ALGORITHMIC MUSICAL IMPROVISATION FROM 2D BOARD GAMES

Dale E. Parson

Kutztown University of Pennsylvania
Department of Computer Science

ABSTRACT

Scrabble™-to-MIDI is a computer-emulated two dimensional board game that generates MIDI music from game rules and state as the game is played. Software modules include the board graphical user interface, rule engine and game state, a translator that maps game state to musical events, a second graphical user interface for manipulating translator configuration parameters, and a software synthesizer for rendering MIDI events. The plug-in translator and its configuration parameters comprise a composition. Improvisation consists of playing a unique game and of making ongoing adjustments to mapping parameters during play. Statistical distributions of letters and words provide a basis for mapping structures from word lists to notes, chords and phrases. While pseudo-random tile selection provides a stochastic aspect to the instrument, players utilize knowledge of vocabulary to impose structure on this sequence of pseudo-random selections, and a conductor uses mapping parameters to variegate this structure in up to sixteen instrument voices.

1. INTRODUCTION

Scrabble™-to-MIDI consists of a set of Java software modules that plug into a musical board game design framework. The primary goals of the framework are the creation of an interactive software environment in which non-musicians can learn important musical concepts such as meter, tempo, intervals and scales while playing a familiar game, and in which electronic musicians can investigate a novel, exploratory approach to algorithmic composition and improvisation. The framework is not tied to any particular two dimensional board game, and the development plan is the design of novel games intended for musical composition and performance.

Figure 1 illustrates the flow of data in the game-to-music software process. A player places a word on the Scrabble board, possibly forming new crosswords. The board transmits word lists to the mapping process. The mapping process may reside on a different computer in order to support concurrent play and manipulation of parameters by different people. The mapping process loads a textual configuration file at start-up time that includes definitions for static structures such as interval sequences for named scales / modes (e.g., "dorian," "minor pentatonic," etc.) along with initial values for mapping parameters such as tonic, accent patterns, tempo and instrument voices. Up to sixteen MIDI instrument voices support up to sixteen parameter sets for most parameters. For example, each instrument can have its own tonic, accent pattern, tempo, and so on. A human conductor can vary most parameters during a performance.

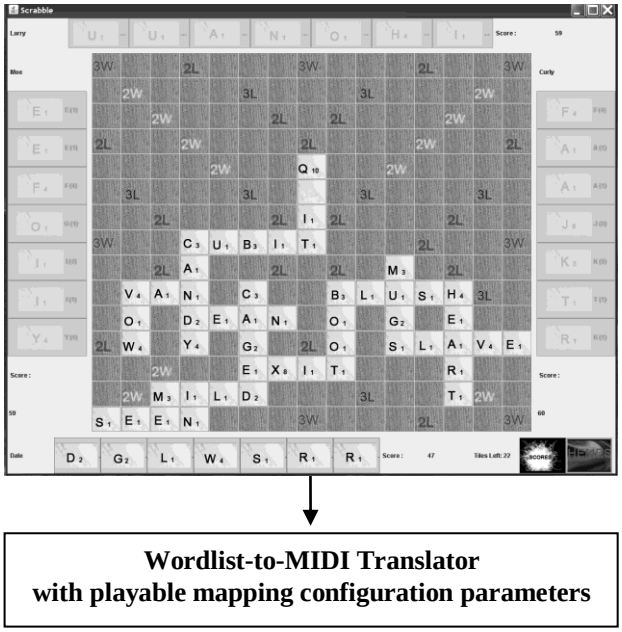

Figure 1. Game state data flow to the wordlist-to-MIDI translator after every move.

A specific plug-in translator determines the game properties such as word lists, letters on tiles, tile weights and player scores to use in translation. The translator used in the game of Figure 1 applies a *depth first search* of the words on the board, starting with the most recently placed word. The metaphor for this search is serial exploration of a maze, with the search algorithm treating each word on the board as a corridor in a maze. The translator explores the maze, mapping individual letters or groups of letters, up to a full word for a block chord, to groups of one or more simultaneous notes. Other search algorithms such as *breadth first search* allow generation of alternate phrasing.

