A Networked Visual Instrument
" VENUES "

James Tompkin
james.tompkin@gmail.com

26th April 2006

Abstract

Networked Visual Instrument is a project to create an audio instrument with a painterly, play-evoking user interface, that allows for multi-user real-time collaboration. The software created will be known as VENUES.

During development, VENUES matured from a simple instrument to a more featured sequencer. VENUES' interface is built on-top of Processing. It is an attempt to provide immediate experimentation with a fine granularity of control, so as to blur the line between its use being a technical skill and being a creative skill. Collaboration is responsive and easy to initiate, and is accomplished by input passing through OpenSoundControl. Thus, bandwidth usage is low, and audio output is of a high quality.

This report is included in colour on the CD as the hyperlinked document `Final Report.pdf`.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary.

Signed,

Date:

Acknowledgements

Ian Mackie, for his support and advice as both my project supervisor and my computer science good conscience.

Ben Fry, Casey Reas, Krister Olsson, Christian Riekoff, Andreas Schlegel, and the Processing community.

Dominique Tompkin, my mother, for internationalization assistance and for continually improving my French.

This project will be a fun venture into wonderland. Don’t forget your hat!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Collaborative software rarely leaves the o□ce. Be it team management, shared document editing or extreme programming, there is very little collaborative software in the creative arts market. There is potential for collaboration in this field, especially in group arts such as audio creation.

Audio creation software rarely leaves the studio. It is often expensive, complicated and a hindrance to immediate experimentation. Though powerful, manipulating audio software is chiefly a technical skill. There is scope to explore changing this interface ethos from a technical to a creative one.

Much work has been done on such interfaces, notably by Levin [9], but his work is aimed at one-time performance rather than production. Simple audio creation software often looks and feels like a cut-down version because it actually is a reduced version of more complex software. Less is often more during the initial stages of audio creation when properly designed. Thus, an accessible interface within which to experiment without interference increases workflow.

The aim of this project is to explore audio collaboration and play-evoking interfaces so as to produce a piece of software which successfully marries these ideas.

This report presents the processes applied to complete the project, from the topic's background to Venues' potential future. A review of existing noteworthy software was first conducted (page 5). Following this investigation, development was undertaken. This is documented in the design, implementation and validation sections (pages 13, 25, and 37). An in-depth evaluation and conclusion completes this work and gives insights into possible extensions (pages 39 and 45).

Chapter 2

Background

2.1 Collaboration

Computer-supported collaboration has existed in one form or another for over 30 years. The most broad-minded onlooker may consider the creation of ARPANET and the following communication tools (e-mail, ftp) in the early 1970's to be the first instances of computer-supported collaboration [1]. The creation of USENET in 1980 and specifically the advent of news bulletin boards and news software may be a more sensible starting point - enabling user interaction grouped around specific topics (e.g. sci.research or soc.culture.african). USENET became an environment where online communities could develop and prosper.

Alongside USENET, with the introduction of the minicomputer in the mid 1970's, attempts were made to support group working in software (so called O□ce Automation - involving integrating and extending word processors and spreadsheets to support teams and departments) [2]. In 1984 the term Computer Supported Cooperative World (CSCW) was coined, and along with the adoption of the term Groupware (denoting the research and commercial technology aspects respectively), gave focus to the development of collaborative systems in the workplace.

Since then, although USENET still exists, it has been superseded by internet forums and web logs. So called 'social software' is emergent in its opt-in approach, and exists primarily to enable interaction between individuals and groups (through whatever commonality). Social software often includes some 'six degrees of separation' aspect to link users, semantic tagging to categorize content, and publication tools such a photograph sharing. Websites such as `MySpace.com` package this into an online community.

Groupware has developed and is now an integral part of the modern corporate office. The groupware approach places people into groups defined organizationally or functionally. Management software keeps track of schedules, calendars, tasks and documents for the hierarchy of teams and workgroups. Communication collaboration allows remote conferencing and document editing. Tools such as SubEthaEdit allow for extreme programming and collaborative note taking in conferences.

Social software networks are flourishing - `MySpace.com`'s user base has quadrupled since January 2005 to 40 million members, and in October 2005 it accounted for 10%

of all advertisements viewed online [3]. In contrast, groupware has had a difficult development and a troublesome integration. Collaboration is often at odds with the competitive corporate environment, and convincing people to use the software is often the tallest hurdle to jump - groupware often fails to achieve the critical mass necessary to be useful [4] (a lack of interoperability and individual benefit are cited as common reasons for failing to achieve critical mass) [5].

2.2 Audio

Electronic instruments date back to the late 19th century, when Thaddeus Cahill developed the monstrous Teleharmonium, a multi-ton polyphonic multi-keyboard harmonic organ. Post-war, the analogue synthesizer (made famous by Moog) dominated the initially experimental electronic music scene for 35 years. By the end of the 1970's, musicians were using many different synthesizers to produce music, each using a proprietary connection system. The need to control many synthesizers at once coupled with the growing interest in digital sequencing led Dave Smith of Sequential Circuits and Roland to develop the MIDI standard in 1982. MIDI provided a common communication between synthesizers, and when using Roland's famous MPU-401 PC interface and MESA software, allowed for digital sequencing and synchronization. Twelve Tone Systems soon released their Cakewalk software, and computer music was born.

By the mid to late 1990's, desktop computers were becoming powerful enough to replace musical hardware. Software synthesizers and samplers replaced their dedicated analogue and digital counterparts. Graphical user interfaces tended to replicate the devices they were replacing - visual knobs and buttons needed to be turned using the mouse or via a dedicated MIDI controller. Sequencing software presented a grid interface of note against time.

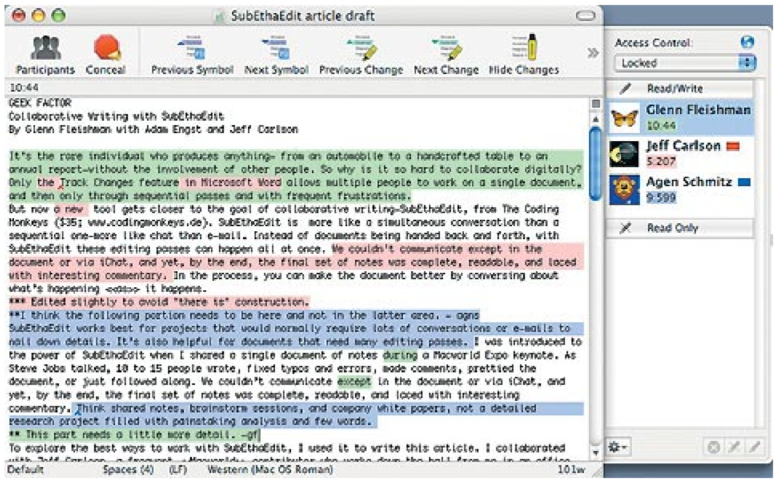
As computers became more powerful, music software allowed for more features. It has been said that, in the 21st century, audio is 'basically done' [6] - although it may not be real-time, whichever way you'd like to manipulate audio, it's possible. The digital studio can be compressed to a laptop with no real loss of function or quality. However, whilst ProTools and Logic are synonymous with the modern studio, in the home computer music is rarely exploited, leaving the pc to be more of a jukebox than an instrument. When Apple introduced GarageBand in 2004 they went some way to increasing the number of people using their computers for audio creation. Critchley comments that GarageBand allows creative thought to flow much more freely than Logic because "it has so little going for it" - the simplicity is "exactly what you need when you have an idea" [7]. Musicians are benefiting in the early stages of music creation from having stripped-down, simpler software.

Chapter 3

Review

3.1 SubEthaEdit

http://www.codingmonkeys.de/subethaedit/



```
10:44
GEEK FACTOR
Collaborative Writing with SubEthaEdit
By Glenn Fleishman with Adam Engst and Jeff Carlson

It's the rare individual who produces anything- from an automobile to a handcrafted table to an
annual report-without the involvement of other people. So why is it so hard to collaborate digitally?
Only the Track Changes feature in Microsoft Word allows multiple people to work on a single document,
and then only through sequential passes and with frequent frustrations.
But now a new  tool gets closer to the goal of collaborative writing-SubEthaEdit, from The Coding
Monkeys ($35; www.codingmonkeys.de). SubEthaEdit is  more like a simultaneous conversation than a
sequential one-more like chat than e-mail. Instead of documents being handed back and forth, with
SubEthaEdit these editing passes can happen all at once. We couldn't communicate except in the
document or via iChat, and yet, by the end, the final set of notes was complete, readable, and laced
with interesting commentary. In the process, you can make the document better by conversing about
what's happening <edit> it happens.
*** Edited slightly to avoid "there is" construction.
**I think the following portion needs to be here and not in the latter area. - agns
SubEthaEdit works best for projects that would normally require lots of conversations or e-mails to
nail down details. It's also helpful for documents that need many editing passes. I was introduced to
the power of SubEthaEdit when I shared a single document of notes during a Macworld Expo keynote. As
Steve Jobs talked, 10 to 15 people wrote, fixed typos and errors, made comments, prettied the
document, or just followed along. We couldn't communicate except in the document or via iChat, and
yet, by the end, the final set of notes was complete, readable, and laced with interesting
commentary. Think shared notes, brainstorm sessions, and company white papers, not a detailed
research project filled with painstaking analysis and few words.
** This part needs a little more detail. -gf
To explore the best ways to work with SubEthaEdit, I used it to write this article. I collaborated
```

Figure 3.1: *Each participant's contributions can be seen clearly labelled with their individual colours in SubEthaEdit.*

As previously mentioned, SubEthaEdit is a collaborative text editor for MacOS X. Each member's work is highlighted in their particular colour. The group's creator controls access rights to the people working on the text, but does not hold authority over anyone else in the group. SubEthaEdit also shows each person's location within the document by placing a representative coloured block in the vertical scroll bar. Whilst being a fully fledged editor, it only supports plain text (not rich text). SubEthaEdit prides itself on being a lean, high performance text editor without superfluous features - the key idea being that bloat complicates collaboration.

Similar software of note:

Emacs has supported collaboration through the talk-connect and make-frame-on-display functions for many years, though Bray suggests that their names have hindered their use [8].
http://www.gnu.org/software/emacs/emacs.html

MoonEdit is a multi-platform collaborative text editor. It has many of the features of SubEthaEdit, and includes useful tools such as a calculator. Its networking is built on a client/server architecture, and allows for collaborative sharing and editing of whole directories.
http://moonedit.com/

There is currently much fascination with collaborative editors, especially concerning wiki integration[1]. There exist only a handful of programs (approximately 20) that will edit text in real-time, and although uptake is slow, there is much enthusiasm over the possible applications of this technology. All three pieces of software support this impressive function (as opposed to near real-time or simple version control). Emacs' support of this feature is relatively unknown, as is MoonEdit, but SubEthaEdit is quietly famous. SubEthaEdit's advantage is in contact list integration, whereas MoonEdit only supports using a server's IP address. It could be said that the majority of SubEthaEdit's renown is not due to it supporting more features, but due to it being a darling of the Mac community.

3.2 ArtPad

http://artpad.art.com/artpad/painter/

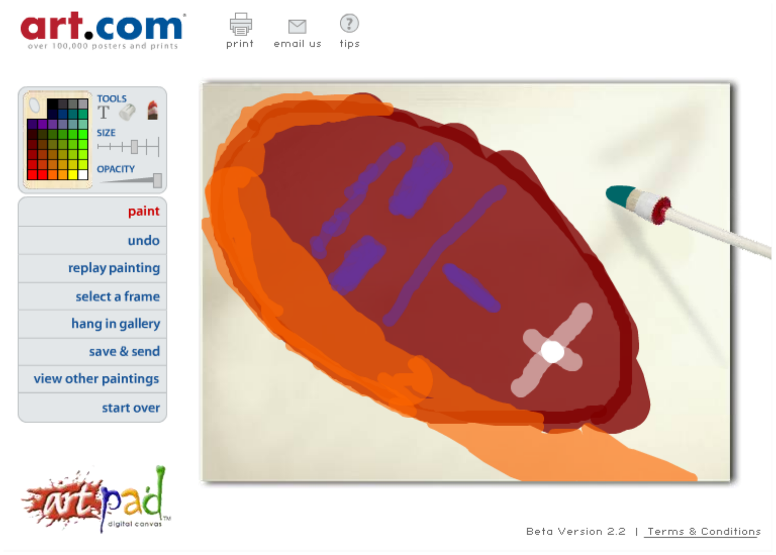


Figure 3.2: *ArtPad has many tools for sharing creations - here we can see 'hang in gallery' and 'save & send'.*

ArtPad is an online virtual canvas. It allows a single user to use simple tools to create a piece of art, and share it with their friends (through email or via a hyperlink). Mouse movements are recorded so that paintings can be replayed - each stroke can be viewed in order at various speeds. Thus, the viewer gains an insight into how the painting was created. The transferable nature of these paintings elevates ArtPad to a piece of collaborative software: once a painting has been received, it is still an open canvas - the receiver can add new paint or even remove existing paint from the canvas.

[1]Many users collaboratively editing an article inside a wiki, rather than each user in turn obtaining a lock on that article.

Similar software of note:

Imagination Cubed, released by General Electric, is an online whiteboard written in Flash. It allows users to collaboratively draw in real-time, allows the image to be shared and saved by email, and allows for simple printing. The interface is minimal, using a menu bar and a floating (though minimizable) tool window.
http://www.imaginationcubed.com/

Beauty and Chaos is a very simple digital canvas that allows for real-time collaborative drawing. Only one tool is provided: a yellow pen. No user interface exists so to speak, the only information given is whether you are connected to the server and how many people you are drawing with. Other users on the canvas can be seen as small cursors when they are drawing.
http://ericdeis.com/content/beautyandchaos/beautyandchaos.php

Imagination Cubed trumps ArtPad by supporting most of ArtPad's useful functions whilst allowing for real-time collaborative painting. Imagination Cubed provides the user with a much larger canvas, although it does have a restriction on the number of strokes you can place upon this canvas before it starts removing the oldest strokes. Unfortunately, Imagination Cubed suers because it is seen as an advertising activity for General Electric, and its merits are drawn from an exercise for a corporate community. Beauty and Chaos is somewhat dierent from the other software in that its interface is so sparse as to be experimental. The amount a user can do with just one tool is so restrictive that it forces them to approach the activity with a dierent frame of mind. This can make Beauty and Chaos' output more interesting than that of its counterparts; however, it also means that often it produces nothing constructive at all as users struggle to realise their ideas.

3.3 The Fridge

http://www.melbs.org/projects/fridge/index.cgi

An example of a simple piece of social software, the online fridge magnet allows people to compose poetry. Although the actual process is solitary, you can title and save your composition for others to read. Reading other people's poems is more than just reading text - the poem is presented as the final state of the fridge, allowing the poet to be spatially creative and providing the reader with a greater insight into the poet's workflow. Dierent magnet styles are available, from Shakespeare to E.E. Cummings. Code and poetry is distributed under the Creative Commons licence.

