A Framework for Developing Music-Generated Games

Erik Azzarano

Rochester Institute of Technology[1]

July 30, 2018

Abstract

This work proposes a framework for developing music-generated games from real-time audio input. Games of the music genre today are largely adaptive, such as ones with a set game soundtrack that changes based on what the player is doing or where they are, or interactive, such as the well-known rhythm games like *Guitar Hero* [7] where a pre-loaded audio track creates game components that the player must match in time with the beat of the music. A less commonly produced subgenre of music games is reactive music games, where the game itself reacts to the background music being played in the game. This framework focuses on the reactive game type and aims to create an intuitive mapping between the features of the audio input and the game's mechanics by using parts of audio such as frequency, amplitude, and beats. An arcade-like 2D Shoot 'em Up called *Audio Attack* was built with the Processing programming language and the Minim sound library as an example of the framework. Using the Fast Fourier Transform (FFT), it analyzes the incoming audio and extracts information about average amplitude, amplitudes in different frequency ranges, and beat and note onsets. This information is mapped to different game parameters to drive the experience, such as when enemies spawn, their location, and how fast they move. Overall, this framework will enable future research to experiment with games of this nature and expand upon it by having participants test games created by the framework. This will lead to a better understanding of what the best mappings are, and if music-reactive games are a viable sub-genre of music games that should be focused on more in the game industry.

1. Introduction

1.1 Background

Sound has been an integral part of digital games since their creation in the mid twentieth century. From the simple but revolutionary sound effects of early games like *Pong* [14], to the highly complex composed music and soundtracks for games like the *Legend of Zelda* [12], sound adds a unique depth to any gameplay experience. It evokes emotions that guide a player's experience in a way the developers intended, as well as provide interactive user feedback for what normally might be a passive experience. The recurring theme, however, is that with these games and many others, the music and sound effects were created specifically for the game. This is sometimes known as an adaptive music game, where sounds in the game change depending where the player is and what they are doing. The

[1] Work done at the University of Rochester during Summer REU program.

sounds only occur at calculated points in the game. Another type of game in the music game genre is interactive music games. The more well-known examples of interactive music games are contemporary rhythm games such as *Guitar Hero* [7] and *Rock Band* [16], where a music file is analyzed and the extracted data creates notes that the player must match in sync with the music. When the player matches the note with the controller, the appropriate sound of the note is played to accompany the song playing.

Still, music in games can be taken further. It becomes particularly interesting in cases where the gameplay is driven or even generated by the game's music and sounds, which can be classified as a reactive music game. The game *Vib-Ribbon* [21] released in 1999 on the original Sony PlayStation was one of the first members of the "music-reactive" game genre. In *Vib-Ribbon*, the player can either load a default audio file, or a file from their own music library to play along to. The game processes the audio file and creates different obstacles and other game components based on the music that the player must respond to. In this case, knowing the music could give a player an advantage as they begin to see how the music maps to the game. About a decade later the game *Audiosurf* [2] was released. Likewise, it processes a music file and creates the game level based on the data. The goal of the player is to hit colored bricks that flow in sync with the beat of the music. While reactive music games are a fairly unexplored area of game development, there has been some research done on this genre that will be discussed as it pertains to the overall goal of this work.

1.2 Literature Review

Previous research has looked into mapping music data to 3D space to create a visualization of audio [19]. By analyzing a MIDI file, information about individual tones and instruments can be extracted. For each tone, the visualization application can get information about pitch, volume, and timbre. The pitch refers to where the tone falls on the frequency range. The volume is how loud the tone is. Timbre is less quantitative and refers more to the quality of how a tone sounds. The rhythm of each individual instrument can be found based on the rate at which a tone is detected. The specific tone will also last for a short period after it is detected, so that length can be used as well. Ultimately, all of this information can be mapped to particular features of the visualization that react as the audio file plays.

