Transforming 8-Bit Video Games into Musical Interfaces via Reverse Engineering and Augmentation

Ben Olson
Signal Narrative
Madison, WI, USA
ben@signalnarrative.com

ABSTRACT

Video games and music have influenced each other since the beginning of the consumer video game era. In particular the "chiptune" genre of music uses sounds from 8-bit video games; these sounds have even found their way into contemporary popular music. However, in this genre, game sounds are arranged using conventional musical interfaces, meaning the games themselves (their algorithms, design and interactivity) play no role in the creation of the music.

This paper describes a new way of creating music with 8-bit games, by reverse engineering and augmenting them with run-time scripts. A new API, Emstrument, is presented which allows these scripts to send MIDI to music production software. The end result is game-derived musical interfaces any computer musician can use with their existing workflow. This enhances prior work in repurposing games as musical interfaces by allowing musicians to use the original games instead of having to build new versions with added musical capabilities.

Several examples of both new musical instruments and dynamic interactive musical compositions using Emstrument are presented, using iconic games from the 8-bit era.

Author Keywords

Computer Music Controllers, Game Modding, Emulation, Audio Programming, Active Score Music, Algorithmic Composition

ACM Classification

• **Applied computing—Sound and music computing** • **Applied computing—Computer games** • *Social and professional topics—Software reverse engineering*

1. INTRODUCTION

Video games and music have long been intertwined. In addition to having a musical soundtrack, many games have rhythmic or compositional elements to them [13]. Video games, as a large part of today's culture, have in turn influenced music. The chiptune genre of music involves crafting music using sound chips from old game consoles, either to invoke nostalgia or as a way to follow a minimalist aesthetic [12]. Many musicians today across all genres use game hardware or software simulations to augment their music with these sounds.

Chiptune music, however, does not use actual games in its composition or performance, even if retro games are evoked in the listener's mind. Incorporating these games into the musical process could lead to many new possibilities, for reasons both artistic and technical.

Artistically, creating music by interacting with a game adds a unique visual component to a traditionally non-visual experience. Furthermore, there is a lot of cultural value in 8-bit games. Many artists' work features a game-influenced "8-bit" aesthetic [7], and countless mass media is game-influenced, including a recent multi-million dollar film, *Pixels*. The personal value of these games to many listeners adds another dimension to the art.

From a technical viewpoint, using retro games as musical interfaces leads not only to new ways to create music, but also musical creations that have "character" based on the algorithms, design and interactivity of the original games. They present user interfaces that are widely accessible to non-musicians and indirectly communicate the internal workings of the game via sonification.

This paper will look at one approach to incorporating games into the creation of music: using run-time scripts to augment them with MIDI output, for integration into computer music workflows. This extends prior efforts to convert games into musical interfaces by allowing musicians to convert closed-source, seemingly non-extensible games using simple scripting. The scope of this work is only emulated 8-bit games, but the same principles can be used for later games, albeit at greater difficulty.

Figure 1: Super Mario Brothers keyboard instrument. The key being pressed is highlighted

NIME'16, July 11-15, 2016, Griffith University, Brisbane, Australia.

2. PRIOR WORK

There is plenty of existing work in the area of repurposing non-musical games for musical expression. One such example is ChucK ChucK Rocket [15], which recreated the puzzle game ChuChu Rocket as a multiplayer musical canvas. However the approach of rebuilding a game from the ground up is time-consuming; in addition the recreated game will not behave exactly the same as the original game, possibly altering the musical output. A work closer to this paper is UDKOSC, which allows musicians to create musical interactions in games based on the Unreal engine [9]. However, UDKOSC and other similar projects [5, 8] require a game or game engine that is open-source or designed to be modifiable; this precludes using any games designed for a closed game console. This paper describes a framework allowing musicians to transform non-extensible console games into musical interfaces, using their original binary code and algorithms. This unlocks a wide variety of games for musicians to use as expressive interfaces.

