A COGNITIVE DIMENSIONS APPROACH FOR THE DESIGN OF AN INTERACTIVE GENERATIVE SCORE EDITOR

Samuel J. Hunt
Creative Technologies Laboratory
UWE Bristol
Samuel.hunt@uwe.ac.uk

Tom Mitchell
Creative Technologies Laboratory
UWE Bristol
Tom.mitchell@uwe.ac.uk

Chris Nash
Creative Technologies Laboratory
UWE Bristol
Chris.nash@uwe.ac.uk

ABSTRACT

This paper describes how the Cognitive Dimensions of Notation can guide the design of algorithmic composition tools. Prior research has also used the cognitive dimensions for analysing interaction design for algorithmic composition software. This work aims to address the shortcomings of existing algorithmic composition software, by utilising the more commonly used score notation interfaces, rather than patch based or code based environments. The paper sets out design requirements in each dimension and presents these in the context of a software prototype. These principles are also applicable for general music composition systems.

1. INTRODUCTION

The primary focus of this research is to engage traditional composers with generative music systems. Existing tools for generative music composition require that users transition from traditional score editor workflows [1], into programming based environments, either graphical (like Max [2]) or textual (like Sonic Pi [3]). Little research has explored how algorithmic music techniques can be integrated into contemporary digital music composition work-flows (e.g. score editor-based, and digital audio workstation-based composition).

This paper discusses an ideal usability profile for a system that integrates generative music elements into a score editing workflow, considering the features required, under the Cognitive Dimensions of Music Notation framework. A prototype generative music system called the Interactive Generative Music Editor (IGME), is presented that considers the Cognitive Dimensions of Music Notation in its design in order to support interactive generative music. The research also shows directions in which digital score editors might develop to improve general usability.

2. COGNITIVE DIMENSIONS FRAMEWORK

Green and Petre [4] proposed the Cognitive Dimensions of Notations framework, as an evaluation technique for visual programming environments, interactive devices and non-interactive notations. Nash [5] has adapted this framework for use in designing and analyzing music notations and user interfaces for digital and traditional music practice and study. Bellingham [6] presents similar work, using the dimensions approach for analyzing a representative selection of user interfaces for algorithmic composition software. Finally, the cognitive dimensions can also be thought of as discussion tools for designers [6].

3. INTRODUCTION TO IGME

IGME is a score editor-based music sequencer that incorporates using generative and algorithmic techniques, designed to promote a human and computer cooperative creative system. A more detailed overview of IGME (previously named IGMSE) is given in [1].

The core design principles of IGME are as follows:

- Integrates algorithmic techniques for musical composition inside familiar score editing and music sequencing workflows.
- Provides full version control, for revisiting and comparing material.
- Uses graphical controls (WIMP) rather than code based interfaces.
- Takes a modular approach to composition, while retaining a linear timeline.
- Uses a multi-layered assembly stage that assembles the final score from individual parts.

IGME considers composition in terms of three distinct musical parts: human created content, computer generated content, or a mixture of both. The IGME program is divided into two main views: the arrange view (Figure 1) and edit view (Figure 2). The arrange view focuses on arranging and sequencing individual parts, using design principles found in other sequencers such as Logic Pro X [7]. The edit view (or detail view) allows the user to edit the individual music sequences, and/or specify the algorithmic effects for each part. A range of algorithmic effects are implemented by IGME, that can either augment human composed music, or generate computer created music. These are presented using an audio plugin metaphor, whereby control is given though simple graphical controls (see figure 4).

