Note Code – A Tangible Music Programming Puzzle Tool

Vishesh Kumar
Indian Institute of Technology
Guwahati, India
k.vishesh@iitg.ernet.in

Utkarsh Dwivedi
Indian Institute of Technology
Guwahati, India
d.utkarsh@iitg.ernet.in

Tuhina Dargan
Indian Institute of Technology
Guwahati, India
tuhina@iitg.ernet.in

Poorvi Vijay
Indian Institute of Technology
Guwahati, India
poorvi@iitg.ernet.in

Abstract
We present the design of Note Code – a music programming puzzle game designed as a tangible device coupled with a Graphical User Interface (GUI). Tapping patterns and placing boxes in proximity enables programming these 'note-boxes' to store sets of notes, play them back and activate different sub-components or neighboring boxes. This system provides users the opportunity to learn a variety of computational concepts, including functions, function calling and recursion, conditionals, as well as engage in composing music. The GUI adds a dimension of viewing the created programs and interacting with a set of puzzles that help discover the various computational concepts in the pursuit of creating target tunes, and optimizing the program made.

TEI '15, Jan 16-19 2015, Stanford, CA, USA
ACM 978-1-4503-3305-4/15/01.
http://dx.doi.org/10.1145/2677199.2688817

Author Keywords
Computational thinking, puzzle based learning, tangible interaction, tangible music, education

ACM Classification Keywords
H.5.2 - User Interfaces, L.1.2 - Learning Objects.

Introduction
Computational thinking refers to a problem solving technique that uses computer science techniques. Over the past few years, it has gained recognition as a skill not just for computer scientists; but a basic skill required in modern society in domains from art to science and blended into people's daily life, as described by Wing and Papert [1,2]. Computational thinking involves finding patterns, decomposing problems into smaller pieces, and using the provided tools.

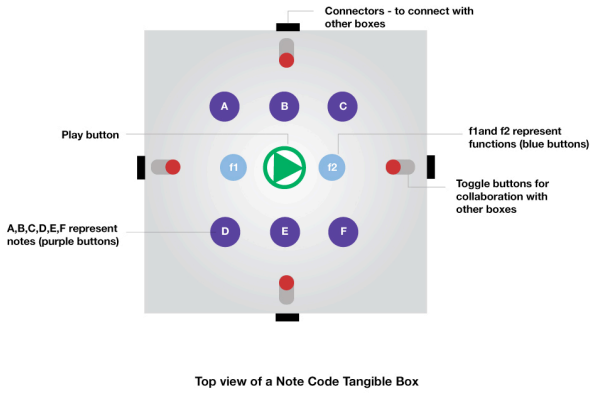

Figure 1. Concept diagram of Note Boxes

There is an incessant construction of tools and systems to make learning programming, or its basic paradigms, simpler for children. Starting with Logo [3], other tools for the same would include AlgoBlock [4], ToonTalk [5], AgentSheets [6], and the largely successful Scratch developed at the MIT Media Lab [7].

Tangible Programming

Piaget and Bruner have showed in the past that it is often more useful to give children concrete examples and objects to enable problem solving, before teaching them the associated symbols. [8,9] This speaks to both, a need for going away from typical syntax, and using more approachable elements (like graphic icons etc.), as well as making tangible tools for the same. McNerney et al reinforce the observation that introduction with syntax is not a suitable way to begin learning to program, especially for primary-middle school aged children [10]. In this spirit, numerous TUIs have been developed to enhance student learning of technology, via technology. Most of these, would be called Resnick's "digital manipulatives" or "programming construction kits" – like 'kinetic recorders' such as Curlybot [11], Topobo [12], or Algorhythm [13]; or helping storytelling by children, such as Telltale [14], and so on.

Tangible music

Numerous tangible music devices have been designed, and prototyped [15], for a variety of purposes. Beginning with the highly popular Reactable, that allows for collaborative music creation by moving tangible markers on a table being followed by sensors – we find BlockJam[16]: a system that involves 'blocks' that store musical pieces, are linked with each others, and are sequentially played by using play blocks. This comes quite close to the tangible aspect of our designed tool as well, though with some crucial differences with respect to stress on the programming paradigms one can interact with. Similar sequencers like Siftables Music Sequencer, based on similar principles, have also been developed.

Music and Computational Thinking

Performamatics [17] is a workshop with a plethora of activities aimed to bridge computational thinking and music, and interest people from both disciplines. Music, and relatedly *sound thinking*, is seen as a ripe avenue to integrate with CT – as both are fertile with patterns, repetition and identification of patters.
Additionally, using computational thinking and programming paradigms allows for an innovative and new approach for music composition, especially in Note Code.