On the non-musical side, there is a lot of relatively recent work which makes this paper's ideas possible. The reverse engineering work of ROM hackers and speed-runners has demystified a lot of the internal workings of early console games [3]. Their work also led to the inclusion of scripting capabilities into many video game console emulators [1]. These emulators have the capability to run scripts which interact with the game as it is played. The scripts are generally used to augment the game in ways that cannot be accomplished by modifying the game binary (e.g real-time mouse input). The ability to use scripting to draw additional graphics in a game is used extensively by the musical interfaces demonstrated in this paper.

3. IMPLEMENTATION

In order to implement the ideas described above, a new Lua module, Emstrument (a portmanteau of 'emulator' and 'instrument'), was created. Its API allows scripts running in video game console emulators to send MIDI messages to external audio software, allowing games to be converted into musical interfaces. Lua was chosen as the API language as it is supported by many popular emulators [1] for its compact, portable design and ease of use [11]. As a Lua module, installation simply consists of downloading the binary (`emstrument.so`) to the Lua install directory; no linking or recompilation is required for use with an emulator. Emstrument currently only outputs MIDI messages and does not generate or manipulate in-game sounds; this was a conscious decision to allow electronic musicians to use their existing MIDI-compatible software and hardware.

Building a musical interface with Emstrument is fairly straightforward. A script is written that observes some values in the console's RAM while the game is played. These values correspond to various in-game properties, such as the player's location, score, level, etc. When those values change, MIDI messages such as notes, control changes, and pitch bends can be programmatically sent over the 16 MIDI channels to control audio software. Utility scripts which do not interact with a game can also be written for uses such as mapping MIDI controls to audio parameters.

3.1 API Design

The Emstrument API was created for ease of use and to work specifically with scripting systems that iterate once per video frame. It queues MIDI messages, which are consolidated and sent all at once, for precise timing and to eliminate redundant or conflicting messages (e.g multiple note off messages per note on). To do this, Emstrument keeps

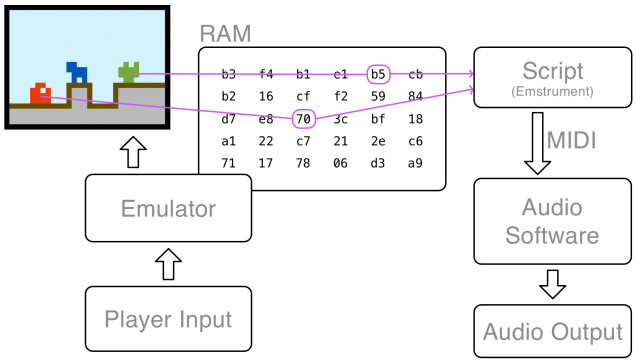

Figure 2: Emstrument data flow

extensive internal state; it is not simply a thin wrapper for low-level MIDI functions.

The API functions are designed to make scripts concise and readable. `noteonwithduration()` sends a note on message, then sends the corresponding note off message after the desired duration, so the script is not bogged down with timing code. The `allnotesoff()` function turns off all the notes currently active on a channel (and only those notes), preventing the need to keep track of which notes are playing[1]. The `notenumber()` function translates a string into a MIDI note value, allowing for code which can easily be read as chords and melodies. The rest of the functions are described in the API documentation found at the link in Appendix A.

3.2 Game Reverse Engineering

In a vacuum, the most difficult part of using Emstrument would be reverse-engineering games to find the RAM addresses of in-game properties. However, this is not usually difficult in practice for a few reasons:

- In modern computing the address of any value in RAM is dynamic and potentially changes every time the program is run. This makes variable detection an active area of research [4]. However, 8-bit video games generally do not perform any dynamic memory allocation. All the variables are stored in the same places every time the game is run, for performance and because there is no need for code portability.

- Much of the work of finding the RAM addresses of these values has been done by the ROM hacking and speed-running communities [3]; RAM maps for many games are easily found online. There are several resources for doing further reverse engineering [2].

- In the case where a RAM address for a value is unknown, a secondary value often can be used. For example, an increase in the player's score can be tracked instead of a change in an undiscovered "power-ups acquired" value.

