Implementation and Evaluation of a Background Music Reactive Game

Khalid Aallouche
Tampere University of Technology
khalid.aallouche@insa-lyon.fr

Homam Albeiriss
Tampere University of Technology
homam.albeiriss@gmail.com

Redouane Zarghoune
Tampere University of Technology
redouane.z@gmail.com

Juha Arrasvuori
Nokia Research Center
P.O. Box 1000
FIN-33721 Tampere, Finland
+358 (0)50 486 7978
juha.arrasvuori@nokia.com

Antti Eronen
Nokia Research Center
P.O. Box 1000
FIN-33721 Tampere, Finland
+358 (0)50 482 1942
antti.eronen@nokia.com

Jukka Holm
Nokia Research Center
P.O. Box 1000
FIN-33721 Tampere, Finland
+358 (0)50 596 9177
jukka.a.holm@nokia.com

ABSTRACT

This paper discusses further work on the authors' "background music reactive games" concept, where background music is used to modify video game parameters and thus actions on the screen. Each song selected by the player makes the game look different and behave variedly. The concept is explored by modifying an open-source game called Briquolo, which is based on the well-known arcade game Breakout. Several audio signal features such as magnitude, energy, centroid, and spectral flux are calculated from the background music MP3 file and mapped to relevant game parameters. In order to verify how well the features work in practice, a user study with 20 participants was arranged. The results suggest strongly that people appreciate the concept of background music reactive games. The selected analysis algorithms and mapping worked nicely, and 90% of participants felt that the music truly affected the game. In addition, 90% of participants also felt that the modified version was more entertaining than the original.

Keywords

Games, music, background music reactive games, musically controlled games, MP3, music signal analysis, Breakout, Briquolo, AudioAsteroids.

1. INTRODUCTION

Music contributes to creating an immersive atmosphere for gaming, and can effectively emphasize the actions on the screen. It is common in contemporary video games that the background music is adaptive, meaning that it changes according to game events and between different parts of the game. However, the development of an adaptive music soundtrack takes a lot of effort. Due to this, many games simply loop the same relatively short musical material over and over. Repetition has its cost in the overall experience of the game: The gamer may become bored with the non-adaptive soundtrack and turn it off after a while. The study conducted by Cassidy et al. [13] suggests that the best player experience emerges when a player can choose a game's background music to something that he or she prefers. However, the scope of that study was limited to driving games.

Since the 1990's, in parallel with the introduction of multimedia computers and powerful video game consoles, we have seen the emergence of musically oriented games. As Blaine points out in [15], the majority of these are so called "rhythm games" that prompt a single player or a group of players to perform rhythmic actions in time with a predetermined musical sequence. This game genre has also led to the development of new low-cost musical interfaces such as drum and guitar controllers that make the games more enjoyable to play. The "Guitar Hero" [6] series has been a recent success in this game and accessory genre. Many rhythm games suffer from the same problem as non-adaptive game soundtracks: As the number of music files is limited, the games may have quite short lifecycles. To improve the situation (and sell sequels and extension packs), some developers have started offering game upgrades such as catalogues of popular songs.

This paper describes an alternative way to use music in video games. Instead of relying on adaptive background music that reacts to the game events, the authors propose the concept of games that react to their background music. Throughout this paper, these games are referred to as "background music reactive games." In [11] and [14], the authors have also used the term "musically controlled games."

The contents of this paper are as follows: Chapter 2 discusses some relevant related work and Chapter 3 presents an overview of the basic concept. Chapter 4 discusses in detail the implementation of a background music reactive Briquolo game, various sound analysis and processing related issues, as well as mapping music to game parameters. Results of the arranged user study are iterated in Chapter 5. Finally, Chapter 6 draws some conclusions and Chapter 7 suggests directions for future work.

2. RELATED WORK

Background music reactive games are such a new phenomenon, that so far only a handful of such games have been implemented.

