BeatTheBeat
Music-Based Procedural Content Generation In a Mobile Game

Annika Jordan, Dimitri Scheftelowitsch, Jan Lahni, Jannic Hartwecker, Matthias Kuchem, Mirko Walter-Huber, Nils Vortmeier, Tim Delbrügger, Ümit Güler, Igor Vatolkin Student Member, IEEE, Mike Preuss Member, IEEE

Abstract—We present a multi-player mobile game that employs fully automated music feature extraction to create ‘levels’ and thereby produce game content procedurally. Starting from a pool of songs (and their features), a self-organizing map is used to organize the music into a hexagonal board so that each field contains a song and one of three minigames which can then be played using the song as background and content provider. The game is completely asynchronous: there are no turns, players can start and stop to play anytime. A preference-learning style experiment investigates whether the user is able to discriminate levels that match the music from randomly chosen ones in order to see if the user gets the connection, but at the same time, the levels do not get too predictable.

Fig. 1. Screenshots of the board (left, colors indicating player owning field (inner) and k-means “genre” clustering (outer, see fig. 5), symbols standing for associated minigames) and the minigame Music Fighter (right, the player controls the ‘ship’ resembled by the bottom triangle, red notes emerging from the center are the enemies)

I. INTRODUCTION

This work brings together three different streams of research and development, all of which are currently very active: mobile games, music games, and procedural content creation. Undoubtedly, the mobile game market is growing very rapidly, as is the number of sold smart phones. Meanwhile, all important platforms (Android, iOS, BlackBerry OS) welcome video games as regular apps and provide means to distribute them. Music-related games have been very successful especially on game consoles in the last years as they also appeal to a typically non-gaming audience (e.g. Guitar Hero or SingStar). However, content for these games has to be prepared manually (by ‘tagging’ videos with additional game-relevant information), so that usually, only a very limited number of songs is available. Only very few music games leave the grounds of simple reaction-based schemes and feature more complex interactions. Notable exceptions are

Rez (Sega) from 2001, a rail shooter that tightly binds the visual impression (and appearance of enemies) to the composed music and also feedbacks user actions acoustically.

Electroplankton by Nintendo from 2005, here the player interacts in various ways with the game to create audio and visual experiences.

More recently, there is also some research interest in mobile music games, e.g. [1] present the turntable-like game Scratch-Off that reminds a bit of the minigame Tap by Tap we present later on. Recent mobile game which also use music to generate levels are e.g. The Impossible Game and Bit Trip Runner.

While on consoles, games have to provide their own music as we cannot expect to find any other on the device, the situation is completely different for smartphones. As smartphones are multi-purpose devices that combine the functionalities of an organizer, a personal computer, a camera, an mp3-player, a navigation system, and last but not least, a telephone, music can be expected to be already there. It is clear that some problems come with using the available songs:

- We have to presume that the music is legally there (the smartphone user has obtained the right to store it).
- Song collections will be largely different for different users, and we cannot simply copy songs from one device to another due to copyright issues.
- In order to make use of the music it may be necessary to start computations (e.g. feature extraction) which are not possible on the smartphone itself due to a lack of computational power or software compatibility.

1E.g., it is possible to access the music library on recent versions of the PS3 console, but this is not utilised by many games

2http://www.sonicteam.com/rez

3http://www.mobygames.com/game/electroplankton

all authors are with the Computational Intelligence Group, Chair of Algorithm Engineering, Computer Science, TU Dortmund, Germany ; email: firstname.lastname@tu-dortmund.de
For the time being, the presented game *BeatTheBeat* is rather at the stage of a scientific study than that of a professional product, thus we ignore these problems now and envision a situation where every user has the same pool of songs available, together with a complete set of extracted features.

**Procedural content generation** (PCG, see e.g. [2] for a recent survey and taxonomy) is possibly the most important research direction in video games at the moment. Its main idea is to use algorithms to extend the possibilities of manual content creation as the rising demands towards the complexity and detail of game worlds renders organization and timing of large game projects more and more difficult. On mobile platforms, there are several new possibilities to obtain context information which can also be used as information source for PCG, e.g. the current location of the devices. A framework that supports this concept is MUPE by Nokia4.

In our case we basically employ a ‘correlated’ source of information (the features extracted from a currently chosen song that is also played as background music, see section III) for creating game levels. That is, the game concept (section II) allows the game to adapt to the music in a way that can be perceived by the player. However, it is challenging to on the one hand use only information that can be automatically extracted (so that in principle every user could provide new music), and on the other hand accomplish a gameplay that is not trivial. With respect to player satisfaction, especially the flow concept of Csikszentmihalyi [3], the influence of the extracted features shall be visible, but not predictable, so that the user is neither overly challenged nor bored.