Continuing to look at how audio can be used to create a visualization or experience, other research has delved into the possibilities of creating music reactive games from existing games originally unrelated to music [9], such as *AudioAsteroids*, a modified version of the game *Asteroids*, and *Briquolo*, an open source 3D spinoff of the game *Breakout*. This work provides design recommendations for how to create a music-reactive game from a non-music-reactive game. The first and probably most important step is figuring out the best way to translate the audio data to the game's mechanics. For example, with *Briquolo*, the paddle size and ball size are altered by different aspects of the music, such as the signal energy and magnitude. Additionally, for *AudioAsteroids*, the music's tempo is mapped appropriately to the speed of the game, and drum hits result in the spawn of a new asteroid for the player to destroy. The next steps in the design process are selecting the music, followed by analyzing the music, and lastly applying the extracted audio data to the game itself by altering the parameters specified in the first step. Ultimately, this work found that the music reactive version of both of these games were viewed as more fun and engaging than the originals because events seemed to have meaning and were not as random. Additionally, knowledge of the music being

played can have implications on how the player controls the game. More recently, a similar idea was proposed by a few researchers looking into how music could be implemented into the mechanics of a non-traditional music game [4]. They hoped to create a platform game where a level is generated by a song(s) from a player's music library.

A slightly different take on music-generated games can be seen in a game called *Celestia* [22], which was created using the Processing programming language and the Minim sound library. The researchers created a game which the player controls with vocal input by altering pitch and volume which is analyzed with a Fast Fourier Transform. Instruments can be used in place of vocals with the same rules applying. Specific vocal inputs are mapped to different game mechanics, such as attacking or dodging. Another game that requires audio input to control the game is *Cello Fortress* [3]. This is essentially a tower defense game, where towers are placed around a path in order to defend an area from incoming enemies. However, in this game the different towers are controlled by the pitch and frequency of a cello being played into a microphone. Four other players use standard gaming controllers to drive tanks that must destroy the towers and avoid their attacks. Overall, these are incredibly interesting attempts at music-reactive games, however, they are restricted to those who are competent vocalists or musicians. The current work aims to provide a more universal experience by creating a game that can work with any form of audio input.

Additional research has provided information on ways in which audio or other media data can be analyzed and mapped to game parameters [10]. With music, there tends to be three standard features that define the structure of any given song. They are: events, states, and transitions. An event can be classified as something that only occurs periodically and lasts temporarily, such as a cymbal crash. The events of a song could be used to trigger particular features of a game, such as an enemy or obstacle spawning to impede the player. A State is the opposite of an event, meaning that it is common and tends to last for a while. This includes the tempo or the average loudness of the music which could be used to control something that should be also be fairly stable, like the speed of a game. Lastly, transition is the movement from one feature to another in music, such as switching from silence to a more prominent part of the song. Transitions could be used similarly to events for controlling gameplay since they tend to happen infrequently. Overall, these three common components of music can help to create an intuitive mapping between audio data and a game's mechanics.

1.3 Current Work

The current work aims expand the domain of reactive music games by creating a framework for actually developing a "music-generated" game of this nature. The goal is to provide insight into how audio features and game mechanics should be mapped to each other in order to create an intuitive and enjoyable experience that is compatible with most types of music. After establishing the framework, a simple arcade game will be built with the Processing programming language and its sound analysis libraries to show how the mappings between audio and games can work. Overall, this framework should provide a base for further experiments and development of music-generated games.

2. Framework

2.1 Overview of Mapping

The proposed framework focuses on the mappings between a game's mechanic

and features of an audio source. Mapping, meaning the relationship between the audio data and the game parameters, is the critical component for creating a music-generated game. Figuring out what is and is not intuitive can be difficult, as there are many aspects of audio that could be mapped to the same game mechanic. The following framework provides suggestions on how to make this work, starting with game mechanics, and then moving on to how audio features should be mapped to them.

2.2 Game Mechanics

Space: Spaces are the existing places in the game. Different spaces within the game may provide different experiences or have different rules. Typically spaces are either discrete, meaning they end somewhere, or continuous, and have dimensions and boundaries that define the extent of the space [18].

Time: Just like space, there is discrete and continuous time in games. Discrete time may be used in a turn-based game where the time between turns can be infinite, while sports simulation games use continuous time, as the game clock keeps moving forward. Time dictates the length of a game for better or worse. Time limits can feel oppressive while infinite amounts of time may make the game boring [18].