One of the pioneering game titles of this type was the PlayStation game "Vib-Ribbon" (NanaOn-Sha 1999) [8]. It is best described as an obstacle track game, in which the background music (any song chosen by the gamer) affects in some way the appearance of obstacles, the points in time when these obstacles appear, and the spawning of certain additional objects. Player's character seems to walk along the stylized waveform. One issue with Vib-Ribbon is that the correspondences between characteristics of music and obstacle track are not that obvious for casual players. The obstacle track just appears a bit different with different pieces of music.

"Dance Factory" [7] (Codemasters 2006) for PlayStation 2 is a dancing game that is intended to be played with a dance mat controller. The game can be played with any piece of music provided by the gamer, as the game allegedly can match the dance steps to the beat of the music. Dance Factory has received criticism because it does not always synchronize the dance steps precisely enough to the beat.

Kuju's music-puzzle game "Traxion" is a background music reactive game that was planned for PlayStation Portable. The idea of the game was that players could use their own MP3 files as the basis for more than 20 minigames. Unfortunately, the development was ceased in the beginning of 2007. [16]

In [11], [12], and [14], the authors have described their previous experiments with a background music reactive game called "AudioAsteroids." In the game, the player controls with the keyboard a spaceship that must avoid colliding with asteroids and other objects flying around. The player must attempt to shoot dangerous objects such as asteroids and enemy ships, and collect some bonus objects in order to get more points, lives, etc. Two versions of the game were implemented: the first one was based on MIDI music and the second one on Wave files. The players were able to define the connections between musical control parameters and game events by themselves. A generic mapping, which was as illustrative as possible for several types of background music, was also provided with the game.

In the case of MIDI, the music analysis was done in real-time. The authors learned that the most important and easily noticeable mapping was connecting song's tempo to modify game speed. Another very perceivable connection was mapping a certain drum being played to generate a specific game object.

In the case of digital audio, the files were analyzed beforehand and the results stored to a file that was then used as an input to the game engine. The tempo of used Wave files was calculated using a simple beat-tracking algorithm. It was mapped to game's speed and seemed to work almost as nicely as in the case of MIDI files. Other low-level signal features (see Section 4.1) such as energies at various frequency bands, energy changes, spectral centroid, and so forth were used to spawn different objects to the screen. Most modern music is compressed really hard so it was difficult to find useful differences between musical styles. The goal with this experiment was to find such mapping that e.g. heavy metal would be more difficult to play than 70's disco, and this was achieved to some degree. However, it was concluded that a general problem when implementing background music reactive games using sampled audio input is that high-level musical features cannot be analyzed robustly using current methods. Compared to MIDI files, one must use rather low-level signal features that describe the characteristics of the signal rather than musically meaningful attributes. A challenge therefore is to find suitable mappings and post-processing steps for the low-level features in order to create meaningful changes in the game. In this paper, the authors present an example where the use of low-level features and suitable post-processing steps creates exciting variations to the Briquolo game.

3. OVERVIEW OF BACKGROUND MUSIC REACTIVE GAMES

As the name implies, "background music reactive game" refers to games that somehow react to their background music. Starting this kind of game, see Figure 1, differs somewhat from traditional video games. In the beginning, the player must first select a music file or collection of files to be used in the game. The selected music is analyzed for relevant musical or audio signal features either in real time, in larger buffers, or the whole song can be processed before the game starts. The resulting control data is then sent to the game engine in a suitable format, and mapped to selected game parameters. Depending on the implementation, the mapping can be either a fixed set of connections delivered with the game, or a new set specified by the player. When a mapping has been defined, the player can start the game and try to play through as many music files as possible. Depending on the game type and implementation, each music file may be considered as one unique game level.

When music is analyzed to produce control data for the game, novel game ideas can be found. Even a very trivial game can be made interesting if the player can affect the difficulty level by changing the background music to his or her favorite tune. As shown in this paper, musical control can also bring new life to existing game implementations. The concept offers also many imaginative possibilities for the game designers. Numerous actions that would usually be controlled by a random generator or artificial intelligence (AI) can now depend on the selected background music, meaning that the actions do not appear random but seem to happen in synchrony with the music.