3.4 Just Letters

http://web.okaygo.co.uk/apps/letters/flashcom/index.htm

Just Letters is another example of fridge magnets. The participants share a virtual fridge surface and can move individual letters. The fridge's state cannot be saved,

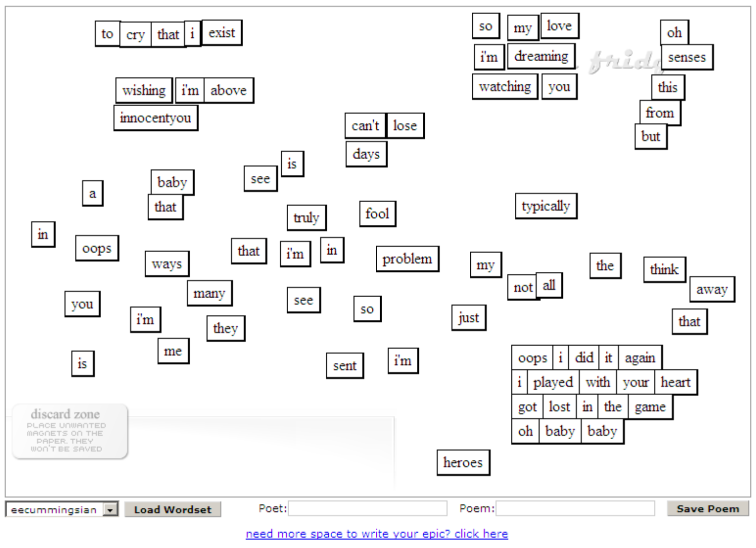


Figure 3.3: *One of the many magnet styles The Fridge offers is the Britney Spears style.*

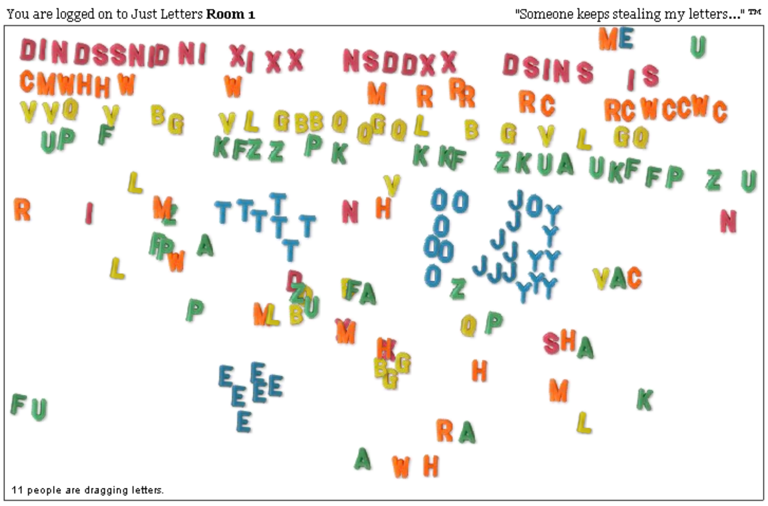

You are logged on to Just Letters Room 1
"Someone keeps stealing my letters..." ™
11 people are dragging letters.

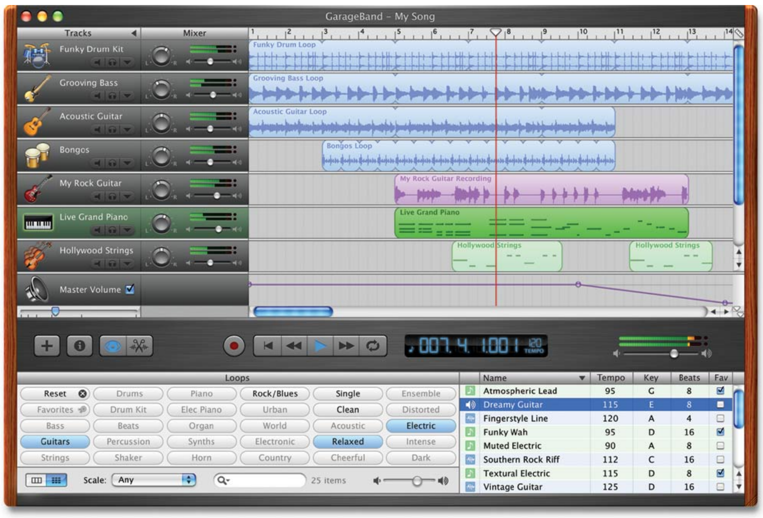
Figure 3.4: *The tagline, "Someone keeps stealing my letters...", encapsulates the frustration Just Letters creates.*

and there are no rules or conditions. The environment causes competition over highly sought after letters, and the sabotage of words is common.

The Fridge could not be said to be collaborative software, although it is social software. The key definition difference between the two virtual fridges is the direct interaction with other participants that makes Just Letters collaboratory (even if most users tend to competition). Similarly, Just Letters could not be said to be social software. All interaction is anonymous, there are no saved fridges - it could be said that social groups could use Just Letters only as a play activity.

3.5 GarageBand

http://www.apple.com/ilife/garageband/



Name	Tempo	Key	Beats	Fav
Atmospheric Lead	95	G	8	☑
Dreamy Guitar	115	E	8	☐
Fingerstyle Line	120	A	4	☐
Funky Wah	95	D	16	☑
Muted Electric	90	A	8	☐
Southern Rock Riff	112	C	16	☐
Textural Electric	115	D	8	☑
Vintage Guitar	125	D	16	☐

Figure 3.5: *On first glance, GarageBand's interface looks intimidating. However, compared to CakeWalk, it is both a piece of cake and a walk in the park.*

As previously mentioned, GarageBand is music creation software for MacOs X. GarageBand supports multiple instruments across multiple tracks. GarageBand supports audio (waveform tracks) and synthesized instruments (grid tracks - Live Grand Piano and Hollywood Strings from the screenshot). GarageBand supports plug-in synthesized instruments and loop creation. GarageBand is simple - there are no menus (let alone multi-nested menus), there are few button controls, and the interface relies on drag-and-drop.

Similar software of note:

CakeWalk Sonar Home Studio for Windows is of similar complexity to GarageBand. It supports a finer grain of control and co-operates with a greater variety of plug-ins and software. Sonar's interface is much more complex, however, and is full of

buttons, menus and tabs.
http://www.cakewalk.com/Products/HomeStudio/default.asp

Mixxx is an open source multi-platform DJ tool. As a digital DJ it provides channel mixing from file or source, beat estimation, and parallel displays. Of note is its very clean interface and clear labelling.
http://mixxx.sourceforge.net/

Apple currently has two major pieces of music creation software on the market: Logic and GarageBand. Whist Logic is designed to accommodate almost any audio task, GarageBand was designed separately from the ground up as simple software to enable the average consumer. Logic's competitors, one of these being CakeWalk, produced cut down versions of their software to compete with GarageBand - herein lies the difference: CakeWalk retains much of the control (from its big brothers) unnecessary for simple production and as such appears intimidating; whereas GarageBand appears inviting. Csikszentmihalyi's studies of the thought processes of creative people manifesting as flow are reiterated in Critchley's complaints about complicated electronic musical instrument interfaces [7]. His now altered workflow, using GarageBand as a whiteboard for ideas instead of Logic, demonstrates the very real need for interfaces that benefit expression and promote flow.

3.6 YellowTail

http://www.flong.com/yellowtail/

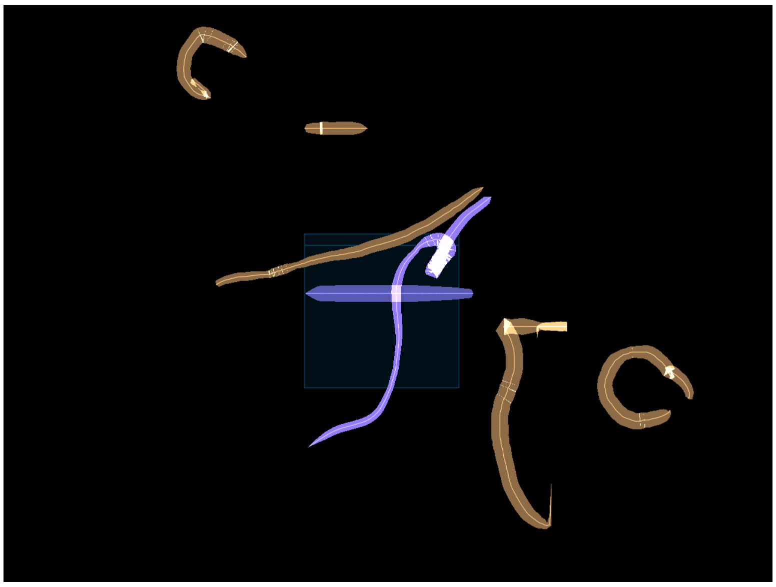


Figure 3.6: *YellowTail presents almost no interface at all - the user is free to experiment.*

YellowTail is an experiment by Golan Levin in interactive gestural creation and performance. YellowTail allows the user to draw lines on the screen, which are then repeated end-over-end so that they travel across the screen. As part of a suite of similar “painterly interfaces for audiovisual performance”[2], YellowTail interprets the moving lines on the screen and generates audio by applying an inverse spectrogram at points of intersection along a sweeping time indicator. YellowTail permits the real-time creation and performance of dynamic image patterns.

YellowTail is not music creation software in the same vein as GarageBand. The output from the inverse spectrogram method has been described by its creator, Frank Cooper, as “highly intelligible” speech [10], and whilst this makes sense in the abstract, many people would consider YellowTail’s output to be nothing more than ‘blips and blops’. As an instrument YellowTail succeeds in generating interesting audio samples for many different types of music (though it could be said none of them in the mainstream of listening habits). With prior knowledge of YellowTail’s process it is quite possible to match the audio to the visual.

YellowTail’s attempt to marry a free flowing interface with audio generation is to be commended, but more often than not the user’s intended effect is not forthcoming. It could be said that the same results could be generated by a pseudo-random algorithm, and for that reason the software is unfulfilling. Levin performed a number of composed concerts using YellowTail and the rest of his suite - whilst I would be enthusiastic to hear and (I predict) enjoy them, I would feel at a loss to ascertain the accuracy of the rendition.

All software features presented are correct as of October 2005. Newer versions of the software presented may now be available, however these updates did not factor in the design process.

[2]YellowTail was the first of Levin’s AVES suite, and his progression of ideas is demonstrated through the remaining four pieces of software. However, their discussion is not warranted here as their basis does not deviate from abstract expression. Interested readers are referred to [9] for more information.

Chapter 4

Specification and Design

4.1 Requirements

After review, here are the high level requirements of the software:

1. Provide an instrument for audio output.
2. Provide an interface for this instrument that removes clutter and promotes flow - to be as simple as necessary in an attempt to evoke play.
3. Provide a system for collaboration within this interface - for users to interact through a shared interface, to modify each other's work.
4. Provide a system for initiating collaboration with minimal networking knowledge.

1. The software is not required to be able to create music. It will provide manipulation of an instrument of limited complexity (e.g., a tone generator). The aim of the project is not to develop a realistic software synthesizer, nor to develop a fully functioning sequencer. However, if progress is swift, the ideal would be to provide functionality similar to that of a simple sequencer.

2. The interface should provide intuitive access to all of the instrument's features and attributes. Given collaboration, the interface should be accommodating to multiple users and provide information on the actions of the other users.

3. Collaboration should be location independent - collaboration does not require users to be in the same physical space. I have not stipulated that collaboration should be real-time. I will attempt to find a solution so that real-time collaboration is possible, but the outcome may be near real-time or worse.

4. Similarly, I have not stipulated the exact method for initiating collaboration. As much as it would be desirable to have a fully functioning website replete with community tools and contact tracking, it will be likely that I simply do not have time to implement this. A more likely implementation is client-server or peer-to-peer networking.

4.2 Functional Specification

Use Case diagrams were created to model the functionality available to a user (1 and 2). The first, third and fourth requirements (4.1) are functionally modelled by these Use Cases.

Functionally modelling the second requirement is a little trickier. It would be naive to specify an interface this early into development, but we can specify design principles that are to be adhered to. Using my review research, the design techniques set out by Ambler [11] and the design principles established by Constantine and Lockwood [12], here are the key interface design points I have identified that are specific to VENUES:

Canvas-like working area The system should maximize the available working space - this is vitally important to evoke play. If possible, there should be no space in the software that isn't workable area. (See YellowTail 3.6, Beauty And Chaos 3.2)

Floating/disappearing interface To maximize the canvas space, other interface elements should be able to 'float', and, if possible, entirely disappear when not in use. (See Imagination Cubed 3.2)

Simple interaction The most common tasks should be the simplest to perform. All instrument playing (phrase creation, modification, deletion, *etc.*) should be available by only using the main input device (in most cases, the mouse).

WYSIWYN Separate from WYSIWYG, "What You See Is What You Need". Everything required to create should be instantly available, and everything additional should not be visible [13]. This also applies to the amount of information presented to the user. (See SubEthaEdit 3.1)

Low screen density Musical score notation is clear and concise. To allow easy readability using non-standard notation (or, in the abstract, no logical notation at all), the ratio of objects to screen space should not exceed 1:2.5 [14].

Consistent The interfaces on and off the canvas should behave consistently. Given the specialized nature of the canvas, it is not expected that the canvas interface will be consistent with, say, the collaboration initialization interface (for instance, you shouldn't have to draw the address of the collaboration session you wish to join). (See ArtPad 3.2)

Intuitable Users should be able to make educated guesses as to how to use the system. Even when guesses are wrong, the system should provide reasonable results from which the user can learn. (See GarageBand 3.5)

Frequent response-based revision Create the interface through evolution - as new features are added, solicit feedback from potential users and integrate suggestions that are consistent with the interface design principles [15].

4.3 Design

Non-Functional Design

Awareness of the importance of the non-functional design of software is crucial. A bad design can impede tasks of all types. Although most of these tasks do not relate directly to the user, a bad non-functional design increases code complexity during implementation, and hence increases the time it takes to maintain and extend the system.

- Modularity - the system should be designed with a modular approach and the responsibility of each module should be clearly separated.
- Low Coupling and High Cohesion - internal modules should be highly related and the design should aim to minimize the dependencies between modules.
- Flexible and Extensible - it should be easy to extend and maintain the system.
- Comprehensible and Verifiable - the system's structure and components should be easy to understand and each component should have a clear purpose and meaning.
- Coding standards should be followed throughout.

Satisfaction of the non-functional design requirements is neither an easy task to perform nor a result easily measured. Using a metrics analysis package throughout development would assist in meeting these requirements. We can use the LCOM[1] metric to measure cohesion; the AC[2] and EC[3] metrics to measure coupling; and MCCC[4] metric to indicate the comprehensibility and verifiability of the method procedures.

Input Interface

The input interface for the system is critical. The method by which a user creates phrases for the instrument should not be slow or cumbersome. Analysing the review material shows that the problem found with YellowTail (3.6) is not intrinsically tied to its painterly interface - the significant usability problem of finding it unfulfilling falls squarely on the inverse spectrogram technique. Comparing YellowTail to the collaborative painting applications shows the benefits that a painterly interface could bring: it takes a chiefly technical skill (audio production in software) and translates it to a chiefly creative skill (non-technical drawing and painting). Painterly interfaces are also fast - in the case of YellowTail, it is possible to create a cacophony within seconds. In traditional software sequencers (such as GarageBand, 3.5), it is common to define envelopes for volume using vertices joined by edges - mapping out a visually mountainous landscape across the track (see figure). It could be seen as a small side step to take this idea and develop it into a painterly interface.

[1]Lack of Cohesion of Methods - a low LCOM value represents high cohesion.
[2]Afferent Couplings - the number of other packages that depend upon classes within the system.
[3]Efferent Coupling - the number of other packages that the classes within the system depend upon.
[4]McCabe's Cyclomatic Complexity is a measure of the linearity of code (i.e., how much it branches).