Objects, Attributes, and States: Objects are the people, places, or things within the game spaces. They tend to have attributes that can be manipulated during gameplay, and each attribute moves between states that are triggered by events in the game. For example, an enemy (object) has a position (attribute) and either moves or stands still (states). An attribute can have numerous states allowing for objects to become extremely complex, but having to keep track of a large number of states as a player could be distracting and overwhelming [18].

Actions: Actions are the things that objects can do in the game space. Objects usually have basic and strategic actions. Basic actions are the simplest abilities of the object, like moving or jumping, while strategic actions are much more complex and usually requires thinking several steps ahead. Strategic actions are not always explicit things that can be done, or even part of the game's rules. Rather, they emerge by experiencing the game as the player finds new and interesting ways to play. The more basic actions there are, the more opportunities there are to find strategic actions. It also helps if there is more than a single way to achieve the end goal. Overall, actions allow for a self-sustaining game [18].

Rules: Rules are what define all of the previous mechanics. They also decide the goals of the game, giving it structure and really making it a game. There are many different types of rules; some created by the developers, some by the laws of the game world, and others by the players themselves. The game should have a way to enforce these rules to prevent cheating. Still, the most important rule is about how the player can achieve goals in the game. How this is done should be clear so the player never feels stuck or confused [18].

Skill: Skill is the first mechanic thus far that is largely dictated by the player, but it is still facilitated by the game itself. Skill defines the necessary physical, mental, and social abilities that the player must have to get by in the game. Sometimes they are real skills, such physical coordination to control the character's movement, which can improve just by practicing the game. There can also be virtual skills, such as an in game attribute like "sword-fighting" which may increase as the player wins more fights [18].

Chance: As the last of the main game mechanics, chance interacts with the previous six mechanics by bringing uncertainty to them. If everything in the game

is sure to happen every single time, it can be boring. A game needs some surprises to be fun. Chance deals with the probability of events happening, for example, an action like attacking with a sword or shooting with a gun may depend on the player's level with those particular skills. A higher skill level or a better weapon usually corresponds with a greater hit probability. Overall, the same game could be made completely different based on how chance is defined in the game mechanics [18].

Overall, determining a game's mechanics is only a small part of the challenge of creating a good game. More importantly, the mechanics must be balanced in a way that certain things do not give players, or enemies, an unintentional advantage. Balance allows for strategy because the player can calculate actions based on what they know about the other mechanics. If there the game is unbalanced and the mechanics are loosely defined, the player will never be able to improve or get better at the game, which does not bode well for its success. Now that the standard mechanics of a game have been described, the different audio features that could relate to them in a music-generated game will be discussed. (See table 1 for reference of main game mechanics)

Table 1: List of Main Game Mechanics

Game Mechanics [18]
Space (places that exist in the game)
Time (how the game moves forward)
Objects, Attributes, and States (people, places, and things in the game)
Actions (what can be done in the game)
Rules (defines the other mechanics)
Skill (abilities needed to play the game)
Chance (provides surprise and uncertainty to the other mechanics)

2.3 Audio Features

Tempo: Using the tempo of a song to set the speed of the game is common in the interactive rhythm games previously touched on [7] [16]. If one was to tap their finger to the beat of a song, it would make sense that the speed of a game would match the rate of the tapping. Tempo is essentially a direct measurement of the speed of a song, so it makes sense that game speed and tempo should be matched up. If loading an audio file with no concern for loading times, finding the tempo using beat detection algorithms should not be a problem. However, if using live or real-time audio input to create the game instantaneously, finding the tempo quickly and efficiently could prove to be difficult. Luckily, tempo is not the only audio feature that is capable of representing game speed.