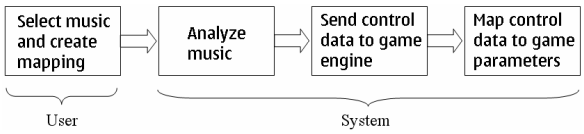

Figure 1. Starting a background music reactive game.

Examples of background music reactive game elements and characteristics include e.g.

- Speed and difficulty level of the game;
- Location of game objects (enemies, ammo, guns, bonus objects, obstacles, balls, bats, etc.);
- Number, size, type, color, and shape of objects;
- Timing and frequency of appearance of new objects;
- Movement (e.g. speed, direction, rhythm, starting point, trajectory) of objects;
- Properties of avatars (e.g. skills and endurance);
- Properties of gameworld (e.g. location of game objects, time of day); and
- Camera angle to gameworld.

The set of suitable musical and audio signal features depends largely on the used sound format (MIDI, digital audio, or compressed digital audio), gaming platform, game type and design, available processing power, memory, and so on. Interesting alternatives include at least e.g.

- Tempo;
- Signal magnitude and energy;
- Occurrence of certain instruments and pitches;
- Intervals and harmonies;
- Spectral content (perceived as e.g. brightness); and
- Polyphony.

For the purposes of background music reactive games, these features and the control data calculated based on them can be divided into three principal groups: "Event" (musical or signal features that occur occasionally and last for a brief time), "state" (features that stay more or less the same for a longer time), and "transition" (significant changes from one value to another). These groups are discussed in more detail in earlier papers ([11], [12], and [14]) by the authors.

3.1 Mapping

Mappings between musical control data and game parameters are most effective when players immediately understand the relationship between what they hear and what they see. If a player knows a certain piece of music well and understands the mapping used in the game, he can anticipate some of the actions that will occur in the game. Thus, the game can be enjoyed in a novel way.

In order to support a large number of players having different musical tastes, the mapping should not be tailored to a single musical genre. There are undoubtedly interesting differences between musical styles, but in most cases the game designer should aim at a generic mapping that works nicely with any genre. It is beneficial if distinct musical genres produce distinct game experiences, but this should not be done at the cost of the overall playing experience.

While most musical parameters affect how the music sounds like, some of them may not be easily observable. Parameters that are heard by most players should usually be connected to major foreground events in the game, while the less obvious ones can be used to modify less important things like background graphics and so on.

4. IMPLEMENTATION

The implemented background music reactive game is based on an open source game called "Briquolo" [1], which is an enhanced, 3D version of the well-known arcade game "Breakout" [9]. Briquolo works on both Windows and Linux platforms, and is written using C++ programming language. The game was selected because it is simple and intuitive, has been released under the GNU/GPL license [4], and includes several parameters that can be made to react to musical parameters. In the game, the player controls a paddle with the computer keyboard's arrow keys or with a mouse. A ball travels across the screen and destroys the bricks that it hits. The goal is to destroy all the bricks on the screen. The paddle is used to avoid the ball falling to the bottom of the screen. The player has a limited number of lives, and every time the player misses a ball, a life is lost.

The original Briquolo game was modified to support the selection, playback, and analysis of MP3 files. Sound processing was done using Maaate open source library [2], and MP3 files were handled using irrKlang [3]. Some changes to the game's GUI were also made, as players had to be able to select the music and make mappings between selected game and music parameters.

Figure 2. Screenshot from the Briquolo game [1].

When the player launches the game, the main window appears and the player can see the following menu items: 'Play', 'Level Editor', 'Settings', and 'Quit'.

If the player selects Play, he can choose either the original Briquolo game or its modified background music reactive version. He can also select which MP3 file to use as background music in the game and one of several game levels. The selected MP3 file is analyzed for those signal features that have been chosen in the Settings menu.

Using the Level Editor, the player can create new game levels e.g. by determining the number and location of blocks.

