Introduction

Making music in a documented fashion using computers, is a well-established process. Countless Digital Audio Workstations (DAWs) are available for users to create music on their machines using components such as real and virtual instruments, vocal tracks and other types of samples. In terms of collaborating, DAWs allow multiple users to work on a project, although it is convention for this to be done in the same location and machine – further flexibility is however possible, as users are also able to collaborate by sharing project files.

In terms of composition of music in a fashion through which users collaborate remotely (for example over a network), the options are far more limited. Ableton created “Link” functionality for their DAW and other Link-compatible software. Link allows musicians using compatible software to play in sync (live) whilst connected to the same local network – by ensuring that all players adhere to a global sense of time through tempo synchronisation, beat alignment and phase synchronisation. This project sets out to overcome the local network restrictions that are part of Link and provide a Link-inspired functionality that works in a wide area network, whilst still focusing on the element of live performances by make use of Sonic Pi – thus providing a distributed method for musicians to collaborate.

A wide area approach naturally introduces challenges involving dealing with networking issues such as latency and jitter, as well as (very critically) the problem of making sure everyone involved in the performance, observes the same events in the same order. This distributed sense of coherence is vital to ensuring that everyone perceives the performance in the same way (so that it sounds the same for all users) and is the main focus of this project.

Starting Point

Relevant Courses include Networking and Distributed/Concurrent Systems from Part IB.

Access to Ableton Link’s developer resources (including GitHub repository with source code)

Access to the source code for Sonic Pi through GitHub

Substance and Structure *i.e. Key concepts, Major work items + their relations and relative importance, Data structures and algorithms*

Link

Link allows users to play in sync by synchronising tempo, aligning beats and synchronising phase.

Tempo synchronisation

Tempo can be described as the rate at which a musical piece progresses, measured in beats per minute (BPM). In order for different musicians to play with each other in a coherent manner, it is essential that their contributions to the musical piece advance at the same rate – simply making sure that all musicians adhere to the same predefined tempo.

Beat alignment

Advancing at the same rate is not sufficient for contributions to a specific piece to sound coherent – it is also essential that the beats occur at the same exact time (that they are aligned). In order to clarify what is meant by this, we adopt a numbering notion for beats such that the first beat is denoted by number 1, the second by number 2, the third by number 3, etc. Beat alignment ensures that the difference between the beats of any two users, is always an integer, such that it is possible for one user to be at beat 4 whilst another is at beat 2, but it is not possible for third user to be at beat 2.5 (halfway between beat 2 and 3).

Phase synchronisation

When dealing with loops or bars (consisting of a set number of beats) it is also useful to ensure that there is some consensus regarding the points at which they start and end. This is so loops created by different users start/end at sensible places relative to each other. For loops of the same size, this means starting and finishing together (which may involve delaying the triggering of loops so their start in sync with others). For loops of different sizes, there may be more than one point at which a loop can be triggered – for example a 4-beat loop can start either at the start or the middle of an 8-beat loop.

Open Sound Control (OSC)

Open Sound Control defines a message-based protocol that allows communication between different multimedia devices such as computers and synthesisers. OSC messages can be used to encode descriptions of sounds (or musically related events) that can then be used in various scenarios. We will make use of OSC messages in order to communicate between Sonic Pi instances (being run on different machines by different users) in order to ensure that they all agree on which sounds to produce.

Sonic Pi

Sonic Pi defines a well-established language and software that allows users to produce music through live-coding. OSC messages are used within the software in order to describe music related events that need to be relayed to a speaker (through an audio synthesis platform called SuperCollider, integrated into Sonic Pi). Sonic Pi also defines temporal semantics which provide ways of reasoning about when sounds produced are by the software as well as making sure this occurs in a deterministic manner (or relaying to the user when that is not a possibility).

Structure

Implementation

A user will produce a script (using Sonic Pi) that will in order generate OSC messages describing the sound they intend to make -- said messages are then interpreted within the SuperCollider synth to relay the correct sound to the user’s speakers. These messages need to be sent from a Sonic Pi instance to all other instances that are part of the performance so that other contributors are also able to hear the sounds that the original user intended to produce. A server will be used to setup a connection between Sonic Pi instances so that OSC messages can be transported between them. Since we are focusing on a collaborative performance, it is vital that all users hear exactly the same sounds -- or in other words, that the lists of OSC messages interpreted by each user’s SuperCollider synth, are exactly the same. It is therefore important to account for aspects such as latency and jitter in the network to ensure that we are able to provide a deterministic order of events for all users when they are all transmitting OSC messages to each other.

Evaluation

After setting up a scenario where at least 3 users are connected via the server, we will examine the list of OSC messages that each user’s SuperCollider synth receives and see whether that said lists are identical for all users.

We then aim to focus on the deviation between the timelines of the users - even though users may be hearing the same thing, the events are unlikely to occur at the exact same points in real time. The idea is to find the limit at which the deviation is too great for a suitable interactive experience (so that users can react to each other’s contributions.

We also aim to find the point at which our idea of determinism regarding the distributed consensus of events breaks down. We want to investigate the point at which network issues such as latency are too great for messages to be scheduled at the same time for all users and thus, under what conditions we are unable to provide a deterministic view of events across users.