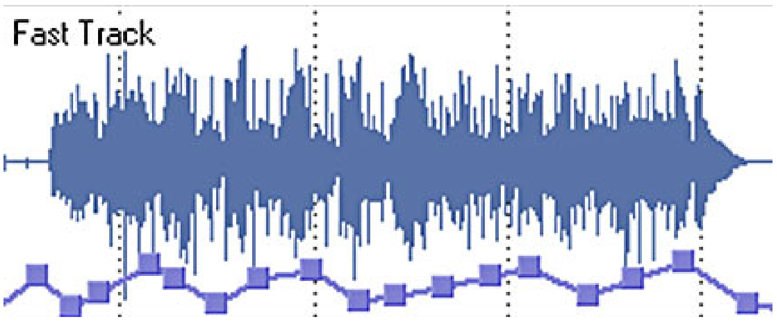


Figure 4.1: *We can see the volume envelope of an audio file being edited in Sony Sound Forge 8.0.*

YellowTail's interface could be seen to be missing important structure. Beauty and Chaos' lack of tools (and subsequently the interface for these tools) occasionally causes a creative problem for its users, and a lack of any control (to edit) could cause frustration. Providing users with the most important basic functions from a sequencer whilst still providing a minimal interface would strike a subtle balance. After referencing GarageBand, Cakewalk, *et al.*, I have identified what I believe are key features below[5]:

- Multi-tracked
- Volume control per-track
- Pan control per-track
- Tempo per-track
- Mute per-track
- Time bar control
- Edit control over created phrases (move/delete/modify)

YellowTail creates its worms based on the speed of input - the slower the creation, the thicker the worm. When the inverse spectrogram is applied, the thickness relates to the variance in the distribution of frequencies that are amplified and fed into the additive synthesizer. It would be possible to take this idea and extend it to modifying an attribute of the instrument - for instance to make the speed of phrase input (the thickness of the line) affect the volume of, say, the non-fundamental harmonics of a tone. MIDI users will no doubt see the potential for a relationship between the speed of phrase input and the velocity of a note.

I am not confident that modifying key attributes of the instrument based on the speed of input is useful. When implemented, it may in fact bring no benefit to the user if it is unwieldy and frustrating. For Venues to be fulfilling, the user must feel as though the system is responding accurately to the user's input. I will investigate this possibility in the implementation (5).

[5] Although I stated (4.1) that a simple sequencing environment would only be developed if time permitted, I now believe that it is a necessary feature for the system and it will be built into the design. Attempting to retro-fit a system to be multi-tracked that was initially hard-coded for only a single track may have proved difficult.

The proposed representation of a phrase (ignoring thickness variation) can be seen in figures 4.2, 4.3 and 4.4. If, for instance, the figures were mapped to volume and pan (4.4), we would have per-phrase volume and pan - this increases the granularity of control over the proposed key features (4.3). If a track has a default attribute then these would be applied to any phrase within the track that did not specify that attribute.

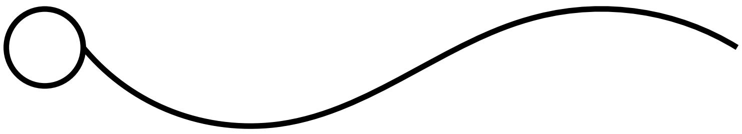


Figure 4.2: *The circle acts as a control point for the phrase - right clicking on the control point allows the user to drag the phrase, left clicking on the control point produces the following:*

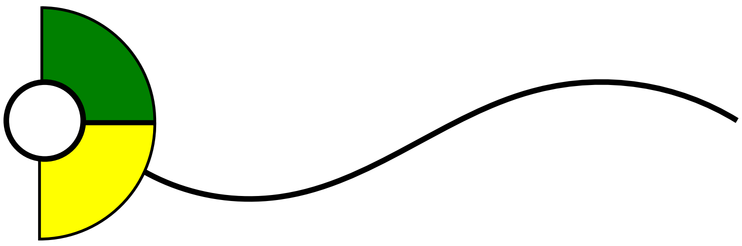


Figure 4.3: *A semicircular shape pops up allowing the user to select different attributes of the phrase. The user then draws another line from the control point that controls this attribute. The semicircle could expand to hold more segments depending on requirements (for example, a per-phrase mute).*

Data Representation

Given that the software is to allow multiple tracks, caution must be taken when thinking about the internal representation of phrases. If the user is to be allowed to change the number of tracks on-the-fly (a great boon for a flowing interface), the screen space allocated per track should expand and contract to maximize the working space. Storing the phrases as pixel positions would make this di□cult. Take this example: the user draws a phrase then adds a track. The software calculates the new pixel positions for the phrase (in the now reduced screen space of the track) from the old pixel positions. The user then removes a track, and the software calculates the new positions for the original phrase (for the now increased screen space). To not lose precision when up-sizing the phrase, a second non-changing representation of the phrase would have to be stored. Instead, storing the phrases as a ratio of the track's screen space would eliminate this complication. When rendering, the software takes the ratio and multiplies it by the screen size of the current track to give the pixel position.

A phrase will be stored as a set of figures - each figure represents an attribute of the instrument and is set at a certain 'depth'. Each figure will be stored as a set of points (represented as floats, as explained above). This can be seen in figure 3.

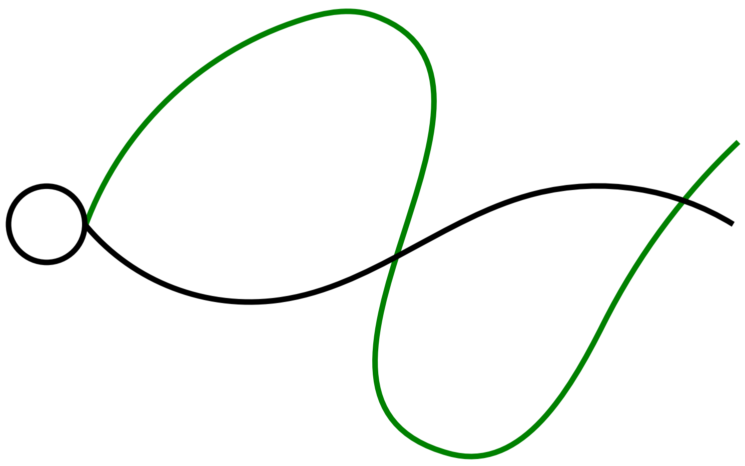


Figure 4.4: *The second line, in the corresponding colour and with less weight, may represent the volume or (in this case, a likely example of a) pan envelope of the phrase. Using such a system consequently brings about per-phrase, rather than per-track, volume and pan.*

Networking

The underlying networking architecture doesn't arbitrarily restrict the functionality of the application, but the architectural choice does depend on the amount of bandwidth the system requires and the amount of bandwidth available.

Bandwidth Requirements

There is a discussion to be had about exactly what data gets transmitted between users. The options available are:

Transmit audio The audio output from each client is transmitted to all clients and mixed and synchronized client-side.

Transmit data and receive audio The data[6] from each client is processed as audio on one machine, and this audio is transmitted to all clients.

Transmit data The data from each client is transmitted to all clients and processed as audio client-side.

Uncompressed audio at CD quality[7] requires 1.4Mbps of bandwidth. Accounting for moderate 20% network overheads [16] raises this to 1.7Mbps. In the first case, each client would require 1.7Mbps per client. In the second case, each client requires only 1.7Mbps of bandwidth, but the machine processing the audio again requires 1.7Mbps per client. Compression techniques could be used, requiring real-time compression

[6]The state of the system - a set of all values needed to reproduce that state.

[7]16 bit, 44100Hz Stereo.

Mouse Input	Bits
Mouse Pressed	boolean 1
Mouse Button	byte 8
Mouse X	short int 16
Mouse Y	short int 16
Total	41
x 125	5125

Table 4.1: We can see how much bandwidth is necessary to transmit the mouse input from a client.

(not a trivial task for a CPU) and decompression. Although this would reduce the bandwidth requirements to 230kbps[8] per client, it would damage the application's credibility with musicians because compressed audio is almost never desirable.

If that much bandwidth were available (this is certainly the case for LANs), both the first and second cases guarantee that all clients would hear exactly the same output. This is a very significant property, as audio hardware will undoubtedly vary from machine to machine.

The third option requires considerably more processing power per client (each client's data must be processed as audio client-side) but reduces bandwidth requirements significantly. It is di□cult to estimate exactly how much bandwidth would be needed, but if each client's mouse input were transmitted 125 times a second[9] it would require just over 6kbps per client including overheads (see figure 4.1). The third option does not guarantee that each client's output will be identical (this would only be the case if each client had identical hardware) but it has bandwidth requirements that are (with optimization) within the reach of dial-up users.

Broadband adoption in the UK has been fast, but not 'synchronous 10Mbps per household' fast. As transmission of uncompressed audio is not feasible, and transmission of compressed audio is undesirable (and has its own significant overheads), the most reasonable option is to transmit client inputs (and other significant data where necessary).

Network Architecture

The two architectures available for real-time systems are client-server and peer-to-peer. Given that we will be transmitting client inputs, the network processing requirements of both architectures are the same - each client must process all other clients' input. However, client-server architectures become more bandwidth e□cient when the number of clients rises above two. This is a significant result for a real-time system - with four clients, client-server uses 40% of the bandwidth that peer-to-peer uses. With eight clients, client-server uses only 16%.

[8]192kbps: Average value for variable bit rate compression using LAME's CD quality setting –alt-preset standard.

[9]Most USB mice poll at 125hz, PS/2 mice poll anywhere from 30-200hz.

Client-server does have a natural single point of failure and the server is liable to become a bottleneck. The code is also more complicated to implement, however I have past experience with client-server architectures and do not anticipate having architectural problems.

In the games industry, where many products include real-time networking, it is common for products to ship with dedicated servers[10]. Whilst a dedicated server provides more options for networking setup, they usually become necessities when client numbers climb into the tens and beyond. Although I can imagine a collaborative musical instrument with 16 users, it is not a pleasant thought imagining what it might sound like (perhaps a virtual conductor could orchestrate). I do not plan to support that many users, and so a dedicated server is unnecessary.

On the other hand, a master server[11] would provide many benefits. A master server would meet requirement four (4.1) aptly. When a client wishes to connect to a collaboration session, they would query the master server, which would return all available sessions. The client then joins the most suitable session with minimal fuss.

Synchronization

For the occasion that a client's input goes astray, the collaboration session must be able to synchronize itself. Assuring that clients are synchronized after every transmission would be overkill. A more suitable strategy would be for the server to regularly send 'synchronization checks' with the most current state of the session. If any client returns false, the server can initialize synchronization with all clients[12].

User Access Rights

Requirement three (4.1) states that users should be able "to modify each other's work". In Just Letters (3.4), the environment is a competitive one, as users regularly sabotage other user's attempts at creating. An initial reaction to prevent competition and encourage collaboration might be to implement content write locks or access rights.

Beauty and Chaos (3.2) does not implement write locks on content - users can edit any other user's work. However, Beauty and Chaos does not provide users with any means of removing paint from the canvas, and so the edit functionality is limited to 'overpaints'. As the brush strokes are thin, sabotage is made ineffective and attempting to completely overpaint another user's work is painstaking.

SubEthaEdit (3.1) has per-user access defined by the collaboration leader, but once a user has access rights they are free to edit any content. This is a 'trusted' write lock that depends on the validity of the identity of the user that is invited into a

[10] As opposed to a listening server, where a client and server is run on the same machine, a dedicated server runs solely to be the 'middle-man' between clients.

[11] A master server holds the details of all the servers currently running, globally.

[12] It is necessary for the server to synchronize with all clients, rather than just those out-of-sync, to stop those clients once again becoming out-of-sync during the time that it takes to synchronize.

collaboration session. Write privileges may be revoked at any time. The 'undo' facility included in SubEthaEdit neuters any potential saboteur.

When viewing VENUES as a tool for artistic expression, it is diﬃcult to justify the presence of content write locks. Beauty and Chaos is primarily an artistic tool (and something of an experiment), and content write locks are at odds with its ethos. It could be suggested that including an option for the server to toggle write locks would satisfy both those users wanting to experiment and those users wanting to work. However, the mere inclusion of a content write lock option defines the way in which a user views the system. If the concept of ownership is introduced into a collaborative environment it immediately secludes users from one another and ultimately hinders collaboration.

SubEthaEdit's access rights provide an eﬀective solution to enable trusted collaboration without enforcing content write locks.

The kick facility outlined in the functional specification (4.2) provides a way for the collaboration leader (i.e., the server) to remove users. Trusted access similar to SubEthaEdit would require either a contact list (with unique id and authentication server) or integration with an existing 'contacts' system (instant messaging). I do not believe I have time to implement either of these two non-trivial solutions. However, a much simpler way of enabling trusted access is by allowing the server to specify a password. Although it is not as elegant as SubEthaEdit's solution (which allows for 'spectating'), it is still absolutely eﬀective.

Graphics Rendering

The system's rendering technique must be able to deal with the complicated user interface design rules (4.2), such as floating and disappearing components, in real-time. The user interface must also update fast enough to match the speed of client inputs so that the networking is transparent. A typical 'windowed' GUI will most likely not provide suﬃcient update speed to cope with client inputs, and it may be necessary to use a graphics API such as OpenGL to accelerate the rendering. Once the speed of rendering has been established I can then match this to the client input speed.

Audio

The design of the audio subsystem is heavily dependent on the development environment. It is impossible to design any details until I know what is feasible with the audio interface I will be working with. Audio interfaces will be discussed when the development environment (5.1) is decided upon.

Internationalization

Requirement three (4.1) dictates that the system must allow collaboration over a network. It is shortsighted for any piece of software that is networked and/or distributed on the Internet to not at least support different user locales (even if the users must themselves translate). Internationalization (henceforth known as 'I18n' or 'i18n'[13]) options should be available for all relevant fields. This includes providing easy translation for all text, providing location specific number, date, measurement and currency formatting, and providing non-culturally bound images.

4.4 Structural Specification

A traditional three tiered architecture consisting of a GUI, Functional Core and Data Repository is not a good model to follow for this software. Drawing the GUI in real-time is one of the significant tasks that the functional core must perform. As an alternative, two tiers may be represented - a 'real-time processing' tier and a 'data repository' tier (figure 4.5). The processing layer deals with network, graphics, and audio processing, whilst the data repository stores the state of everything that needs to be processed. In each real-time tick[14] the software will handle client inputs, draw the GUI, and process the audio.

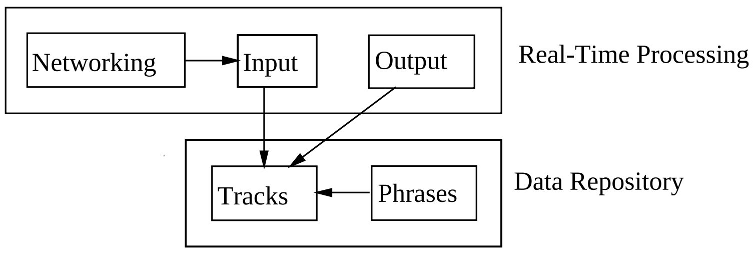


Figure 4.5: *Although the software uses many fields of computing, its architecture is simple.*

A class diagram of the proposed structural design can be seen in figure 4. It is worth noting that:

- the main class can be both a server and a client (known as a listening server).
- each client's input is of the type `UserInput`. The main class has many UserInputs.
- the master server is a separate entity. It is not connected in any way to the main application and runs as its own application.
- the master server has many server objects. Similarly, the client has many server objects.

[13]Internationalization is generally shortened to i18n as there are eighteen letters between the first and last letters.
[14]Similar to the tick of a clock, each tick represents a frame of the system.

4.5 Behavioural Specification

As modern hardware is capable of both rendering 2D and processing audio at speed, it should not be necessary to thread the GUI drawing separately from the audio processing. However, the client input collecting should run in a separate thread. Client inputs will be received at any time (between ticks, within ticks, and of any number), and these need to be collected and ordered so that no input is lost. A statechart of the processLoop() method can be seen in figure 5.

Figure 6 shows the threads of the system, their lifetimes, and their activities as a UML Sequence Diagram. The diagram also shows the predicted network activity (ignoring master server communication).