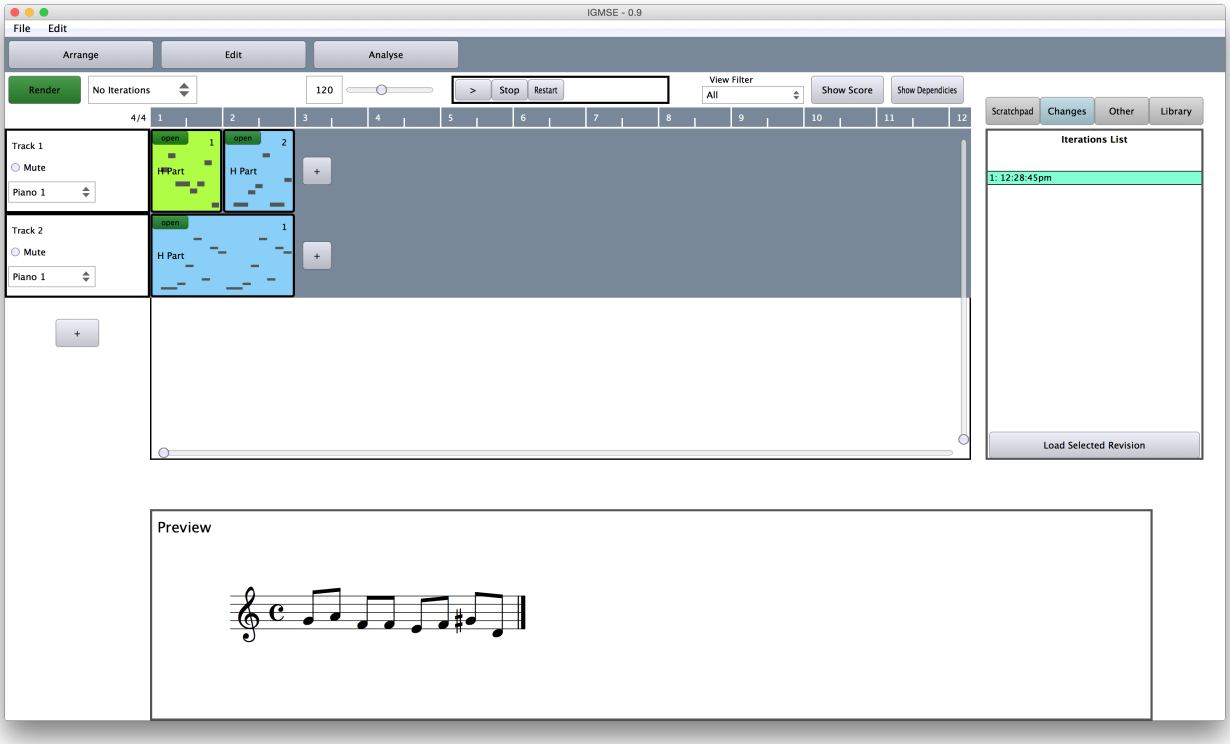

Figure 1. The arrange view in IGME.

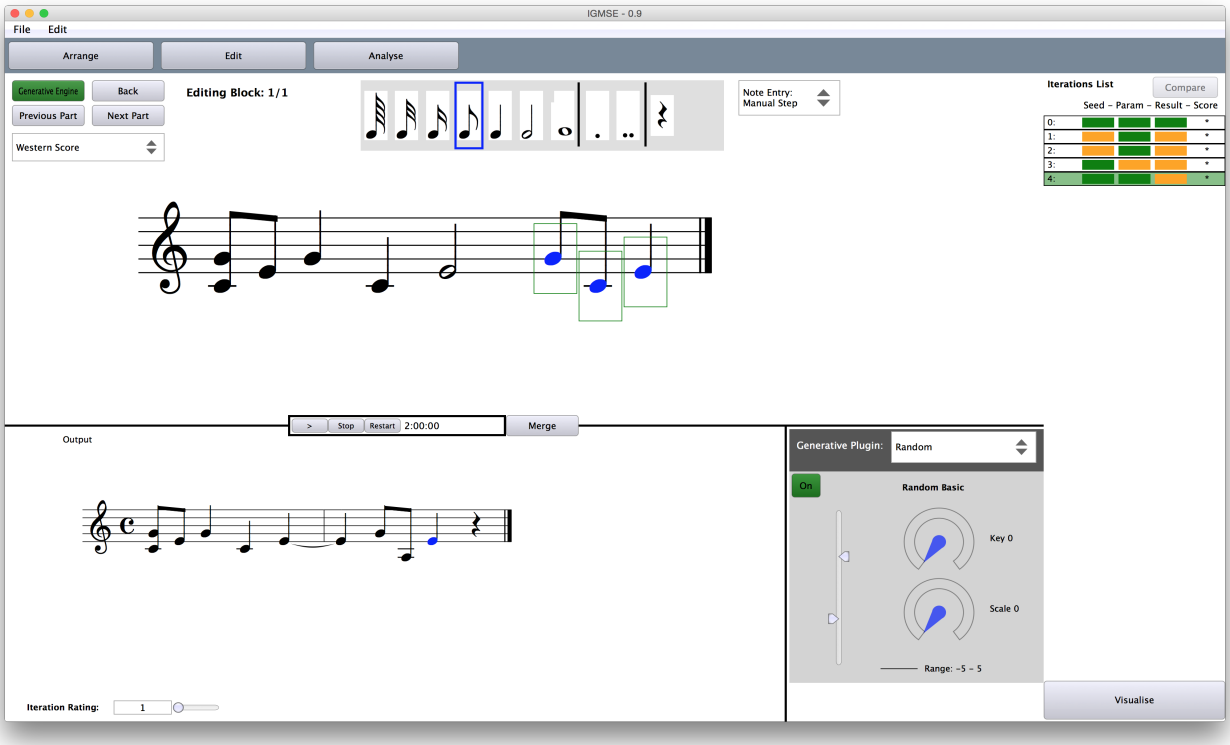

Figure 2. The edit view in IGME.

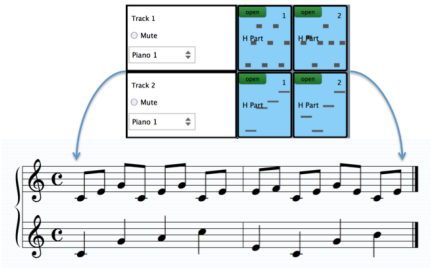

Figure 3. Each of the parts gets concatenated to form a final score.

