# Procedurally Generated, Adaptive Music for Rapid Game Development

Timothey Adam, Michael Haungs, Foaad Khosmood
Dept. of Computer Science
California Polytechnic State University
1 Grand Avenue
San Luis Obispo, California
tiadam@calpoly.edu, mhaungs@calpoly.edu, foaad@calpoly.edu

## ABSTRACT

Audio design is an important aspect of game development which may be neglected in time-limited rapid prototyping game creation events. In such environments, members of small development teams often multitask or switch roles, but they may not possess the necessary time, resources or skills for original music compositions. In this paper, we present AUD.js, a system developed for procedural music generation for JavaScript-based web games. By taking input from game events, the system can create music corresponding to various Western perceptions of music mood. The system was trained with classic video game music. Game development students rated the mood of 80 pieces, after which the Markov Chains of those pieces were extracted and added into AUD.js. AUD.js can adapt its generated music to new sets of input parameters, thereby updating the perceived mood of the generated music at runtime. We conducted a user study during Global Game Jam 2014 at Cal Poly. We find that while the quality of the audio is lower than hand-picked or composed pieces of music, AUD.js is capable of being a useful and cost-saving tool for game developers working with constraints.

## Categories and Subject Descriptors

J.5 [**Arts and Humanities**]: Music; K.8 [**Personal Computing**]: Games; H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing

## Keywords

Procedural Composition, Music Mood Adaptation, Game Jams, Computer Music, Emotion

## 1. INTRODUCTION

Procedurally generated music has been a topic sparsely researched but frequently revisited. Early work, like Iannis Xenakis' Illiac Suite [11], is mostly limited to simple probabilistic methods, such as Markov chains and probabilistic grammars [6]. Other areas of research include exploring topics from mathematics and artificial intelligence, like the use of fractals and genetic algorithms. For example, machine learning has been used to generate jazz music [8]. Some systems even combine multiple techniques, like genetic algorithms and Markov chains [2]. Recent research, like the work of David Cope, seeks to model creativity when generating music in an attempt to have computers generate pieces that sound as if they were composed by a human [4].

Procedural audio has previously been used in games, though not for generating full melodies. Game audio needs to meet the burden of being pertinent and impactful towards gameplay, something procedurally generated audio rarely achieves [3]. Instances of procedural music in games are usually only to modify or adapt the composed music to avoid sounding repetitive when listened to many times [3]. A solution for this problem is using complex logic to create the music, but very few projects have the budget to dedicate to engineers for these purposes [3]. Current algorithms for procedural music generation typically also have the issue of being less appealing aesthetically than composed pieces. Unless the chosen system can interact on a raw signal or even a sample level, the music would doubtfully sound composed. That kind of low-level interaction is costly, both from a real-time performance and a financial point of view [5].

During game jams [1], audio integration can be challenging. It is rare for a team attempting to create a game in the span of a few days to have a dedicated composer. Even if the team does, due to the stressful nature of the event, coordination between the composers, audio designers, and the programming team can be sparse. This can lead to subpar audio if the team cannot find a way to dedicate a significant amount of time to audio development and integration. In addition, having a musician or sound designer on the team doesn't necessarily lead to good integration with the game's tempo, logic, or user input. An example of a solution to this problem is 2013's audiodraft.com "Create Music for Game Jam Team 2" contest, where participants from all over the world could submit music to be used in game jams [1]. Even if a Game Jam team has outstanding audio, that audio is likely static and unchangeable at their game's runtime. Trying to

[1]A game jam is a gathering of game developers for the purpose of planning, designing, and creating one or more games within a short span of time, usually ranging between 24 to 48 hours.

make the static music adaptive would also likely break the flow of the track, thereby detracting from the quality of the music experience. Truly adaptive music, which changes with the mood of the game, would require an exponentially larger commitment from the composers as well as careful planning of the game events. Those both can be problematic in a Game Jam setting.

