UNLOCKING THE DECIBEL SCOREPLAYER

Aaron Wyatt
Monash University
Melbourne, Australia
aaron.wyatt@monash.edu

Lindsay Vickery
Edith Cowan University
Perth, Australia
l.vickery@ecu.edu.au

Stuart James
Edith Cowan University
Perth, Australia
s.james@ecu.edu.au

ABSTRACT

This paper discusses recent developments in the Decibel ScorePlayer project, including the introduction of a canvas scoring mode, python ScorePlayer externals, and enhancements to the ScoreCreator application. Firstly, the canvas scoring mode of the Decibel ScorePlayer app allows for other applications, such as Max, to send drawing commands to the ScorePlayer via OSC. Several examples of implementations of generative and animated notation scores are discussed and evaluated. An object model has been developed allowing for the creation of hierarchies of drawn elements. The object model defines a framework of commands that can be used to create and control these objects, and supporting examples describe the way in which scores can be developed to take advantage of this new scoring mode. Secondly, a python *scoreplayer-external* library has been developed, defining two python classes: *scorePlayerExternal* that makes a connection to the iPad, opening a UDP listening socket and letting the iPad know which port to send its replies to, and *scoreObject* which is responsible for creating and drawing objects populated on the canvas display window of the Decibel ScorePlayer. It acts as a wrapper to the raw OSC commands so that programming can be done using object-oriented paradigms. Thirdly, the ScoreCreator, an application developed for Mac OSX for automating the process of making scores for the Decibel ScorePlayer, has been expanded allowing for the defining of a range of score types and functionalities.

1. INTRODUCTION

Upon its public release in 2013 [1], the Decibel Scoreplayer app offered a limited number of score presentation formats. At this early stage of development, it was considered a priority to implement robust networking capabilities which served the demands of live performances involving the synchronisation of multiple devices. The ScorePlayer app continued to be developed over some years, based on an iterative methodology, and tested in rehearsals and performances with Decibel, an Australian new music ensemble. The ScorePlayer's preeminent mode of functionality was the scrolling score format in which an image file could be synchronously moved from left to right over a prescribed period of time on multiple devices. In 2012, while the ScorePlayer was still being used by the Decibel ensemble, a ScoreCreator app was developed for Mac OSX to simplify the creation of scores utilising the scrolling score mode of the ScorePlayer, a user-friendly solution for those less tech savvy. As development of the app continued, further formats and functionalities were added [2, 3, 4, 5, 6, 7], however many of these enhancements were inaccessible to the public: and while some features were periodically made available in public releases of the app, they were not well documented and composers often could only access them by reverse engineering existing scores. This paper documents a range of developments intended both to expand the functionalities of the ScorePlayer app and to open the platform for experimentation to interested parties. These developments include expansion and support for non-linear score formats in the ScoreCreator app, implementation of a canvas scoring mode for the sandboxing of non-standard notations, and a Python library to allow for the control of the ScorePlayer via the command line, in addition to OSC-based visual programming.

2. THE CANVAS SCORING MODE

The Decibel ScorePlayer was initially conceived as a modular platform [1], assigning control of the user interface and networking functions to the main player window while drawing tasks were enacted by rendering modules that could be alternated to allow for the implementation of different score types. New score modalities were implemented, including scrolling in all planar directions, nonlinear, two dimensional [1], rhizomatic [5] and generative [4] scores, but with standalone or very restricted user control of their functionality. While it would perhaps be ideal for the platform to be open allowing users to create and compile their own rendering modules, iOS (like most tablet computing environments) forbids Apps that are distributed via the App Store to permit libraries to be dynamically loaded [8]. We attempted to overcome this limitation by allowing composers to directly control the drawing surface in the canvas scoring mode (publicly released in 2018) from an external application using commands sent via the Open Sound Control (OSC) protocol [10, 2, 7]. This arrangement permits composers to develop, prototype, and even distribute new score paradigms and idiosyncratic realtime score generation without the need for any code to be accepted into the App Store. The public release of canvas mode implements six different types of objects:

- Layers - consist of a simple rectangular region on the canvas which is empty when first created, into which a flat colour or an image can be loaded;
- Scrollers - that allow for a larger image to be scrolled horizontally at a set rate through a fixed viewing window;
- Text - using iPad supported fonts;
- Glyphs - a special case of the text object used as a convenient shortcut to display symbols from the bravura music font (Steinberg Media Technologies GmbH. 2018);
- Staves – creating a five-line staff of a defined size;
- Lines - drawn within the coordinate space of their parent object. (These are the only type of object that cannot be used as a container for other objects.)

An external device can send drawing commands via OSC to any networked canvas score. Each connected score, for example on a set of iPads, has an OSC address starting `/Renderer/Command`, followed by the name of the object to be manipulated, and the command to be sent to that object: for example, to add a layer to the score's initial canvas, the command "`/Renderer/Command/canvas/addLayer`" would be sent. This command is followed by arguments defining the name of the layer to be created, the part for it to be assigned to (or 0 for the layer to be placed on all parts) and then the objects' display coordinate data. A full list of drawing commands can be found at http://www.psi-borg.org/canvas.html.

