Tweakable

Julian Woodward
Visual Systems Ltd
42 Veda road London
jw@vsys.co.uk

ABSTRACT

In this paper, I describe some of the core features of *Tweakable*, a new interactive algorithmic music system. Live examples can be found on the web at https://tweakable.org/.

1. INTRODUCTION

Tweakable is a web-based visual programming system (VPS) designed to lower the barrier of entry for creating algorithmic music, while still offering vast possibilities for experimentation.

From a palette of components, users can quickly design an algorithmic system, and expose parameters through a user interface that enable the algorithm to be 'tweaked' in real time.

2. INCLUSIVITY

A key goal of *Tweakable* is put into anyone's hands the tools to experiment, play, discover and share ideas about how mathematics, sound and music can interrelate, acting as a catalyst for learning. Web-based, it is easy for users to share projects without worrying about missing dependencies. Much of the software already available for making algorithmic music requires specialist knowledge and sometimes complex software / hardware setup. Hugely successful, *Max MSP*[1] "provides an open and experimental environment for artistic expression, (however) it also restricts access to many users through a steep learning curve and requirement for low-level DSP knowledge". [2]

As a painter/programmer I aim to enhance engagement and user experience by infusing the user interface with a strong visual quality.

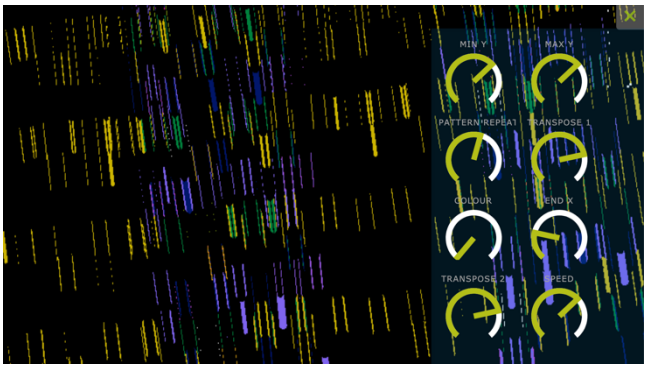

Figure 1. Visualization example - Tweakable Musical Tapestry

Web Audio Conference WAC-2019, December 4–6, 2019, Trondheim, Norway.

3. COMPONENTS

Available components fall into three categories: Data input / flow, Sequencing, and Audio. In addition, it is possible to create custom modules to encapsulate and reuse behavior.

3.1 Data input / flow

Handling user input and controlling the way information flows through the system.

3.1.1 Data input

Control input components (Slider, Knob etc) enable user interaction and can be connected to parameters to enable the algorithm to be tweaked.

Some components automatically detect touch, enabling the x and y coordinates of the touch to be wired to parameters.

Midi input & output allows midi to be received from and sent to other applications.

3.1.2 Data flow

Data flow components include state machine, switch, counter, splitter, and filter, providing ways to control the way data flows through the system. State Machine can be connected to multiple components' parameters to create cross-component presets which can be selected or performed as a set by transitioning between each setting at specified intervals.

3.2 Sequencing

3.2.1 Sequence source

A sequence is a list of instructions that can be scheduled to generate events or control automation.

Graph components generate sequences from a mathematical function, or instead receive user input from touch or mouse.

Grid components generate sequences from user interaction with a piano-roll style grid.

Script components generate sequences using a custom notation language. Letters of the alphabet correspond to notes of the scale, lower / uppercase changes corresponding to lowering / raising the pitch / octave.

Rhythm components generate sequences from note / step parameters using the Euclidean algorithm [3].

3.2.2 Transformation, modulation, composition

Sequences can be passed through various transformations: reverse, invert, crop, transpose, scale, stretch, constrain etc. which can be applied conditionally or periodically, parameters changing in real time. Users can also code their own transformations in JavaScript.

Complex patterns can be generated by modulating one sequence with another. Sequences can be combined in series using letter notation (for example ABCBA describes a Rondo form). A

sequence can be exploded by the Harmonize component, which generates multiple contrapuntal parts taking a chord progression as input.

3.3 Audio

Sequences are rendered into audible sound by connecting them to a scheduler, and from the scheduler to a pre-built instrument (sampler or synth), or combinations of lower-level components such as oscillators, envelopes and LFO's, whose parameters can also be automated using sequences. Audio effects are implemented as objects that can be connected to audio outputs.

3.3.1 Automation

Individual parameters of audio components can be automated by sequences either by connecting the scheduled sequence directly to the parameter input, or by using Controller components which listen for events on specified controller numbers on the instrument.