Once the individual musical parts have been arranged on the timeline and any generative systems set up, the final score can be assembled. This process, referred to as the assembly stage, evaluates all of the computer generated parts and along with any human parts assembles them into a final score that can be auditioned and further inspected (see Figure 3).

A prominent feature necessary for supporting IGME is the version control system (Figure 1 & 2). This feature keeps track of all of the various edits made at both the individual part level and global level. This encourages experimentation with the generative system as it is not possible for the user to overwrite or loose material. This feature also allows the user to revisit previous material at will. Such a feature was originally proposed by Duignan [8], noting that such features are non-existent in music composition systems.

Two related and existing systems worth noting that also use score interfaces and have similar design themes are InScore [9] and OpenMusic [10]. InScore is an environment for the design of interactive augmented music scores, it is mainly used for the real-time playback of dynamic music. OpenMusic is a visual programming language, that is a convenient environment for music composition. Although both systems use score interfaces, InScore requires the learning of syntax, and OpenMusic has many of the issues associated with patch environments [6].

4. DIMENSIONS OF MUSIC NOTATION

The remainder of the paper takes each of the dimensions in turn and discusses them in context. The description for each dimension is taken from Nash's [5] work.

5. VISIBILITY

"How easy is it to view and find elements or parts of the music during editing?"

The software presents varying hierarchical levels of visibility, these can be loosely thought of as the detail view, arrange view, and global view. The detail view (Figure 2) shows the individual notes within each part, the processes attached to them, and the net result. The arrange view (Figure 1) shows the sequential order between parts, giving an indication of which parts are human created, or computer

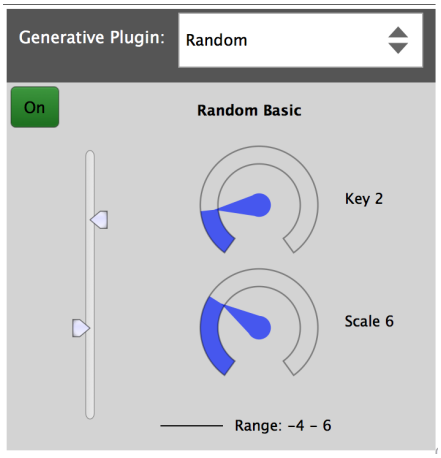

Figure 4. Random algorithmic effect, alters the pitch within a given range, also applies key/scale quantisation.

generated. Finally the global view (Figure 3) shows the final output as a score, after all processes have been applied, and is the musical content auditioned by the user, this is very similar to the kind of view seen in programs such as Sibelius [1] . There are few steps required to get from the arrange view down into the edit view. A criticism of patch based interfaces as stated by Bellingham [6] is that this structure is not often clear with many layers often spread across different windows. Finally selecting an individual note in the global view will highlight its parent part, so that it can be edited.

Generative effects in IGME are shown with graphical controls (Figure 4) similar to those found in audio/MIDI plugins, rather than using variables or number boxes. The values of the plugins are easier to see, compared with programming based systems. In programming based generative systems the value of a variable can sometimes be defined far away from where it is used, thus presenting additional debugging challenges.

Following design principles in similar programs, different tracks can be isolated in the view and shown against each other, giving great control over what is shown at once. The program can be split into two windows, one for the arrange and edit view and the other for permanently showing the overall score.

6. JUXTAPOSABILITY

"How easy is it to compare elements within the music?"

Specific iterations can be compared with a dedicated compare command. As each iteration and edit is retained, it allows different parts to be swapped in and out quickly and the result auditioned in context with the parts around it. The included *diff* tool (Figure 5) shows the explicit difference between parts. Bellingham et al [6] note that form-based systems such as Tune Smithy and the Algorithmic Composition Toolbox do not allow users to see older entries as they are replaced, this having a negative effect on

[1] http://www.avid.com/sibelius

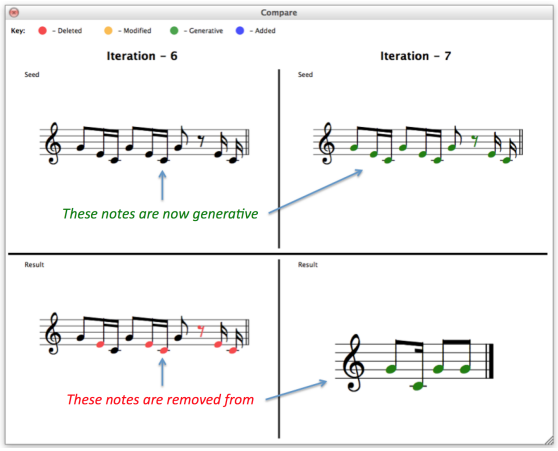

Figure 5. Diff tool comparing two different iterations.