This paper presents AUD.js, a tool for procedural music generation based on an input 'mood', and capable of adapting the music to different moods at runtime. The system provides a small but powerful API, making it capable of accelerating game development. This would reduce pressure on developers, and the mood-based approach helps the system be applicable to multiple genres and types of games.

## 2. AUD.JS

This section presents a description of the input elements and API of AUD.js such that potential developers could understand how to use this software.

### 2.1 Input Model

Previous work has examined the mood of music by analysis on several spectra, including loudness of the music, tempo, mode, harmonic complexity and variety of timbre. [7]. Based on these qualities, the music can be placed in an "emotion space" - a visual representation of human emotion. One such representation is the Russell mood model [10], which categorizes emotion in terms of two axes: arousal & valence. Other terms that have been used for these axes include {active/passive} & {positive/negative}, activity & pleasure or stress & energy in the Thayer model of mood [9].

Since AUD.js seeks to generate music based off of a representation of music mood, the generation process starts with the user entering five parameters (see figure 1):

1: Stress is the first component of the model of mood this system uses[2] and typically governs the level of harmony and dissonance of the music.

2: Energy is the second component of the model of mood this system uses[2], and typically governs the rhythm, tempo, and pace of the music.

3: The seed ensures a unique, re-creatable piece is generated as it seeds the random number generator which governs the random aspects of the generation process. Each unique seed will result in a unique sequence of random numbers used to choose notes in the melodies, thereby resulting in a unique piece.

4: The number of patterns indicates the number of unique 4-measure strings of notes should be generated. Each measure contains up to 8 eighth notes.

5: The repeat of patterns indicates how many times a generated 4-measure string of notes should repeat. Longer repeat values will result in minor variation in the notes of the repeated patterns.

[2]See in general: Livingstone & Brown (2005)

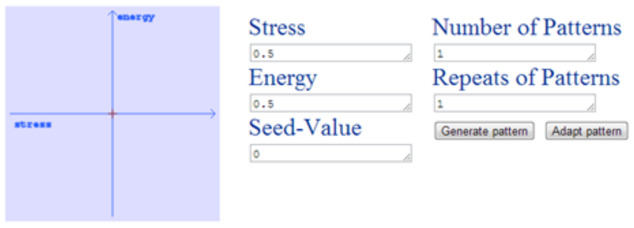

**Figure 1: AUD.js input UI elements.**

### 2.2 Software Structure

AUD.js is a browser-based framework built on Webkit Audio. This allows AUD.js to populate the audio buffers used by the browser to play sound, and manipulate that raw audio. The AUD_interface module reads AUD.js' generated music, and converts it to digital audio as needed (indicated by Webkit Audio's node.onaudioprocess callback function calls). The interface creates the audio signal at runtime, thereby allowing the music to adapt when the melodies in AUD.js change. When a change in AUD's generated music occurs, the AUD_interface linearly interpolates between the old melody and the new melody over approximately one second (40000 samples, where the typical sample rate is 44100 samples per second).

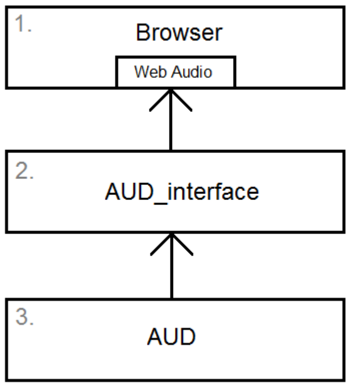

**Figure 2: AUD.js software design. 1: The web browser on which AUD.js runs. 2: The interface module that converts arrays of notes (generated melodies) into digital signals, to be played through the browser. 3: The AUD module itself.**

### 2.3 API

AUD.js provides an API for other developers to include the functionality of this system in their own applications. This API consists of:

**Table 1: AUD.js API description.**

| | |
|---|---|
| generatepattern (stress, energy, numberpatterns, repeatspatterns, seed) | Starts the generation process from the beginning: all the parameters are reset and the piece that's playing gets overwritten and reset to the beginning of the new piece. |

| | |
|---|---|
| adaptpattern (stress, energy) | Adapts the current piece to a new mood by regenerating the entire piece, but not changing the length or the seed. The random number generator is reset, thereby ensure the same piece is regenerated with the new given mood. The net effect is that the piece keeps playing, except it will now have a perceivably different mood (provided the new stress and energy are sufficiently different as to warrant a perceivable difference). |
| togglepause() | Pauses the song if it's playing or resumes it from where it was paused. |
| toggleplay() | Resets the song to the beginning and either pauses it or resumes is depending on whether it was playing. |
| isplaying() | Returns whether the piece is playing as a boolean value. |
| setvolume (newvolume) | Sets the volume of the system's outputted music to the new given value (it should be between 0 and 1, where 0 is muted and 1 is the maximum volume possible). |

A full tutorial on how to use AUD.js is provided at *timotheyadam.com/AUD*.

## 3. IMPLEMENTATION

This section explains the algorithm AUD.js uses to generate music based on mood and AUD.js' software structure.

### 3.1 Music Mood Representation

Once the input is received, a base note is selected. The base note is a randomly chosen number between 12 and 24, representing the C1 to C2 notes (the system's full range is C0 [0] to C8 [96], representing frequencies of 16.35 Hz to 4186 Hz). Individual instruments can be up to 5 octaves higher: all instruments share the same base note phase shifted to different octaves.

Next, a map is constructed. The map is a two dimensional array of floats, akin to a Markov Chain[3]. This map indicates note prevalence chances starting at the base note and ending at eleven half steps above the base note, thereby representing probabilities of any of the 12 uniquely named notes occurring with any of the twelve notes. This map is generated by linearly interpolating the stress and energy values between maps corresponding to all possible combinations of low, moderate, and high stress and low, moderate and high energy. As shown in figure 1, stress and energy can be plotted on a two dimensional square. For interpolation purposes, low stress/energy corresponds to 0, moderate stress/energy corresponds to 0.5 and high stress/energy corresponds to 1.

When generating, to linearly interpolate between maps, the stress and energy values are placed in the appropriate quadrant of a stress-energy grid (see figure 1). The 4 closest maps are averaged together based on their proximity to the stress and energy values: the closer the map is to the desired stress and energy values, the more contribution it will have on the resulting map.

[3]A Markov Chain is a state machine where each state transition has an associated percentage change. All outgoing transitions from any state have probabilities totaling 100%.

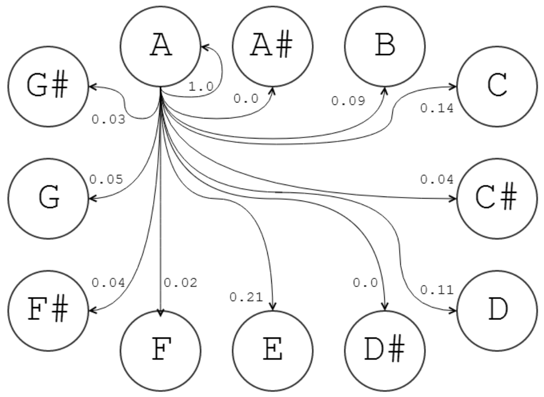

**Figure 3: One of 12 entries in the map, showing the likelihood of an 'A' note being played with any other named note.**

#### 3.1.1 Music Mood Data

The maps used for interpolation were extracted by parsing note prevalence of classic video game soundtracks. A total of 80 pieces were rated by first-year game developers (potential end-users of the system) as having either low, moderate or high stress and low, moderate or high energy on a scale from 1 to 5. The average stress is 2.9 / 5 and the average energy is 3.3 / 5, meaning high energy is slightly overrepresented (see figure 4). Each piece was rated by at least 30 people, and a total of 93 people participated. For a full list of pieces used, see Appendix 1. These studies were conducted by the authors on December 3, 2013 and February 18, 2014 during Cal Poly's Intro to Game Design and Intro to Interactive Entertainment resprectively.