Chapter 5

Implementation

5.1 Development Environment

The development environment should be defined by the easiest path to implement the most challenging subsystems. In Venues' case, these are the graphics and audio subsystems.

Graphics Subsystem

Java Cross-platform. Provides extremely accurate 2D software rendering through Java2D with a fine granularity of control. Supports per-pixel AA[1]. Rendering comes at a cost, however, and a blank canvas of size 400x400 pixels fails to reach 60fps[2]. Zero cost SDK and IDEs.

OpenGL Cross-platform. Industry standard. Provides accurate 2D rendering through fixed 3D viewports with hardware support. AA support is per sub-pixel and hardware dependent. Rendering is fast - modern hardware renders a blank canvas of size 400x400 in excess of 1000fps. Zero cost SDK and IDEs.

Processing Cross-platform. BETA. Provides accurate 2D rendering and fast inaccurate 2D rendering through a fixed 3D viewport (P3D). Provides an interface for JOGL[3]. AA is supported per-pixel in accurate 2D and JOGL modes.

DirectX Windows. Proprietary suite of technologies. Component Direct3D is equivalent to OpenGL in low-level api functions and rendering speed, however Direct3D also provides high-level access to different rendering techniques. Requires a Microsoft development environment such as Visual Studio, and use of a supported language (Basic,C++,C#,J#). £299 cost SDK and IDE.[4]

Torque Cross-platform independent game engine. Provides 2D and 3D rendering through OpenGL. Microsoft Visual C++ is the only compiler supported for Torque source. £70 cost SKD and IDE plus cost of compiler (£299, within Visual Studio).

[1]Anti-aliasing.
[2]Frames per second.
[3]Java bindings for OpenGL 1.5.
[4]On 7th November 2005, Microsoft announced Visual Studio Express Edition - a free development environment for students and hobbyists based around a cut-down Visual Studio 2005.

Audio Subsystem

Java Cross-platform. Provides 32 2D channels in software through JavaSound.

OpenAL Cross-platform. Provides abstracted access to hardware 3D channels.

Processing Cross-platform. BETA. Provides extremely limited 2D audio support, though libraries Sonia and Ess extend support to make Processing competitive.

DirectX Windows. Component DirectAudio provides abstracted access to hardware 2D and 3D channels. Feature rich.

Torque Cross-platform. Provides multi-channel (unspecified) 2D, and 3D through OpenAL.

It is difficult to separate these subsystems as different environments require/advise using certain complementary subsystems. Whilst it would be possible to use, say, Direct3D for graphics rendering and OpenAL for audio processing, it is easier (specifically, it is painless) to integrate Direct3D with DirectAudio. All solutions offer high-level networking interfaces, and so this is not a factor.

Anti-aliasing is necessary because, unlike most GUIs, VENUES' GUI will not be constructed primarily of straight lines. Users may input phrases arbitrarily - anti-aliasing is necessary to make these phrases look like lines and not stairs. Hardware AA is desirable as the task is CPU intensive.

The Torque engine would provide the easiest path, and as a benefit the engine centres a helpful community of hobbyists. However, the cost is prohibitive. Java's 2D rendering is too slow to be useful. OpenGL is sufficiently fast for rendering, however OpenAL provides no 2D channel support. It would be possible to create a flat model for a 3D sound space, but this is unnecessary and overly complicated.

I will be developing VENUES in Processing. However, attention must be paid to its drawbacks:

Primitive IDE Processing's IDE is extremely primitive. It is designed for people unused to programming. I will be using Eclipse[5] instead.

BETA status Processing is in development and has bugs. Although it is quite possible that I will not encounter any problems, I will pay attention to regression testing to ensure that Processing updates do not break existing functionality.

Limited documentation Much of Processing is undocumented. I will interact with the Processing community and make use of forum archives so that I do not repeat the mistakes of others.

Difficult Integration Processing runs as an applet, which makes it particularly difficult to integrate with other applications and the shell (not least for security signing reasons). This reinforces the difficulty of contact integration noted in the networking design.

[5]http://www.eclipse.org

Networking will be implemented using the OpenSoundControl [17] protocol [6]. OSC is a modern networking protocol with support for pattern matched messages, concurrently executed message bundles, URL-style naming scheme, and simple message identification. OSC is designed to support a client/server architecture and is transport-independent. One major advantage of using OSC is that it is gaining major support within the computer music industry - Native Instruments support OSC messages across all their products[7]. With little e□ort it would be possible to interface VENUES with any other OSC supporting system (be it hardware or software). This is discussed in the Future Work chapter (8).

To monitor how well VENUES is satisfying the non-functional design, I will be using Team In A Box's Eclipse Metric Plugin[8].

As VENUES is a distributed system, it would be useful to have remote troubleshooting and debugging tools. I will setup a VPN using VNC to access the master server. Eclipse also supports remote debugging and hot-swap through JPDA, further reducing downtime.

5.2 Development Model

As an individual implementing a system from scratch, I will be using a staged evolutionary prototyping model. Subsystems will be prototyped and integrated in sequence. Unlike standard prototyping a working system will not be available early in the development process, because to develop Venues to a 'working' stage, the integration of three complex subsystems is required. Evolutionary prototyping is also a good fit for the user interface development, as it was not possible to specify this in advance.

Prototyping makes verification di□cult as the specification and design is significantly abstract. It is possible to validate that the system meets the requirements by demonstrating the adequacy of the system. I will be writing JUnit test cases where useful alongside system code, making use of David Sa□'s Continuous Testing plug-in for Eclipse[9] to alert me to test case failures as I type. The results of this will be covered in the Validation chapter (6).

5.3 Implementation

GUI

Each `Track` has a `ScreenSegment` that specifies where the track is located and how large a space it occupies. This is a generalized rectangle solution and allows for arbitrary placement and size of tracks. For ease of use, tracks are placed vertically and occupy the full horizontal width of VENUES. This is similar to standard musical notation and sequencing software such as GarageBand (3.5). Existing tracks are

[6]http://www.cnmat.berkeley.edu/OpenSoundControl/
[7]Intakt, Reaktor, and Traktor.
[8]http://www.teaminabox.co.uk/downloads/metrics/
[9]http://pag.csail.mit.edu/continuoustesting/

automatically resized when a new track is added. Tracks are toggleable full-screen by use of a keyboard button - this is context sensitive depending on which track the mouse is currently in. Track height can also be increased and decreased - other tracks automatically adjust to fill the entire canvas.

The current active track[10] is highlighted, all other tracks become slightly faded through a transition animation. Track information is displayed in the upper right corner of each track. This fades away when the mouse is in proximity to allow all areas of the canvas to be used. It is important that track information is displayed most of the time as it diﬀerentiates between visibly similar tracks. Buttons to add new tracks and remove tracks are displayed in the lower right corner of each track. These only appear if the mouse is in proximity, and should the user be creating/modifying a phrase, they do not appear (so that they do not intrude).

If no tracks exist (if the user removes all tracks) then the user is presented with a screen containing buttons to add new tracks. The screen also informs the user how to add new tracks via the keyboard shortcuts.

Phrases are implemented as a series of ArrayLists. Figure selection[11] is implemented through a complicated series of animations with fading via alpha-blending.

Message output has been redirected to the GUI. When the user invokes message output display, a column of text slides down from the top left of the screen and fades in. Keyboard help can be invoked (in a very similar fashion) - this slides and fades in to the right of the screen. When the user dismisses these informations, they slide and fade out.

All animations are framerate independent. VENUES creates a `Clock` thread which handles animation timing. The GUI is then free to render as fast as it can.

The GUI supports colour schemes. All colours used are redefinable via a simple interface. Colour schemes may be loaded at any time and changes are both immediately eﬀective and saved to file as a preference (the preferred colour scheme is loaded when Venues starts). A selection of colour schemes are provided, however users can save new colour schemes and edit existing colour schemes. A default colour scheme exists. Should a user not specify certain colours in their scheme, the corresponding default colours are used. A user cannot overwrite the default colour scheme. Were the user to attempt to edit their preferred colour scheme directly by editing the `.csc` file but make a mistake (or if the stored preference failed to load for whatever reason), Venues uses the default colour scheme stored as a hard-coded backup.

Certain GUI elements too time consuming to custom code have been borrowed from Swing - most notably, file choosers. However, Swing components share the same colour scheme as Venues' custom GUI. This mapping is defined in a separate file and is loaded at start-up. It is possible for a user to manually specify which Swing elements they would like recoloured (and which colours they would like to be used) by editing this file.

[10]The track that contains the mouse.
[11]Which attribute of the instrument we are editing.

Audio

Time signatures, along with a direction (legato, allegro, *etc.*) are used in musical notation to define a tempo. Venues defines the beats per minute, beats per bar (the quantization) and the number of bars per-track. This is in line with software sequencers. The user can visually segment each track from these values - this is toggleable per-track and fades in and out. These three values (all variable by the user) define the speed of the sweeping bar (or time bar). The sweeping bar can move through a track from left to right and vice-versa - this is toggleable. As the bar moves within a track, it crosses phrases. The height within a track at which the sweeping bar intersects a phrase determines the value of the corresponding instrument attribute. The clock handles sweeping bar position calculation and bar/phrase intersection. This occurs 60 times a second.

Venues supports three types of tracks: Wave (or Tone), File and MIDI. Wave tracks can be either Sine, Square, Triangle or Sawtooth. Wave tracks have a frequency range, variable from 0-40Hz to 0-10040Hz. If the user holds down the Control key, the frequency relating to their current mouse position and the closest musical note to that frequency are displayed.

File tracks are created from an external file. This can be of type AU, AIFF, WAV or MP3. File tracks can be set to play once or loop for the length of the phrase. The loop points are designated by notches along the phrase.

The first time Venues is started, it will ask the user which MIDI synthesizer they would like to use (this is dependent on the hardware in the computer). This is stored as a preference. The user can change synthesizers at any time. Should Venues fail to initialize the stored synthesizer, it will prompt the user to make another selection. Each MIDI track has a MIDI channel and a MIDI patch (or instrument). Channels are auto-managed (the MIDI standard supports 16 channels). If the user holds down the Control key, the musical note is displayed (or, for the drum channel, the drum type). Speed based input is accepted on MIDI tracks - this varies the velocity of a phrase. However it is di□cult to achieve the desired results, and so it is toggleable. Speed based input has not been removed from Venues so that users can experience this idea, but it is classed as an experimental feature and does not work in a collaboration session (Venues automatically turns it o□).

Venues supports 16 MIDI tracks and 32 audio phrases. The number of tracks has been capped at 8 to provide a manageable interface.

Each phrase spawns its own `PhraseThread` which is controlled from `Clock` through methods in `Phrase`. PhraseThread handles the audio processing and dispatching for a specific phrase. If a phrase's thread is busy, a user is notified by the relevant phrase being greyed-out.

Control

Phrase creation is bound to the edges of the containing track. Phrases can be duplicated (copied). Phrases can be dragged within a track. If a track is displaying its segmentations (bar/quantization markings) the user can, whilst dragging a phrase, snap to the nearest quantization. Phrases can also be dragged from track to track (of any type). A phrase's constituent POINTS can be manipulated in two ways: points may be deleted using a rubber (an eraser), and points may be dragged. Deleting the starting point of a phrase removes the phrase entirely. Phrases can also be deleted by picking up a phrase (through dragging) and pressing a key. The points that make up a phrase are not usually drawn - the user can toggle their display for easier editing of phrases. If a user taps on a phrase starting point, they hear a preview of the phrase. The user can undo any phrase manipulation - the undo list is per-user, and is currently set to store 10 manipulations (though this is an arbitrary restriction).

Phrases can be drawn past the edges of the track, however these are capped visually (and audibly) at the limits of the track. This allows you to draw invisibly past the limits of the track, and then through dragging the phrase, bring that detail back into the track.

The sweeping bar can be directly manipulated. Whether the sweeping bar progresses is toggleable. Tapping a track will snap the sweeping bar to that position. The sweeping bar can also be dragged in any direction. This enables the sweeping bar to be used as a scrubber when it is moved through a phrase[12]. The sweeping bar can also be thrown left or right - if the user is dragging the sweeping bar and the mouse is not stationary when they release, the velocity of the mouse is transferred to the bar. Physics then kick in and the bar accelerates/decelerates to the tempo defined by the track. Manipulation of the sweeping bar like this is not a feature regular software sequencers o er. It helps evoke play in the user as it is fun to throw the sweeping bar around and hear what is produced. It also adds an element of performance to VENUES.

Should the sweeping bar intersect a phrase at more than one point, the value of the attribute assigned is the highest point of intersection. The user is free to draw any phrase they wish, however should they wish to transform the phrase to only that which is read by the sweeping bar, VENUES can COLLAPSE the phrase. This uses a geometric slicing algorithm, and is worth considering in more detail:

```
COLLAPSE(Phrase $p$)
1   For all pairs of adjacent points within $p$:
2         do if proper intersection exist between two pairs
3               then Add point of intersection to separate array.
4
5   Add all intersection points to $p$.
6   For all pairs of adjacent points within $p$:
7         do Remove all points underneath the lowest point in the pair within slice.
8
9   Sort the remaining points in $p$ by $x$.
10  Reposition the starting point to be the point with lowest $x - coord$.
```

Proper intersection testing is performed using left turn testing. The fact that the line defined by the phrase is arbitrary means that no implementation quicker than $O(n^2)$ exists. COLLAPSE runs into problems when an overhang exists. An overhang is caused when:

[12]The user hears the exact frame in the audio buffer underneath the sweeping bar. By moving the sweeping bar through the phrase (or 'scrubbing the phrase') the user can reproduce the phrase at varying speeds, directions, *etc.*

1. The first point of a phrase is not the smallest x-point.
2. The last point of a phrase is not the greatest x-point.
3. The phrase snakes so that the bend of a curve is not either the smallest or greatest x-point, and there exists more phrase below with smaller or greater x-point.

Overhangs can be dealt with by projecting x-points of high y-values onto the next highest y-value of the phrase (and creating a new point). However, the diﬃculty comes in distinguishing which of these projected points is a valid point on the line of greatest y, and which can be discarded. As the line is arbitrary, and can be either clockwise or anti-clockwise, this is not a simple task. Even if we can isolate the correct projected points, it is diﬃcult to know how this point should be sorted in relation to the other points. A correct collapse can be seen in figures 5.1, 5.2 and 5.3. Collapse produces accurate results except on overhangs, where an x-point is not projected. In the time I allotted to Collapse, I could not devise a solution to this problem. However, I devised an alternative.