Design and Functionality

Note Code's design and functioning is heavily influenced by concepts of puzzle based learning, since Puzzle-based learning shares many of the pedagogical goals of the emerging paradigm of Computational Thinking [1, 18]; inspired from the drum-bots of Algo.Rhythm, which is conceptually similar to BlockJam and other tangible music systems – recording information in individual blocks, and calling neighboring blocks to play.
Further, the programming concepts we attempted to embed in our interactions and systems, were those of

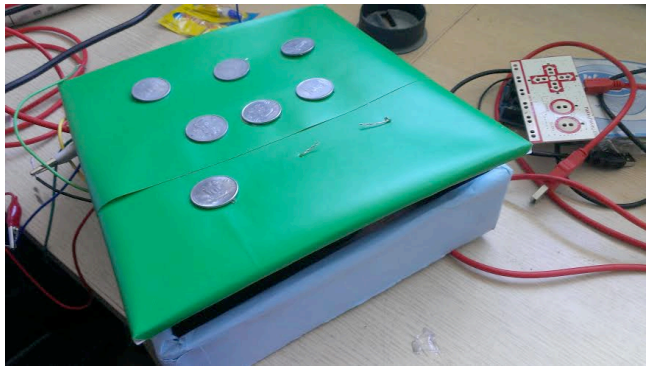

Figure 2. An initial limited prototype prepared

functional modularity, and conditionals – functions being the only non-intuitive construct for beginners to need to understand and grasp, while being one of the most fundamental concepts in computational thinking – along side control structures like conditions and loops.
To embody these, we built a note-box with buttons for sets of notes, and function buttons to hold recordings of sequences. This ended up being significantly similar to Perlman's Button Box. [19] The differences being recording music, is equal parts 'programming by rehearsal' (not requiring the player to think in terms of abstracted instructions) [19] as well as programming in a language i.e. with relatively abstract instructions, when puzzles (tasks of target tunes required to be created) are worked upon.
To enable activating different blocks from one, we made 'edge calls' so that each of the edges could be executed as a step just like any of the other notes. The ability of receiving edge calls, also provided an avenue for implementing conditionals – such that instructions are conditional on receiving signals on an edge, at that step. The implementation of conditionals on edge signals, and using this as a paradigm of computational thinking to interact with, is a feature that makes Note Code significantly unique compared to the numerous other tangible music systems.

Detailed Working

In our prototype, each note-box had six note buttons (notes A to F), four edge out buttons (call edges 1 to 4), four edge in buttons (if edges 1 to 4) two function buttons, and one play button.
To start recording a function, one of the function buttons would be pressed (F1 or F2), and then a sequence of notes, edge buttons, and function buttons would be pressed – which would be recorded into the initially chosen function. The play button would be pressed to stop a function's recording, and in case no function is being recorded – the play button would 'play' the note-box i.e. call F1.
The edge buttons send a signal to the outgoing connectors on each of the edges. During recording a function, pressing a function button records a function call. Pressing an out edge button calls the edge, and pressing an in edge button records a conditional – if a signal is received on that edge during play, the following instruction will be implemented, otherwise it'll be skipped. For example, following is a set of button presses, and then the corresponding recorded functions as a result of the same:
F1 – note B – note A – call edge 2 – note D – F2 – if edge 1 – F1 – note C – Play
produces the program:

```
f1() {
    B;
    A;
    edge 2;
    D;
    f2();
    if (edge 1) {
        F1(); }
    C; }
```

Each row of instruction takes a beat to 'play', check, or be executed. When another note-box is placed in connection with the edge of an earlier note-box, the corresponding edge calls and conditionals enables the possibility of complex and layered music.
This entire construction has been prototyped by using a microcontroller (Arduino/MaKeyMaKey) inside a hand-sawn plywood box. The microcontroller programming, and coupled GUI, have been made using Processing.

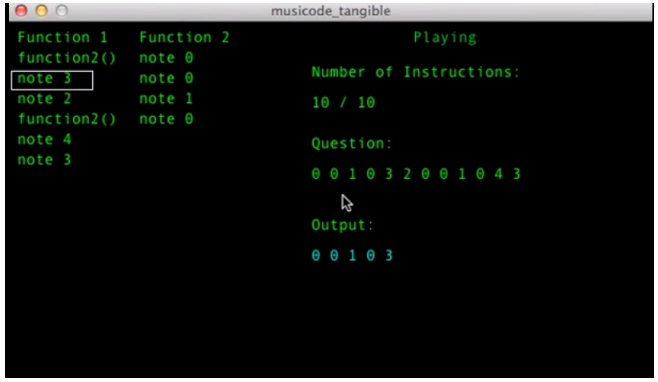

Figure 3. Screenshot of the coupled GUI

Puzzles and GUI

We coupled this system with a GUI (on a connected computer), to help keep track of the contents of the recorded functions. This GUI also had a puzzle component – which would offer the target note sequence, and the minimum number of instructions in which the same should be doable. Our preliminary tests involved seeing how quickly students could grasp the concept of grouping repeated instructions using functions, on a 'puzzle' that involved creating the Happy Birthday tune: " A A B A D C A A B A E D ".
This was preceded and followed by other puzzles to help grasp similar concepts, where the target note sequence was attempted to be melodious as often as possible.
The GUI also helped follow the program flow, by highlighting the instruction being executed at every beat. This helped the players in keeping track of how correctly the program they had made, was behaving.