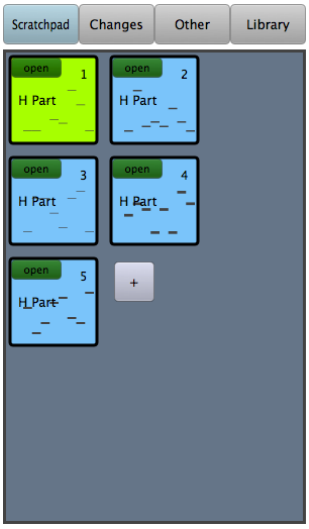

Figure 6. Scratchpad for creating musical ideas.

juxtaposability. Thus users are reliant on either using an undo, or keeping everything in working memory, increasing *hard mental operations*. These issues are mitigated by the version control techniques introduced in IGME.

Nash [5] notes that some musical characteristics are not obvious purely in the visual domain. For example some notes are dependent on key and transposition (for certain instruments) to fully decode, requiring the user to understand these relationships. Two solutions can be proposed respectively. First, the colour or shape of the note head could be modified to explicitly show that a note is for example not an E but an E flat, however this feature is possibly only useful for novice composers. Second, by using the IGME assembly process, transposed instruments can be notated as a normal instrument, and then be transposed into the correct transposition for the overall score during the assembly stage, this does however increase *hidden dependencies*. Both of these features have been encoded into IGME but remain optional at this stage.

It may be beneficial to compare elements in the music in a different domain to assess for example why a generative part is not suitable. IGME includes built in analysis tools that allow for the analysis and comparison of musical elements, showing for example the difference in pitch distributions between different sections. This may for example show quickly that track A is clashing with track B, because of the increased use of accidentals. This feature would be useful for normal, purely human compositions, but increasingly useful for compositions with elements of computer generated parts. For more information relating to visualizing musical elements in reference to the cognitive dimensions see [11].

Using programming based environments for generative music makes comparing elements in the music especially difficult. The program would need to be recompiled and the runtime output recorded into a third-party program for offline comparison. This issue is also true of environments such as Max/MSP. The ability for programming based environments to achieve the same effect in multiple ways can be prohibitive for this dimension. IGME supports the auditioning of music within the software itself; however the music can also be output as MIDI for synthesis or analysis in an external client.

7. HARD MENTAL OPERATIONS

"When writing music, are there difficult things to work out in your head?"

Existing systems for generative music often require the user to design and implement algorithmic techniques using either code or a graphical programming environment (e.g. Max/MSP), therefore placing a high mental load on the user. This software provides such techniques out of the box, focusing on using, rather than designing algorithms for music composition. A knock on effect is that these techniques end up being black boxes, where the user has little knowledge of what each technique is actually doing. There is a certain trade off between making each technique internally accessible, and making it simple to use on the outside.

Scores unlike other forms of notation and music software, require the user to have a fairly high literacy threshold of score notation [5]. As this software is designed for composers familiar with western notation, this issues is not a prominent one. However it is possible to switch from a score editor to a piano roll editor for those more familiar with the MIDI editing workflow, although the focus at this stage is on score editing.

The scratchpad feature of IGME helps reduce the cognitive load further by allowing users to offload their musical ideas, without needing to think about their final location in the overall structure. This feature works by simply allowing the user to create musical fragments in a separate window. These fragments can then later be dropped into the final arrangement (Figure 6). Finally the rapid entry methods discussed in section 12 also aid this dimension even further, allowing users to capture core primitive elements of the music and focusing later on the exactness of these elements.

Bellingham et al [6] stress the need for a clear visualization showing the signal flow between components. These

issues are a common problem for systems without a clear indication of a timeline, for example coding, patching and offline systems. As IGME uses the timeline metaphor control moves from left to right, therefore the user is not required to predict control flow, reducing cognitive load.

8. PROGRESSIVE EVALUATION

"How easy is it to stop and check your progress during editing?"

Individual parts can be rapidly auditioned, however users of this system are required to manually iterate the assembly process so that individual parts can be heard in the context of other parts, adding a small amount of delay to the process. The user is able to toggle these arrange level iterations to happen automatically, mitigating this delay, but it must be first explained to each user. Mute and solo controls are present in IGME and have the same usage as the majority of music software.

Collins [12] states that evaluating the material is obviously important for music software, as iteration is a primary concept in composition. A user interacting with any form of composition software is likely to apply a trial and error approach, testing many different ideas and combinations. Nash [13] notes that a rapid edit-audition cycle contributes to having a high state of *flow*, a desirable mental state for users engaging with creative exercises such as music. The affordances offered by score editing software make it very easy to stop and audition parts at any point, and for the most part make quick edits. Code based environments, make such editing processes more complicated, often due to the need for recompiling. The introduction of compile time errors, can cause the composition workflow to stop all together, such compile and run time errors are prevalent in patch and code based environments. Compile time errors are eliminated with IGME, as all generative effects are pre-compiled. Like other non-generative music software, erroneous data is prevented from being entered by the restrictions imposed by the UI. The software makes heavy use of pop-up warnings. These features of IGME reduce *error-proneness* and increase *provisionality* also.

The version control system further aids in this dimension, as users can not only revisit their previous work, but also see how there compositions have progressed over time using the diff tool (Figure 5).

9. HIDDEN DEPENDENCIES

"How explicit are the relationships between related elements in the notation?"