In this work we present a multi-player mobile game (section II) that employs music-based PCG to create ‘levels’ which are combined into a board by a self-organizing map (section IV). Each level is bound to one specific song and provides an extended experience of the music that goes beyond simple listening. In order to check if players can indeed perceive the influence of the music in the games, we do a blind test user study (section V) where the task is to identify if the ‘right’ features were used to generate the level. This question is synonymous to the one if a level matches the music (or which of 2 does). The two high-level hypotheses we are going to challenge in the remainder of this work are:

- Music (via feature extraction) can successfully be used as information source for game content creation, in a way that goes beyond simple reactive schemes.
- Self-organising maps enable generating well-structured board compositions purely based on automatically derived features (no tagging).

**II. GAME CONCEPT**

*BeatTheBeat* is a two-level multi-player game, where the upper level consists of a board of hexagons each of which can be occupied by one player. Interaction with other players is only possible at the board level. Conquering a hexagon (‘field’) is done by playing an associated minigame and beating the high score on this field. There is also a global score: Players get points for owning as many fields as possible, with a bonus for connected fields (for each connected group this is the square of the number of fields). However, there are no game pieces that have to be moved around, players can obtain some information about the fields they do not own and can choose to try out any field they like. Thus, there is no need for synchronization other than communicating if some player has achieved a new high score and now owns a field. There are no turns so that a player can start (and also stop) a minigame any time.

In the current version of the game, we employ some simplifications that may have to be refined before making the game publicly available:

- There are no defined end conditions. A new game has to be initiated manually by producing a new board (this cannot be done by the player).
- The board itself is considered fully consistent around the players (which has to be ensured by the server).

The realization of the game requires at least 4 different component types: a game engine as background library, a server that is able to ensure consistency for the different players, the board logic that is used to start minigames and record results and the minigames themselves. Currently, there are 3 implemented minigames: *Music Fighter, Tap by Tap, and Music Tower Defense*. In the classification offered by [4], *Tap by Tap* is a rhythm action game, where the two others feature sound agents, controlled by the game in case of the *Music Fighter*, and by the player in case of the *Music Tower Defense*. It is not difficult to integrate other minigames, however there is no well defined plugin interface yet. In the following, we describe all the components and minigames separately.

![Diagram of game layers](http://www.mupe.net)
A. Engine/Server

The game has originally been envisioned to run on Black-Berry OS, but it became clear already at the design stage that nowadays one can hardly ignore Android which currently (January 2012) has a smartphone market share of more than 50%. Our solution to this platform problem is an additional abstraction layer that implements game engine functionality. The engine is based on Java 1.3 (latest common available version of the two platforms) and has been implemented on both systems so that the game is available on both platforms. Figure 2 depicts the layers of abstraction of the engine, on top of which we find the game (BeatTheBeat in this case). The basic functionality of the engine can be separated into 7 categories:

1. Graphics: support for drawing and moving shapes and keeping a complete scene graph
2. Sound: playing songs and providing feature events
3. Input: handling specific devices such as the gyroscope
4. Data management: handling of configuration files
5. Object pooling: prevent steady object creation/deletion for performance reasons
6. Picking: provide events for touchscreen input that is directed towards a specific graphical object
7. Collision detection: provide events for collisions between stored graphical objects

Furthermore, BeatTheBeat is a multi-player game, so that a synchronization mechanism is required. This is provided by a server that is contacted by the mobile devices via TCP/IP whenever an event of global importance occurs (e.g. a player reaches a new high score for a field). The server maintains the board status, that is song, ownership and high score for every field, and also issues new boards when a new game commences. Please see section IV for information about how boards are created.

B. Board

The board is a collection of hexagonal fields, where each of these is associated with a specific minigame and a specific song that have been assigned during board construction and are immutable during one game. Additionally, every field has one or none owner and a current high score. Figure 1 (left) provides an example board. The details of the board construction are discussed in section IV. By clicking on a field, players can obtain information about the current owner, the assigned song and the current high score. They can then freely choose to start the minigame associated to this field or to continue search. A minigame runs as long as the associated song if not preliminary stopped. There is no turn that requires action at some point, players can drop in whenever they have time and play as many minigames as they want. Motivation for this design is threefold:

- It should be easy to conquer ‘some’ field. High scores will rise over time, but if one is not required to follow any specific order of fields but can jump freely over the whole board, chances are good that a field with matching song / minigame / high score is found.
- We enable strategic decisions: When considering the overall number of points, it makes sense to build walls (as in Go) by conquering some fields with minigames the player is very good at.
- In our example board (figure 1), around 80 songs are used. However, the board may be much bigger, depending on the availability of music data. A pinch-to-zoom function enables continuous zooming so that the player can adjust the board display.