Amplitude: Another aspect of audio that game speed could be mapped to is the amplitude, or loudness of the music. Usually, louder parts of the song are more intense, which may help the game become simultaneously more intense by increasing the speed. Conversely, a quieter part of the song may be a good place to slow down the game and allow the player the rest momentarily. The on-screen activity should match the state of the song. As the music gets louder, the overall game speed should increase and the rate at which enemies or obstacles spawn and move should be faster as well. Real-time beat detection to calculate the tempo can sometimes hurt performance, so this method of substituting loudness should improve performance while still keeping the gameplay in touch with the music. Still, amplitude may be for more than just game speed. There is a potential to use the amplitudes of a frequency spectrum to create terrains for a game level or world. For a 2D game, average amplitude may be a good way to define the size or shape of the game space. A jump in amplitude for a period of time could create the appearance of a mountain,

while a part of the song that is consistently quiet may create a more level terrain.

Beat/Note Onset: Beats, which are essentially just onsets of different notes, are considered an event in a piece of music. Typically people tap a finger or foot to the beat of a song, so having the game trigger an event on the beat would make sense intuitively to a player. For this reason, a detected beat should be used as a trigger for an event in the game. Types of events might include: an enemy spawning, a change in a game state, or a new object or obstacle appearing on screen. However, while there is usually a lower frequency beat that provides the tempo of a song, a note onset can come from any frequency at any point in the song. Any note onset could be considered as an event just like the beat. Since note onsets could be in any frequency, this may be a great way to add variety to the game. An onset detected at different frequencies could decide the type of game event or the location of the game event [11].

Beat detection can be very tricky however. Different types of music create the rhythm of a song differently. While today's popular and electronic music typically has a large bass kick throughout, classical music probably does not. Depending on the genre, it may be necessary to analyze different ranges of the frequency spectrum to look for the beat and calculate the tempo. Furthermore, specific frequency ranges could be analyzed to look for specific instruments or vocals, as they tend to produce sound within a particular range. For example, the range of human vocals is from a few hundred hertz to a few thousand [8], so that particular window could be analyzed to detect vocals and alter the game's parameters accordingly. Another option is before running the audio through an FFT, filter the audio file to focus on a specific frequency range of interest. When looking for the bass or kick of a song, a low-pass filter may need to be used to get the lower frequency range without the noise from vocals or higher pitched instruments.

Mood: Emotion, particularly mood, is very relevant when talking about music. People listen to different types of music depending on what mood they are in, or conversely, the music they listen to can alter their mood. Music itself tends to have a mood, and therefore, a game that is generated based on music should match the mood of the music that is playing. Previous studies have looked into automatically categorizing music moods based on features such as intensity, timbre, rhythm, harmony, and lyrics [13][17]. A song can be given values for arousal, meaning how exciting or calming the song is, and valence, how positive or negative the song is, based on these audio features [23]. The arousal and valance values can then be used to categorize it by an emotion. Arousal could be measured with loudness or intensity, as well as tempo or beats per minute. Tempo could also relay a certain valance. Lyrics, though not all music has them, would also be a good way to get a value for valance. The overall genre of a song could help narrow the calculations needed to determine the mood, as songs put into the same genre are usually similar in many ways.

Furthermore, music, like anything in life, has a beginning and an end. The parts in between tell the story of how you get from the beginning and end, just like a game with a well-built narrative would do. In the case of a music-generated game, there is no definitive narrative. Everything is created on-the-fly. The story must come from the music, so how the game portrays the music is critical for creating the game's narrative. Emotion is a big part of this. Intense parts of a song could mean excitement, danger, or power, while slow or silent parts could mean calm, suspense, or sadness.

Events: As described in the literature review, events are a more general part of songs that occur periodically and only last for

a certain amount of time. Beats for example would fall under the umbrella of events. They set the tempo of a song by occurring once every so many seconds. Still, any note onset could be considered an event, whether or not it creates the beat. Like beats, any event of a song should trigger an event in the game.

States: States of music should map well to the states of a game. States are more consistent parts of a song, so they could help tell the narrative of the game as the music may make gradual changes from the beginning, through the middle, and at the end. States should affect the overall look and feel of the game. A self-similarity matrix could be used to find patterns in the music [5], making it easier to determine when to different states change, which moves into the next feature.