Discussions and Future Works

We imagine extending the abilities of the described system to make the music composition abilities far richer. It should be possible, to choose different instruments on different boxes, let alone different scales of notes as well. We chose just 6 note buttons for easier implementation, but more – at least 12 notes spanning one octave (per box) would be desirable.
Also, making these note boxes standalone, i.e. not requiring a connected computer, would also help greatly in making this TUI more mobile, and easily playable.
Enriching the GUI for greater ease of appearance and use, and perhaps enhanced functionality with respect to changing boxes' properties (like sound quality, tempo, instrument) would enhance the experience further.
A preliminary testing showed that the ability of creating melodious tunes by solving puzzles, or open endedly exploring on the boxes – to discover either new music, or computational concepts – is highly exciting for both 9th graders as well as college freshmen. We believe that tying seemingly disparate fields like music composition and learning programming exhibits great promise, and further constructs around this should strive to balance simplicity of usage and representation, along side more paradigms like objects, variables, etc.

References

[1] Wing, J.M. Computational thinking and thinking about computing. Philosophical transactions. Series A, Mathematical, physical, and engineering sciences, 366(1881), 3717-25, 2008.
[2] Papert, S. Teaching Children Thinking (AI Memo 247), MIT, Cambridge, MA, 1971.
[3] Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas.

[4] Suzuki H., Kato, H. 1995. Interaction-level support for collaborative learning: AlgoBlock—an open programming language. In The first international conference on Computer support for collaborative learning (CSCL '95), John L. Schnase and Edward L. Cunnius (Eds.). L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 349-355.
[5] Kahn, K. (1996). ToonTalkTM–An Animated Programming Environment for Children (Vol. 7). (Elsevier, Ed.) Journal of Visual Languages & Computing. Leave & Wegner. (1991). Situated learning: Legitimate peripheral participation. Cambridge University Press.
[6] Repenning, A., Ioannidou, A., & Zola, J. (2000). AgentSheets: End-user programmable simulations. Journal of Artificial Societies and Social Simulation, 3(3)
[7] Resnick et al., M. (2009). Scratch: programming for all. Commun. ACM 52.
[8] Bruner, J. (1966). Theory of Instruction. Cambridge, Mass.: Harvard University Press
[9] Piaget, J. (1973). The child and reality: Problems of genetic psychology.
[10] McNerney, Timothy S. "From turtles to Tangible Programming Bricks: explorations in physical language design." Personal and Ubiquitous Computing 8.5 (2004): 326-337.
[11] Frei, P., & Su, V. (2000). Curlybot: designing a new class of computational toys. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (CHI '00). ACM, New York, NY, USA, 129-136.
[12] Raffle, H. S., Parkes, A. J., & Ishii, H. (2004). Topobo: a constructive assembly system with kinetic memory. Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 647–654).
[13] Peng, H. 2012. Algo.Rhythm: computational thinking through tangible music device. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction* (TEI '12), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, 401-402.
[14] Ananny, M. (2002). Supporting children's collaborative authoring: practicing written literacy while composing oral texts. Proceedings of the Conference on Computer Support for Collaborative Learning: Foundations for a CSCL Community (pp. 595–596). International Society of the Learning Sciences.
[15] Tangible Music. "URL: http://modin.yuri.at/tangibles." Consultado em 24.04 (2009).
[16] Newton-Dunn, Henry, Nakano, H., Gibson J. "Block jam: a tangible interface for interactive music." Proceedings of the 2003 conference on New interfaces for musical expression. National University of Singapore, 2003.
[17] Jesse M. Heines, Gena R. Greher, and Sarah Kuhn. 2009. Music performamatics: interdisciplinary interaction. *SIGCSE Bull.* 41, 1 (March 2009), 478-482.
[18] Falkner, N., Sooriamurthi, R., Michalewicz, Z. Teaching puzzle-based learning: development of basic concepts. Teaching Mathematics and Computer Science, 2012; 10(1):183-204
[19] McNerney, Timothy S. "From turtles to Tangible Programming Bricks: explorations in physical language design." Personal and Ubiquitous Computing 8.5 (2004): 326-337.