C. Music Fighter

This minigame resembles a simple shooter (figure 1) in "Asteroids" style where the player controls a small space ship that needs to avoid collisions with the enemies and their shots. All enemies are spawned in the whirlwind that occupies the screen center and depending on their type, they fly towards the players ship in order to hammer it down, or they fly into another direction and shoot on the players ship. By tapping on the screen, the player issues shots at the enemies, and with a sliding movement or by means of the gyroscope, position and focus of the space ship can be adjusted. However, it is restricted to the area outside of the ‘rail’. While the player cannot ‘die’ during the game, the space ship can be destroyed if it gets too much damage. In this case, the player is penalized by a small time-out after which the space ship appears and has to be positioned on the rail again to restart the game.

Enemy spawning depends on rhythm events (eq. 4), for each event a new enemy is created up to a maximum of 10 concurrent ones. Enemy types depend on the current value of the sensory roughness feature with equal chances for the five possibilities:

1. Artillery: huge, 12 hit points; moves very slowly and fires from time to time in flight direction
2. Shooter: slowly pursues the player and shoots in player direction, 6 hit points
3. Spinner: small and quick, 3 hit points; pursues the player in order to ram the players ship
4. Striker: 2 hit points, very quick, flies in direction of the screen margin but changes direction towards the players ship if the bass feature reaches a certain level
5. Mothership: very huge, 1000 hit points, slow, follows the player ship, only one at a time

D. Tap By Tap

This minigame is depicted in figure 3 (left). It resembles very much popular music games (e.g. Guitar Hero) where players have to press buttons in a predefined order that corresponds to the rhythm or melody of a song. In our case, the matching is not manually encoded but has been generated automatically. A randomized selection of rhythm feature events (with some restrictions in order to enforce a
minimum timespan between two selected events) is used to generate red and blue balls and, as an extension for the later stages of a song, also r/b balls with inscribed direction (arrow to the right or the left). These are issued at the top of the screen and slowly ‘fall down’. For the solid balls, the player has to tap the right color (two areas in the bottom corners of the screen) when the ball exactly fits into the square line. For the ‘direction balls’, the player has to do the same but instead of tap wipe into the right direction. The more exact the player tapping/wiping, the higher the number of earned points. The game finishes when three balls are completely missed or the song ends. Note that the balls need some time to ‘fall down’ until they hit the square, and that this point in time should be synchronized with the triggering event within the music, so that a preview of the rhythm events is necessary.

E. Music Tower Defense

Figure 3 (right) shows a very simple level of this minigame that follows the well known tower defense scheme but adds a dependency of the game contents (tower properties) of the features extracted from the actual song. When the game starts, the player has enough money to build exactly one tower, and the choice is between a bass tower, rhythm tower, and rough tower. These types correspond to the bass feature, rhythm feature, and sensory roughness feature, so that their firing conditions depend on the current properties of the played song. E.g., a bass tower is more suitable if the music is rich in low frequencies, while a rhythm tower pays off most for up-tempo songs. In more detail, the dependencies of the towers on the song features are:
- The rhythm tower shoots once for every rhythm event.
- The rough tower shoots faster for more turbulent songs.
- The bass tower also shoots once for every bass event, but shot intensity depends on bass intensity.
- The rough tower shoots faster for more turbulent songs.

There are also different types of enemies (possessing different movement speeds and hit points) so that achieving a high score requires a tower building strategy that takes the underlying music into account. The game ends if either the song finishes or too many enemies reach the end of the path in the bottom right corner. Instead of Computational Intelligence techniques as suggested in [5], we employ a simple generate and test scheme [2] for obtaining rules for translating feature values to tower and enemy actions in order to generate an appropriate ‘level’ for each song. More automation is subject to ongoing work.

III. Music Features

In general, music can be described by features obtained from different sources:
- Audio features are estimated from the recorded audio signal. Many algorithms are available as intermediate steps: filtering of the predefined frequency ranges, transforms into spectral, cepstral or phase domains, analysis of harmonic and noisy components etc.
- Symbolic features are calculated from the score and provide a fast possibility to obtain exact high-level music characteristics such as musical key and mode, number of simultaneously playing instruments, harmonic characteristics and so on.
- Metadata features are derived from other manually saved information, such as year and place of composition.
- User statistics gather information from music listeners, e.g. the most frequent last.fm tags or similarity-based playlist characteristics.

Apart from the specific pros and cons of these sources all of them may be valuable and there exist some studies which state that the combination of different feature groups is indeed very promising [6], [7]. However we concentrate here only on audio music features because of their major advantage, namely that they can be extracted from any recorded music piece. This can not be guaranteed for other feature sources: score and metadata features are not always available, and user statistics may be erroneous or simply not existing for less popular music - consider a situation where the player himself composes music as new game content.