Any application that can send and receive OSC packets can be used to control the ScorePlayer in canvas scoring mode. Examples of patches written in Max were included in a previous paper [6] and can also function within Ableton's Live using MaxforLive.

A library has also been developed to allow externals to be quickly and easily written in python, and this has been released under the LGPL via the Python Package Index, or PyPI for short [10]. Both of these solutions use Bonjour [11] for service discovery, and are able to find any iPads running the ScorePlayer on the local network, as long as multicast traffic is not blocked. In cases where such traffic is blocked, manual connection is still possible, but an understanding of IP addressing is required.

Canvas scores, like previous ScorePlayer files, consist of a standard zip file with its extension changed to dsz [1] which must be bundled with all of the image resources needed by the score, in jpg or png format, as well as xml files [12] that define the score's metadata and settings for any additional options. This file is then imported into the ScorePlayer via Apple AirDrop, iTunes' file sharing feature or downloaded directly from a web server using either a URL or a QR code that represents a URL. The main xml file that defines the score is named opus.xml [1]. For canvas mode, the <scoretype> tag in this file is set to "Canvas" and duration may be set either to zero or any number of seconds. If zero (or negative), the ScorePlayer only displays the Reset button to the user, and the navigation bar and status bar remain visible. If set to a positive value, then the Play button is also displayed, and pressing it sends the usual /Control/Play command over the network, starts the clock, and hides the navigation bar until the specified time limit is reached. The obvious advantage of this is that it makes a larger drawing surface available, even if a composer has no intention of making use of the timing functions of the ScorePlayer. (The size of the canvas is 1024x768 when in landscape mode, and the status and navigation bar clip reduces this height by 70 pixels). The opus.xml file may also point to an XML preferences file. This file (typically "prefs.xml") is currently used to define the number of available parts that can be drawn to. Like previous ScorePlayer files, parts are accessed by swiping up or down on the screen. In a canvas score this file is also used to define the <clearonreset> setting, which determines whether the canvas is cleared by pressing the Reset button in the ScorePlayer. If undefined this feature is by default set to yes.

2.1 Affordances of Canvas Mode

The canvas scoring mode provides for masking and changes in opacity in addition to a number of forms of object animation. Since new objects are placed in the foreground, existing objects can be modified by placing masks in front of them. The opacity of all objects can be controlled using the `setOpacity` command. Making a parent layer translucent will also affect the opacity of any child layers. Alternatively, the opacity can be animated using the `fade` command, which additionally allows for the duration of the opacity change to be specified. David Kim-Boyle discusses using both masking and opacity generated in Max as compositional tools in his *tunings* (2006) for Cello and Computer [13].

"Flipcard" style animation, of which André Vida's *Vidatone* series is an example [14, 15] can be achieved by repeatedly using a `loadImage` command to load a sequence of evolving images into a layer. All objects can also be moved continuously or discontinuously in any planar direction on the screen. The `move` command sets the position of an object and animates the transition over a period of time specified in seconds by a duration argument, while the `setPosition` command can be used to change the position of an object instantly. The scroller object is most efficient for moving large images. It allows for a larger image to be scrolled horizontally at a set rate through a fixed viewing window. The viewing window itself is defined by the usual position and size coordinates that are passed to the object on creation. A few additional parameters, also passed at creation, are used to define the behaviour of the scroller while animated. The `scrollerWidth` parameter determines the width of the layer to be scrolled through the viewing window, while the `scrollerSpeed` defines the rate at which this occurs in pixels per second. Setting a negative value for `scrollerSpeed` causes the scroller content to move backwards, to the right of screen. These values can be changed at any time, and the scroller can be set in motion or stopped using the start and stop commands.

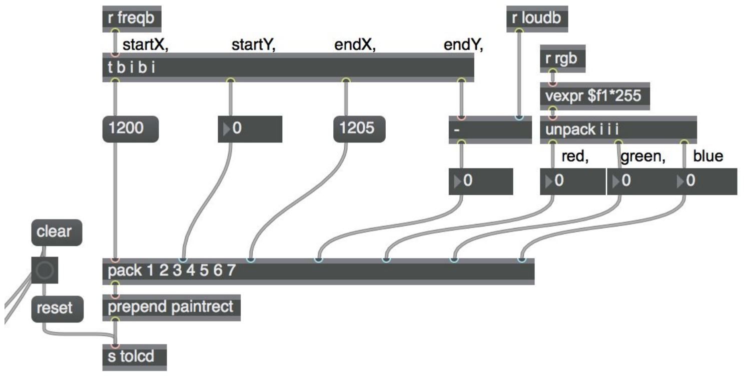


Figure 1. Score generation in the Max-only version of the *Lyrebird Environment Player* using the LCD object.

