Manhattan: Serious Games for Serious Music

MET2016 Submission for Paper

Dr Chris Nash
Department of Computer Science and Creative Technologies,
Faculty of Environment and Technology,
University of the West of England,
Coldharbour Lane, Bristol, UK
BS16 1QY

Email: chris.nash@uwe.ac.uk
Phone: +44 (0)7962 180080

Keywords

music education, serious games, end-user programming, motivation, notation, composition

Abstract

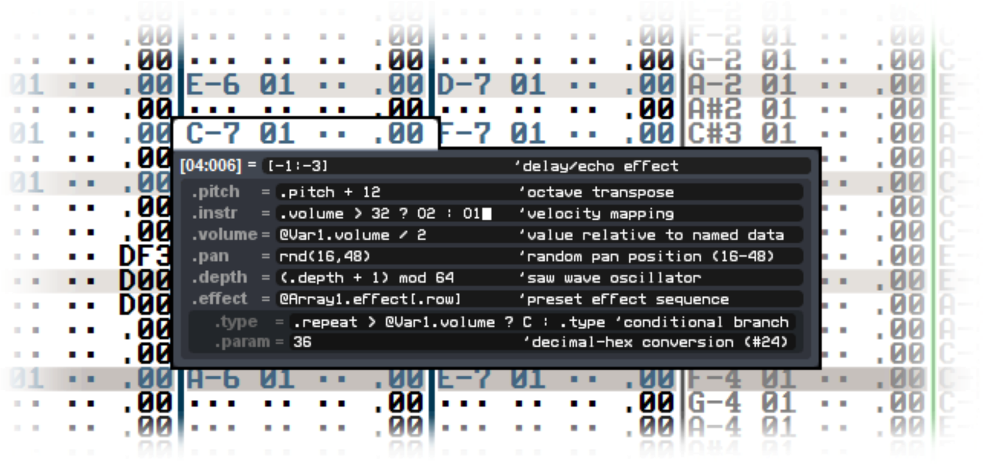
This paper details a digital platform designed for digital creativity, learning, and engagement with new concepts and aesthetics in both music and coding. An open online ecosystem is outlined, connecting users for the purposes of collating, sharing, supporting, collaborating, and competing with works combining music and code – collectively designed to tackle both intrinsic and extrinsic motivational issues in both the learning of music and programming.

Developing on observed practices and aesthetics in digital music subcultures, composing and coding through a unified digital notation is fashioned as a 'serious game'; composers compete against themselves or others, in works that combine creativity and virtuosity in music and code. Mechanisms for scoring pieces with respect to both musical aesthetic (e.g. user reviews) and technique (e.g. code complexity) are considered, proposing a metric that rewards conciseness, in order to encourage abstraction and pattern recognition in both music and code.

The platform develops on *Manhattan* (Nash, 2014), an end-user programming environment for music composition, based on a text-based pattern sequencer, using a grid/cell formula metaphor to integrate programming functionality. A rapid edit-audition cycle improves the liveness of notation interaction, facilitating learning and experimentation. Unlike other music programming tools, it prioritises the visibility and editing of musical data, rather than code; as in spreadsheets, users are able to engage with code expressions as much (or as little) as they wish, offering a lower entry threshold and shallower learning curve for programming – plus a continuum of musical applications from the fixed, structured notation of music (e.g. MIDI sequencing) to more dynamic and experimental elements, such as minimal (process-based) and algorithmic composition techniques. The paper provides musical examples and discusses existing use of the technologies in teaching, with reference to lessons and workshops based on the software, as well as current and future directions for the research.

The Manhattan project explores the use of technology in developing digital pedagogies that bridge learning in both music and programming, through the development of a connected end-user programming environment for music composition. The approach draws upon analogous concepts, issues and practices in each creative domain, including: the mediation of interaction through notation (the scripting of future behaviour); high learning thresholds for entry (notably literacy); extended learning trajectories that start with low-level knowledge (notes, syntax) and progress to more abstract concepts (harmony, processes, structure); and concomitant challenges in maintaining learner motivation.