The translator uses a set of configuration parameters to determine how to map letters on the tiles to MIDI instrument voices, notes and note properties. The translator of Figure 1 uses channel-specific parameters for up to 16 MIDI channels that include *patch* (instrument voice), *tempo* in beats per minute, *period* in 32nd note temporal intervals, simultaneous *number-of-notes* to sound together (0 to convert an entire word into a block chord), *sustain* period, *accent pattern* from which the meter derives, *tonic* as a MIDI note number, *scale* as a named sequence of MIDI note intervals from a configuration file, *arpeggiation* upper and lower bounds and step intervals in MIDI octaves from the tonic, MIDI *velocity* (amplitude) and velocity-sweeping patterns, stereo *panning* patterns and limits, and a global *volume* parameter. The combination of tempo, period, number-of-notes, sustain and accent pattern determines the number and rate at which the translator maps individual letters on game tiles to MIDI note on and off messages. The translator sends MIDI messages in a repeating loop to a software synthesizer included in the Java library [14] or to an external MIDI synthesizer.

Higher order structures derived from translator parameters include canons, polyrhythms, overlapping polyharmonies, and emulated two-channel "tape delays" [11] that configure two MIDI channels identically except for small, cyclic offsets in tempo. One approach to using polyharmonies is to configure MIDI channels to use overlapping but non-identical scales. Most compositions map frequently occurring letters such as vowels to consonant intervals such a tonic, third and fifth at assorted octave offsets, and map outlying consonants such as 'X' or 'Q' to outlying notes in the scale or to chromatic passing tones. These mappings are at the discretion of the composer. Configuring multiple MIDI voices to play a piece in several overlapping but non-identical scales causes harmonies to fly apart on the unshared scale portions and to come back together on the shared ones.

One way to achieve polyrhythms is by configuring MIDI channels to use overlapping accent patterns that are non-identical in length. An accent pattern such as 2:1:1:2:1:1:2:0 for example, giving an emphasis of ONE-and-two-AND-three-and-FOUR-(rest), consumes eight times the number of notes per harmonic interval configured for the MIDI channel before repeating. When that channel's translation process reaches the end of the game word list, if it finds itself midway through the accent pattern, it cycles through the word list again without interrupting the accent pattern. The end of the accent pattern and the end of the word list must eventually align. By giving different channels similar accent patterns that differ in lengths, the accented notes realign cyclically over traversals of the word list, yielding polyrhythms.

A conductor for an ensemble performance typically sets up initial configuration parameters in a file for the start of a performance, and conducts by varying these parameters while game players play. The conductor improvises over the musical space of the composition embodied by the plug-in translator. The timing of the changes entailed by player and conductor moves is determined by the translator software. The translator is essentially a multi-layer structural sequencer that builds MIDI sequences from game state as modulated by translation parameters.

2. RELATED WORK

Iannis Xenakis' works *Duel* and *Strategie* are two music games played by orchestral conductors responding to each other's moves as guided by a game matrix [16] that has recently been implemented using software and computer vision analysis of conductor moves [4]. While the present project was not influenced directly by Xenakis' work, his involvement in game theory and stochastic composition techniques resonate with the central aspects of this project.

Recent game-based compositions in ChucK include an improvisational ensemble piece based on multiple player slot machines [12] and a piece based on a video game [13]. The present system differs from these compositions by exposing music mapping configuration parameters to players for manipulation outside the scope of the game.

A notable commercial example is the Lumines™ interactive software game [5] that integrates a block placement puzzle program with visual effects and music that are synchronized to the game state. The present system extends the game-as-performance concept of Lumines to allow composers to create new plugins and to allow performers to improvise by manipulating configuration parameters directly, outside the scope of the game.

The immediate predecessor to the present system is the author's first-generation game for mapping piece-to-piece relationships in a chess game to music in a Just Intonation scale [7]. That system, which is written in Python and which uses sound generators written in ChucK [15], includes a 2D board game graphical user interface (GUI), a second GUI for manipulating configuration parameters, and multi-player distribution across a network. These properties have carried over to the present system.

The present system makes many significant advances over the chess-to-music system. The chess mapping strategy uses a breadth-first search algorithm from the two most recent moves, mapping piece-to-piece relations (passive and active attack and support relationships) modulated by piece value and inter-piece value differential directly to sound synthesis parameters. Chess piece relationships map almost directly to pitch and timbral qualities of generated sounds. Scrabble-to-MIDI, in contrast, utilizes statistical distributions of letters in a set of Scrabble tiles and words in the English language in mapping words to higher level, more complex phrase structures. Timbral properties in the present system are determined by per-channel MIDI patch numbers and optional external effects processing on the generated audio stream. The focus of this game-to-music research effort is

shifting away from sound generation towards algorithmic improvisation of interleaved phrases on multiple MIDI channels. The organized combinatorial space of intersecting and overlapping words in a Scrabble game is much larger than the basic space of relationships exploited by the chess-to-music program, although deeper analysis of chess relationships could certainly lead to more complex musical mapping. Emergent generation of higher order structures such as canons, polyrhythms, and polyharmonies was not possible with the chess system because configuration parameters related mostly to lower-level sound generation properties.