Transitions: Transitions mark the changes between states. When a new state is detected, parts of the overall game may change such as the rules or controls. This would add variety and uniqueness depending on the type of music that is chosen. Rule changes could include: how the player scores points, specific attacks the player or enemies have, or how the player must win the game.

Other: There are numerous additional audio features not covered in detail by this framework, as previous research does not offer much insight into these areas. The key is to find audio features that are easy to extract but also provide meaningful information that can be mapped to the game. If the extraction is performance intensive or the information extracted does not map well to a mechanic in the game, it probably is not good to use. Similarly, it is important to pinpoint game mechanics that are able to be manipulated easily by the audio data.

Overall, the suggested framework for developing music-generated games should provide a good base for further research by creating music-reactive games and experimenting with them. It may be critical to get any music-reactive game working for a single song of a single genre, then try another song in the same genre, and then move onward. Not every song is going to look natural most likely. Still, how the audio is mapped to the game mechanic should be obvious. The player should be able to learn the mapping quickly so when a certain part of a song comes or a certain sound occurs they know exactly, or approximately, what will occur in the game. (See Table 2 for reference of audio features)

Table 2: List of Potential Audio Features

Audio Features [9] [10] [19]
Tempo (speed of the music)
Amplitude (intensity of the music or frequency)
Beat/Note Onsets (sets tempo or rhythm)
Mood (emotion associated with the music)
Event (temporary; periodically occurring)
State (constant; consistently occurring)
Transition (movement between states)
Others (lyrics, vocals, timbre, harmony, refrain, chorus, instrument solo)

3. Prototype

Using the Processing programming language [14], a 2D Shoot ‘em Up game called *Audio Attack* was created as an example of the framework (See figure 1). The Minim sound library is imported to handle the audio analysis. The prototype was originally tested by loading an audio file from the computer, which worked well but restricted the player to only using music from their own library. It then moved toward taking audio input directly from the computer microphone so any music or sound playing out loud would create the game. However, this causes a slight delay between the live input and the events in the game, and a lot of unwanted noise was picked up. Now, the

game can actually use the computer's audio output as its input [1]. This means that anything being played out of the computer's speakers or headphones will be put directly into the game. By controlling the game in this manner, it eliminates the delay of the live mic input, while also giving player the freedom to use any source of music, rather than limiting them to their own music library.

In *Audio Attack*, the player controls a ship that can move on an x-y plane in a space-like environment. To start making the game, a FFT is used to calculate a frequency spectrum. The spectrum consists of 1024 bands split into 10 octaves. Using a very simple beat detection method, the beat in each of the 10 octaves can be tracked. Each octave is represented by a different colored bar at the top of the game screen. The bars change height slightly based on the intensity of their respective octave. If a beat is detected in any octave, an enemy is created at that position. The enemy then travels in the direction of the player at a speed directly proportional to the amplitude, or loudness, of the song. The player must move around to dodge the incoming enemies. The player has a limited number of bullets that shoot up to a certain range and can destroy the enemies. The bullets replenish after a cooldown period that is also determined by the loudness of the game. The bullets will replenish faster during the more intense parts of the song. If an enemy hits the player, the player loses health. If the player's health reaches 0, the player is destroyed and a "Game Over" screen appears. Additionally, a beat detection for the overall song is implemented to control the background stars flowing down the screen. When the beat is detected, the stars change color and become larger momentarily, making them essentially dance to the music.

There is an option to increase and decrease a few of the game parameters to better control how the game plays out. One thing the player can do is increase or decrease the game speed, meaning how fast the enemies move toward the player. Additionally, the thresholds for the octave beat detection and overall beat detection can be manipulated to control how many enemies spawn and how frequently the background stars pulse, respectively. (See table 3 for full list of mappings for *Audio Attack*)

Table 3: Mappings between Audio Features and Game Mechanics for *Audio Attack*

Audio Features Used	Game Mechanics Used
Beat/Note Onset	Enemy spawn, background object color and size change
Amplitude	Enemy speed, player reload speed
Frequency Bands	Location/type of enemy when spawning