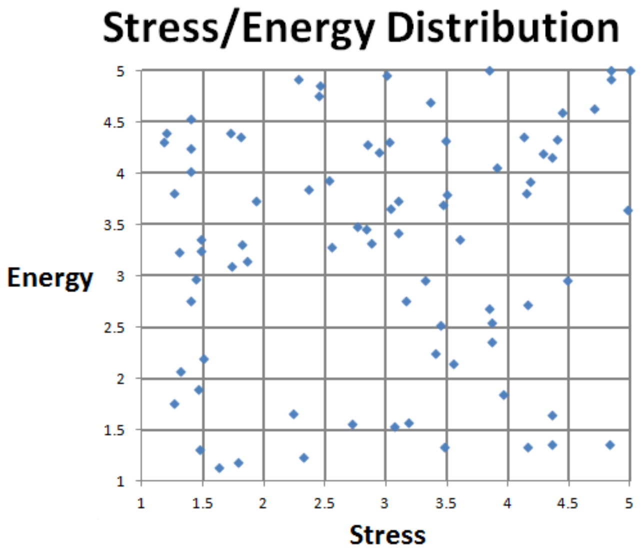

**Figure 4: The distribution of the rated value of each piece used in training AUD.js.**

## 3.2 Fractal Maps

The next step of the generation process is creating a percussion pattern based on the energy. Percussion in AUD.js takes the form of a short hit of a noise wave, representing a snare drum or kick, and a longer, fading out noise wave, representing a high hat. The higher the energy level, the faster the attack and decay of both will be. The percussion is randomly generated according to a fractal probability, given a 32-tick string to fill (4 measures or 8 eighth notes each):

- Extremely likely to play percussion on multiples of 8
- Highly likely to play percussion on multiples of 4 (that aren't multiples of 8)
- Moderately likely to play percussion on multiples 2 (that aren't multiples of 8 or 4)
- Not very likely to play percussion on multiples of 1 (that aren't multiples of 8, 4 or 2)

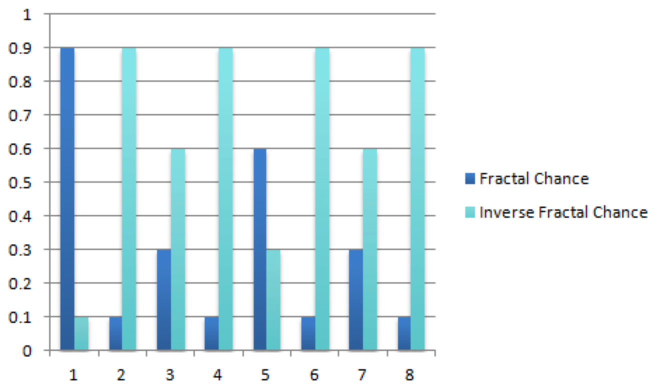

**Figure 5: Fractal probabilities over 8 ticks.**

The likelihood of percussion being played is amplified by energy: higher energy means more percussion. A percussive hit will occur if the following expression is true:

*random number + energy > [percentage calculated from fractal map]*

## 3.3 Creating Melodies (Note by Note)

The next step in the generation process is to create tracks of melody. AUD.js currently supports 4 unique tracks, each capable of producing 4 different length notes (1,2,4 or 8 eighth notes long). Each one is randomly assigned a waveform: triangle wave, sawtooth wave or square wave. The envelopes for their notes are dependent on energy (like with percussion, higher energy means sharper attack and decay, whereas lower energy means more fade-in and fade-out). To determine when a track should play a note, for each note length, it checks the same formula as percussion does, but with increased probability if there was a percussive hit:

*random number + energy > [percentage calculated from fractal map] – [percussive hit value]*

When a track needs to play a note, the note is chosen based on the earlier generated map and other notes that are playing.

A set of notes will be created that contains all notes that are valid to be played (given the base note and other notes that are playing). For a note to be valid, its entry in the Markov Chain map must be greater than the fractal cutoff for each note in the set it's being compared against:

*Candidate note valid if:*

$map[candidatenote][comparisonnote] > [fractalpercentage]$

This gives a list of candidate notes, from which one is selected based on the prevalence table probabilities.