3. EXAMPLES OF IMPLEMENTATIONS OF WORKS IN CANVAS MODE

Two generative scores by Lindsay Vickery, *Lyrebird Environment Player* (2014) and *The Semantics of Redaction* (2014) originally implemented in Max were adapted to utilise the canvas scoring mode. In these works audio files, of field recordings and speech respectively, are analysed in order to generate a score [4, 5]. In both works the intent was to emphasise the capability of generating a score permitting interaction by performers with recent and locally recorded sonic environments or a current topical speech in "near real-time". The data for generating the score in these works is derived from the `sigmund~` object in *Lyrebird*. Mapping of the audio analysis to score elements is shown in Table 1.

The works presented different challenges for implementation in the canvas scoring mode. In the case *of Lyrebird*, the principal parameter of interest was the network-rate/draw-rate of the OSC connection. In the original Max-only version, *Lyrebird* drew coloured rectangles to an LCD object at an average rate of between 21 and 22 messages per second These `paintrect` messages, as shown in Figure 1, are driven by the output of a `sigmund~` object with an FFT analysis window size of 2048 audio samples.

Updating the score to support the canvas scoring mode on the iPad required translation of the LCD drawing commands. Whilst the original version used a single draw command, the new implementation requires four messages: an `addLayer` command, followed by a `setColour`, `move`, and `remove` command. As the sequence of these commands is crucial in procedural programming they are sequentially executed by a trigger object (Figure 2). The sending of the remove command is delayed by 12 seconds to allow the move animation for each rectangle to complete. Each rectangle is named according to a variable number '`line%s`' with a replaceable suffix of any number between 0 and 500, as in this case with a draw rate of 21 or 22 rectangles per second, up to 259 separate rectangles might appear at any time over a scroll time of 12 seconds.

Spectral Descriptor	Graphical specification
Frequency	rectangle vertical position
Amplitude	rectangle size
Brightness	rectangle hue
Noisiness	rectangle saturation
Bark Scale deviation	rectangle luminance

Table 1. Spectral Descriptor to Graphical specification mapping in *Lyrebird.*

To measure the responsiveness of this scoring mode, the Max application was stress tested whilst overdrive mode was enabled. The timing and quantity of packets sent from Max to the iPad were analysed during these tests. Between 84 and 88 renderer commands were consistently generated every second over a total time of 60 seconds. The networked iPad also comfortably drew at this rate. At this stage it is assumed by this that as a scoring paradigm, this method would prove to be consistently reliable.

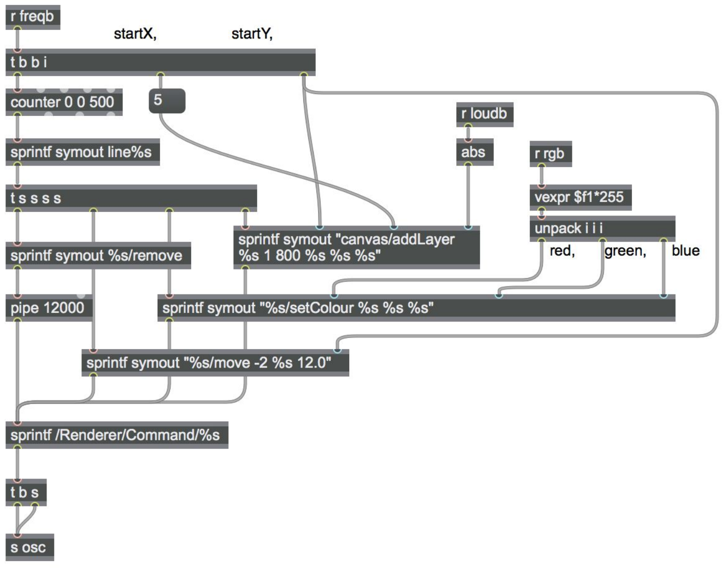


Figure 2. *Lyrebird Environment Player* canvas score mode version.

There are some minor differences between both rendered versions of the score, LCD versus the iPad canvas scoring mode in the ScorePlayer app (Figure 3), however as the sizing of the canvas window in the LCD and on the iPad are known, and both systems use standard RGB color space to define their colors, on the whole the scores translate reasonably closely with the added benefit of the smooth graphical rendering capabilities of the core animation graphics rendering and animation infrastructure available on iOS devices.

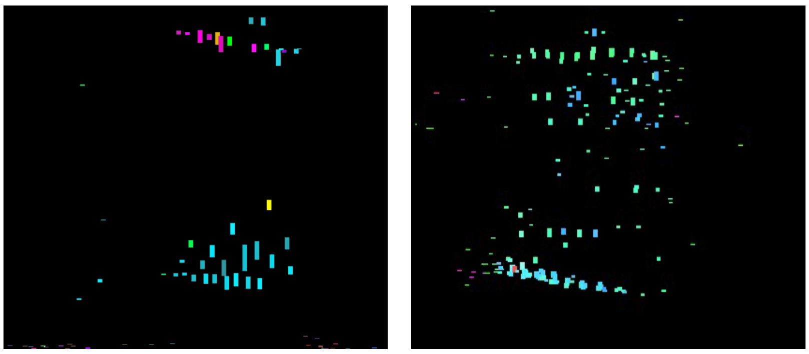


Figure 3. Comparison between *Lyrebird* ScorePlayer canvas score (left) and LCD (right) score.