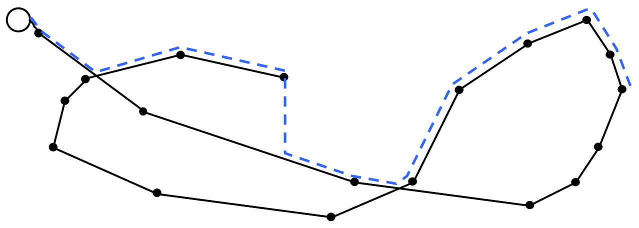


Figure 5.1: *The dotted line shows the peaks that would be read by the sweeping bar.*

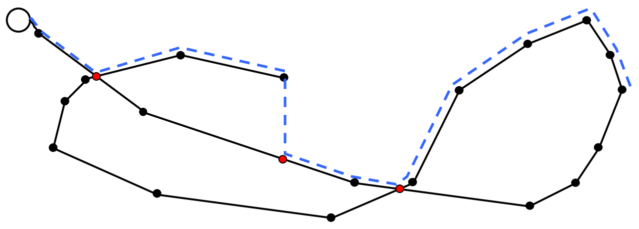


Figure 5.2: *Here we can see the intersection points and overhang point at the end of the phrase.*

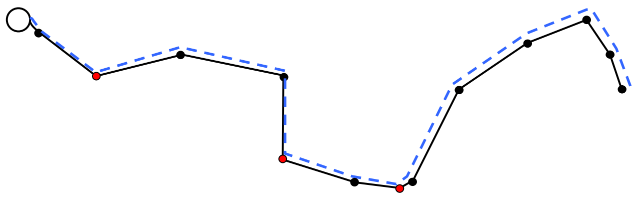


Figure 5.3: *The final collapsed phrase.*

Collapse2 is an $O(n^2)$ algorithm that produces exact results. It has a much larger constant than Collapse, though there is no diﬀerence in perceived performance on modern hardware. Collapse2 is also a slicing algorithm, but each slice is only one unit large.

```
Collapse2(Phrase p)
1  For each pixel in the width of $p$:
2        do Add point of highest y to separate array.
3
4  Replace all points in $p$ with those in separate array.
5  For all pairs of adjacent points within $p$:
6        do Check gradient between two points.
7            if gradient is the same as the previous pair
8               then remove point.
9  Reposition the starting point to be the point with lowest $x - coord$.
```

Collapse2 generates a point for each pixel, and so the second half of the algorithm (lines 5-8) removes all unnecessary points. The user can switch between these two algorithms at will.

Venues supports input devices. A user may plug in a device, such as a joystick or joypad, and use it as a replacement for the mouse. If the device a user plugs in does not meet the requirements for a mouse replacement (two axis, three buttons) it may be set up as an auxiliary device to control less important functions. Figure selection and phrase starting points are scaled up to accommodate the differing precision between devices. Should a user be using a device during collaboration, figure selection and phrase starting points are scaled for every user to provide a consistent interface across users.

Networking

The master server requests a refresh of information from each server every 20 seconds. Should a server not have sent a refresh respond to the master server within a minute, the server is removed from the master server listings. This situation may come about if a server crashes (i.e., hardware failure) without managing to send its 'server quitting' message to the master server. The master server constantly outputs to a log file for easy monitoring. The master server runs completely separately as its own application.

The master server address is specified in a file. If the master server is down, the client is notified. The client may still directly connect to a server by specifying an address. Once a client has received the list of servers, it pings all servers on the list. The server responds with either a thumbnail picture of its current state[13], a user-defined logo, or no image. As servers return responses to the client, they are added to the user's `ServerBrowser` in real time. As servers are added, they are sorted by round-trip-time. The user is able to filter servers if they are full, refresh all servers, and specify a personal name and server password.

[13]Thumbnail creation is threaded, though the server's GUI does pause for approximately 200ms as it writes the pixel information to an array. Unfortunately this pause is unavoidable.

lower	upper	%deviation	prime
2^5	2^6	10.416667	53
2^{12}	2^{13}	0.113932	6151
2^{13}	2^{14}	0.008138	12289
2^{19}	2^{20}	0.000127	786433

Table 5.1: Displaying the closest lower and upper power of two's, and the deviation from the optimal middle of the two.

As client `UserInputs` arrive, they are added to a queue of inputs of type `QueuedInputs`. The queue is then processed by the `Clock` thread at each tick. This prevents client inputs getting lost should more than one arrive between each tick. The queue's implementation allows for concurrent access (so that the `Client` may write new inputs to the queue whilst the `Clock` processes them).

Users can communicate to each other outside of the canvas using text through the `ChatDialog`. Chat runs in its own thread, which is created when a user joins a server. The chat display can be toggled on and o (whilst still receiving messages), and when new chat messages are waiting unseen users are notified by a small on-canvas icon.

Should a user try to load an audio file into a track during a collaboration session, it is likely that not all clients will have the file. VENUES oers the ability for the user to send the file to all other clients in the background.

The server sends synchronization checks every five seconds to all clients. This packet includes hash checks for the game state. Hash creation and checking is very fast[14]. Should hash codes not match, the server synchronizes with all clients.

Java requires that hash codes are guaranteed to be equal if two objects are equal, but are not guaranteed to be not equal if two objects are not equal. This creates a problem whereby two unequal objects could potentially generate the same hash code and collide - this would be an unacceptable situation for our synchronization.

We can reduce the possibility of collisions by deterministically generating our hash using properties of the object. Using a specially selecting prime number as the base for hashing minimizes clustering in hash tables (and hence reduces the possibility of a collision over time during synchronization). Using a prime as far as possible from the nearest two powers of two yields good results in practice [18]. We need to consider the length of the hash code. Hash codes should be at least as large as the square of the number of input messages. For a track, the number of input messages is

$$tc + (p * (f * (2 * n) + fc) + pc)$$

where p is the number of phrases, f is the number of figures, n is the number of points, and tc, pc, fc are respective track, phrase and figure constants equalling the sum of the specific object attributes to be hashed.

A hash length of 32bits allows for 65536 input messages. Assuming each phrase is full and has 500 constitutent points per figure, this allows for

[14]Sub 1ms.

$$\lfloor\frac{\sqrt{2^{32}}-8}{5017}\rfloor = 13$$

thirteen phrases per track[15]. A 32bit hash code is not enough. A 64bit hash code allows for

$$\lfloor\frac{\sqrt{2^{64}}-8}{5017}\rfloor = 856082$$

a lot of phrases indeed, and is more than su□cient for VENUES. A 64bit hash can be used as a per-session hash rather than a per-track hash. If eight tracks are in use (VENUES' imposed limit) then a 64bit hash allows for

$$\lfloor\frac{\sqrt{2^{64}}}{8*(8+(p*5017))}\rfloor = 1,\ p = \lfloor\frac{\sqrt{2^{64}}-64}{8*5017}\rfloor,\ p = 107010$$

83 times more phrases than would fit on an 800x800 canvas.

Other Operations

Users can save and load their compositions. If a user attempts to load a composition that references files they do not have, they are prompted for alternatives. If the server loads a composition, it is sent to all clients and loaded. Clients cannot load a game. Clients and servers may save at any time.

Users may capture screenshots - these are stamped with the current date and time and saved as PNGs.

Exceptions are handled by a custom exception handler which logs errors and advises the user.

Internationalization

All information has been externalized and localized. The first time VENUES is run, it prompts the user to select a locale. This is then stored to file as a preference. The user may change their locale at any time with immediate e□ect. VENUES includes a full French translation (excluding help).

Help

A user guide is included within VENUES and is displayed as html. This is created in a threaded dialog so that users may switch between the help dialog and the canvas without restriction. The help system supports i18n and loads the current locale when invoked.

[15] tc = 8, pc = 7, fc = 2

Implementation Complete Diagrams

Figure 7 shows the structure of Venues post-implementation. We can see the following major diﬀerences (discussed in the implementation) when compared with the structure proposed in the specification:

- Audio processing has been threaded separately from GUI drawing and is per-phrase in `PhraseThread`. This allows for one phrase to be busy, whilst the GUI continues to draw and other phrases continue to be processed.
- All other key real-time subsystems are threaded separately from GUI drawing and are contained within `Clock`. The GUI halts when Swing components are overlayed and so threading is a necessity. It also separates the GUI from the functional core - in eﬀect this returns the system to a typical three-tiered architecture model.
- `UserInputs` are queued before processing - these are stored as `QueuedInputs`.
- Each track has a `ScreenSegment` that defines and distributes the track's canvas space.
- `ServerBrowser` provides a threaded server browser to the user. `ServerDialog` provides a threaded dialog for server creation options.
- `Chat` provides a threaded chat interface for users.
- `ColoursDialog` provides colour scheme editing to the user.
- `I18n` provides internationalization support to the user.
- `ScreenShot` provides background threaded screenshot scaling and compression.
- `ConsolePrintStream` redirects console output to an internal message stream.
- `HelpSystem` and `LinkFollower` control the threaded help interface and hyperlink navigation.

Figure 8 shows thread creation internal to Venues and external to the environment. Method calls are simplified (for instance, the incoming call `queryServer()` is handled by the `Client` thread before the `MainThread` creates a new `ScreenShot` thread).

Figure 9 shows the major communications between client, server and master server. The messages displayed are data-simplified and reduced to one message for each operation (for instance, sending a server's image totals many messages). When a server is created, the listening client[16] does not communicate with the server externally during connection at all (all messages are internal, we shortcut the process).

Figure 10 shows navigation around the GUI from the canvas. Components to the left of centre are Swing elements, and components to the right are instrument/sequencer functions.

[16]The client created automatically when a server is created - the client of the user who created the server.

Chapter 6

Validation

Venues does not lend itself to simple validation. It is applet-based, highly graphical, and its output (audio) is not easily measurable. It is real-time, highly threaded and distributed. The system is susceptible to race conditions, which are NP-hard to predict statically [19] and virtually impossible to detect dynamically[1].

Unfortunately, race conditions must be debugged by hand. However, other testing methods were implemented to reduce the work load. As planned, JUnit test cases (white box tests) were written for those features that could be easily deterministically evaluated. These proved invaluable during network programming. When I was implementing synchronization code I ran into a defect whereby the synchronization check was failing consistently, but the test cases for synchronization were passing successfully. The test cases allowed me to trust the synchronization code, and instead directed me to investigate other areas where the problem may have lain[2].

JUnit test cases do not work well for GUI-intensive code. Defects related to the way Venues looks on a particular graphics card or the way JOGL is handled cross-platform cannot be easily tested using JUnit. JUnit works best for testing programming logic. Similarly, whilst JUnit test cases can be used for networking test simulations, they do not work well for real networking testing.

Functionality was regression tested throughout development. Low cohesion between subsystems and comprehensive threading isolated regression testing. If the system exhibited high cohesion, regression testing would have taken too much time and would not have been feasible. To aid this process, JUnit cases were written to monitor defects and ensure that fixes introduced were not subsequently broken. This successfully tracks ‘fragile’ fixes[3]. Table 1 documents a subset of regression tests.

Black box testing for Venues was a demanding task. The system contains close to *six hundred*[4] possible user functions (not including the master server). Table 2 documents a subset of black box tests.

[1]Every memory access would have to be examined.

[2]The problem actually lay with the client input queuing.

[3]A fix that no longer works if another change is made to the system.

[4]592. (50 functions per track x 3 track types x 3 modes (offline/server/client)) + (43 other functions x 3 modes) + 13 miscellaneous mode specific.

User testing took place throughout the development of the system. This was necessary to follow the ‘Frequent response-based revision’ principle of GUI development (4.2). Features and changes forthcoming from this process included:

- Point rubber (eraser) and point dragger.
- Sweeping bar reverse.
- Depth selection (phrase attribute) sizes and timing suggestions.
- Point rubber and point dragger size and behaviour suggestions.
- Keyboard shortcut mapping.

Two points came up during user testing that are worth discussion. The first concerns the method to select which phrase attribute we are changing. When first presented with the selection, users instinctively click to select (when in fact they only need hover). Without initial training, users objected to not having to click and suggested that selection should work through clicking. However, once accustomed, the motion became natural. To accommodate users who wish to click, a click at any stage of depth selection will accelerate the process. The second point concerns content write locks. One user suggested that they would like content write locks. After reasoning with the user, I did not go back on my decision (discussed in the design (4.3)) and did not refit Venues with content write locks.

Network testing deserves a special mention due to the environments in which the system was developed. King’s College DCS labs have an internal software firewall, which had to be disabled before any tra□c could flow between machines. All DCS tra□c also flows through a proxy server, which blocks Venues’ port[5]. Similarly, at my residence (in one of King’s College’s halls) almost all ports are blocked, and policy does not allow ports to be opened. Recruiting enough PC’s outside of restricted networks with which to test Venues was less an engineering problem than a social one.

[5]Port 4660

Chapter 7

Evaluation

7.1 Technical Evaluation

Venues di￿ers substantially from its initial design, in both scale and execution. The project initially started as merely an instrument, with no specified collaboration means or goals. The decision to turn Venues into a more featured sequencer increased the design complexity of the system. During implementation a similar shift in thread complexity occurred.

Whilst originally envisioned as occupying two threads of execution, the system is now heavily threaded. When in a collaborative session, Venues occupies five more threads than there are phrases. This was a necessary design change[1], and the benefits are immediately apparent to the user. Separating the GUI from the functional core lowered cohesion and made the system easier to work with as more subsystems were added.

The networking design did not change. Added features, such as user chat, sit happily on top of the existing networking structure. Bandwidth usage was estimated to be 6kbps for mouse input. Table 7.1 shows the implemented use. OSC messages have a 32 byte address pattern, and a 4 byte type tag. An OSC argument is zero-padded to the nearest 4 bytes. OSC uses UDP as a carrier, which has a 28 byte header for IPv4 (48 bytes for IPv6). This gives us a value of 37.5kbps for 60 transmissions per second (note that the original estimate did not include keyboard input or implementation-level overheads). We can clearly see that packet overheads dominate usage. However, client inputs are only transmitted when necessary (up to a maximum of 60 transmissions per second). Average use is di￿cult to estimate, however we can use results from simulated collaboration (table 4) to plot a box and whisker diagram (shown in figure 7.1) to demonstrate a five-number summary.

Regardless of real-world performance, there is clearly scope for optimization. On a canvas 800x800, only 9 bits[2] are required to represent an input's axis, saving 3.2kbps. 10 bits will accommodate a canvas with a greater resolution than that of most monitors (2047x2047). Similarly, 3 bits can be used to represent the mouse

[1]Given that Ess halts Processing's draw() thread. A two-threaded design would pause the GUI rendering when Ess was busy.

[2]Unsigned. $2^{10} = 1024$. $1 - 2^{10} = \Sigma\ all\ bits = 1023$.

Input	Bits
Device Pressed	boolean 1
Device Button	byte 8
X	int 32
Y	int 32
Key	char 16
Key Code	int 32
Key Pressed	boolean 1
Total	122
+ zero padding	128
+ osc overhead	416
+ udp carrier	640
x 60	38400

Table 7.1: *We can see how much bandwidth is actually used per transmission. The input device is usually a mouse. The key code is an identifier for non-ascii characters.*

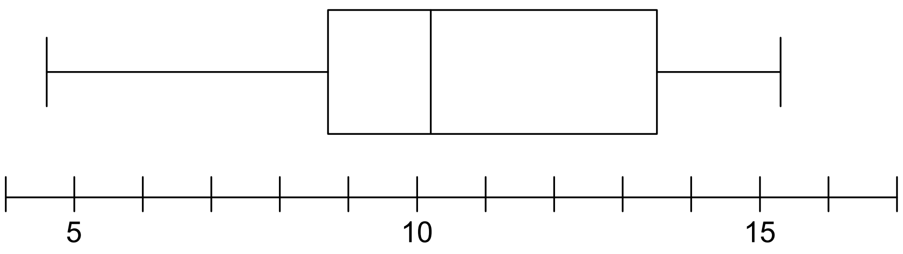


Figure 7.1: *Values denote input messages per second. Median 10.2, Q1 8.7, Q3 13.5, Min. 4.6, Max. 16.3, Mean 10.72, Std.Dev. 3.43.*

buttons (as 3 bits gives 8 permutations[3]), saving another 0.2kbps. Adaptive Huffman coding would be suitably fast and inexpensive for real-time compression of the entire OSC message and would improve the situation further.

If we compare these result to the best case alternative design (of transmitting audio from a central server) we can see the implementation-level improvement. Assuming audio is compressed using the excellent aacPlus[4] codec, which features CD quality[5] encoding at 48kbps (without overheads), Venues uses 65% of the bandwidth. When we factor in average use (mean), Venues uses 12% of the bandwidth. This equates to supporting 8 times the number of users. With the optimizations described and average use, Venues would use 10.5% of the bandwidth.