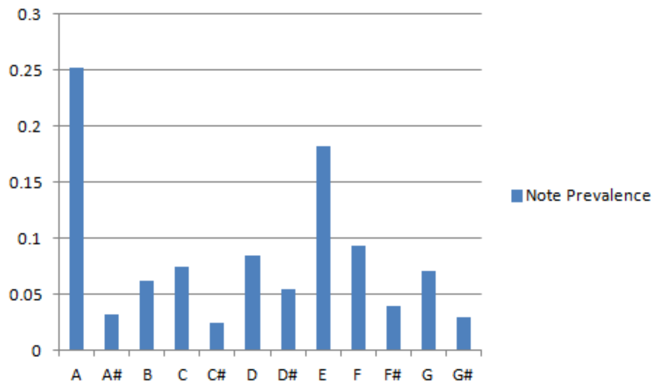

**Figure 6: Note prevalence given that the base note is an 'A'.**

This process is repeated until the 32-tick string is filled.

## 3.4 Other Aspects

To ensure that the generated music doesn't sound too cacophonous, AUD.js will randomly prevent tracks from generating notes for a 32-tick string, thereby reducing the amount of instruments playing at once at any time. This is also dependent on energy: with higher energy, more tracks will be playing simultaneously.

To create repeating motives, AUD.js may randomly forward a generated pattern ahead to a future pattern of the full piece. This ensures the 32-tick string in the track is reused when the generation process gets to that particular future string. This will only happen if there is sufficient space to move a pattern forward (i.e. the generative process is not nearing the end of the desired piece's length).

To provide slight variation in repetitions of a single string, AUD.js may randomly change some notes when copying over strings into following strings. This depends on the inverse fractal (see figure 5): it's very unlikely to change something on a multiple of 8 out of 32 ticks, but highly likely to change a note on the multiples of 1 (that aren't multiples of 2, 4, or 8). The new note is chosen using the same method as listed above. This process will only be applied if the user selects to have more than patterns repeat more than once (i.e. 32-tick strings will need to be copied to perform the repeat).

## 4. EXPERIMENT

This section presents the hypothesis and expirement for assessing AUD.js' effectiveness.

### 4.1 Hypothesis

With the following experiments, we will try to ascertain that AUD.js effectively provides musical backing for a video game. We will also attempt to show that use of a procedural music generation software library augments the development of a game during a game jam.

### 4.2 Method

AUD.js' evaluation took place at Cal Poly's 2014 Global Game Jam site in San Luis Obispo, California.

During the Jam, participants had the option to use the AUD.js system to provide them with music. Once the jam was over, the developers that used the system were given the following survey. If participants building a game in javascript chose not to use AUD.js, they were instead given a survey to assess why they chose not to.

People interested in games developed at Cal Poly's Global Game Jam site played the games that had the AUD.js system integrated, and filled out a survey assessing their opinion of the game and its music. For comparison, survey results were also collected on games that didn't use the AUD.js system.

### 4.3 Developer Survey

Out of the four teams at Cal Poly's Global Game Jam that considered using AUD.js, one ultimately decided against keeping it in the final project.

#### *4.3.1 Games with AUD.js*

Three teams used AUD.js: Bullet'space (2 person team), Anellu Moore (6 person team) and Phancy Adventures of Charles the Cat (6 person team). All games can be found at *http://globalgamejam.org/2014/jam-sites/cal-poly/games*.

Out of these teams, 12 people filled out the survey.

**Table 2: Developer Survey Results A**

| | |
|---|---|
| Rate how easy you felt the AUD.js API was to use. | 4.41 / 5 |
| Rate how easy the music mood model (stress, energy) was to understand. | 4.5 / 5 |
| Rate the quality of the generated music. | 3.58 / 5 |
| Rate how well the music responded to the input mood you gave. | 3.91 / 5 |
| Would you use AUD.js again in the future? | 91% yes |

#### *4.3.2 Games without AUD.js*

One team that created a game with javascript decided against using AUD.js: 5Alive (5 person team). This game can be found at *http://globalgamejam.org/2014/jam-sites/cal-poly/games*. Four members of this team filled out the survey (duplicate/highly similar feedback is omitted below).