4. MUSICAL INTERFACE EXAMPLES

The following sections discuss the design of 4 selected game modifications which use Emstrument. Two are game-based single musical instrument controllers, which leverage expert technique (gameplay expertise), one of Cook's design principles for computer music controllers [6]. There are also

[1]The MIDI channel mode message for "All Notes Off" was not universally supported on tested audio software, so it was implemented manually

two multi-instrumental interactive compositions, following the principle of "Create a piece, not an instrument". These scripts are all under a few hundred lines of code and show promising results. There were not any great difficulties in translating musical ideas into code using Lua and Emstrument.

4.1 New Musical Instruments

4.1.1 Super Mario Brothers - Keyboard

Super Mario Brothers is an iconic early 2D run and jump game. This script superimposes a diatonic keyboard onto the bottom of the screen, which is played by jumping on keys at different horizontal positions (Figure 1). The instrument resembles a fully digital version of Andante [16], a system overlaying walking figures over a keyboard[2].

The keyboard covers 2 repeating octaves, enough to play simple melodies. As the player travels right in the game world, the keyboard scrolls left, making it feel like an object in the game's virtual world. The virtual physicality of the keyboard leads to ideas for further physical interactions (e.g ducking the character to trigger aftertouch).

The instrument works for both percussive sounds (piano, marimba) and sustained sounds (strings, pads), as the player can hold a note by standing in the same place after triggering a note. Early versions of the instrument also triggered notes when the player walked over them, but this resulted in reduced control and mechanical-sounding runs of notes.

Playing the instrument requires some practice, just like learning the game for the first time, since making precise jumps and landings requires adjustment to the game's internal physics.

4.1.2 Arkanoid - Drum Machine

Arkanoid is a clone of Breakout which is converted here into a percussion generator. Standard MIDI drum notes are triggered when the ball changes direction in response to hitting various objects. The player has some control over the drum pattern by directing the ball in different directions by hitting it with different parts of the paddle.

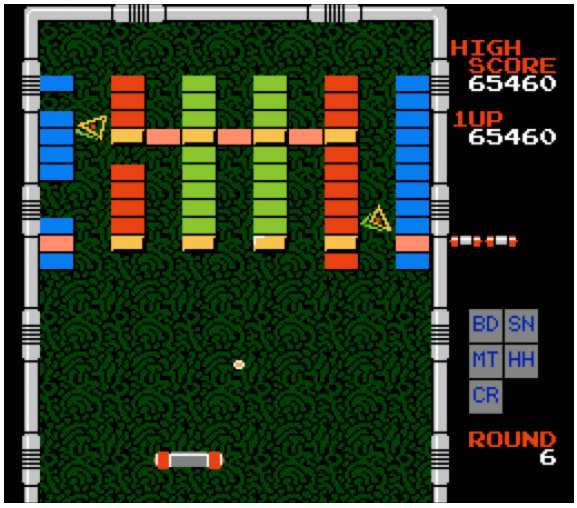

Figure 3: Arkanoid drum machine. Triggered notes are drawn on the right below the score

One challenge faced when converting games into musical interfaces is keeping a regular rhythm, as in-game events occur at arbitrary times. That is accomplished here by slightly quantizing the events, as is done in many rhythm games [13].

For this instrument a further game modification was added to always keep the ball in play to keep the drum pattern going. The result is the ball continually increasing in speed creating more and more intense rhythms, not unlike a drummer playing a solo. This emergent behavior gives the instrument character, giving life to code written decades ago.

[2]This was not discovered until after the instrument was implemented

4.2 Interactive Musical Compositions

Emstrument can be used to play many instruments simultaneously to create full compositions. This is accomplished by sending messages on multiple MIDI channels (one for each voice), and filtering the messages at the endpoints. These algorithmic compositions are created by the interaction of the algorithms in the script and the algorithms in the game; they thus can be described as inter-algorithmic compositions.

These pieces can be recorded as audiovisual art, or played live, transforming the game into what is referred to as active score music [13].