Venues' real-time performance can be evaluated by looking at the output frames per second across different hardware. Table 7.2 demonstrates the results. As the system uses OpenGL, graphics rendering performance is GPU dependent. The

[3]8 permutations map to 3 mouse buttons as button presses are not mutually exclusive. If presses were exclusive, we could use only 2 bits for 4 buttons.

[4]http://www.codingtechnologies.com/products/aacPlus.htm

[5]16 bit, 44100Hz Stereo.

CPU	GPU	FPS (No AA)	FPS (AA)
A64 3500+	nVidia GeForce 6800GT	64	64
P4HT 2.8GHz	Intel 82865G	52	52
P4 2.8GHz	Intel 82865G	49	49
P4 2.4GHz	nVidia GeForce 2MX	42	42
K7XP 1.4GHz	SiS 740	35	32
K7XP 1.4GHz (Java2D)	SiS 740	17	14

Table 7.2: *FPS is measured against a consistent 8 track, 16 phrase composition.*

work assigned to the GPU is minimal, however AA is a hit to performance on older hardware. We can see that VENUES' performance is CPU bound. Hyper-threaded or multi-core CPUs naturally perform better than single-core CPUs due to the threaded nature of the system.

Comparing alternative designs, rendering the same composition in Java2D on the last machine in Table 7.2 produces 17fps with no AA, and 14fps with AA. Clearly this would have been unacceptable, and the decision to hardware accelerate was correct.

The COLLAPSE algorithm runs in $O(n^2)$ time-complexity. The coe�cient of n in the implementation is 3, however as n is generally $50 < n < 200$ the coe�cient is insignificant. COLLAPSE2 also runs in $O(n^2)$ time-complexity, however it's coe�cient is bound by the width of the canvas (currently 800). This is a significant di�erence, even though it is not perceivable during execution. COLLAPSE and COLLAPSE2's implementations did not receive a large amount of development time. Whilst they are still provably e�cient[6], they are implementation e�cient in neither space nor execution time.

Table 3 shows the results of metrics analysis for VENUES. We can directly assess the non-functional design (4.3) - low coupling is demonstrated by the low AC and EC means, and high cohesion is demonstrated by the low LCOM mean. Of note are the Nested Block Depth and McCabe Cyclomatic Complexity metrics. NBD is self-explanatory, MCCC counts the number of flows through a piece of code (branches and boolean operators) [20]. The high maximums in both `Clock.run()` and `NVI.draw()`, whilst initially alarming, are caused by the Track/Phrase/Figure/Point structure. For each track, each phrase must be checked; for each phrase, each figure must be checked; and so on. Further metrics analysis showed that for the occasions where this structure must be traversed, we have high NBD and MCCC counts. Only once we are at the `Point` level is any work conducted, and so for analysis this traversal can be abstracted and ignored.

MCCC is known to artificially inflate method complexity when switches (or many if/else statements) are used[7]. Discounting switches (such as those used in input handling and network message passing) and abstracting traversal reduces the MCCC metric fairly. We can see the adjusted metrics at the foot of the table. Whilst still undesirable, the di�erences are considerable - particularly for MCCC. This di�erence reduces VENUES from the 'moderate risk' to the 'without much risk' categorization [20]. A better metric to have used is Essential Complexity[8], which

[6]In that there does not exist an algorithm quicker than $O(n^2)$.

[7]A switch with 64 cases would have a complexity of 64, however the switch itself is not a complex function nor is it difficult to understand.

[8]http://xsun.sdct.itl.nist.gov/HHRFdata/Artifacts/ITLdoc/235/chaptera.htm

ignores structured programming primitives (such as switches).

7.2 Usability Evaluation

Venues' GUI has been sectioned into a) the canvas (that which is hand-crafted and rendered in OpenGL), and b) Swing components (that which would be time-ine□cient to hand-craft). An annotated composited screenshot of the canvas is shown in figure 7.2. All bar one of the Swing components are accessible via a single button press (as demonstrated in figure 10). By looking at the interface design rules (4.2), we can evaluate how strictly these have been followed.

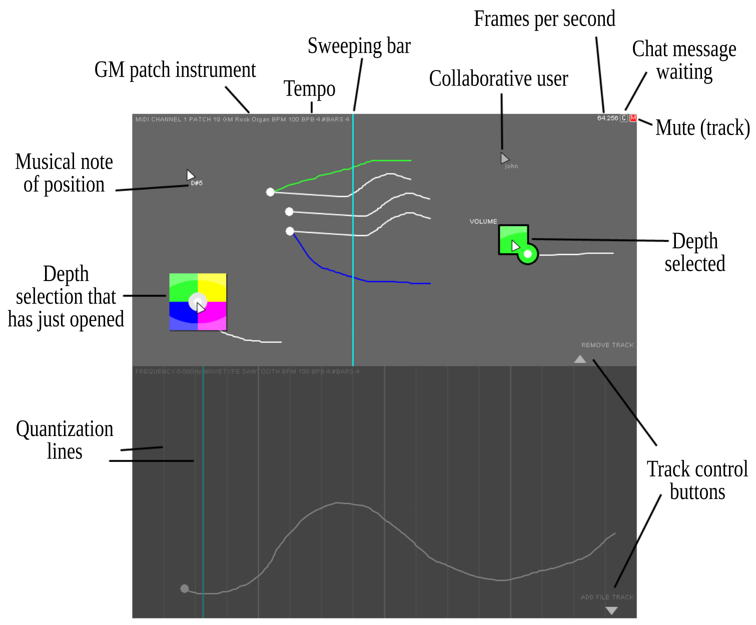


Figure 7.2: *This screenshot has been constructed to show many elements of the canvas GUI.*

All application space (that isn't a Swing component) is canvas space. Canvas items (information, buttons) disappear when in the way, and only appear when necessary. All sequencing tools are available either via the mouse or a keyboard button. Everything necessary to create, modify and delete phrases is on the mouse. All additional functions are not visible from the canvas. This behaviour follows the first four rules exactly.

Screen density is not something that can be defined once[9] - if the user wished to draw phrases in all of the screen space, then the ratio of objects to screen space would be 1:1. However, it could be said that a phrase's object is its starting point,

[9] As is the case with typical windowed applications.

in which case it is impossible that a density of 1 would occur[10]. Ratios closer to 1:10 or 1:20 are common.

Although the rules specify that the interface should be consistent on and off the canvas (which it is), VENUES would feel more closely integrated if all interface elements were on the canvas (fading and sliding into view on invocation). This would increase flow as the user's concentration is not taken from one window to another when performing secondary functions.

User testing brought up a point concerning intuitiveness over figure selection. Whether this point is actually conditioning rather than intuitiveness is moot - the important aspect is that the system provides reasonable results from which the user can learn. It may be wise to add an interactive 'trainer' just for this situation, and to prompt users should they attempt and fail to select a figure a number of times.

Speed based input using a mouse is not viable where accuracy is paramount. It is not fulfilling, it is unpredictable and is ultimately frustrating. The mouse is a device that can generate large acceleration values and is relatively imprecise for work that is typically pen or brush based. Experimenting with a device more like a pen, such as a tablet, may yield better results.

'Flow' and 'play' as concepts have no measure, and as such cannot be quantifiably evaluated. Save from recruiting fifty volunteers for two weeks, the true test of VENUES' interface will come when it is released into the wild. How quickly it can be used to produce, and how enjoyable it is to use, will depend on how well I have understood and applied these concepts.

7.3 Environment Evaluation

Working with Processing was both a joy and a pain. Processing enabled extremely rapid development of GUI elements and input handling. Its OpenGL support, whilst limited, was exactly what the project required, and was almost effortless to integrate.

However, Processing's BETA status is well deserved. VENUES currently has four major[11] and four minor[12] defects that are Processing related and require an update to Processing. I am in communication with Processing's developers to identify and isolate these defects, and with the community to raise awareness and find 'work-arounds', however as of now the problems are unavoidable and out of my hands.

[10]Circles do not tessellate.
[11]OpenGL text rendering; wave generation clicking; pitch buffer cut-off; applet embedding.
[12]Bezier curve fill; inconsistent startup; Ess busy halting; Ess loading external files.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

Some students (and indeed some professionals) believe that Java, and Swing interfaces especially, are somehow inherently slow. Much of this belief can be accredited to the misunderstanding of `ActionListeners`, where people nest work instead of threading it. Whilst it is certainly the case that Java's VM uses more memory than native code, Java is not slow. I believe I have demonstrated that with careful environment selection and e□ective use of threading, Java based systems can be responsive.

OpenGL is an excellent API for rendering cross-platform due to the fact that hardware vendors write platform-specific drivers. Observers of Venues have been surprised at the speed and quality of rendering - this is OpenGL's compliment. JavaSound, being primarily a virtualized solution, has no such hardware-specific, platform-specific drivers, and whilst it is a low-level[1] API, performance is restricted. JavaSound 1.5 implements optimized direct audio access through systems which offer native hardware mixing[2], however it exhibits compatibility issues and falling back to a software mixer is necessary. JavaSound does have excellent MIDI support, though this would be expected for a 24-year-old standard.

OSC is very easy to work with. Once I had overcome the initial learning curve of adjusting to a new protocol (not made any easier by some dubious documentation) I had no problems to speak of. Throughput is excellent, and even though the overheads are a little higher than straight UDP, it was much quicker to work with - a great benefit for a project such as this.

Input passing is the most bandwidth e□cient, highest quality audio producing method of collaboration, where exact audio synchronization[3] and exact audio output[4] are not required. Although bandwidth use is less than any of the proposed alternative designs, and audio remains at CD-quality, significant gains should be possible. It is clear that there is scope for future work in reducing tra□c.

[1]In that it provides direct buffer access for manipulation, as used for wave tracks.
[2]Linux ALSA, Windows DirectSound, Solaris Mixer.
[3]All client's hear output at exactly the same time.
[4]All clients hear the same output.

8.2 Future Work

VENUES was originally envisioned as being embedded in a piece of social software. For this purpose the system runs as an applet. Creating a website around VENUES as a hub for musical collaboration would take significant work, and would be an excellent path upon which to extend requirement four (see 4.1). Many of the currently windowed Swing components could be absorbed into the website. For instance, chat could be dedicated its own space on the canvas page, and server creation/browsing could be handled on page prior to the canvas.

It is *de rigueur* for websites not to create pop-up windows. Certain Swing elements (which currently pop-up) would require hand crafting - synthesizer selection, input selection, *etc.* This would also focus the attention of the user on the canvas, rather than any additional windows or screens, increasing flow within the system. Such complexity, such as that seen in visually dense file choosers, might warrant a hierarchy of generalized canvas interface components.

An alternative direction for VENUES is to become a more fully featured application within the computer music domain. OSC support is a major step, and the ability to act as a host for OSC devices and software would be a major boon - the system would have to respond correctly to OSC query messages for all sequence aecting functions. The fact that the system already generalizes input could make control assignment quite simple, by providing users with a way to map device controls to system inputs. A profiling system for devices and software, coupled with a networked database of profiles (user uploadable) would streamline support.

Throughout the project I have been considering attempting to support VST instruments. With two months to go, I realized that it would not be possible as I was suciently time-pressed with my existing workload. Writing a VSTi host is a significant task made more complex because the SDK is written in C++. A wrapper, jVSTwRapper[5], exists for VSTi development, however host development would require significant porting. Although current MIDI support provides a wealth of instruments through internal patches, SoundBanks, and hardware extension (such as Yamaha's XG series), VSTi support would allow high-fidelity instrument synthesis, both from samples and from simulation.

I am disappointed by how much bandwith VENUES needs. Even if I'd had time to conduct the work necessary to reduce bandwidth usage, packets would still be massively overhead heavy. It should be possible to reduce the combined input messages (keyboard and mouse) further, as table 8.2 shows. Although we could only use 42 bits for input[6], we cannot rely on compression to always reduce this to 32 bits - 4 byte zero-padding means that we might as well increase certain input values to saturate the OSC argument (supporting twelve device input buttons, and using 16 bits for a key code). We can do away with the boolean signifiers altogether - the bits for the device buttons include every possible on/o combination for all buttons, and unicode has an empty value which can signify no key press. The key remains 16bits long for UTF-16. Splitting keyboard and mouse input would actually make the situation worse, as overheads dwarf the message content.

[5] http://jvstwrapper.sourceforge.net/

[6] 3 bit button input, 9 bit axis input * 2, 16 bit key, 5 bit key code.

Input	Bits
Device Button	12
X	10
Y	10
Key	16
Key Code	16
Total	64
+ zero padding	64
+ osc overhead	352
+ udp carrier	576
x 60	34560

A solution is to purposely de-synchronize and delay client input processing, similar to how media is streamed. This allows us to bundle many inputs together, reducing the eect of message overhead. Although UDP allows for packet sizes larger than 512 bytes, certain devices (e.g., routers) between client and server may not. 512 bytes allows for 448 bytes of content, which equates to 56 messages. Each client then only requires 4096 + 274/1024 = 4.2kbps. This is a vast improvement, using 8 times less bandwidth with a latency of just less than a second.

We could exploit the inherent latency between clients and calculate how many input messages can be bundled. A 200ms latency (typical for a 56k modem) would allow 12 input messages to be bundled without any loss in perceived responsiveness. Each packet would then be 1280 bits long, sent five times a second, and each client would require 6.25kbps of bandwidth. This is an excellent result - a 56k modem user could support 4 clients with no perceived latency. This would be a marked improvement on the current implementation, which requires 150kbps to support 4 clients. The assumption that latency decreases with bandwidth is not necessarily true, but is often the case. If we made this assumption, we could implement a dynamic latency-based message bundling system. Users with lower latency (and more bandwidth) would then also perceive no loss in responsiveness.

Bibliography

[1] S. Boyd. Are you ready for social software? 2003. http://www.darwinmag.com/read/050103/social.html.

[2] J. Grudin. Cscw: History and focus. 1994. http://www.ics.uci.edu/~grudin/Papers/IEEE94/IEEEComplastsub.html.

[3] J. Hempel and P. Lehman. The myspace generation. *BusinessWeek*, 2005. http://www.businessweek.com/magazine/content/05_50/b3963001.htm.

[4] M. L. Markus and T. Connolley. Why cscw applications fail: Problems in the adoption of interdependent work tools. In *Proceedings of the Conference of CSCW*, pages 371–380, New York, NY, 1990. ACM.

[5] T. Brinck. Groupware design issues. 1998. http://www.usabilityfirst.com/groupware/design-issues.txt.

[6] John Carmack. Keynote speech. In *Game Developers Conference*. GDC, 2004.

[7] S. Critchley. I want less and im willing to pay for it. 2005. http://oreillynet.com/pub/wlg/6757.

[8] T. Bray. Things that just work: Subethaedit. 2005. http://tbray.org/ongoing/When/200x/2005/03/14/Sub-Etha-Edit.

[9] Golan Levin. Painterly interfaces for audiovisual performance. Master's thesis, Massachusetts Institute of Technology, 2000. http://acg.media.mit.edu/people/golan/thesis/.

[10] M. Slaney. Pattern playback from 1950 to 1995. In *Man and Cybernetics Conference*, Vancouver, Canada, 1995. IEEE Systems.

[11] Scott W. Ambler. User interface design tips, techniques, and principles. 2006. http://www.ambysoft.com/essays/userInterfaceDesign.html.

[12] Larry L. Constantine and Lucy A.D. Lockwood. Usage centered design. 2003. http://www.foruse.com/.

[13] Larry L. Constantine and Lucy A.D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of UsageCentered Design*. Addison-Wesley Professional, 1999.

[14] Randolph G. Bias and Deborah J. Mayhew. *Cost-Justifying Usability*. Morgan Kaufmann, second edition edition, 2005.

[15] Scott W. Ambler. User interface prototyping. 2006. http://www.ambysoft.com/essays/userInterfacePrototyping.html.