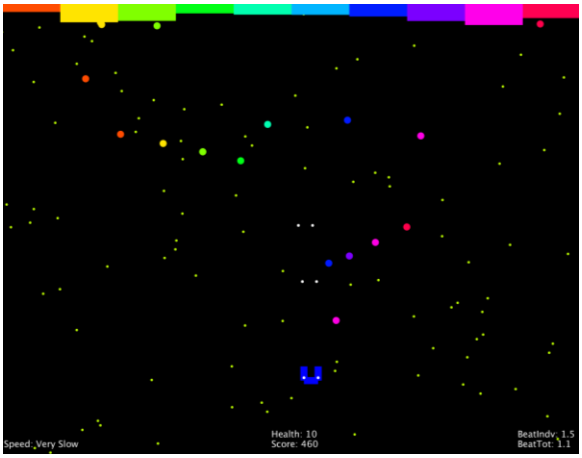

Figure 1: Gameplay of *Audio Attack*

4. Future Direction

While this is just a simple game created with a somewhat limited software development tool in Processing, the idea of using music to create or modify gameplay can be expanded upon in other domains and genres of game development. First and

foremost, an experiment where people actually test the game is necessary to determine a few things. Currently, there is no way to know for sure if the proposed mappings translate to fun or engaging gameplay. Feedback is necessary to determine what mappings work the best in different situations. Also, researchers can create an experiment to find out if a music-generated game is perceived as better than a non-music-generated game, which would be telling about the validity of this subgenre of music games in the game industry.

The main two options with this framework for future researchers are: to develop new and original games, or to take non-music-reactive games and make them reactive. Either way, it is important to note that not any game genre would be suitable for being music-reactive. It would be best to look at games that use procedural generation, as that is essentially what the music would be doing. Side-scrolling arcade games like the prototype is one option that works because the game is continuously updated with the song. Similarly, an infinite runner type game would work, since the player would constantly have to react to the changes of the game based on incoming audio. Lastly, rogue-like games where a cave or dungeon is procedurally generated for exploring could be another option. While it might not be best for the music to actually generate the dungeon, as that is usually done carefully with algorithms, the enemies or other objects in the dungeon could move to the music, like in the game *Crypt of the NecroDancer* [5].

Additionally, more sophisticated audio analysis software could be used to extract the audio information in real-time that may allow for more complex gameplay mappings. Tempo, for example, was not looked at by the prototype, but has been used in the past and should be tested in the future. Also, while it may be restricting to the players, using a MIDI file to create the game may allow for greater variety in gameplay. MIDI files hold a large amount of information about a given song, such as exactly when a specific instrument is played, and at what pitch and tempo. This could allow for a large amount of very unique gameplay between different songs and genres.

Furthermore, it is possible that traditionally time consuming and computationally expensive processes for level design and generation could be made simpler by using music to create the level on the fly. Researchers have looked at automate level design for 2D platform games and 2D adventure games using a genetic algorithm [20]. While this method offers more control to the design of the level, a music-generated level may prove to be just as capable and unique.

Lastly, future researchers could connect to music streaming sites such as Spotify or Pandora to live stream music. The game could intercept the audio signal before it plays out of the speaker and use it just as before to generate the game. This would let players still feel like their music is controlling the game, but allow for better audio analysis in real-time. The audio could be delayed slightly so everything still happens in real-time to the player, but underneath, beat detection or other algorithms can run to get information about a point in the song before it actually happens on the screen. Making a game like this would probably mean looking into platforms like web and mobile, which should be a good things as they are readily available or easily accessible to most people.

5. Conclusion

The intention this work is that the proposed framework can be used to answer additional research questions about this genre of music-reactive games. This was done by providing general information and suggestions about mappings between game

mechanics and audio features, the core component of a music-reactive game. The developed prototype exemplified several aspects of the framework by showing how exactly a game can change based on music. Ultimately, the goal of future researchers should be to experiment with the framework and the games they develop or modify with it. Human testers should be able to provide insight into the soundness of the framework and music-reactive games as a whole.

6. Acknowledgements

This research was funded by the National Science Foundation. I would like to give a special thanks to Prof. Al Biles and Prof. Ian Schreiber from Rochester Institute of Technology, and Prof. Ehsan Hoque and ROC HCI, the Kearns Center staff, and my faculty mentor Prof. Sreepathi Pai at the University of Rochester.