In the Settings menu, the player can configure the original game parameters or make connections between selected game parameters and implemented signal features (see Section 4.1.1). In order to save processing time, only those features that are mapped to some game parameter are analyzed from the selected MP3 file. The analysis results are saved to a file so the analysis has to be done only once before the first play.

4.1 Sound Processing

After the background music MP3 file and used mapping have been selected, the game starts to analyze the file by extracting information which will then be applied during the game. In the following, this sound processing step is discussed in more detail.

An MP3 file is made of multiple frames composed by a header and data. The header consists on a sync word used to identify the beginning of a valid frame. It is followed by a bit indicating that the MPEG standard [18] is being used, and then two bits indicating that layer 3 is being used, hence MPEG-1 Audio Layer 3 or MP3.

During the sound processing step, the selected MP3 file is decoded and its frames are extracted. For each extracted frame, a number of audio analysis features selected according to player's own settings are calculated. The equations for the used features are described in the following.

4.1.1 Description of low-level features

Signal energy is calculated as

$$E(t) = \frac{1}{I \cdot M} \sum_{i=0}^{I-1} \sum_{m=0}^{M-1} s_i{}^2 (Mt + m) \cdot h(m), \quad (1)$$

where $s_i(n)$ is the value of the subband i of the MPEG-1 Audio Layer 3 polyphase filterbank at time instant n, I is the number of subbands, M is the size of the analysis window in samples, t is the index of the analysis window, and $h(m)$ is the Bartlett window function.

Signal magnitude is an approximation of the perceived loudness of the file, and is calculated using the equation

$$\mathrm{M}(t) = \frac{1}{I \cdot M} \sum_{i=0}^{I-1} \sum_{m=0}^{M-1} \left| s_i (Mt + m) \right| \cdot h(m). \quad (2)$$

Spectral centroid is the balancing point of the subband energy distribution. It gives a rough measure of the perceived brightness of sounds: the higher the centroid, the brighter sounding the signal is. The centroid is calculated as:

$$C(t) = \frac{\sum_{i=0}^{I-1} (i+1) r_i(t)}{\sum_{i=0}^{I-1} r_i(t)}. \quad (3)$$

where $r_i(n)$ is the root-mean-square (RMS) energy of subband i at time instant t.

Rolloff point R determines the frequency point where 85% of the window's energy is achieved. It measures the "skewness" of the spectral shape, and is calculated as:

$$\sum_{i=0}^{R} r_i(t) = 0.85 \cdot \sum_{i=0}^{I-1} r_i(t). \quad (4)$$

Spectral flux measures the variation of the spectral energy distribution between two successive windows:

$$\Delta(t, t+1) = \sqrt{\sum_{i=0}^{I-1} \left| r_i'(t) - r_i'(t+1) \right|^2}, \quad (5)$$

where $r_i'(t)$ is the RMS energy of subband i normalized with the maximum value:

$$r_i'(t) = r_i(t) / \max(r_j(t) : 0 \le j \le I-1).$$

Thus, in music with lots of rapid onsets the spectral flux will be large, whereas in music with long sustained sounds the flux will be small.

Drum track: the intent of this parameter is to detect energy peaks, such as drum hits, from the music file. The signal energy is first calculated as described in Equation 1, and then values smaller than a threshold are truncated by setting them equal to the threshold. The threshold was selected to be $1/(I \cdot M)$. The values higher than this threshold are further compressed by taking a logarithm.

Silence: the game is able to detect silences in the music and stops the game as long as the silence lasts. A threshold of one percent of the average signal loudness is used as a threshold for silence. In Maaate, loudness measurement is done by summing the subband scale factors.

4.1.2 Postprocessing of low-level features

Once the low-level signal features have been extracted, a couple of post-processing steps are implemented to create smooth and controlled changes into the game. As the values of the low-level features may change rapidly and in an uncontrolled manner, mapping them directly into game parameters would cause too strong and noisy effects making the game unplayable.