[16] Charles M. Kozierok. Theoretical and real-world throughput, and factors affecting network performance. 2005. http://www.tcpipguide.com/free/t_TheoreticalandRealWorldThroughputandFactorsAffecti.htm.

[17] Adrian Freed Matthew Wright and Ali Momeni. Opensound control: State of the art 2003. 2003. http://www.cnmat.berkeley.edu/Research/NIME2003/NIME03_Wright.pdf.

[18] Aaron Krowne. Good hash table primes. 2002. http://planetmath.org/encyclopedia/GoodHashTablePrimes.html.

[19] Robert H. B. Netzer and Barton P. Miller. On the complexity of event ordering for shared-memory parallel program execution. In *International Conference on Parallel Processing*, pages 1193–1197, 1990.

[20] Edmond VanDoren. Cyclomatic complexity. 2000. http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html.

[21] K. H. Burns. History of electronic and computer music. 2000. http://eamusic.dartmouth.edu/~wowem/electronmedia/music/eamhistory.html.

[22] M. Czikszentmihalyi. *Flow.* S.O.S. Free Stock, 1991.

[23] U. Essler. Adoption of groupware. 1998. http://www.dsv.su.se/publikationer/Rapporter-1998.html.

[24] Raymond Lewallen. Patterns and practices. 2005. http://codebetter.com/blogs/raymond.lewallen/archive/2005/07/14/129236.aspx.

[25] Ian Sommerville. *Software Engineering*. Addison Wesley, seventh edition edition, 2004.

Appendix

User Guide

VENUES includes online help, and so the user guide is not reproduced here. Once VENUES is running, press 'F12' for immediate key help, or 'H' for the user guide. The user guide is presented as HTML, and may be viewed externally by accessing `venues/resources/help/$LANGUAGE$/$COUNTRY$/index.html`.

Use Cases

Play Instrument: See 2.
Initiate Collaboration: The user can allow other users to collaborate in their session.
Kick User: The user can remove a particular user from the collaborative session (if the user is the creator of the session).
Join Collaboration: The user can join a collaborative session.
Save Composition: The user can store the composition in some way - this may be saving the file for later loading, or it may be outputting the audio data to a file. In the first case, the user would also have the ability to Load Composition.

Create Phrase: The user can create a musical phrase to be played by the instrument.
Modify Phrase: The user can modify the musical phrase. This is highly dependent on the phrase representation - the representation should try not to restrict any potential for modification.
Delete Phrase: The user can delete a created phrase.
Modify Instrument: The user can change the attributes of the instrument. If the instrument were a tone, an example of this would be changing the frequency of the tone.
Modify Environment: The user can change the attributes of the musical environment, such as the output volume.

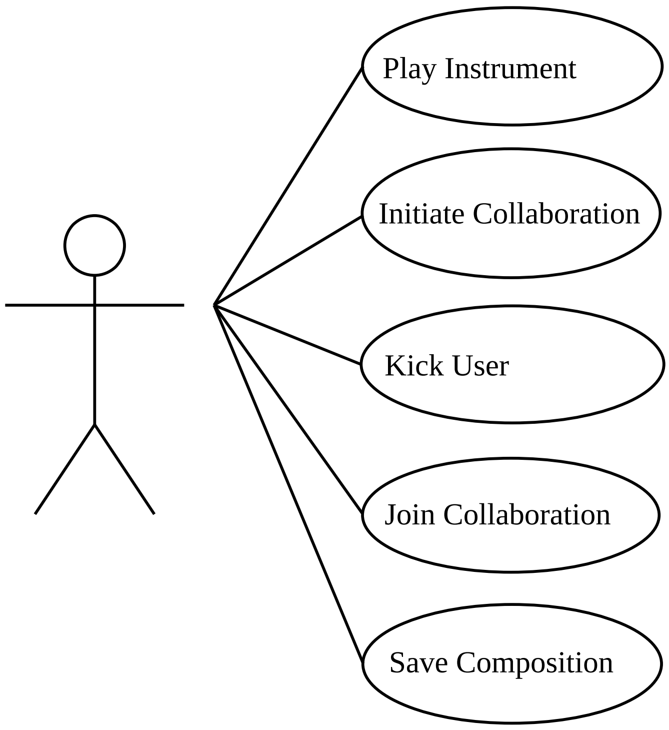


Figure 1: *A Use Case outlining top-level behaviour for a user acting as a server.*

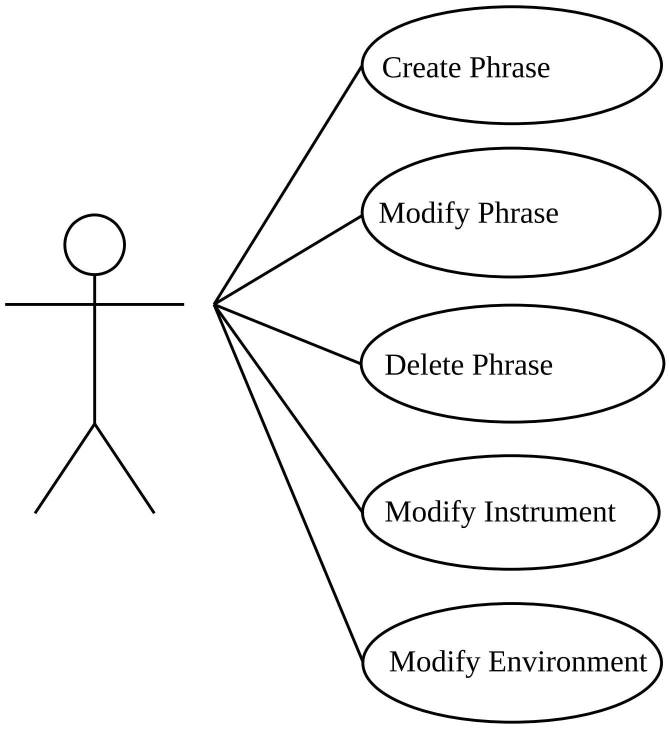


Figure 2: *A Use Case outlining audio affecting commands.*

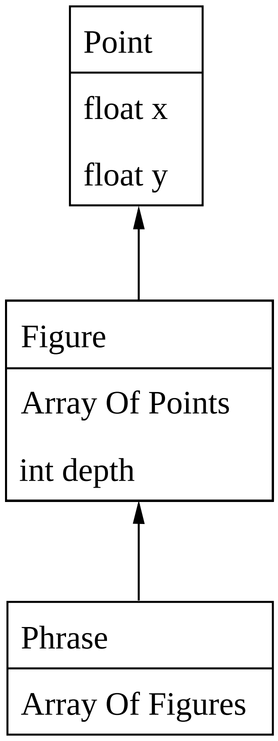


Figure 3: *The internal representation of a phrase.*

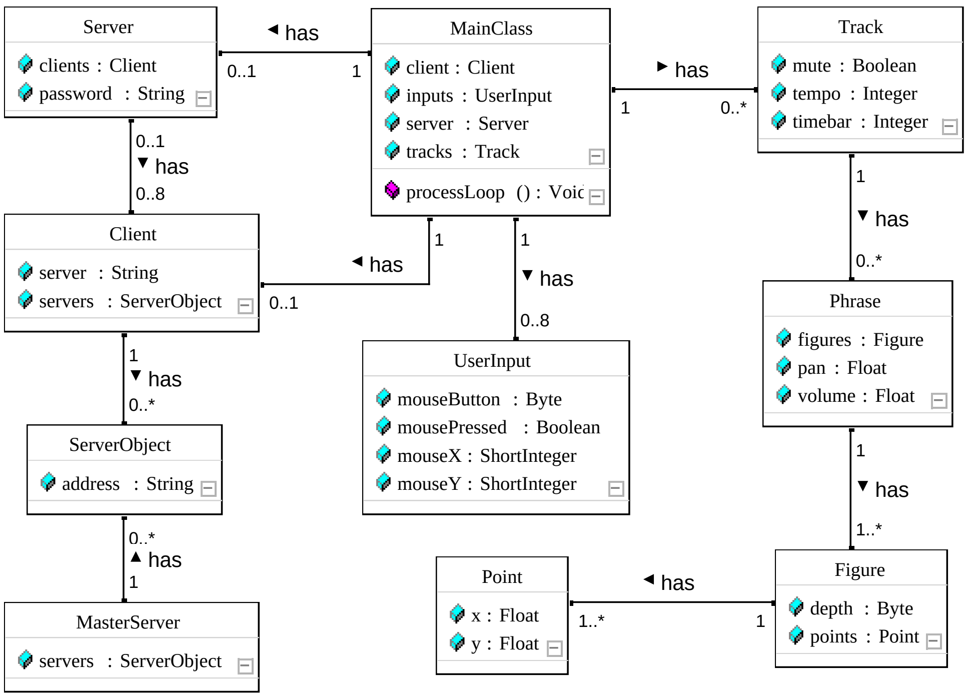


Figure 4: *A class diagram demonstrating the proposed structure of the system.*

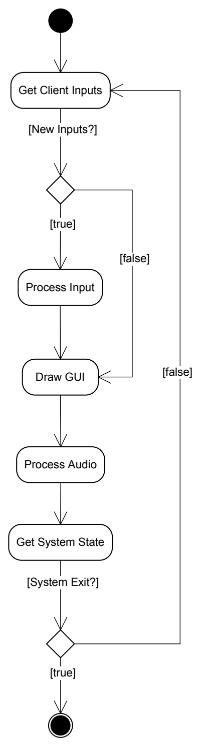


Figure 5: *A statechart demonstrating the main processes that occur in each system tick.*

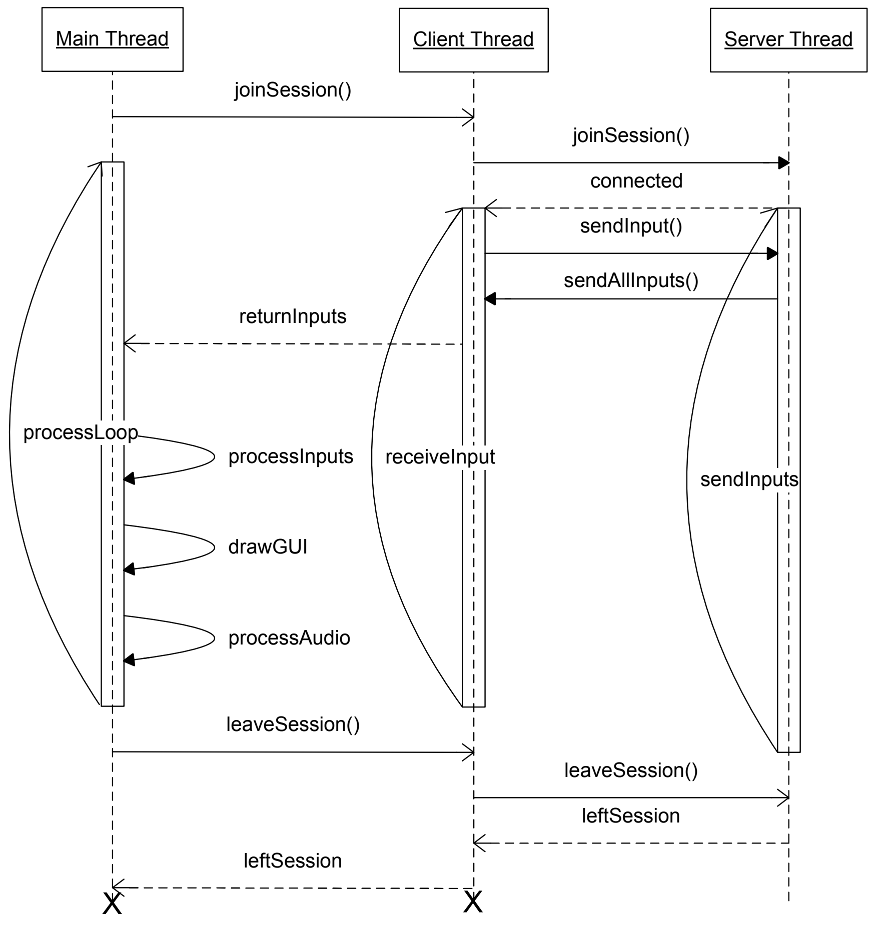


Figure 6: *A sequence diagram demonstrating thread interaction. The server thread could be running locally (in the case of a listening server) or remotely.*