In the case of *The Semantics of Redaction* five categories of score elements needed to be drawn using separate processes: score setup, noteheads, note stems, graphical symbols and text (indicating the beginning of each section). In each section the audio data is mapped in five varied "modes" (Table 2).

The score setup includes a black line in the centre of the score that functions as a "beam" for all of the generated noteheads/stems and a "playhead" giving the position at which a note is executed. These scored objects are created as part of the initialisation routine as children of a layer object using the commands:

```
/Renderer/Command/canvas/addLayer sor 0 0 0
1024 698
/Renderer/Command/sor/setColour 255 255 255
/Renderer/Command/sor/addLayer beam 0 0 348
1024 6
/Renderer/Command/beam/setColour 0 0 0
/Renderer/Command/sor/addLayer playhead 0 70 0
4 197
/Renderer/Command/playhead/setColour 255 255 0
```

The data for generating the score in these works is derived from the `analyzer~` object in *The Semantics of Redaction*. Mapping of the audio analysis to score elements is shown in Table 3.

The rate of draw commands in *The Semantics of Redaction* are considerably slower than in *Lyrebird*, however, there are a greater variety of draw commands in the work, ranging from rectangular colored noteheads with thin black stems connected to the central beam, to graphical notations, and text. Each of these processes draws on the same process as seen in Figure 2, except the `line%s/setColour` command is substituted for a `%s/loadImage` command when loading png image files. Text includes a number of further variables, and therefore requires more than four commands: `addText`, `setFontSize`, `setColour`, `setFont`, `setText`, `move`, and `remove`.

The iPad copes well with the realtime drawing commands over the network, provided the UDP OSC packets are received. Again because of discrepancy between the iPad screen/LCD dimensions, the score is more horizontally compressed in the Canvas mode implementation (Figure 4).

Spectral Descriptor	Generated graphical specification
Frequency	notehead vertical position and hue
Amplitude	notehead size
Brightness	notehead colour saturation
Noisiness	notehead colour luminance

Table 3. Notehead and graphical symbol drawing behaviour and audio playback behaviour in *The Semantics of Redaction*.

3.1 Flipcard Style Animation

An experiment to reproduce animated notation using "Flipcard" style animation was conducted with Ryan Ross Smith's work *Study No. 55* (2016). The work was a challenging candidate, requiring colour full-screen frames (1920 × 1080px on the iPad) with an average size of 180KB. 500 frames of the 13-minute work (2.6% of the total) were rendered and bundled into a .dsz file of 46MB. A Max patch was made using the canvas `loadImage` command to draw and then remove successive image layers (Figure 5). The optimal removal rate for images was shown to be 1.2 times the draw rate. The draw command rate was tested across a dedicated wireless network between a 2018 Macbook Pro running High Sierra and an

	Note head drawing behaviour				graphical symbols				Audio Playback			
Mode	triggered by detected "attacks"	noteheads extended horizontally	continuous steps drawn between noteheads	interrupted by black rectangles (redactangles)	graphical symbols are superimposed	draws black rectangles (redactangles)	noteheads become increasingly transparent	playback audio normal	playback audio muted	interrupted by redaction "bleeps"	audio superimposed at varying playback speeds.	continually pitch-shifted downwards
Opening/Body	x							x				
Commentary	x	x			x			x				
Interlude	x		x		x				x			
Redactio	x	,		x	x	x				x	x	
Closing	x				x		x					x

Table 2. Notehead and graphical symbol drawing b and audio playback behaviour in *The Semantics of Redaction*.