4.2.1 Tetris - Multi-Sequencer Composition

Tetris is a seminal puzzle game where falling blocks are arranged by the player, who tries to avoid filling up the game board by clearing horizontal lines. Using Emstrument the game board becomes the input for 2 independent sequencers, which trigger a bassline, drum pattern, and lead synth based on the content of each of the 10 columns (resulting in the odd meter of $\frac{10}{4}$ or $\frac{5}{4}$).

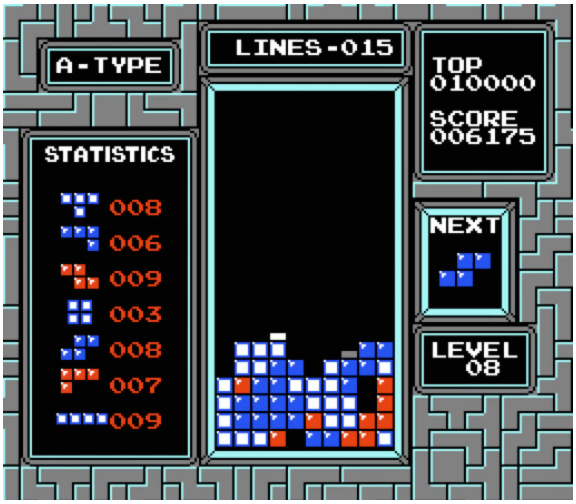

Figure 4: Tetris multi-sequencer. The 2 sequencer playheads are drawn above the blocks

At the start of a game, the bass and lead synth play a single note corresponding to the bottom of the game board. As the board fills up, they start playing notes corresponding to the top block of each column. When the player creates a difficult situation by introducing a gap in a column (where a horizontal line can no longer be formed), that column then triggers percussion sounds. As the game proceeds and the player inevitably stacks the columns higher and higher, the music becomes more intense to match the game's increasing intensity. Various techniques are used to keep the music dynamic and reduce repetition, including altering sequencer behavior when lines are cleared.

This composition is an example of how a new auditory experience can change the way a game is played. Instead of keeping the board as clear as possible as they would in the original game, the player is tempted to make "mistakes"

to introduce new rhythms and sounds, while still trying to avoid filling up the board completely.

4.2.2 Pac-Man - Generative Soundtrack[3]

Pac-Man is a maze game where the player is pursued by 4 ghost characters. The game's carefully written AI algorithms cause tension to build as the ghosts chase the player, then release when the ghosts (sometimes) retreat. To reflect that tension, in this composition each ghost controls a synthesizer voice. These voices are filtered based on the player's distance from each ghost. Thus, in dangerous situations when multiple ghosts are near the player, their notes combine in dissonant ways to create tension. To fill in musical space, other sounds are played when the player eats various objects, and to avoid repetition the various notes and sounds are changed based on player and ghost directions and locations across different levels.

The end effect is that the playful chirping sounds found in the original game are replaced by ominous bass-heavy and distant sounds, darkening the mood of the game. Future game-based generative compositions could be used to further explore the connection between audio and how a game is experienced.

5. SUMMARY

This paper describes Emstrument and 4 game modifications made possible with it, as a starting point in creating music using the algorithms, design and interactivity (rather than the sounds) of 8-bit video games. Using Emstrument, music can be created using games unexplored by prior research in repurposing games for musical expression.

These musical interfaces create synesthetic experiences where the visuals illustrate the music in ways non-musicians can understand, as well as communicating the underlying architecture and behaviors of the game via rhythm and harmony. In addition, they create a new form of gameplay in which the player not only has to accomplish their in-game goals, but do so in a "musical" way (e.g playing sub-optimally for better musical results). Playing music written for conventional instruments with these interfaces is not easy, due to the player's limited control over the game, forcing the creation of new unconventional forms of music.

Beyond any other consideration, these new musical interfaces are fun to use, as "Everyday objects suggest amusing controllers" [6].

6. FUTURE WORK

First and most importantly, work needs to be done exploring the usability of these new musical interfaces by external users, as a study could not be carried out while this project was in development. Musicians and composers familiar with gaming could provide insight into how these game-derived interfaces either succeed or fail at allowing them to express their musical ideas in new ways, and help further develop the core concept.