An important consideration is the relationships between different musical elements at varying structural levels. A specific use case of IGME is that the individual parts can reference other parts. For example part 2 on track 1 can take its initial content from the output of part 1 on the same track. This facilities simple repeats, or more complex processed based music. To reduce the complexity of using reference parts, the specific dependencies can be

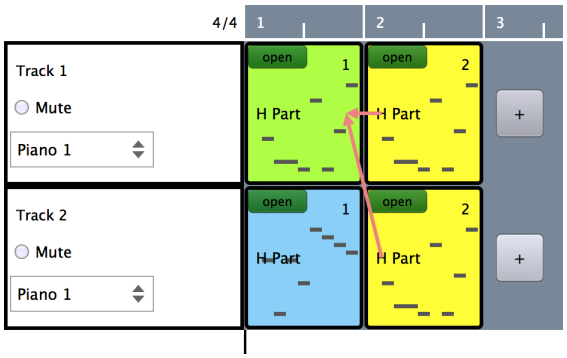

Figure 7. Part 1 on track 1 is being repeated (referenced) by 2 other parts.

Figure 8. The show dependants features, highlights all parts that are dependant on part 1 track 1.

highlighted by using unidirectional coloured arrows, an example of this is given in Figure 7. This feature is similar to the patch cable metaphor used in Max/MSP [2] and Reason [14]. Deleting events that have references triggers a warning ensuring the user is aware of the knock on effect of doing this.

Generative systems such as Max make dependencies more explicit by using patch cables [6]. However, the variables used in code based music systems, have more complex hidden dependencies. For example without manually searching for a variable it can be a challenge to see exactly where it is later used, and what effect changing it has for the overall output. Changing the internals of a particular function or patch can introduce knock-on effects if other parts of the program are dependant on the original behaviour. Even though IGME does not use code or patch based workflows, it can still have dependencies, especially for more complex arrangements. For example some of the purely generative effects work by analysing the surrounding musical content, changing or removing a musical part will alter the output of such generative processes. By shift clicking on a part in the arrange view, it will show what other parts are dependent on this part (Figure 8). Furthermore, pop-up warnings are presented at the point where making an edit would have a knock-on effect.

10. CONCISENESS / DIFFUSENESS

"How concise is the notation? Does it make good use of space?"

Nash [5] notes that western scores remain a concise form of notation interface, with the material shrinking and growing depending on the number of notes in a bar. However, issues present themselves in that score notation requires expert knowledge to decode the symbolic encoding of time. These decoding issues are not an issue for other forms of digital notation such as a step sequencer or piano roll; however both of these can take up considerable amounts of space.

The multiple views offered by IGME allows the representation of music at different hierarchical levels, improving the *conciseness* of the notation, also aiding with *visibility*.

Bellingham [6] notes that one should increase the verboseness in the language for variable names in coding environments, even if this has a negative effect on this dimension. This idea has been incorporated by giving parameter names more verbose names, for example"num of notes" gets expanded into "number of notes to be generated". The effect on overall user interface space usage is negligible.

11. PROVISIONALITY

"Is it possible to sketch things out and play with ideas without being too precise about the exact result?"

The most prominent idea introduced in IGME is the in-built version control technologies. This encourages experimentation as ideas cannot fundamentally be overridden or lost.

The *assembly* process allows users to rapidly enter note sequences, without the restrictions imposed by bar lines or time signatures. For example eliminating bar lines in the initial note entry process removes the need for tied notes that cross bars, as notes can simply be displayed as their absolute length. The later assembly stage takes care of creating these formalisms (see Figure 2).

A *scratchpad* feature (Figure 6) is presented in IGME that supports the user creating parts outside the scope of the arrange view. In most other software this could only be achieved by placing content many bars in the future or creating a new session entirely. This ability to create provisional material is a feature of Presonus [2] . Older iterations in a given part can be placed into the scratchpad so they can be later re-purposed.

A prominent feature is the note step option. When entering a new pitch for a selected note the user can switch between auto and manual step. Manual step means that entering a new note alters only the selected note, and the cursor does not increment to the next note. Auto step allows rapid entry as the cursor increments each time a new note is entered.

Nash [5] states that digital score notation interfaces are weak for supporting this dimension compared with paper

[2] https://www.presonus.com/products/Studio-One

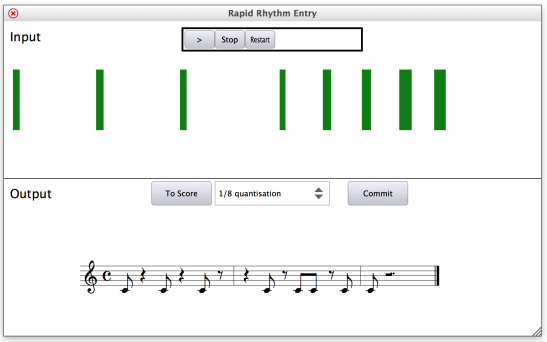

Figure 9. The Raid Rhythm Entry window allows user to enter rhythm patterns quickly.

notation which is far more flexible, as it allows for informal sketching. Programs such as Sibelius are beneficial for preparing final scores, and not necessarily for rapidly experimenting with ideas. A feature of IGME is the ability to rapidly enter notes, and this is supported by providing interfaces supporting a sketching metaphor. See *secondary notation* (Section 12) for more information.