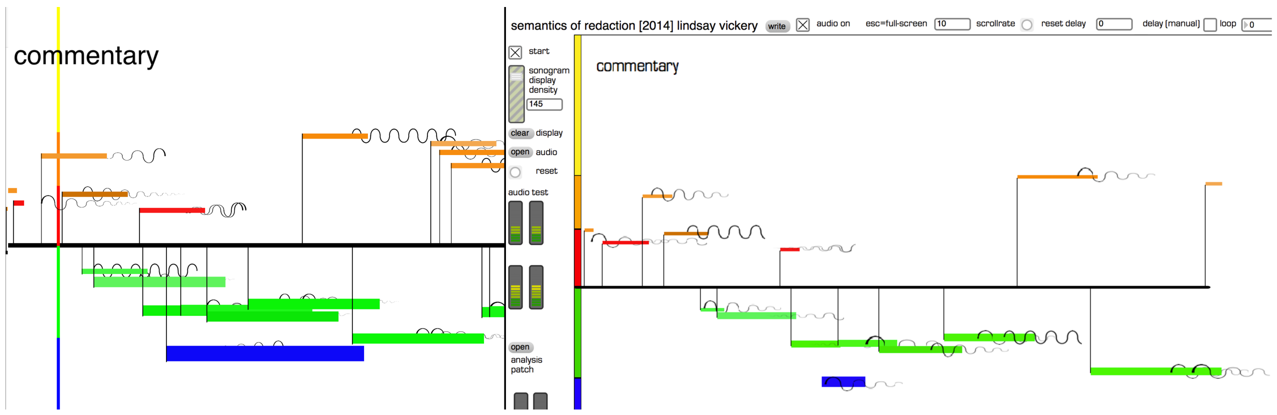
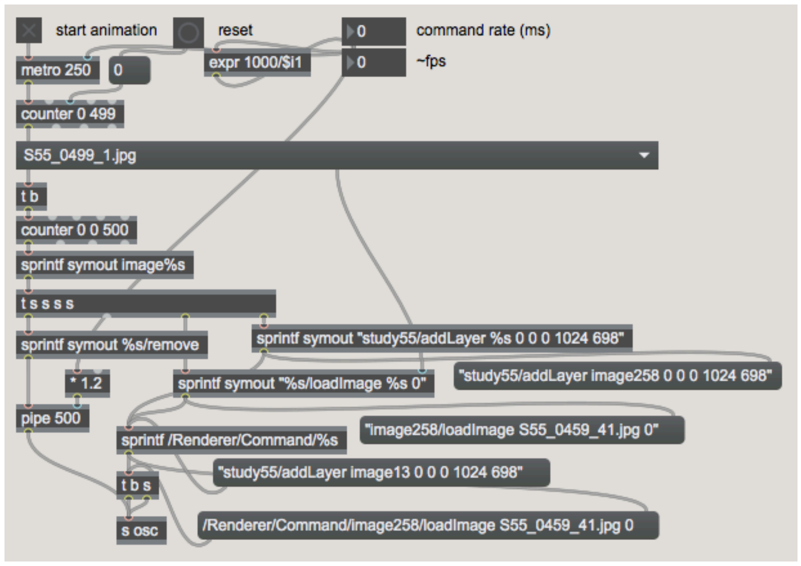


Figure 4. Comparison between *The Semantics of Redaction* ScorePlayer canvas score (left) and Max/LCD (right) score.

iPad Pro running OS12.2. Draw commands were successfully communicated at a rate of 15ms (approximately 66FPS) and demonstrated to be stable at a rate of 41ms (approximately 24FPS) the frame rate of Smith's original video. A second test using an iPad 2 crashed the Scoreplayer application at rates of 4FPS.

Although the results were encouraging, they suggest that flipcard style animation in canvas scoring mode is more suited to the implementation of smaller, shorter frame animations, perhaps in combination with other, less size and CPU intensive, processes. (A dsz bundle combining all 18 thousand frames of *Study No. 55* is estimated to be a more unwieldy 880MB in size).



```
metro 250
0
expr 1000/$i1
0
0
counter 0 499
S55_0499_1.jpg
t b
counter 0 0 500
sprintf symout image%s
t s s s s
sprintf symout %s/remove
sprintf symout "study55/addLayer %s 0 0 0 1024 698"
"study55/addLayer image258 0 0 0 1024 698"
* 1.2
sprintf symout "%s/loadImage %s 0"
pipe 500
sprintf /Renderer/Command/%s
"image258/loadImage S55_0459_41.jpg 0"
"study55/addLayer image13 0 0 0 1024 698"
t b s
s osc
/Renderer/Command/image258/loadImage S55_0459_41.jpg 0
```

Figure 5. Max patch using the canvas loadImage command to draw and then remove successive image layers for "flipcard" style animation.

4. PYTHON SCOREPLAYER EXTERNALS

First released in 1991 and created by Guido van Rossum, Python is generally recognized as an interpreted, high-level and general-purpose programming language. There are several advantages to using Python as an alternative to Max for driving the Decibel ScorePlayer's canvas mode features on the iPad, particularly due to it being open-source and based on a community development model. As a result, it is freely available on most platforms, and has a convenient distribution mechanism for libraries in the Python Package Index, or PyPI [10]. It also has a plethora of existing libraries, including ones specific to service discovery [17] and OSC [18], which greatly simplified the development of our own scripts. These scripts have since been developed into the *scoreplayer-external* library which is now available on PyPI following its initial release and presentation at the Australasian Computer Music Conference in December 2018. The Python *scoreplayer-external* library defines two python classes: *scorePlayerExternala* and *scoreObject*.

The first class is used to make a connection to the iPad, opening a UDP listening socket and letting the iPad know which port to send its replies to. The second object is designed to represent the objects of music notation that populate the canvas display window. The *scorePlayerExternal* class also encapsulates a Bonjour service browser that can be used to find an iPad to connect to. This design allows for the process of finding and connecting to an iPad to be done in very few lines of code, allowing the composer the freedom to focus almost entirely on drawing commands.