3.4 Custom modules

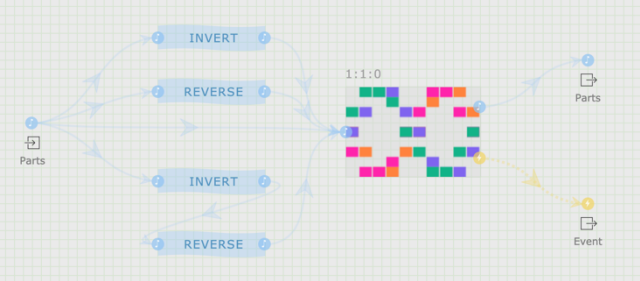

Figure 2. An example of a module that takes a sequence as input and outputs four sequences: the original, inverse, reverse, and inverse-reverse

Behavior can be encapsulated by creating modules, which act as containers for other components and can expose input and output parameters. See Figure 2 for an example of a module.

Custom modules can be exported and imported into other projects, and shared between users.

4. APPLICATION ARCHITECTURE

A key design concern was to ensure a clean separation between the User Interface ('UI' / 'front end') and the conceptual 'back end' functionality so that music and sound can continue to play when switching between views, and to minimize dependency on the front-end framework. Maintaining independence from the UI, audio output can be prioritized where necessary, and animation disabled where it becomes a problem for low performance devices.

Components are implemented as *TypeScript* classes that talk to each other using the Reactive Extensions for JavaScript library (*RxJS*). Subscriptions are created between them when connections are made. Component classes are data-bound to a corresponding UI component (implemented with *Angular*).

4.1 Aesthetic choices

The UI has been designed to minimize clutter and be as clear as possible. In the node-graph view, components each have a strong visual identity which hopefully helps with the understanding of their function.

A built-in palette system enables users to change the overall color palette. Rather than choosing colors separately for each element, palette indices are assigned, helping to ensure an overall visual consistency by limiting the palette of colors in use.

4.2 Mobile devices

Users can design their UI to be responsive by defining both narrow screen and wide screen views. *Tweakable* automatically selects the version which matches the end-user's device width. All components work with touch devices.

4.3 Visualization

Visualization is employed to aid user understanding of the patterns controlling the audio output. Each UI component implements an 'animate' function. Rather than running multiple requestAnimationFrame (RAF) loops, a single RAF loop runs, calling each component's 'animate' function in turn. Visualizations use HTML5 canvas. Audio visualizations are provided for waveform 'oscilloscope' and audio envelope & partials. Sequences are visualized by the Piano Roll component. Custom visualizations can be coded in JavaScript.

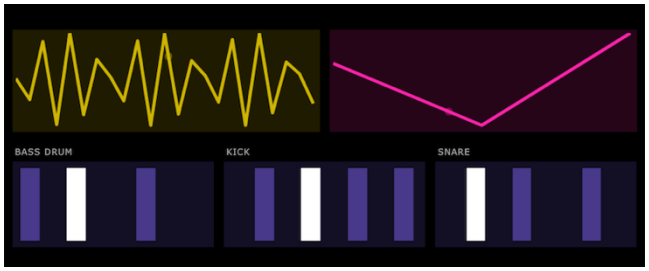

Figure 3. Tweakable algorithmic music example.

Melody and bassline and can be controlled by touch-dragging up/down (pitch) and left/right (shape), rhythm can be modified and rotated by dragging horizontally / vertically

5. ACKNOWLEDGMENTS

Acknowledgements are due to many open source software libraries, notably:

- RxJS (https://rxjs.dev) handles all data flow between components
- Tone.js (https://tonejs.github.io/) for audio instruments and effects (see 3.3), as well as envelope, filter and LFO are implemented as components
- jsPlumb (https://github.com/jsplumb/jsplumb) - the ability to drag connections between nodes in the 'node graph' GUI
- Codemirror (https://codemirror.net/) - a coding window enabling live coding in JavaScript or the in-built custom music notation language
- Angular (https://angular.io/) - the front-end framework that provides data-binding capability

6. REFERENCES

[1] Cycling '74. *Max MSP*.[Software] Cycling '74. Available from: https://cycling74.com. 2019

[2] Bullock, Jamie. 2018. *Designing interfaces for musical algorithms*. The Oxford Handbook of Algorithmic Music, The Oxford University Press, 2018

[3] Toussaint, Godfried. *The Euclidean Algorithm Generates Traditional Musical Rhythms*. School of Computer Science, McGill University Montreal