Finally, Bellingham [6] notes that Impro-visor's [15] preset algorithms can be used for quickly creating musical sketches based on chord progressions. A prominent use case hypothesis for algorithmic music is that it can be used to generate new ideas, or to suggest augmentations to existing ideas, the extent to which such a feature is useful, is one of the major objectives of planned future research in evaluating IGME. Bellingham also notes that Logic Pro's in-built loops facilitate *provisionality*, as the content can be used as a place-holder and replaced later.

12. SECONDARY NOTATION

"How easy is it to make informal notes to capture ideas outside the formal rules of the notation?"

Nash notes that handwritten scores have an almost unlimited ability to make informal notes that can be interpreted by the user across the score. However, digital scores make this much harder through limited interaction with the keyboard and mouse. Staff Pad [3] offers a trade off between paper based and digital notation, in that scores can be sketched on a digital screen using a stylus, and the content typeset via advanced handwriting recognition, this is perhaps only useful for those already familiar with handwritten notation, it is also unclear how high the error rate would be. A key principle of IGME is providing ways to quickly enter sequences of notes, as briefly discussed in the previous dimension (*provisionality*).

IGME has a range of in-built tools for quickly capturing ideas, that can be formalised at a later stage. Figure 9 shows an example of one of these techniques that use secondary notation to quickly input material. The "Rapid Rhythm Entry" window allows user to simply tap a rhythm in using the space bar. The ascoiated pitches and exact rhythm can be edited later. These techniques support a

[3] https://www.staffpad.net/

kind of sketching [5] metaphor where informal ideas can be recorded quickly and effortlessly.

Bellingham [6] notes that adding colour to different elements of the interface can aid the usability of a program. For example logic pro can display each track as a different colour. Max allows different patch cables to be given different colours which could for example represent different types of signal flow (e.g. MIDI, mathematical, GUI controls). Bellingham [6] and Nash [5] both note that programming and patching environments support adding comments that can better explain the program between users and subsequent uses. Nash emphasises that limited provision is made in digital audio workstations for annotating music in any of the sub-notations or views. Both the parts and generative effects in IGME allow the user to write informal notes about their current choice of arrangement.

13. CONSISTENCY

"Where aspects of the notation mean similar things, is the similarity clear in the way they appear?"

A key design feature of IGME is that its design elements are consistent with other music sequencers. For example the arrange view and edit view are borrowed from similar elements in Logic Pro X, Cubase and Pro Tools. Editing music in score notation shares many features with Sibelius and other notation packages. An important design principle is that someone who is familiar with Logic Pro and Sibelius should find it easy to pick up and use IGME. The generative effects operate much like plug-ins, with presets and graphical controls.

Bellingham [6] notes that a consistent interface is easy to learn, so for IGME it is important that the interface is consistent so the new features introduced are easily picked up, so they can instead focus on exploring the novel features of the software. A criticism of many existing generative music systems is that they require the user to learn a new, often unfamiliar workflow.

This is one dimension where the changes made for other dimensions have had a knock on negative affect in this area. For example offering different forms of input notation to rapidly record new ideas, reduces *consistency* (as there are now multiple ways to achieve similar things) for an improvement in *provisionality* and *closenss of mapping* (to other interfaces more familiar to users).

14. VISCOSITY

"Is it easy to go back and make changes to the music?"

The editing stage in IGME has been designed with low *viscosity* in mind. The removal of bar lines for editing notes, means that users are not required to supply tie lines for notes that cross bar lines. Attempting to increase the length of a note in existing score editors, has knock-on *viscosity* [5], where the resultant effect will often discard notes from the end of a bar. Guitar pro [4] solves this in a novel way, by not removing anything, instead highlighting the bar as an error (in red), requiring a manual fix from the user. This suggestion of removing bar lines for speeding up the edit process is not novel, as it is also used in Dorico [5].

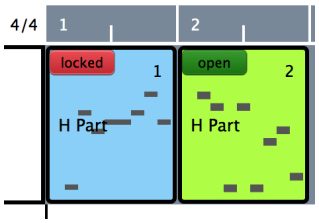

Figure 10. Part 1 is locked and cannot be moved or edited, whereas part 2 is open.

Repetition viscosity [6] becomes an issue where sections of the music are copy and pasted to create repeats, and the user wishes to change the initial content. IGME's reference feature can mitigate this issue, as changing the initial part, causes all parts that reference it to update. A downside is there is a slight increase in hidden dependencies, although this has been addressed in other ways (see Section 9). By providing users with an easy way to use repeats, it could have consequences for both the musical quality and variety. The non-automated method may create happy accidents, whereas the proposal here may simply encourage rigid repetition. The exact effects cannot be determined at this time, but will be considered in future analyses of user interaction with the software.