4.1 Connecting to the Decibel ScorePlayer using the scorePlayerExternal class

The following example code demonstrates how a link can be established between a Python script and the ScorePlayer app using the *scorePlayerExternal* class.

```
  #!/usr/bin/env python
  from scoreplayer_external import scorePlayerEx-
ternal
  from threading import Event
  finished = Event()external
= scorePlayerExternal() ex-
ternal.selectServer()canvas
=    external.connect(onCon-
nect)
  finished.wait()     exter-
nal.shutdown()
  external.shutdown()
```

Once all of the necessary libraries have been imported, a new *scorePlayerExternal* object can be created (and in this case is assigned the name external). Running the `selectServer()` method on this object prints the list of available servers to the console, and prompts the user to either select one or refresh the list to check for any new servers that might have become available. Once the user makes a choice, the `connect(connectionHandler)` method can be called. This sends a registration message to the iPad server, letting it know of our listening port, and on return of a confirmation message it runs the function passed as the `connectionHandler` argument. This function (not shown here) should contain the drawing commands that the composer wants to use to render their score. The connect function also returns an instance of a *scoreObject* that is a reference to our canvas. It is by calling the methods of this returned object that new objects can be added to the score.

4.2 Creating Python-driven scores using the scoreObject class

The *scoreObject* class encapsulates the sending of the raw OSC commands used to manipulate objects on the ScorePlayer canvas, and allows these remote objects to be treated as if they were Python objects. Instances of the *scoreObject* class should not be created directly by the user, but should instead be stored and used when returned from the various methods that add objects to the canvas. As outlined earlier, for example, the initial connect method returns a *scoreObject* that represents the canvas. Other objects of varying types can be added to this by calling the `addLayer`, `addScroller`, `addText`, `addGlyph`, `addStave`, or `addLine` methods of the canvas. This is demonstrated in the following code example (which assumes that the initial canvas object has been stored to the variable canvas):

```
  rdef onConnect():
  canvas.clear()
  stave = canvas.addLayer('stave', 0, 0, 0,
1000, 800)
  stave.loadImage('Stave.png', 1)
  score = canvas.addScroller('score', 0, 90, 0,
1000, 800, 0, 60) score.loadImage('Score.png',
1)
  score.start()
  global finished
  finished = True
```

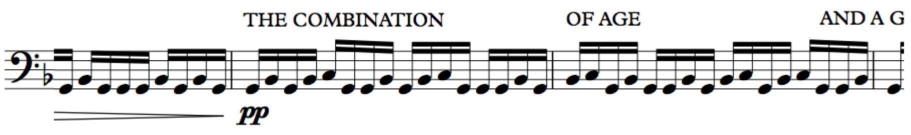


Figure 6. A layer object on the canvas that contains a sublayer.

The addLayer method is used first to add a layer named ‘stave’ to the canvas. A separate scroller layer is added named ‘score’, and the script then proceeds to scroll the score. (Figure 6).

There are some methods that are common to all objects. `setColour` can be used to change the colour of any object. (Whether this command applies to the foreground or background colour is dependent on the type of object.) All objects apart from the canvas can also have their opacity set, and all bar the canvas or line objects can be moved within the coordinate space of their parent object, either instantly or animated over a specified time frame using the `move` command. Additionally, any object (except for lines) can be used to hold other objects, allowing for the creation of complex hierarchies of objects that can be manipulated as one with relatively few commands. Score objects also have commands that are specific to their type listed in the Figure 7 tree diagram.

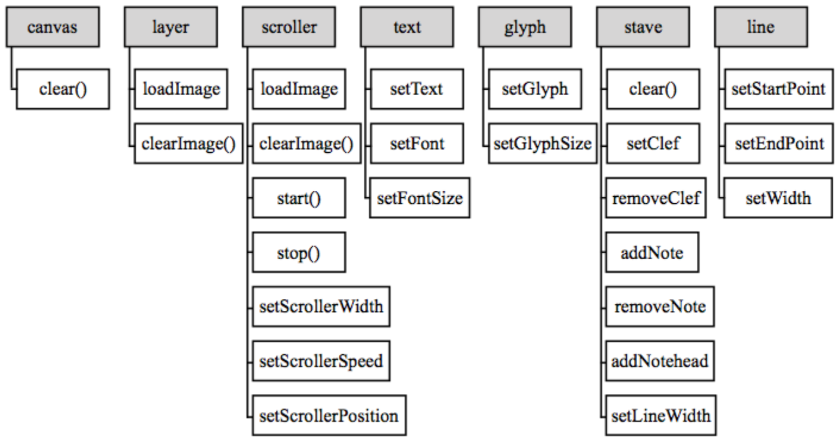


Figure 7. A tree diagram outlining the commands specific to canvas score object type.

Using the type specific commands of the stave object, for example, Common Practice notation can be added to a score. The following code example adds a stave to the canvas and then draws a sequence of clefs and a randomly populated sequence of stemless note-heads, the output of which can be seen below it in Figure 8.

```
  def onConnect():
  canvas.clear()
  stave = canvas.addStave('stave', 0, 10, 100,
1100, 72, 2)
  pitches = ('C4', 'C#4', 'C#+4', 'D4', 'E4',
'F4', 'F#4', 'G-4', 'A#+4', 'B4')
stave.setClef('treble', 40)
  stave.move(-1000, 100, 10)
  for i in range(0, 20):
  note = stave.addNotehead (pitches[random.rand-
int(0, 9)], (i*50)+100, 1)
```

```
global finished
finished = True
```

Since notes are drawn on the stave using pitch notation, a clef needs to be defined before any notes can be added. To avoid this, note drawing could also be achieved, albeit in a much more manual way, with either the `addGlyph` command or the `addText` command, with the font set to the Bravura font [16]. These commands also allow the composer to override the default geometries used by the stave object. With all of these varied objects and commands, composers can create a hybrid system of notation combining not only Common Practice and graphic notation, but also animated notation paradigms using both the scroller object and the more general animated movement and fade commands.