Codeorgan [1] is a website that generates music from text within any web page interpreted as a score. While it is not an interactive game and while its translation algorithm is hard coded with no performance configuration parameters, it does share with the current project the use of statistical distributions of text properties in generating music.

Chessynthesis [6] uses vision analysis of a physical chess game to generate synthesized music. Physical interaction with musical instruments is an important aspect of performance, and the present system is somewhat lacking in its sole reliance on mouse-driven computer graphics. This limitation was initially intentional, with the goal being portability and availability to as many players as possible without the need for special board or video hardware. Plans include extension of the user interface to a widely available touchpad screen for physical interaction.

3. DEVELOPMENT HISTORY

3.1. Game Development

Game development grew out of programming projects in a sophomore-to-senior undergraduate Java programming course. Students learned sufficient concepts regarding chromatic scales and MIDI note representations to allow them to write letter-to-note mappings. The initial student composition used depth-first traversal of the game word list to map entire words to block chords in a single MIDI piano voice. Once this basic software game-instrument was working, the author expanded the music translator to use the configuration parameters and musical structures outlined above while the students worked on graphical user interfaces for the game and the parameters. Enhancement of the translator by the author is an ongoing process. Both the source code and a browser-hosted Java applet are available on line [8]. A 30-minute solo performance by the author as part of a Summer 2009 international musical webcast is also available on line [9].

3.2. Game Demonstrations and Discoveries

The author has introduced several populations of players to this system, including high school students considering computer science as a field of study, musically inclined software engineers, and software inclined composers and musicians. A game distributed across two computers served as an interactive installation at college recruiting fairs and at an electronic music festival [3].

Players typically start out by concentrating on Scrabble playing. Players with some musical background are quick to begin manipulating plug-in configuration parameters. At some point a switch in focus by a waiting player to the configuration parameters occurs. Once that switch takes place, even musical novices can begin exploring concepts such as tempo, sustain, harmony, transpositions and phrase structure in a manner that is seamless with playing the game. Both solo and ensemble performance are possible, with performers taking turns manipulating translation parameters on the conductor laptop. There has been no opportunity yet for long term observation of a population of players to determine effect on musical skills. However, non-musician players are clearly learning musical concepts related to both composition and performance. Musical skill improves with practice, as with a conventional musical instrument. Plugins with configuration parameters designed as a music tutorial laboratory could be written to lead players through composition and performance concepts beyond those embodied in current parameters.

Expert Scrabble players have uncovered interesting aspects of the game that were not anticipated by the author. One expert player made her first move in a game-in-progress by putting down a word that created a tight cluster with three new crosswords. When the translator traverses these overlapping words, treating them as block or arpeggiated chords, it repeats the notes from the reused letters. The result is an interleaved polychord transition across a subset of shared notes that is difficult to achieve any other way in the game. The virtuosity of an expert Scrabble player translates to the music! Other players have developed game-playing strategies aimed at musical complexity at the cost of player scores. For example, placing odd-length words that have high least-common-multiple lengths in common with accent patterns generates more complex rhythmic permutations of the word-accent space. Placing outlying letters such as ‘X’ or ‘Q’ early in the game, often at the expense of waiting for higher scoring board positions, introduces appreciated harmonic variety. Players acquire such music generation skills through practice as they would with any instrument.

In Fall 2009 the author and several students performed a 30-minute composition at a public computer audio seminar at Kutztown University of PA that led to a second-page article with photographs in the local newspaper with subsequent web news service coverage [2]. This exposure has led to invitations to perform at a state-level conference of computer science educators in Spring 2010 [10] and at several smaller regional symposia.

Because of mass familiarity with crossword puzzles and Scrabble playing, the system lends itself to lecture-style demonstrations, interactive demos and installations.