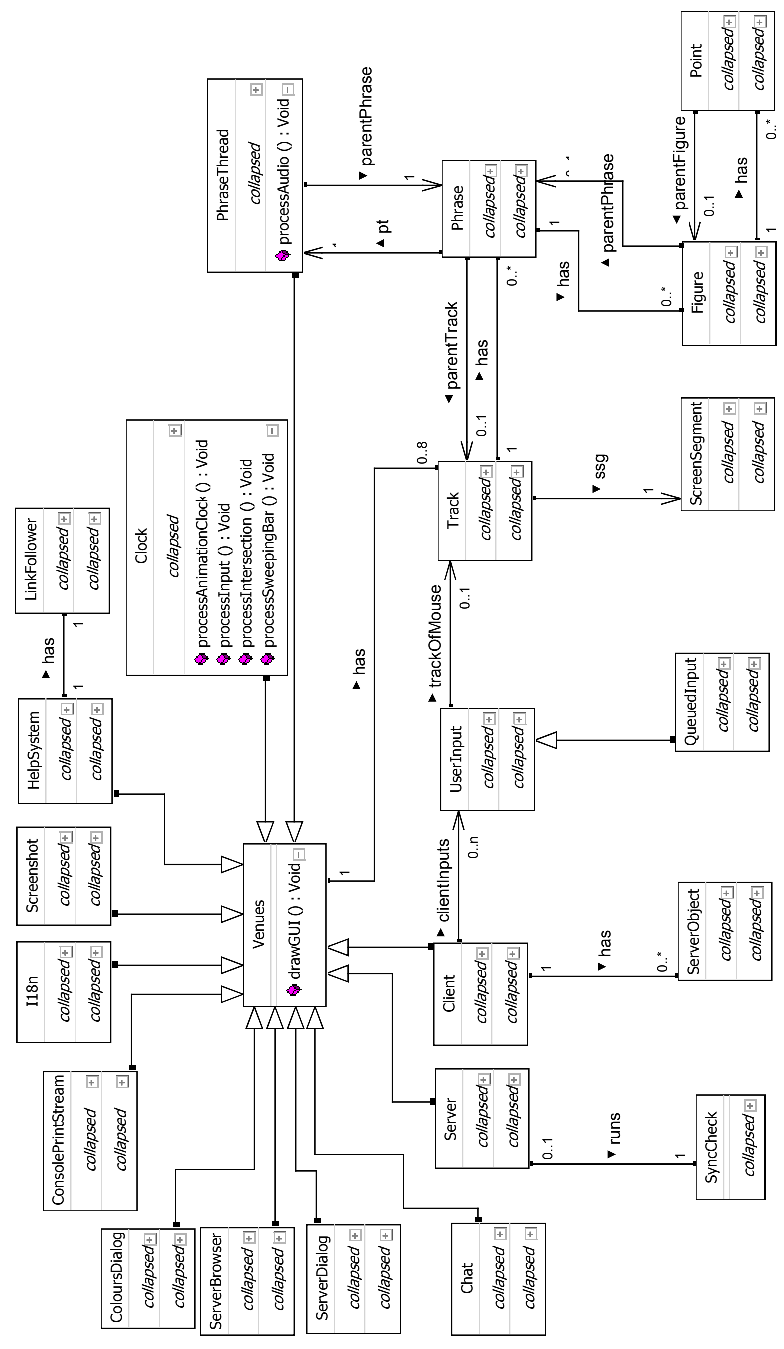


Figure 7: *A class diagram demonstrating the implemented structure.*

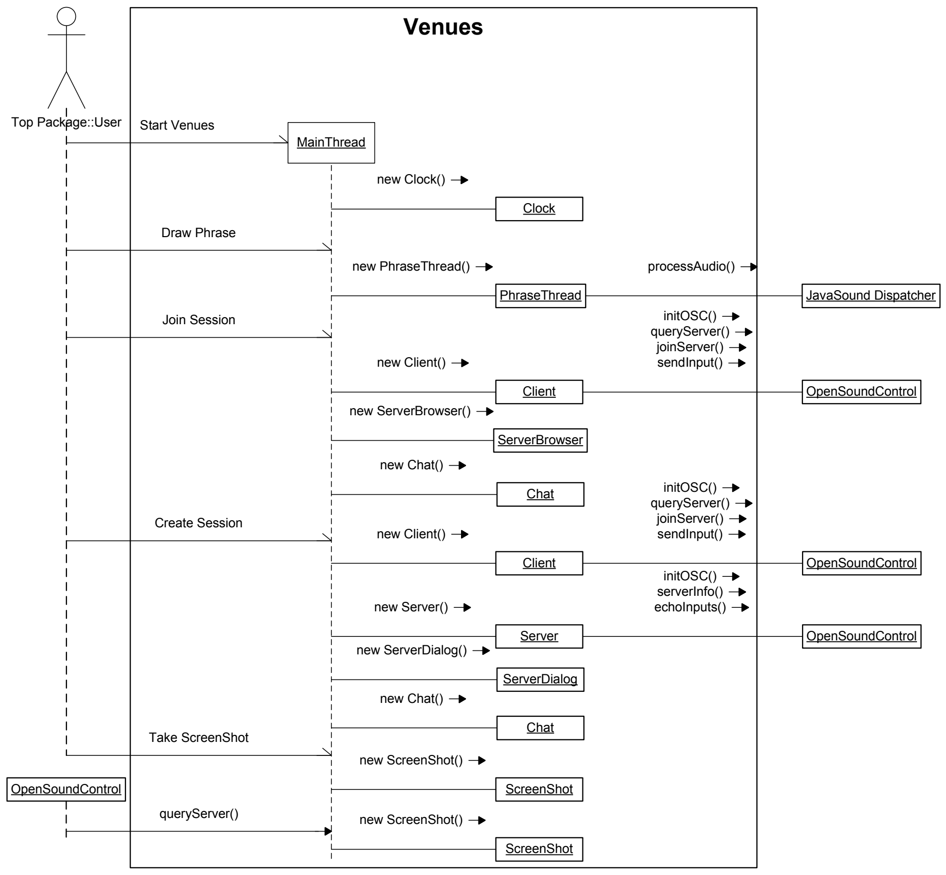


Figure 8: *A sequence diagram demonstrating thread creation and use within Venues and outside.*

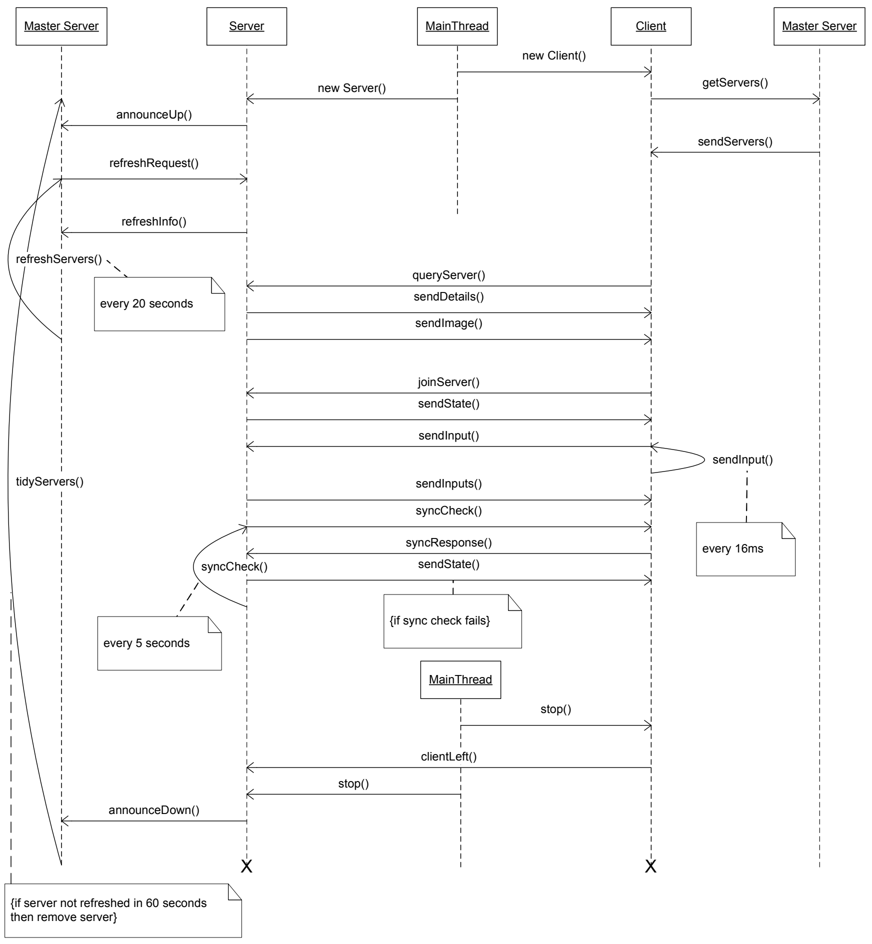


Figure 9: *A sequence diagram demonstrating the major communications between client, server and master server.*

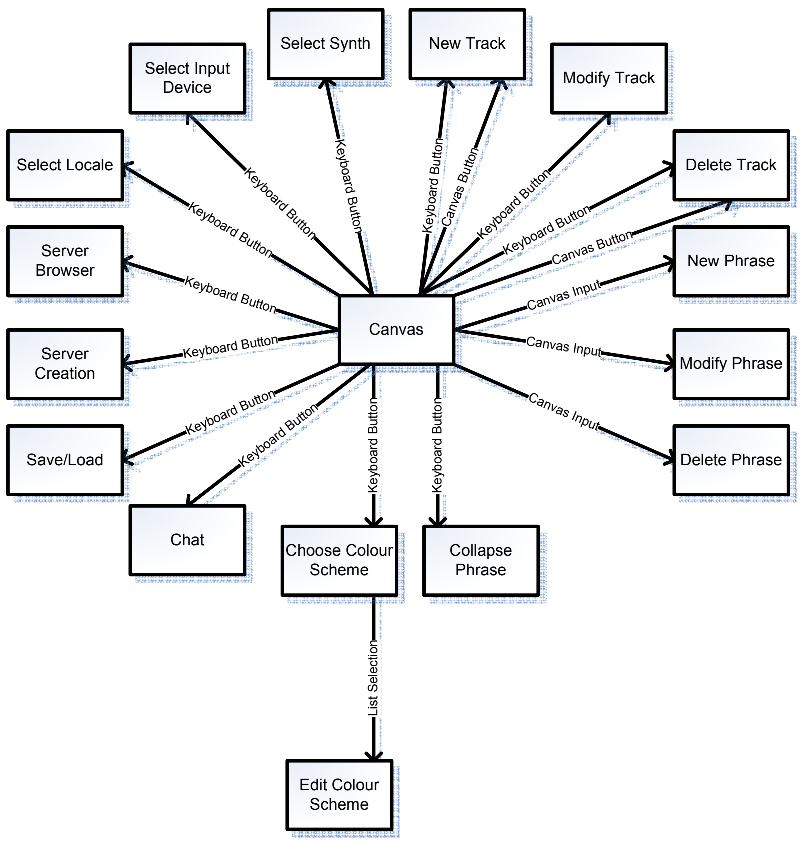


Figure 10: *A flow diagram demonstrating functional navigation. Components to the left of centre are Swing elements, and components to the right are instrument or sequencer functions.*

Previously Failed Test	Previous Outcome	New Outcome	Action Required
Draw a phrase onto another track.	Phrase crosses boundary.	Phrase is bound.	No.
Draw a phrase outside the canvas.	NullPointerException: trackOfInput is null.	Phrase is bound.	No.
Draw a phrase on top of another.	Two phrases are created.	Cannot draw new phrase	No.
Draw a phrase whilst no tracks exist.	A phrase is created in memory but not on the canvas.	No drawing is possible when no tracks exist.	No.
Draw a file track phrase.	Phrase drawing is x & y even though pitch does not vary with y.	Drawing is limited to x only.	No.

Table 1: *A table showing a subset of the regression testing process, concerning the drawing of phrases.*

Test	Result
F1: Show message output.	Toggles message output. Output slides in and out of view.
F2: Show FPS.	Toggles FPS display in top-right.
F3: Show points.	Toggles showing of points on phrases.
F4: Select input.	Dialog appears containing devices. Current device is selected.
F5: Save.	File chooser appears with filter.
F6: Client.	Server browser appears. When in session, client options appear.
F7: Server.	Server dialog appears. When in session, server options appear.
F8: Load.	File chooser appears with filter. If client in session, information dialog appears informing user that client cannot load whilst in session.
F9: Select colour scheme.	Dialog appears with list of schemes. Current scheme is selected.
F10: Select locale.	Dialog appears with list of locales. Current locale is selected.
F11: Take screenshot.	Screenshot appears in 'screenshots' folder date stamped.
F12: Show key help.	Toggles key help. Key help slides in and out of view.

Table 2: *A table showing a subset of the black box testing process, concerning the functions bound to the F-keys.*

Metric	Total	Mean	S.D.	Max.	Location
Lines of Code	9435				
Method Lines of Code	7463				
Nested Block Depth		**1.68**	**1.376**	**12**	Clock.run()
Lack of Cohesion of Methods		0.364	0.406	1.006	
McCabe Cyclomatic Complexity		**5.242**	**18.595**	**214**	NVI.draw()
A[illegible]erent Coupling		2.25	2.773	7	
E[illegible]erent Coupling		4.25	3.345	8	
Instability		0.55	0.36	1	
Adjusted Nested Block Depth		**1.518**	**0.949**	**7**	Server.sState()
Adjusted McCabe Cyclomatic Complexity		**2.279**	**2.895**	**24**	NVI.load()

Table 3: *Metrics. Values in bold are undesirable.*

Run	Time (seconds)	Number of Input Messages
1	600	8108
2	600	2754
3	600	9785
4	600	4822
5	600	5644
6	600	8629
7	600	5218
8	600	6447
9	600	5792
10	600	7145

Table 4: *Simulated Collaboration Results.*

Execution Instructions and Notes

Due to a bug in Processing that stops the launching of applets (that use OpenGL) in a browser, it is not as simple as it should be to launch VENUES. A fix for this bug was released just two weeks before completion, but integrating the fix is not a simple task and updating to the latest version breaks existing functionality.

I will be working to update VENUES to launch from a browser in the future. Unfortunately, for now, it is necessary to jump through one or two hoops. I apologize for this inconvenience. To compensate somewhat there are video and audio files of VENUES in action included on the CD.

As mentioned at the end of the evaluation (7), along with this bug there exists another two (also Processing related) that are immediately apparent and hence are worth mentioning here. The first is that text renders inconsistently, and is di cult to read. The second is that Ess induces clicks at the end of wave generation. This causes wave phrases to sound ‘dirty’ or ‘poppy’. To reduce this e ect the width of bu er quantization has been increased. You may notice that wave phrases are slow to respond to sudden changes in direction - this is the cause.

If there are any questions or problems please do not hesitate to contact me at *james.tompkin@gmail.com*.

On Windows

The easiest way to launch VENUES is to load it from within an IDE, such a Eclipse, and run the NVI class as an applet. Importing the `Venues` folder should be su cient, as project files are included.

On Linux and Mac

Similarly, importing into an IDE is the easiest way. However, there exists a further complication. Library files that load from `bin` on Windows do not load on Linux, and so these must be placed either in JRE/bin or a system bin and added to the CLASSPATH. The cross-platform library files can be found in `bin`, and are stored as backup in `bbin`.

Master Server

The master server can be run by executing `vms.bat` from within `Executable/ Venues Master Server`. The master sever can also be run by executing `VenuesMasterServer.jar` from the same folder, however in this case it will not run with a console and the only output will be to a log. The master server must run on a separate machine. If this is di cult, forgo use of the master server and specify the ip address of the server you wish to connect to. This process is outlined in the *Collaboration* section of the user guide.

Program Listings

I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary.

Signed,

Date:

Venues

core

ChatDialog provides the user with a Swing interface for chat during a collaboration session.

Client handles client-side networking, such as input sending.

Clock keeps track of animation, input and sweeping bar clocks. It also calculates sweeping bar/phrase intersection.

ColoursDialog provides the user with an interface for changing all the colours used in VENUES.

ConsolePrintStream handles redirection of `System.out`. `NVI` handles the display of this stream.

Figure contains a set of `Points` and has a depth.

I18n handles i18n duties such as string fetching from a .properties file. I18n selection is handles by `NVI`.

NVI is the main class and renders the canvas. It contains mutex objects for all threads, acts as a router for network messages, handles input, saving and loading, and does everything that isn't easily placed elsewhere.

Phrase is a phrase on the canvas, consisting of five figures. `Phrase` also contains the two collapse algorithms.

PhraseThread processes and dispatches all audio tasks for its relevant `Phrase`.

Point is a point on the canvas, usually part of a figure but it could also be a starting point of a phrase (stored in `Track`).

QueuedInput is an object representing a client's input. These are added to a concurrent queue in `UserInput`.

ScreenSegment defines the screen area a track occupies, and includes timers to regulate fading.

Screenshot scales and compresses a screenshot in a background thread.

Server handles server-side networking such as echoing, and server tasks such as tallying for various features.

ServerBrowser creates a Swing component that is displayed when a user wishes to connect to a collaboration session. Handles real-time display of incoming servers.

ServerDialog creates a Swing component that is displayed when a user creates a server.

ServerObject is an object internally representing a server, as used by a client.

Tools provides static methods to perform a variety of tasks, including such geometric standards as leftTurn() and properIntersection(). It also provides static `HashMaps` for musical notes, drums, and GM patches.

Track contains all information that defines a track, including a track's `ScreenSegment`. It also includes methods for starting and adding to phrases.

UndoObject is an object containing every piece of information necessary to undo a phrase manipulation.

UserInput contains all user controls, counters and options. It contains an input queue, an undo list, and methods to add and remove from these data structures.

VenuesException provides three custom exception handling methods that present the user with a dialog and also print the error to a file for easy problem solving.

filefilters

FileTypeFilterAIFF filters files by type AIFF in file choosers.

FileTypeFilterAU filters files by type AU in file choosers.

FileTypeFilterMP3 filters files by type MP3 in file choosers.

FileTypeFilterNVI filters files by type NVI in file choosers. NVI is the extension of VENUES' save format.

FileTypeFilterSounds filters files by all supported sound extensions in file choosers.

FileTypeFilterWAV filters files by type WAV in file choosers.

help

HelpSystem constructs a help dialog and renders html pages within using an `HTMLEditorKit`.

LinkFollower handles hyperlink navigation within the help dialog.

tests

TestForBootUp tests for successful boot.

TestForFigureHashCode tests figure hash code validity.

TestForPhraseHashCode tests phrase hash code validity.

TestForPointHashCode tests point hash code validity.

TestForScreenSegment test for screen segment validity.

TestForTrackHashCode tests point hash code validity.

TestForUserInput test for user input validity.

Venues Master Server

MasterServer runs as an application. It creates a master server, schedules server refreshes, tidies servers, and outputs to a log.

ServerObject is an object internally representing a server, as used by the master server.