The inbuilt source control technologies inside IGME ensure that going back to make changes is encouraged. Some issues can arise whereby the user inserts a note that expands the part , for example from 1 bar into 2 bars, therefore requiring the surrounding parts to be moved around in context. However sync points can be used to ensure that future sections are preserved. The lock feature of IGME (Figure 10) prevents the position or internal content of a part being modified. These sync points and part locks create a trade off between *viscosity* and *provisionality*. The software does not therefore impose either option but hands over responsibility to the user.

Certain parts of the software have a *viscous* workflow, for example the generative plugins have a limited degree of control. Bellingham [6] notes highly viscous workflows can improve stability and create well defined use cases. This is not to say the effects are not powerful but have carefully designed user interfaces that facilitate a fluid experience for the user. Finally, Nash [5] notes increasing *viscosity* is a trade-off for avoiding *hidden dependencies*, this can be observed in patch and code based environments.

15. ROLE EXPRESSIVENESS

"Is it easy to see what each part is for, in the overall format of the notation?"

A key design requirement of IGME is that each design element makes use of existing metaphors, including staff no-

[4] https://www.guitar-pro.com/en/index.php?pg=buy-guitar-pro

[5] https://www.steinberg.net/en/products/dorico/start.html

tation (Sibelius), arrange/edit view (digital audio workstations), graphical controls (plug-ins) and part editing (most music sequencers). Using metaphors and graphical controls, allows a user to quickly understand the potential uses of each editor [5] [6].

In general, code and patch based environments are not as role expressive, unless the user is already familiar with the system. For a traditional score editor user, transitioning into a code/patch based environment presents a considerable learning curve. For example variables replace graphical controls, and without ensuring these have suitable names, confusion can arise in each variable's role. This issue can be prominent in MAX/MSP where some objects appear as text boxes [5].

A key consideration of integrating generative music is ensuring the user is aware of what notes are going to be altered by any generative process. This is done by changing the colour of the note, where green is used to indicate notes that may be processed, and black to show ones that are fixed (Figure 2). Parts are also coloured depending on their use, i.e. blue for a normal part or orange for a purely generative part (Figure 1).

16. PREMATURE COMMITMENT

"Do edits have to be performed in a prescribed order, requiring you to plan or think ahead?"

The timeline metaphor in IGME somewhat encourages a linear left-to-right workflow. However like score editing IGME supports various forms of development, including part-by-part, bar-by-bar or top down arrangements (form) [5]. The scratchpad (Figure 6) feature allows parts to be created offline and then placed back into the arrangement later on. The flexible editing methods offered by not enforcing bar lines, ensures that for example notes in the middle of bar can be edited easily at a later stage.

Bellingham [6] stresses that it should be important to have an option that states I don't know what is going here. Many sequencer based systems including score editors, inherently allow such gaps in the arrangement. More specialised generative music systems without the concept of a timeline are weaker in this category, and in general code based systems are not supportive of more structured or orchestrated compositions. Nash [5] notes that an advantage of using an arrange view metaphor, is it allows musical parts to be easily inserted, moved and copied.

Digital score editors inherently force several commitments from the outset, for example tempo, key and time signature. A planned feature of IGME is these can be autocompleted, whereby the user enters a sequence and the software analyses the musical features to predict tempo, key and the time signature. See Temperley [16] for more work in this area.

17. ERROR PRONENESS

"How easy is it to make annoying mistakes?"

As all generative effects are presented through a graphical UI rather than patching or code, users are generally protected from doing things that would otherwise break the underlying generative models. The assembly process takes care of adding formalisms that might otherwise be seen as errors, for example fixing bar lines with tied notes, and ensuring harmonic consistencies.

A feature of any generative music system is that the algorithms themselves can generate annoying musical mistakes, or wildly inappropriate musical material. The version control tracker features of IGME ensure many iterations can be experimented with, encouraging the user to tune the model to produce a more desired effect. It is difficult to appropriately tackle a subjective area such as music, as musical qualities deemed annoying mistakes by one composer, may be wholly appropriate by another.

18. CLOSENESS OF MAPPING

"Does the notation match how you describe the music yourself?"

A key advantage of the IGME assembly stage, is that different parts can have different notations for representing the musical material during the editing stage. At the assembly stage these parts can be converted into a single notation format for viewing and performing. Nash [5] notes that score representation is not an intuitive representation, but remains widespread especially for performers. It is therefore important to ensure that whatever notational interfaces are offered by the program it can still produce a score based output, especially when the programs musical output is to be performed by musicians. IGME does not yet support any novel notation interfaces but is considered for future versions.

As the generative effects are similar in nature to plugins, the individual effects values are expressed as simple GUI controls rather than variables or number boxes. For certain effects, terminology that aligns with how the sound is described is used. The use of graphical controls aligns with the visual metaphors offered by audio plugins and virtual instruments. Nash [5] notes that DAWs score highly in this dimension due to having interaction paradigms based on recording studio workflows.

19. ABSTRACTION MANAGEMENT

"How can the notation be customised, adapted, or used beyond its intended use?"