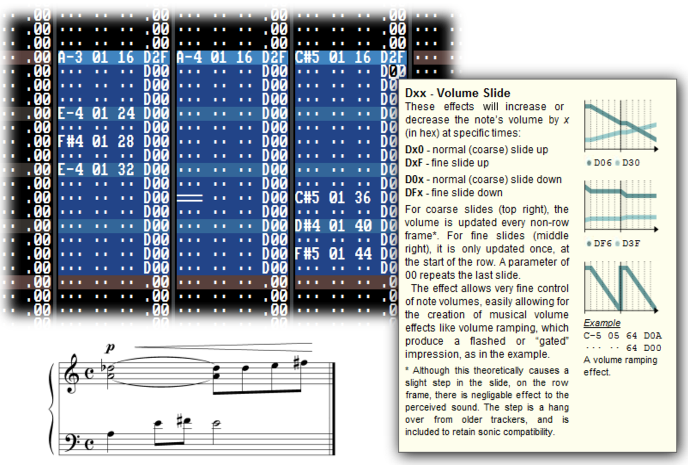
The aim of the research is to develop an integrated platform for musical creativity and learning, through engagement with programming concepts: to teach programming concepts in a conventional music context; to explore musical concepts and aesthetics through programming; to develop critical, abstract, and analytic thinking skills in music and code; and, to explore models of virtuosity as a mechanism for intrinsic and extrinsic motivation.



```
[04:006] = [-1:-3]                          'delay/echo effect
 .pitch  = .pitch + 12                      'octave transpose
 .instr  = .volume > 32 ? 02 : 01           'velocity mapping
 .volume = @Var1.volume / 2                 'value relative to named data
 .pan    = rnd(16,48)                       'random pan position (16-48)
 .depth  = (.depth + 1) mod 64              'saw wave oscillator
 .effect = @Array1.effect[.row]             'preset effect sequence
   .type  = .repeat > @Var1.volume ? C : .type 'conditional branch
   .param = 36                              'decimal-hex conversion (#24)
```

Figure 1 – Code Formulae for Music, in Manhattan

The research is based on the development of *Manhattan* (Figure 1; Nash, 2014), a software ecosystem that integrates a music editor (sequencer) with end-user programming functionality (code formulas), enabling a scalable level of engagement with programming and a continuum from traditional composing styles (e.g. MIDI arrangement), to the incorporation of simple generative, dynamic, or reactive elements, through to process-based (e.g. minimal) and algorithmic composition. In contrast to other music programming tools, the user interface prioritises the visibility of the music (data) over code, which is integrated using a spreadsheet formula metaphor. In this model, expressions are situated (hidden) in the cells of the sequencer pattern (a musical grid with time progressing downwards) that define how events are created and manipulated, using a mixture of simple maths, functions, and references to other cells (including past, present, or future notated events). In computing terms, this model of interaction combines elements of *declarative* programming (coding in terms of *what* is output) and more traditional *imperative* styles (ordered sequences of instructions that define *how* it is output), which respectively improve the accessibility of the programming environment (cf. spreadsheets), while also enabling key concepts from common programming practice to be explored and applied (e.g. iteration, conditional statements, control flow, functions, concurrency, variables, arrays, pointers).



Dxx - Volume Slide

These effects will increase or decrease the note's volume by *x* (in hex) at specific times:

Dx0 - normal (coarse) slide up
DxF - fine slide up

D0x - normal (coarse) slide down
DFx - fine slide down

For coarse slides (top right), the volume is updated every non-row frame*. For fine slides (middle right), it is only updated once, at the start of the row. A parameter of 00 repeats the last slide.

The effect allows very fine control of note volumes, easily allowing for the creation of musical volume effects like volume ramping, which produce a flashed or "gated" impression, as in the example.

* Although this theoretically causes a slight step in the slide, on the row frame, there is negligable effect to the perceived sound. The step is a hang over from older trackers, and is included to retain sonic compatibility.