After the analysis stage, each signal frame is represented by a value for each feature. The values of each feature are then filtered using a user adjustable sensitivity threshold. The threshold controls how large an absolute change must two adjacent feature values have in order to trigger a change in the game. If the user sets the sensitivity threshold high, it means that there will have to be a large absolute change between the feature values of two adjacent frames in order for a change to occur in the game. Correspondingly, setting the sensitivity threshold low allows even small changes in feature values to modify the game parameter. The sensitivity parameter also controls the rate of the changes: as setting the sensitivity threshold low allows even small changes of the feature value to vary the game, it is likely that changes will happen more frequently than with a high sensitivity threshold.

The sensitivity threshold is applied to detect significant level changes in the feature values. This is done according to the method below:

```
// sensitivity_user::sensitivity chosen by the user(percentage)
// delta: difference between the minimum and maximum value of the feature in the signal
// value: tabulated values of the feature over the whole signal
delta = max(value)-min(value);
sensitivity = sensitivity_user * delta;
lastValue = value[0]; //first parameter value
for i ← 1 to TABLE_SIZE-1
    if abs(value[i]-lastValue) ≥ sensitivity
        // lastValue is set to the new level
        then lastValue= value[i];
    else
        // force the previous level value to be used
        value[i]= lastValue;
```

Then, a mean filter is used to smooth the variations in the array 'value'. The longer the window, the more the values are smoothed, and rapid variations are removed. A length of 15 for the mean filter was found to work well. Finally, the results are normalized between 0 and 1, and sent to the communication interface.

Genre extraction

Compressed digital audio files may contain various type of metadata, including title of the song, name of artist, album, composer, genre, production year, and so on. In MP3 format, there are two kinds of tags:

- ID3v1, which occupies 128 bits and is placed at the end of the file; and
- ID3v2; which occupies variable size and usually occurs at the beginning of the file.

In addition to the low-level signal features presented in Section 4.1.1, a software module which extracts the musical genre

directly from an MP3 file tag was implemented. MP3 ID3 meta-data [5] specifies over 100 different musical genres. As it would have been impractical to support them all in the game, the genres were divided into six main groups: "Classical", "Electronic music", "Folklore", "Rock", "Metal", "Rap, funk, and jazz", and "Country and folk." The division was based on [19]. These six genres were mapped to water color as explained in the next section.

4.2 Mapping Music to Game Parameters

The connections between game parameters and supported signal features are defined in Settings menu's CoMuGame submenu, see Figure 3.

Each of 'Paddle size', 'Ball size', and 'Ball speed' parameters can be associated with signal features such as 'Signal magnitude', 'Drum track', 'Signal energy', 'Centroid', 'Rolloff', or 'Spectral flux'. The corresponding equations for each feature were given in Section 4.1. If the player does not want to connect some game parameter to any audio signal feature, he can select 'No interaction' for that parameter value. The player can also adjust the 'Sensitivity' parameter, which has an effect on all the active parameters.

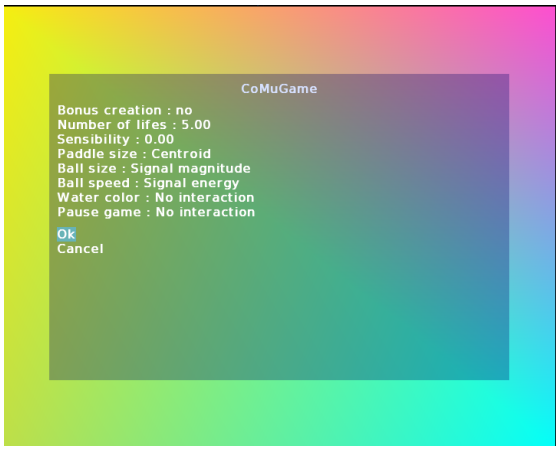

Figure 3. Defining connections between game and music parameters.

Other game related parameters include 'Water color' and 'Pause game'. Pause game is a very intuitive parameter when it is associated with signal feature 'Silence.' If the water color is connected to 'Genre', the background color of the gameworld will change according to the musical genre of the selected MP3 file. The mapping from genres to colors is depicted in Table 1, and cannot be reconfigured by the player.

