIT license. This is compatible with the MIT licenses of the libraries. The Salamander piano samples come under a creative commons license[10], so credit shall be given in the Excel add-in window.

[10] https://creativecommons.org/licenses/by/3.0/.