References

[1] Ambulatore and Theformand, (2016). Doing an FFT on system audio currently playing with minim [online forum comment]. Message posted to https://forum.processing.org/one/topic/doing-an-fft-on-system-audio-currently-playing-with-minim.html

[2] Audiosurf, Invisible Handlebar, 2008

[3] Cello Fortress Web Site. http://www.cellofortress.com, accessed on July 17, 2018

[4] Christian Hahn, Paul Diefenbach, Surge: an experiment in real-time music analysis for gaming, Proceeding of the SA '10 ACM SIGGRAPH ASIA 2010 Posters, Article No. 16, December 15-18, 2010, Seoul, Republic of Korea

[5] Crypt of the NecroDancer Web Site. http://necrodancer.com/ accessed on July 17, 2018.

[6] Foote, J. "Visualizing music and audio using self-similarity." Proceedings of the Seventh ACM International Conference on Multimedia, p. 77-80, October 30 – November 05, 1999, Orlando, Florida.

[7] Guitar Hero Web Site. http://www.guitarhero.com, accessed on July 17, 2018.

[8] Human speech spectrum, frequency range, formants, http://www.bnoack.com/index.html?http&&&www.bnoack.com/audio/speech-level.html accessed on July, 17, 2018.

[9] Juha Arrasvuori, Jukka Holm, Background music reactive games. Proceedings of the 14th international academic MindTrek conference: Envisioning future media environments, p.135-142, October 06-08, 2010, Tampere, Finland

[10] Jukka Holm, Kai Havukainen, Juha Arrasvuori, Personalizing game content using audio-visual media, Proceedings of the 2005 ACM SIGCHI international conference on advances in computer entertainment technology, p. 298-301, June 15 - 17, 2005, Valencia, Spain

[11] J. R. Parker, J. Heerema, Musial interaction in computer games, Proceedings of the 2007 conference on future play, p. 217-220, November 14 - 17, 2007, Toronto, Canad

[12] The Legend of Zelda Web Site: http://www.zelda.com, accessed on July 17, 2018

[13] L Lu, D Liu, and H. J. Zhang, "Automatic mood detection and tracking of music audio signals." *IEEE Transaction on Audio, Speech, and Language Processing*, vol 14, no. 1, pp. 5-18. Jan. 2006.

[14] Pong, Atari, 1972

[15] Reas, C. and Fry, B., *Processing: A programming handbook for visual designers and artists*. Cambridge, MA, The MIT Press., 2014.

[16] Rock Band Web Site. http://www.rockband.com, accessed on July 17, 2018.
[17] R. Panda and R. P. Paiva, "Using support vector machines for automatic mood tracking in audio music." Proceedings of the 130th Audio Engineering Society Convention. May 13-16. 2011. London, United Kingdom.
[18] Schell, J., The Art of Game Design. Natick, MA, CRC Press, 2014
[19] Smith, S. M. and Williams, G. N. "A visualization of music." Proceedings from the 8th Conference on Visualization, p. 499-503, October 18-24, 1997, Phoenix, Arizona.
[20] Sorenson N., Pasquier P. (2010) Towards a Generic Framework for Automated Video Game Level Creation. In: Di Chio C. et al. (eds) Applications of Evolutionary Computation. EvoApplications 2010. Lecture Notes in Computer Science, vol 6024. Springer, Berlin, Heidelberg
[21] Vib-Ribbon, Sony Computer Entertainment, 1999
[22] Yang Shi, Cheng Yang, Celestia: A vocal interaction music game, Proceedings of the CHI '13 extended abstracts on human factors in computing systems, p. 2647-2650, April 27 - May 02, 2013, Paris, France
[23] Y. E. Kim, E. M. Schmidt, R. Migneco, B. G. Morton, P. Richardon, J. Scott, J. A. Speck, and D. Turnbull, "Music emotion recognition: A state of the art review." Proceedings of the 11th International Society for Music Information Retrieval Conference, p. 255-266, 2010.