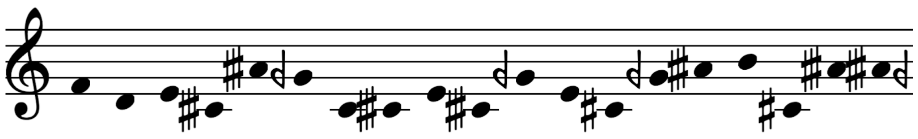


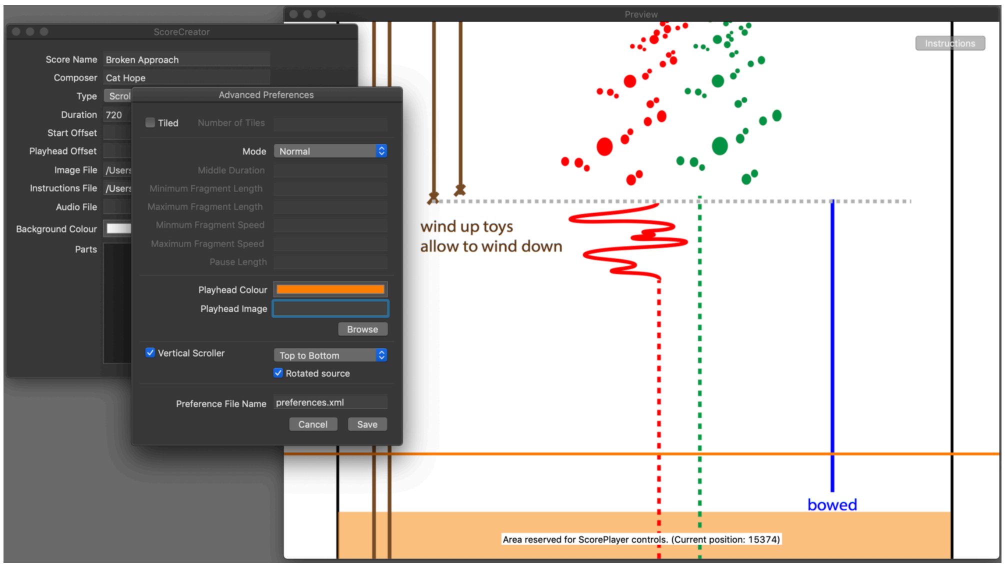
Figure 8. A python script responsible for creating a stave, generating a sequence of 20 pitches randomly gathered from a user-specified list, and scrolling the system off screen over 10 seconds.

5. ENHANCEMENTS TO THE SCORECREATOR APPLICATION

Early versions of the ScoreCreator consisted simply of a form window that allowed for the entry of the data that would populate the relevant xml tags of the score's opus file. When the "create score" button was clicked, this data was written out to the xml file, and this was zipped together with any image resources that the composer had specified. While much more convenient for a composer than editing a raw xml file, this still did not present much of an idea of what the score would look like in the player during the creation process. Refining the settings of a score could potentially require a lot of movement back and forth between the two apps. The ScoreCreator was also only capable of creating scrolling scores, and didn't offer access to some of the more advanced features of these.

ScoreCreator version 0.5.1 addresses both of these concerns. The creation of Talking board [1] style scores has now been added as an option, and work is currently underway to add Slideshow, Flash card and Canvas mode scores. The interface has also been improved to allow for more advanced preferences to be selected. To take scrolling scores as an example, it is now possible to customize the playhead, either by changing its colour or using an image in place of the usual line, and it is possible to make vertical as well as the standard horizontal scrolling scores (Figure 9). For vertical scores, the score can either be set to scroll from top to bottom or bottom to top. Most importantly, the new version has a preview window, so that composers will be able to view what their score will look like once imported into the player. While this preview isn't animated, the composer can still scroll through the image to be used, seeing how it will be affected by any scaling, and how much of it will be visible on the iPad's screen at any one time in both the landscape and portrait orientations of the device. When any settings that change the display of the score are altered in the main window, this is reflected instantly in the preview window.