The last two parts of the evaluation can be carried out using dummynet (or a similar tool) to emulate poor network conditions (controlling packet delay, loss, etc.).

Server

* Build module in between Sonic Pi and Network module
  + so users can specify what connection they want to make
  + this would ideally have a nice GUI (maybe MAX or Pure Data would work, although I know little about them, could sure learn)
  + This then connects to a network module to get incoming OSC messages
  + Handles logic regarding telling Sonic Pi when to act on said messages as well as making sure it is doing the correct thing (i.e. all ‘clients’ work together to ensure agreement) – dealing with OSC ‘merge conflicts’
* Build network module
  + handles outgoing and incoming OSC messages
  + WebRTC as a method of transport (or server approach)
* Server
  + Connect different Sonic Pi instances
  + Will deal with the logic involved in
* Client
  + This will act as a bridge between Sonic Pi and the Server
  + Will pass OSC events that need to be scheduled
* Point
  + Sub-point
* Evaluation: look at timestamps for OSC messages between Ruby and SuperCollider and just check they’re the same across different users

Success Criteria

Fully implement a way for at least three users to setup a connection and play together using Sonic Pi.

Ensure determinism/coherence of each user’s notion of the performance (when network conditions are acceptable) so all users hear the same thing.

Ensure that users’ timelines don’t drift so far apart that they are unable to react to each other’s contributions (even when determinism holds).

Possible Extensions

Integrate WebRTC into the project instead of using a server to provide a peer-to-peer implementation -- if such an approach were to reduce latency significantly, it may be possible to implement features such as making it possible for different users to update (different) aspects of the same sound at the same time (whilst still keeping a deterministic view of events across all users).

Trying out different schemes for getting coherence.

Timetable

Weeks 1-2

* Read Lots advisor

Weeks 3-4

Weeks 5-6

Weeks 7-8

Weeks 9-10

Weeks 11-12

Weeks 13-14

Weeks 15-16

* Finish progress report
* Prepare presentation
* Start dissertation planning

Weeks 17-18

Weeks 19-20

Weeks 21-22

Weeks 23-24

Weeks 25-26

Weeks 27-28

Resources Required

I will be primarily relying on the use of my own laptop for most of the project – a laptop with Intel® Core™ i7 4720HQ Processor, 16GB RAM, 512GB SSD, NVIDIA® GeForce® GTX 960M GPU running Windows 10.

If my laptop should fail, I will use one a MCS workstation in the Computer Laboratory.

I will run weekly backups to a personal USB stick as well as to Google Drive.

I will require access to a VM Server in order to connect Sonic Pi instances (running on different machines) to each other.

Timeline

Ableton Link

Ableton’s Link functionality is designed to allow musicians using separate instances of Link-compatible software, to play in sync in a local network, by keeping a global notion of time – this is done by synchronising tempo, beat and phase alignment. To understand Link, it is first necessary to define what is meant by playing “in time”.

The starting point in allowing two or more users to play in time, is the synchronisation of tempo. It is possible to describe music as being bound by beats, signifying points at which notes should (or should not) be played. The rate at which these beats occur in relation to time, is deemed to be the tempo of the musical piece – this rate is sensibly measured in beats per minute (BPM for short). A collaborative performance, requires all musicians to adhere to a predefined tempo, so that there is no divergence of the rate of the individual contributions – any divergence leads to an incoherent piece as components sound out of place.

It is not sufficient to synchronise tempo in order for users to play in time, as it is possible to imagine a scenario in a collaboration where differences in starting points exist amongst users – such that one user starts their contribution slightly before or after the others. Even with a synchronised tempo, it is likely that the beats of each individual user don’t match up – even though they may be occurring at the same rate. The solution to this issue is making sure that the beats of a user’s contribution, align (occur at the same time) with the beats of all other users – implicitly, ensure that this alignment is done for all users. This can be further clarified if we use integers to number each beat (starting from 1 denoting the first beat). Beat alignment aims to ensure that the difference between the current beat number of any two users, is itself always an integer, such that it is possible for one user to be at beat 4 whilst another is at beat 2, but it is not possible for third user to be at beat 2.5 (halfway between beat 2 and 3).

The final step to playing in time involves phase synchronisation. The need for this comes from the idea of music being made up of bars consisting of a specified number of beats – Ableton’s software also has the notion of loops made up of bars – and the problems that arise when bars or loops are not aligned. For example, assuming synchronised tempo, if one user starts playing a 4-beat loop at beat number 1 (in a global sense) and another starts to play a 4-beat loop one beat later (at global beat number 2), even though the beats are aligned, the loops start and end one beat apart. This is often problematic as the start and end of bars/loops, often assign a sense of meaning to a musical piece. Phase synchronisation deals with making sure that the start or end of bars or loops are somehow aligned by synchronising the beat position of individual contributions. This means that different users playing a 4-beat loop will always observe the same start and end to their loops (even though their separate beat numbers may be different e.g. if one user starts their loop 4 beats before the other). For different sized loops, phase synchronisation still works if the loops in question are multiples of each other, such that a 4-beat loop will be synchronised to start either at the start or the middle of an 8-beat loop.

OSC

OSC defines a message-based protocol that allows communication between different multimedia devices such as computers or synthesisers. OSC messages can be used to encode descriptions of sounds that can then be used in various scenarios.

SonicPi

WebRTC