**Table 3: Developer Survey Results B**

| | |
|---|---|
| Why did you elect not to use AUD.js? | "A day into the jam a musician volunteered to do our music." "The song we had fit better, and the game's feel did not depend on the music adapting." "Was cool, but didn't fit the tone of the game well." |
| How did you get your music for your game? | A day into the jam a musician volunteered to do our music. |
| Would you consider using a system like AUD.js for a different project? | 100% yes |
| What would need to be improved in AUD.js before you would consider using it in a Game Jam? | "Not having a musician, as a musician will be able to craft a more coherent melody to match the tone of the game." "We'd need a game that depended more on it and it would be if the audio quality was higher." "Possibility for different styles. No console logs." *(A developer build of AUD.js had been distributed during the GGJ; debug logs had been left in.)* "As hard as it is, the quality of the songs. For people who don't realize the music is procedural, it can be off-putting." |

### 4.4 Player Survey

23 people rated games developed at Game Jams at Cal Poly. These people were randomly assigned to one of four games, two of which used the AUD.js framework. These people were not told they would be reviewing the music and weren't told that the music may be procedurally generated and adapting. Only the most representative responses of the written responses are included.

#### *4.4.1 Games with AUD.js*

15 people rated either Bullet'space or Anellu Moore, which both had AUD.js integrated.

**Table 4: Player Survey Results A**

| | |
|---|---|
| Rate how immersed you felt in the game. | 1.93 / 5 |
| Rate how effective you found the music to be. | 1.64 / 5 |
| Rate how much you enjoyed the music. | 2.21 / 5 |
| Would you listen to the music if it wasn't in a game? | 13% yes |
| Did you feel the music was responsive to game events and user controls? | 33% yes |

| | |
|---|---|
| Was there an event in the game, or a specific control that triggered a music change that you noticed? | 40% yes |

**What could be improved about the music?**

- "Music is too loud/screachy (too high pitched)"
- "Chiptune isn't my thing"
- "[The music] didn't sound like it belonged in the game"

#### 4.4.2 Games without AUD.js

8 people rated either Tricollide or heart4u (games from previous Cal Poly game jams, with comparable gameplay quality as the AUD.js-integrated games [4]).

**Table 5: Player Survey Results B**

| | |
|---|---|
| Rate how immersed you felt in the game. | 2.25 / 5 |
| Rate how effective you found the music to be. | 3.87 / 5 |
| Rate how much you enjoyed the music. | 3.75 / 5 |
| Would you listen to the music if it wasn't in a game? | 12.5% yes |
| Did you feel the music was responsive to game events and user controls? | 37.5% yes |
| Was there an event in the game, or a specific control that triggered a music change that you noticed? | 12.5% yes |

**What could be improved about the music?**

- "Not sure"
- "More variation"
- "A bit of reaction to what I was doing"

## 5. CONCLUSION

From the results we can draw a number of conclusions. AUD.js seems to have accelerated the development process: it was easy to use and pick up, and according to developers it provided reasonable quality sound. The instrumentation quality could use some improvement though. Most developers would be open to using a system like AUD.js, especially if the team doesn't have a composer or is creating a game where the perceived mood of the music changes frequently.

As for the results from people who played the games, AUD.js' generated music quality is markedly lower than that of composed music, which is unsurprising. Again, the surveys show the instrumentation quality must be improved. The AUD.js version used during the game jam used only basic waveforms (square, triangle, sawtooth, noise) to create a primitive 'chiptune' sound. This style does not work for every type of game. As the survey showed, more variety in instrument timbres would be an important addition.

[4]Other games from Cal Poly's 2014 Global Game Jam either had additional hardware requirements, were platform-specific or were largely unfinished.