Example

```
C-5 05 64 D0A
··· ·· 64 D00
```

A volume ramping effect.

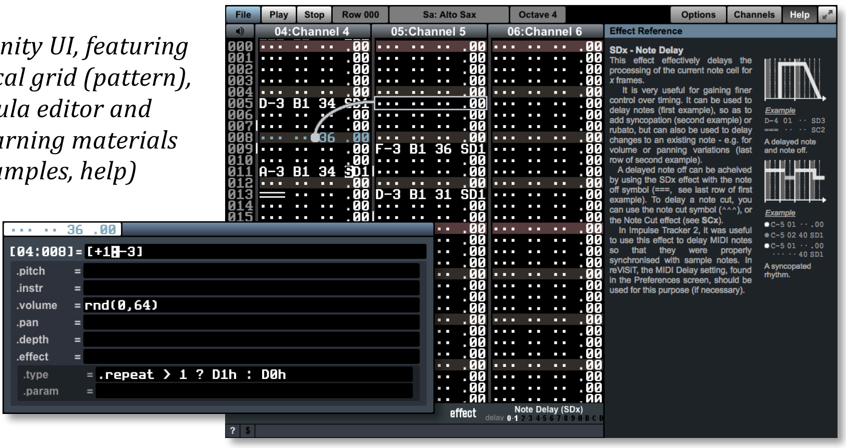
Figure 2 – Score vs. Tracker Notation (inset with effect reference)

The UI encapsulates music in a text-based notation (Figure 2), supported by rapid edit-audition cycles, designed to improve the conditions for intrinsic motivation ("flow") by accelerating musical feedback: lowering the learning threshold, physically immersing the user in music (sound), and increasing the "liveness" of interaction with the notation through the computer keyboard (see Nash and Blackwell, 2014). To focus users on composing and coding, this study uses a simplified version of the original software (dubbed "Trinity", Figure 3), removing all features save the pattern editor (to maintain focus on the notation), restricting songs to a single looped pattern (encourage code-driven evolution of pieces), and bundling a library of 100 high-quality multi-sampled instruments (to avoid users needing to source or configure sound sources).

Initial testing has been conducted with individual composers and educators, using early feedback to refine aspects of the programming language (syntax and semantics) and user experience. To evaluate the flexibility of the tool, established musical works are being implemented in Manhattan – each as short, minimally notated pieces of music that use code (formulas) to generate and evolve the notated musical data to engender significantly longer, more detailed, semi-generative performances of familiar pieces in various styles (see Table 1). This growing list of examples highlights concepts of abstraction, process, and patterns in music (e.g. musical form, theme and variation, chaconne, passacaglia, etc.) that can be explored through fundamental programming concepts of iteration and control flow (conditional branching). Emerging from these tests, a particular strength of the environment is the flexibility it offers for integrating generative elements within more traditional musical forms, structures, and arrangements.

style	***piece***	***music concepts***	***code technique***
Minimalism	*Piano Phase* (1967) by Steve Reich	counterpoint, phasing, process music	arrays and pointers, iteration
	Fratres (1977-92) by Arvo Pärt	harmony (tintinnabuli), melodic progression	arrays, iteration, conditions
Popular	*Blue Monday* (1983) by New Order	progressive house, repetition	arrays, iteration, conditions
Baroque	*Canon in D* by Johann Pachelbel	canon, chaconne, ground bass	arrays, iteration
Traditional	*12 Days of Christmas*	musical form (cumulative song)	control flow, conditions

Table 1 – Selected musical examples, with related music and coding concepts



```
[04:008]= [+1 -3]
.pitch   =
.instr   =
.volume  = rnd(0,64)
.pan     =
.depth   =
.effect  =
  .type  = .repeat > 1 ? D1h : D0h
  .param =
```

Effect Reference

SDx - Note Delay

This effect effectively delays the processing of the current note cell for *x* frames.