Table 1. Mapping colors to genres.

Color	Genre
Blue	Used if water color is connected to 'No interaction'
Grey	No genre found
White	Classical
Cyan	Electronic music
Yellow	Folklore
Red	Rock
Dark red	Metal
Orange-red	Rap, funk, and jazz
Green	Country and folk

Although the player can design his or her own mapping before starting the game, this can be an iterative and very time-consuming process. Therefore, a representative set of connections (which would be as illustrative as possible for several types of background music) was defined and listed in project's web page [17] for the test users. The following generic mapping was defined:

- Paddle size = Spectral centroid;
- Ball size = Drum track;
- Ball speed = Signal magnitude;
- Water color = Genre;
- Pause game = Silence; and
- Sensitivity = 0.

5. USER STUDY

In order to test how well the game works in practice, a user study with twenty participants was arranged. The purpose of the study was to find out if the players felt that the game truly reacted to its background music, what were the most intuitive musical parameters, and what kinds of mappings the players created themselves.

Participants were recruited by sending e-mail invitations. 90% of the participants were young adults (20-30 years old), while the rest were younger. Out of the 20 participants, only four were female. None of the participants had any prior experience in sound processing, so they may have not understood all terms used in the Settings menu. All participants had already played some version of the Breakout game, and 20% of them had not liked it.

The participants downloaded the game from project's web pages, and were able to play it for one week after which they filled in an online questionnaire. In the first part of the test, all users played with the same song and the same default mapping. In the second part of the test, they were free to choose their own mappings and songs.

The results of the questionnaire were really positive. 90% of participants felt that the music truly affected the game. 80% found the background music reactive version of the game more challenging than the original one. However, 90% of them felt that the modified version was more entertaining than the original. One participant even commented that "It is the best game I have ever played." The game concept was appreciated by the test users: "This musically controlled game is very funny, I did not expect that. It was a good surprise and I enjoyed it", "Good idea! I like it!", and "I think that it is a very inventive game."

In the case of generic mapping, 75% of the participants felt that music affected the paddle size and speed of the ball. 70% felt that ball size varied according to the background music.

35% of the participants also tried their own mapping instead of the default one. Most of them associated the signal magnitude feature with paddle size and kept the drum track connected to ball size. None of the participants left the signal magnitude connected to ball speed. Several other parameters were tried, but none of them was dominant.

According to our experiences, the background music reactive version of Briquolo had superior performance compared to our earlier game AudioAsteroids (see Section 2), which has never been tested in a formal user study. Even if Briquolo does not directly support tempo detection like AudioAsteroids, the

selected audio analysis algorithms created a sensation that the game reacted to the tempo of background music. The study also proves that the addition of background music responsiveness can bring new life to an old and simple video game idea.

6. CONCLUSION

This paper elaborated on the authors' research on "background music reactive games", where a video game's background music is used to affect game parameters and thus actions on the screen. Each song generates a new game level with varying characteristics and difficulty.

An open-source game called Briquolo was modified to support the testing of various audio analysis algorithms and experimenting with different mappings. A generic mapping was defined to test the game behavior with several types of music.

All selected algorithms worked nicely with the game, and together with the default mapping they created a feeling of music affecting the game. Even if beat detection was not directly supported, the selected algorithms created a sensation that the game reacted to the tempo of background music. In fact, modifying game's speed with the *perceived* musical tempo seemed to be the most noticeable feature of the game. This is in line with our earlier findings from the AudioAsteroids game. One new feature that seemed to work particularly well was to make the game pause whenever there was a silent moment in the music.

In order to verify how well the implemented ideas worked in practice, a user study with 20 participants was arranged. The results suggest strongly that people appreciate the concept of background music reactive games. 90% of participants felt that the music truly affected the game, and considered this version more entertaining than the original. The concept seems to introduce a new dimension into the experiences of gameplay when players realize how game content and behavior changes in response to certain characteristics of the background music.