Players seemed to start to pick up on the runtime adaptations of the music created by AUD.js. In the case of composed music, players seemed to desire more feedback from the music. This shows that there definitely is an audience desire for a type of adaptive music system.

In general, AUD.js could be a very valuable tool, in the sense that it speeds up game development time and that it is capable of adapting the music it generates at runtime. In its current state though, the generated audio quality hinders its effectiveness. The next step in developing AUD.js will be to address that weakness.

A working version of the AUD.js system can currently be found at *timotheyadam.com/AUD*, though because of its dependency on WebkitAudio will only run on Chrome browsers.

## 6. ACKNOWLEDGEMENTS

The authors would like to thank Cal Poly Professor John Clements, whose insights greatly helped improve the music quality of the system.

The authors would also like to thank Nick Alereza, Travis Angevine, Taggart Ashby, Ethan Nakashima and Thomas Soria for reviewing the paper and providing helpful insights.

The authors thank the 93 game development students who helped provide the music mood data for AUD.js' training.

The authors also thank all the members of the 5Alive, Anellu Moore, Bullet'space, heart4u, Phancy Adventures of Charles the Cat and Tricollide teams for helping evaluate the AUD.js system.

## 7. REFERENCES

[1] Challenge: Create game music for game jam team 2. https://www.audiodraft.com/contests/190-Challenge-Create-game-music-for-Game-Jam-Team-2, 2013.

[2] C. Bell. Algorithmic music composition using dynamic markov chains and genetic algorithms. volume 27, pages 99–107, 2011.

[3] K. Collins. An introduction to procedural music in video games. *Contemporary Music Review*, 28(1):5–15, 2009.

[4] D. Cope. *Virtual music: computer synthesis of musical style*. MIT press, 2004.

[5] A. Farnell. An introduction to procedural audio and its application in computer games. In *Audio Mostly Conference*, pages 1–31, 2007.

[6] H. Jarvelainen. Algorithmic music composition. *Tik-111.080 Seminar on content creation*, 2000.

[7] S.R. Livingstone and A.R. Brown. Dynamic response: real-time adaptation for music emotion. In *Proceedings of the second Australasian conference on Interactive entertainment*, pages 105–111, 2005.

[8] R. Ramirez and A. Hazan. A learning scheme for generating expressive music performances of jazz standards. In *International Joint Conference On Artificial Intelligence*, volume 19, page 1628, 2005.

[9] J.R. Newman R.E. Thayer and T.M. McClain. Self-regulation of mood: Strategies for changing a bad

mood, raising energy, and reducing tension. *Journal of Personality and Social Psychology*, 67(5):910–925, 1994.

[10] J.A. Russell. A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161, 1980.

[11] I. Xenakis. Formalized music. *Bloomington: Indiana University Press*, 1971.

## 8. APPENDIX 1: LIST OF EVALUATED MUSIC TRACKS

Each piece was reviewed by approximately 30 people. A total of 93 game design students participated in this study.