Beyond that, there are a few straightforward ways to extend Emstrument possibly worth investigating:

- Currently, Emstrument runs in a static script, but support for persistent state could be added to allow for live coding performances, where the script is updated in real time. The computation involved in emulation is light enough to be done in a web browser and could perhaps work in conjunction with a browser-based live coding environment like Gibber [14].
- Emstrument is used to generate sound but does not take it as input. Adding MIDI or audio input to Emstrument would allow even further integration between music and game. Using MIDI input to influence a game was previously investigated in a modified version of Asteroids [10], but the game itself was not used as a musical interface.

[3]This Emstrument script does not have an added visual component, so a screenshot is not shown in this paper.

7. REFERENCES

[1] TASVideos / Lua Scripting. `http://tasvideos.org/LuaScripting.html`. Retrieved 2016-04-15.

[2] TASVideos / Reverse Engineering. `http://tasvideos.org/ReverseEngineering.html`. Retrieved 2016-04-15.

[3] N. Altice. Tool-assisted console emulation and platform plasticity. `http://metopal.com/2011/12/06/tool-assisted-console-emulation-and-platform-plasticity/`, November 2011. Retrieved 2016-01-20.

[4] G. Balakrishnan and T. Reps. DIVINE: DIscovering Variables IN Executables. In B. Cook and A. Podelski, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 1–28. Springer Berlin Heidelberg, 2007.

[5] M. Cerqueira, S. Salazar, and G. Wang. SoundCraft: Transducing StarCraft 2. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 243–247, Daejeon, Republic of Korea, May 2013. Graduate School of Culture Technology, KAIST.

[6] P. Cook. Principles for designing computer music controllers. In *Proceedings of the 2001 Conference on New Interfaces for Musical Expression*, NIME '01, pages 1–4, Seattle, Washington, 2001.

[7] M. Grethe and C. Andrew. Videogame music: chiptunes byte back? In *DiGRA '07 - Proceedings of the 2007 DiGRA International Conference: Situated Play*. The University of Tokyo, September 2007.

[8] R. Hamilton. q3osc: or How I learned to stop worrying and love the game. In *Proceedings of the International Computer Music Association Conference*, Belfast, Ireland, 2008.

[9] R. Hamilton. UDKOSC: An immersive musical environment. In *Proceedings of the International Computer Music Association Conference*, Huddersfield, United Kingdom, 2011.

[10] J. Holm, J. Arrasvuori, and K. Havukainen. Using MIDI to modify video game content. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 65–70, Paris, France, 2006.

[11] R. Ierusalimschy, L. H. de Figueiredo, and W. C. Filho. Lua—an extensible extension language. *Softw. Pract. Exper.*, 26(6):635–652, June 1996.

[12] I. Márquez. Playing new music with old games: The chiptune subculture. *Games as Art, Media, Entertainment*, 1(3):67–79, 2014.

[13] P. Martin and K. Fares. Levels of sound: On the principles of interactivity in music video games. In *DiGRA '07 - Proceedings of the 2007 DiGRA International Conference: Situated Play*. The University of Tokyo, September 2007.

[14] C. Roberts and J. Kuchera-Morin. *Gibber: Live coding audio in the browser*. Ann Arbor, MI: MPublishing, University of Michigan Library, 2012.

[15] S. Smallwood, D. Trueman, P. R. Cook, and G. Wang. Composing for laptop orchestra. *Computer Music Journal*, 32(1):9–25, 2008.

[16] X. Xiao, B. Tome, and H. Ishii. Andante: Walking figures on the piano keyboard to visualize musical motion. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 629–632, London, United Kingdom, 2014. Goldsmiths, University of London.

Appendix A: Emstrument Links

Source code, including scripts: `http://github.com/ben-signalnarrative/emstrument`
Demo video, showing game-based instruments and compositions: `http://vimeo.com/150221247`
API documentation: `http://github.com/ben-signalnarrative/emstrument/blob/master/documentation/doc.md`
Scripting tutorial: `http://github.com/ben-signalnarrative/emstrument/blob/master/documentation/tutorial.md`