7. FUTURE WORK

The obvious future step for the background music reactive Briquolo would be the addition of new music signal analysis methods, and testing different connections between the resulting control parameters and the game. For example, methods introduced for instrumental and vocal music detection could be used to create more event-like changes to the game. Changes like these could be used to introduce major changes to the game when the background music switches from a vocal to an instrumental part, and vice versa. Other potential candidates include e.g. beat and pitch detection algorithms as well as controlling the game based on microphone input. The audio signal analysis parameters should be substituted with more intuitively understandable terms. For evaluating the new features, another user study would be needed.

The game would also benefit from several minor new features and bug corrections. Examples include playing through playlists of songs, selecting and placing background elements based on the music, implementing better genre groupings and color associations, varying the color of other game elements than water color, and so on. The authors are planning to distribute the game and its source code [17] to selected developer forums, and see what kind of ideas other people come up with. Another exciting direction in future development would be to combine the activities of playing a game with those of making music with musical controllers. As an example, one player could control the hero in the game, while the musical performance of others would create obstacles for the hero.

8. REFERENCES

[1] Briquolo. http://briquolo.free.fr, 30.3.2007.

[2] Maaate. http://www.cmis.csiro.au/maaate/, 30.3.2007.

[3] irrKlang. http://www.ambiera.com/irrklang/, 30.3.2007.

[4] GNU/GPL License. http://www.gnu.org/copyleft/gpl.html, 30.3.2007.

[5] ID3 v2.3.0 Specification. http://www.id3.org/id3v2.3.0, 30.3.2007.

[6] Guitar Hero. http://www.guitarherogame.com, 10.4.2007.

[7] Dance Factory. http://www.codemasters.co.uk/games/?gameid=1832, 10.4.2007.

[8] Vib-Ribbon. http://www.vib-ribbon.com/, 25.1.2006.

[9] Breakout. http://en.wikipedia.org/wiki/Breakout, 10.4.2007.

[10] Blaine, T., and Fels, S. Design Issues for Collaborative Musical Interfaces and Experiences. In *Proceedings of New Interfaces for Musical Expression (NIME) Conference*, Montreal, Canada, May 22-24, 2003.

[11] Holm, J., Havukainen, K., and Arrasvuori, J. Novel Ways to Use Audio in Games. In *Proceedings of Game Developers Conference (GDC)*, San Francisco, USA, March 7-11, 2005.

[12] Holm, J., Havukainen, K., Arrasvuori, J. Personalizing Game Content Using Audio-Visual Media. In *Proceedings of Advances in Computer Entertainment (ACE) Conference*, Valencia, Spain, June 15-17, 2005.

[13] Cassidy, G., MacDonald, R., and Sykes, J. *The Effects of Aggressive and Relaxing Popular Music on Driving Game Performance and Evaluation.* Abstract in http://www.gamesconference.org/digra2005/viewabstract.php?id=94, 2.4.2006.

[14] Holm, J., Havukainen, K., and Arrasvuori, J. Using MIDI to Modify Game Content. In *Proceedings of New Interfaces for Musical Expression (NIME´06) Conference*, Paris, France, June 4-8, 2006.

[15] Blaine, T. The Convergence of Alternate Controllers and Musical Interfaces in Interactive Entertainment. In *Proceedings of New Interfaces for Musical Expression (NIME) Conference*, Vancouver, Canada, May 26-28, 2005.

[16] Boyes, E. Traxion Loses Traction. http://www.gamespot.com/psp/puzzle/traxion/news.html?sid=6163839&om_act=convert&om_clk=newlyadded, 13.4.2007.

[17] Background Music Reactive Briquolo Game. http://cmg.redouane.info/, 27.4.2007.

[18] The MPEG Home Page. http://www.chiariglione.org/mpeg/, 27.4.2007.

[19] ACIM Organization Home Page. http://www.acim.asso.fr/, 2.4.2007.