| Franchise | Name | Stress | Energy |
|---|---|---|---|
| CastleVania (Konami) | Bloody Tears | 3.48 | 4.32 |
| | Ending | 1.47 | 1.31 |
| | Heart of Fire | 3.5 | 3.8 |
| | Nocturne of Shadow | 3.18 | 1.57 |
| | Out of Time | 3.02 | 4.31 |
| | Poison Mind | 3.9 | 4.06 |
| | Silence of Daylight | 3.46 | 3.7 |
| | Stalker | 3.03 | 3.66 |
| | Vampire Killer | 3.1 | 3.73 |
| | Walking on the Edge | 4.97 | 3.65 |
| | Wicked Child | 4.28 | 4.2 |
| Donkey Kong (Nintendo) | Candy's Theme | 1.46 | 1.9 |
| | Forest Frenzy | 3.32 | 2.96 |
| | Fungi Forest | 1.26 | 1.76 |
| | Gangplank Galleon | 1.2 | 4.4 |
| | Map | 1.48 | 3.36 |
| Final Fantasy (Square Enix) | Airship | 1.8 | 4.36 |
| | Clash on the Big Bridge | 3.84 | 5 |
| | Golbez Clad in Darkness | 3.96 | 1.84 |
| | Mana's Mission | 2.72 | 1.56 |
| Fire Emblem (Nintendo) | Fortune Teller | 2.83 | 3.46 |
| | Inescapable Fate | 3.44 | 2.52 |
| | Main Theme | 1.73 | 3.1 |
| | Reminiscence | 2.23 | 1.66 |
| | Together We Ride | 3.1 | 3.42 |
| Legend of Zelda (Nintendo) | Bolero of Fire | 3.84 | 2.68 |
| | Boss Battle | 4.15 | 3.81 |
| | Deku Palace | 2.76 | 3.48 |
| | Death Mountain | 3.86 | 2.55 |
| | Dungeon | 4.48 | 2.96 |
| | Ganondorf Battle | 4.44 | 4.6 |
| | Gerudo Valley | 2.55 | 3.28 |
| | Goron Race | 1.39 | 4.02 |
| | Deku Tree's Last Words | 4.83 | 1.36 |
| | Kakariko Village | 1.5 | 2.2 |
| | Kokiri Forest | 1.26 | 3.81 |
| | Lost Woods | 1.72 | 4.4 |
| | Majora's Theme | 4.36 | 1.36 |
| | Middle Boss Battle | 4.7 | 4.63 |
| | Overworld | 2.46 | 4.86 |
| | Stone Tower Temple | 4.36 | 1.64 |
| | Title Theme (OoT) | 1.63 | 1.13 |
| | Zelda's Theme | 1.31 | 2.07 |
| Megaman (Capcom) | AirMan | 1.86 | 3.15 |
| | Bubbleman | 2.53 | 3.93 |
| | CrashMan | 2.36 | 3.84 |
| | Dr.Wily's Stages (MM 2) | 2.28 | 4.92 |
| | Ending (Megaman X) | 4.13 | 4.36 |
| | GeminiMan | 2.94 | 4.21 |
| | Hardman | 2.44 | 4.76 |
| | MetalMan | 3.36 | 4.7 |
| | Sigma's Stage | 3.6 | 3.36 |
| | SparkMan | 2.84 | 4.28 |
| | Title Screen (MM 3) | 2.88 | 3.32 |
| | We Are The Robots | 3 | 4.96 |
| Metroid (Nintendo) | Brinstar | 1.81 | 3.31 |
| | Crashed Frigate | 2.32 | 1.24 |
| | Escape Theme | 3.55 | 2.15 |
| | Kraid's Lair | 4.16 | 2.72 |
| | Ridley | 4.84 | 4.92 |
| | Rocky Maridia | 3.47 | 1.34 |
| | Sandy Maridia | 4.16 | 1.33 |
| | Torvus Bog | 3.16 | 2.76 |
| | Tourian | 5 | 5 |
| Pokemon (Nintendo) | Burned Tower | 3.4 | 2.24 |
| | Celadon City | 1.4 | 4.53 |
| | Champion Battle | 4.18 | 3.92 |
| | Elite Four | 1.93 | 3.73 |
| | Lavender Town | 3.06 | 1.53 |
| | Kanto Gym Leader Battle | 4.84 | 5 |
| | Pallet Town | 1.4 | 2.76 |
| | Pokemon League | 1.78 | 1.18 |
| | Radio Tower Takeover | 4.4 | 4.33 |
| | Route 02 Theme | 1.48 | 3.24 |
| | S.S. Anne Theme | 1.44 | 2.97 |
| | Team Rocket Hideout | 4.36 | 4.16 |
| Super Mario (Nintendo) | Castle | 3.86 | 2.36 |
| | Overworld | 1.4 | 4.24 |
| | Overworld (SMB 64) | 1.18 | 4.31 |
| | Underwater | 1.3 | 3.23 |