It is very useful for gaining finer control over timing. It can be used to delay notes (first example), so as to add syncopation (second example) or rubato, but can also be used to delay changes to an existing note - e.g. for volume or panning variations (last row of second example).

A delayed note off can be acheived by using the SDx effect with the note off symbol (===, see last row of first example). To delay a note cut, you can use the note cut symbol (^^^), or the Note Cut effect (see **SCx**).

In Impulse Tracker 2, it was useful to use this effect to delay MIDI notes so that they were properly synchronised with sample notes. In reViSiT, the MIDI Delay setting, found in the Preferences screen, should be used for this purpose (if necessary).

Example

```
D-4 01 ·· SD3
=== ·· ·· SC2
```

A delayed note and note off.

Example

```
C-5 01 ·· .00
C-5 02 40 SD1
C-5 01 ·· .00
··· ·· 40 SD1
```

A syncopated rhythm.

Figure 3 – Trinity UI, featuring a single musical grid (pattern), pop-out formula editor and integrated learning materials (tutorials, examples, help)

Combined with simple generative exercises (such as random note generation using pentatonic scales), such pieces form the basis of lessons and exercises designed for undergraduates on UWE's music technology degree programmes, delivered in parallel with studies in music theory, composition, and software programming (*Max*, *C/C++*). Initial demos and previews have received positive feedback from students, who often struggle (or lack motivation) in programming classes; exploring coding concepts around a more familiar 'sequencing' paradigm appears to make programming more accessible and less abstract. A broader, controlled user study is scheduled for February 2016, based on classes activities split between structured exercises and exploratory creativity – with findings to be presented at MET 2016. A subsequent conference workshop for TENOR 2016 (in Cambridge) will further explore the potential for the environment and project from the perspective of researchers and educators, ahead of classroom-based studies in local Bristol secondary schools in 2016/17.

The next phase of the project looks at extrinsic sources of motivation, using online connectivity for sharing, collaboration, and competition. Collaboration is both synchronous (in realtime, through shared multi-user workspaces) and asynchronous (e.g. shared offline projects, including reprises, reversions, or remixes). Other users contribute support, advice, and feedback on both music and code – which are shared in open-source works, to help develop a repertoire of pieces that act as learning resources. Competitive elements such as rankings and tournaments are supported by simple metrics, based on user reviews of works (a consensual subjective aesthetic) and automated analysis of music and code complexity. To encourage individuals to consider abstractions of music processes and patterns, the length of manually notated music is limited (such that musical length and variation arise from code-based manipulation of music) and rankings use a reward mechanism that balances musical quality and code conciseness – a simple, practical metric for modelling 'virtuosity' (~ perceived musical quality ÷ music and code complexity).

The mechanisms described here are designed to exploit aesthetics and practices observed in amateur digital music and coding cultures (e.g. the "demo-scene"), where artists and coders share and compete with digital art, whilst imposing artificial restrictions on the notation or techniques available to them (Nash and Blackwell, 2011). As in the related community of video gamers, the challenges and (possibly self-)competition presented by the technology help to engender flow, keeping individuals engaged and motivated – and continually learning and developing their abilities. The Manhattan project is an effort to carry such motivational mechanisms to other areas of music and programming.

References

Nash, C. (2014). Manhattan: End-User Programming for Music. *Proceedings of New Interfaces for Musical Expression (NIME) 2014*. June 30-July 4, 2014. Goldsmiths, London, UK. pp. 28-33.

Nash, C. and Blackwell, A.F. (2011). Tracking Virtuosity and Flow in Computer Music. *Proceedings of the International Computer Music Conference (ICMC 2011)*. July 31-August 5. University of Huddersfield, West Yorkshire, UK. International Computer Music Association. pp. 575-582.

Nash, C. and Blackwell, A. (2014). Flow of creative interaction with digital music notations. In: Collins, K., Kapralos, B., and Tessler, H. (Eds.), *The Oxford Handbook of Interactive Audio* (pp. 387-404). New York: Oxford University Press.