Additionally, the ScoreCreator now handles images in a more intelligent way. It has always been possible to use the app to create a scrolling score from a series of equally sized tiles, but now it can search for and detect if such an image set exists upon selection of the first image. After prompting the user about the discovery, it can then automatically adjust the settings if desired. (It can also adjust settings in the other direction if the required number of images are not found.) On creation of a score, it will also check to see if any images are too large for display on the iPad, and will offer to scale or tile the original image as appropriate. (It won't do this if the user has manually done their



ScoreCreator

Score Name	Broken Approach
Composer	Cat Hope
Type	Scrol
Duration	720
Start Offset	
Playhead Offset	
Image File	/Users
Instructions File	/Users
Audio File	
Background Colour	
Parts	

Advanced Preferences

☐ Tiled	Number of Tiles
Mode	Normal
Middle Duration	
Minimum Fragment Length	
Maximum Fragment Length	
Minmum Fragment Speed	
Maximum Fragment Speed	
Pause Length	
Playhead Colour	🟧
Playhead Image	
✓ Vertical Scroller	Top to Bottom
	✓ Rotated source
Preference File Name	preferences.xml

Area reserved for ScorePlayer controls. (Current position: 15374)

Figure 9. The updated ScoreCreator, showing the advanced preferences window for scrolling scores. The preview window can also be prominently seen, showing how the score will look as a vertical scroller.

own tiling, and will trust them to know what they are doing in that case.) The result of all of these changes is that the composer doesn't need to know quite so much about how the ScorePlayer handles their images internally, and can trust that their score will appear in the player exactly as it does in the preview window, leaving them to focus on the task of composition.

6. CONCLUSIONS

The advancements in the Decibel ScorePlayer project described above are aimed at opening the platform for experimentation by composers and others. It is hoped enhancements in the ScoreCreator app will allow for engagement with a range of novel score formats, and make them accessible to novice users with minimal programming experience. The development of the canvas mode now supports drawing commands sufficient to afford significant experimentation with new score formats and presentation methodologies. The implementations of scores discussed in this paper indicate that the canvas mode will support quite draw command intensive scores in a robust manner. The development of a Python library for the creation of external control apps forms something of a bridge between these two approaches, facilitating control of the canvas scoring mode and other ScorePlayer functions with minimal programming experience via a freely available and accessible language.

7. REFERENCES

[1] A. Wyatt, and C. Hope, "Animated Music Notation on the iPad (Or: Music stands just weren't designed to support laptops)," in Proceedings of the 2013 International Computer Music Conference (ICMC2013), Perth, 2013, pp. 201-207.

[2] C. Hope, A.Wyatt, and L. Vickery, "The Decibel ScorePlayer: New Developments and Improved Functionality, " in Proceedings of the 2015 International Computer Music Conference (ICMC2015),Denton, 2015, pp. 314-317.

[3] C. Hope, A. Wyatt, and L. Vickery, "The Decibel ScorePlayer - A digital tool for reading graphic notation" in Proceedings of the International Conference on New Tools for Music Notation and Representation (TENOR'15), Paris, 2015, pp. 59-70.

[4] L. Vickery and S. James, "Tectonic: a networked, generative and interactive, conducting environment for iPad," in Proceedings of the 2016 International Computer Music Conference (ICMC2016), Utrecht, 2016, pp. 542-547.

[5] L. Vickery, "Rhizomatic approaches to screen-based music notation" in Proceedings of the 2016 New Interfaces for Musical Expression Conference (NIME2016), Brisbane, 2016, pp. 394-400.

[6] S. James, C. Hope, L. Vickery, A. Wyatt, B. Carey, X. Fu, and G. Hadju, "Establishing connectivity between the existing networked music notation packages Quintet.net, Decibel ScorePlayer and MaxScore," in Proceedings of the International Conference on New Tools for Music Notation and Representation (TENOR'17). A Coruña, 2017, pp. 171-183.

[7] A. Wyatt, L. Vickery, and S. James, "The Canvas Mode: Rapid Prototyping For The Decibel Scoreplayer," in Proceedings of the Australasian Computer Music Conference 2018 (ACMC2018), Perth, 2018, pp. 99-108.

[8] D. DeVille, "Dynamic linking on iOS." Last modified April 2, 2014. http://ddeville.me/2014/04/dynamic-linking

[9] M. Wright, "The Open Sound Control 1.0 Specification." Last modified March 26, 2002. http://opensoundcontrol.org/spec-1_0

[10] Python Software Foundation. "Python Package Index." Accessed October 26, 2018. https://pypi.org

[11] Apple Inc. "Bonjour Overview." Last Modified April 23 2018. https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/NetServices/Introduction.html.

[12] WC3. "Extensible Markup Language (XML) 1.0 (Fifth Edition)." Last modified February 7, 2013. https://www.w3.org/TR/xml/.

[13] D. Kim-Boyle, "Real time generation of open form scores." Proceedings of Digital Art Weeks, Zurich. 2006, pp. 3-14.

[14] A. Vida, Accessed October 26, 2018. http://www.andrevida.com

[15] R. Smith, A practical and theoretical framework for understanding contemporary animated scoring practices. PhD Diss. Rensselaer Polytechnic Institute, 2016.

[16] Steinberg Media Technologies GmbH. "SMuFL: Standard Music Font Layout." Accessed October 26, 2018. https://www.smufl.org/fonts/

[17] Python Software Foundation. "Zeroconf 0.23.0." Accessed June 1, 2019. https://pypi.org/project/zeroconf/

[18] Python Software Foundation. "Python-OSC 1.7.0." Accessed June 1, 2019. https://pypi.org/project/python-osc/