4. RESEARCH DIRECTIONS

4.1. Incremental Enhancements

The author has been enhancing the wordlist-to-MIDI translator regularly. Recent additions include per-channel amplitude-sweep parameters, including shape, range and speed of amplitude sweeps around a center point, at the suggestion of a student musician. These parameters are patterned after stereo panning parameters added earlier by the author. Other recent enhancements include using several wordlist construction algorithms, alternated during play, that generate alternate phrases, and an undo/redo mechanism that allows players to *repeat* earlier, simpler phrases during later, more congested stages of a game.

4.2. Long Term Directions

The long term plan is to move away from existing games and create a multi-player distributed game intended for music and spoken word algorithmic improvisation. The most important contribution of the current project is the insight that it is possible to map regularities in one combinatorially complex but organized domain, that of overlapping letter and word distributions in a Scrabble game, to regularities in a combinatorially complex musical domain, with important human contributions to performance-time resolution of algorithmic variables. The planned game will continue to use written language character and word structural distributions in generating musical structural distributions, and it will support shared higher level structures, for example supporting shared syllables-across-words and shared words and phrases across higher level clauses, sentences and paragraphs. We hope to add grammar-based phrase restructuring of spoken word samples and human-like voice synthesis to the present instrumental-only music generation.

The author and students are also prototyping graphical interfaces for visualizing letter-to-note correlations as the notes play. Originally, the author envisioned a pinball-like GUI where letters on the board light up in colors as their notes play, but with up to 16 MIDI channels playing different letters at one time, the display became cluttered and confusing. We are currently exploring ideas for visualizing word-to-music correlation.

4.3. Acknowledgements

The undergraduate students of the Java Programming course at Kutztown University of PA from Fall 2008 through Spring 2010 have helped the author flesh out the potential of this system. Raphael Francis, Ryan Huber, Jeremy Parson, Steve Solomon and Mike Williams in particular have been very helpful in suggesting and implementing new ideas and in participating in performances. The Kutztown University Research and Professional Development Committees and the Computer Science Department have assisted the author in procuring research / performance equipment.

5. REFERENCES

[1] Codeorgan website, http://www.codeorgan.com/, April, 2010.

[2] G. Cuyler, "KU students spell ingenuity: musical Scrabble," Reading Eagle, Reading, PA, Oct. 2, 2009, reprinted in McClatchy-Tribune Info Services http://www.tmcnet.com/usubmit/2009/10/02/4403193.htm .

[3] Electro-Music 2009 Festival, October 29-31, 2009, Bloomingdale, NJ, http://event.electro-music.com/.

[4] M. Liuni and D. Morelli, "Playing music: an installation based on Xenakis' musical games," *Proceedings of the Working Conference on Advanced Visual Interfaces*, Venezia, Italy, May, 2006.

[5] Lumines™, http://lumines.jp/, January, 2010.

[6] M. Marion, "Chessynthesis: interactive installation with realtime sound generation based on chess," http://maniaxmemori.net/chessynthesis.html, 2007-2008.

[7] D. Parson, "Chess-based Composition and Improvisation for Non-Musicians," *Proceedings of New Interfaces for Musical Expression (NIME) 2009*, Pittsburgh, PA, June 4-6, 2009.

[8] D. Parson, computer music web page, April, 2010, http://faculty.kutztown.edu/parson/music/ .

[9] D. Parson, "Music for 2 to 4 Scrabble Players," performed on June 20, 2009 on http://radio.electro-music.com, http://electro-music.com/forum/phpbb-files/em_ss09_20june2009_parson_132.mp3 .

[10] PA Computer and Information Science Educators Conference, April, 2010, http://www.pacise.org/.

[11] Steve Reich, *Writings on Music 1965-2000*, Chapter 1: Early Works. Oxford University Press, 2002.

[12] S. Smallwood, "On the Floor," *PLOrk: Live at Richardson Auditorium*, http://plork.cs.princeton.edu/listen/richardson/.

[13] S. Smallwood and G. Wang, "Chuck Chuck Rocket," *PLOrk in the Round*, May 2, 2006, http://plork.cs.princeton.edu/listen/green/.

[14] Sun Microsystems, javax.sound.midi documentation, http://java.sun.com/javase/6/docs/api/index.html, January, 2010.

[15] G. Wang, *The Chuck Audio Programming Language*, Ph.D. dissertation, Princeton University, 2008, http://www.cs.princeton.edu/~gewang/thesis.html.

[16] I. Xenakis, *Formalized Music: Thought and Mathematics in Composition*, Second Edition, Pendragon Press, 2001.