Green and Blackwell [17] describe three classes of software; abstraction-hungry systems, abstraction-tolerant systems and abstraction-hating systems. Programming and patch based environments rely heavily on abstractions, which has a negative effect on usability, especially for traditional composers transitioning into those new types of in-

terfaces. Nash [5] notes that composers using paper scores are free to invent new notation techniques to describe music more concisely, digital score editors tend to be more limited. In addition Bellingham [6] states that Abstractions can be used to make software more effectively match the users mental model of the music they are notating.

Bellingham [6] states that *"An effective design would be for the software to have a low abstraction barrier but be abstraction-tolerant. Such a design would allow new users to work with the language without writing new abstractions, while more advanced users could write abstractions when appropriate."*

In general IGME's inbuilt processes for generative music are abstraction-hating as they cannot be customised internally, but can only be control through exposed GUI controls. IGME abstracts sequences of events into parts that can have further processes applied to them, and also reference and reuse each other. Overall the system is therefore abstraction tolerant. It is unclear without more conclusive user studies how powerful IGMEs inbuilt generative features and part referencing will be.

20. CONCLUSION

This paper has shown how many of the issues associated with generative music systems can be mitigated by transitioning into more traditional music sequencing workflows, eliminating the deficiencies offered by patch and code based environments. Many of the suggestions made in this paper, for example the version control system, would be beneficial for different music sequencing and composition software in general.

Future work will focus on testing the IGME software with composers, both in longitudinal studies and shorter workshop sessions, then evaluating each cogntive dimension in a similar way to Nash's [5] research. Another theme for future research will be integrating more advance algorithmic techniques for music creation, the primary aim of the overarching research objectives.

21. REFERENCES

[1] S. Hunt, C. Nash, and T. Mitchell, "Thoughts on Interactive Generative Music Composition," in *2nd Conference on Computer Simulation of Musical Creativity*, Milton Keynes, UK, 2017.

[2] V. J. Manzo, *Max/MSP/Jitter for Music: A Practical Guide to Developing Interactive Music Systems for Education and More*. Oxford University Press, 2016.

[3] S. Aaron and A. F. Blackwell, "From Sonic Pi to Overtone: Creative Musical Experiences with Domain-Specific and Functional Languages," in *Proceedings of the first ACM SIGPLAN workshop on Functional Art, Music, Modeling and Design (FARM'13)*, 2013, pp. 35–46.

[4] T. R. G. Green and M. Petre, "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework," *Journal of Visual Languages & Computing*, vol. 7, no. 2, pp. 131–174, 1996.

[5] C. Nash, "The Cognitive Dimensions of Music Notations," in *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR'15)*, Paris, France, 2015.

[6] M. Bellingham, S. Holland, and P. Mulholland, "A cognitive dimensions analysis of interaction design for algorithmic composition software," in *Proceedings of Psychology of Programming Interest Group Annual Conference*, Brighton, UK, 2014.

[7] K. Anker and O. Merton, *Logic Pro X Power!: The Comprehensive Guide*. Cengage Learning, 2014.

[8] M. Duignan, J. Noble, and R. Biddle, "Abstraction and Activity in Computer-Mediated Music Production," *Computer Music Journal*, vol. 34, no. 4, pp. 22–33, 2010.

[9] D. Fober, Y. Orlarey, and S. Letz, "Inscore - an environment for the design of live music scores," in *Proceedings of the Linux Audio Conference*, CCRMA, Stanford University, USA, 2012, pp. 47–54.

[10] J. Bresson, C. Agon, and G. Assayag, "OpenMusic: Visual Programming Environment for Music Composition, Analysis and Research," in *Proceedings of the ACM International Conference on Multimedia: Open-Source Software Competition*, Scottsdale, AZ, USA, 2011, pp. 743–746.

[11] S. Hunt, C. Nash, and T. Mitchell, "How can music visualization techniques reveal different perspectives on musical structure," in *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR'17)*, A Coruña, Spain, 2017.

[12] D. Collins, "A synthesis process model of creative thinking in music composition," *Psychology of music*, vol. 33, no. 2, pp. 193–216, 2005.

[13] C. Nash and A. Blackwell, "Liveness and Flow in Notation Use," in *International Conference on New Interfaces for Musical Expression (NIME)*, Ann Arbor, MI, USA, 2012.

[14] M. Duignan, J. Noble, P. Barr, and R. Biddle, "Metaphors for Electronic Music Production in Reason," in *Proceedings of the Asia Pacific Conference on Computer Human Interaction (APCHI)*, Rotorua, New Zealand, 2004.

[15] R. M. Keller and D. R. Morrison, "A Grammatical Approach to Automatic Improvisation," in *Proceedings of the Sound and Music Computing conference (SMC)*, Lefkada, Greece, 2007.

[16] D. Temperley, *Music and probability*. The MIT Press, 2007.

[17] J. M. Carroll, *HCI models, theories, and frameworks: Toward a multidisciplinary science*. Morgan Kaufmann, 2003.