To name a couple of references as an introduction to audio feature extraction, one of the first extensive works was provided by Tzanetakis [8]. In [9] a set of temporal, energy, harmonic and perceptual audio features was discussed in detail. A generic framework for automatic feature construction based on genetic programming was introduced in [10]. A revised list of features was provided in [11]. Another comprehensive list of features is described in the MIR Toolbox manual [12]. Timbre-invariant enhanced statistics for chroma vectors were developed in [13]. Many features from the last three references are available in the open source framework AMUSE [14] and
were initially extracted for the set of songs employed during game development.

While selecting features, we have to take into account that we need a perceivable influence of the music, that is, it should be possible for humans to visually detect a relationship, but it should not be trivial (e.g. beats). For this task, we employ a self-implemented feature visualizer as depicted in figure 4.

After preliminary experiments, we selected the following audio features to be involved in content generation for our music games:

- Root Mean Square (RMS) of the time signal $x$ with $N$ samples provides a rough estimation of its energy:

$$x_{\text{RMS}} = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} x^2}. \quad (1)$$

- Sub-band energy ratios correspond to the relative spectral energy rate of the four frequency bands $\left[ 0; \frac{f_s}{16} \right], \left[ \frac{f_s}{16}; \frac{f_s}{8} \right], \left[ \frac{f_s}{8}; \frac{f_s}{4} \right]$ and $\left[ \frac{f_s}{4}; \frac{f_s}{2} \right]$ (where $f_s$ is the sampling rate which was set to 22050 Hz for our study).

- Sensory roughness as implemented in the MIR Toolbox based on [15] is measured by determining the peaks of the spectrum, and taking the average of the dissonance between all possible pairs of peaks.

- Average angle in phase domain was introduced as feature for music classification in [10]; together with phase domain average distance it could distinguish very well between music with strong (pop) and weak (classical) percussive pulses. Phase vectors $p_i$ are constructed from the original feature series $x_i$:

$$p_i = (x_i, x_{i+d}, x_{i+2d}, \ldots, x_{i+(m-1)d}). \quad (2)$$

Delay $d$ between the analyzed values was set to 1 and the dimensionality of phase domain $m$ to 2.

- Beat Per Minute (BPM) number is a tempo feature which measures the rate of the strong repetitive music impulses (beats) as implemented in [16].

Furthermore, we derived two high-level characteristics from the original audio signal features in order to capture the intuitive human audio impression of heavy bass and rhythm events. Both are to a certain extent loudness based and thus different from purely periodical features as e.g. the beat: While bass energy stands for the resonance people can feel in their stomachs while playing music very loud, rhythm event somewhat resembles the moves people tend to do unintentionally (often with their fingers) if they listen to music. The reasoning behind creating these features is simply to increase the immersion of the player into the game by means of "easily accessible" features.

- Bass energy was calculated from the weighted RMS components of the first four Equivalent Rectangular Bandwidth (ERB) [17] sub-bands:

$$B_e = 1 \cdot x_{\text{RMS}}\left(ERB_1\right) + 1 \cdot x_{\text{RMS}}\left(ERB_2\right) + 0.7 \cdot x_{\text{RMS}}\left(ERB_3\right) + 0.1 \cdot x_{\text{RMS}}\left(ERB_4\right). \quad (3)$$

- Rhythm event $R_e \in \{0; 1\}$ is calculated as follows: at first $x_{\text{RMS}}(i)$ is normalized so that the minimal and maximum values from the interval $[i - 200; i + 200]$ are mapped to 0 and 1. Then, the normalized derivation $\hat{x}_{\text{RMS}}(i)$ and then $R_e$ are estimated:

$$R_e(i) = \begin{cases} 1 & \hat{x}_{\text{RMS}}(i) > 0.55 \\ 0 & \hat{x}_{\text{RMS}}(i) \leq 0.55 \end{cases} \quad (4)$$

Finally, if $R_e(i) = 1$ and $R_e(j) = 1$ where $|i - j| < 2205$, the rhythm event with the smaller corresponding $\hat{x}_{\text{RMS}}$ is set to zero.

It shall be noted that music feature extraction is currently almost impossible to achieve online when a specific song is selected within the game as this would at least take some minutes. While this resembles an important direction of research, we currently presume that all feature sets needed for a new instance of the game are obtained in a "preparation phase". Possible remedies for this computationally intense step would include time-slicing (analyse only a small part of a song at any time) and or indexing (for looking up exact or approximate feature data in a server-based data base), but are not implemented yet.

**IV. SOM Based Board Construction**

Self-Organizing Maps (SOM) [18] are a special type of artificial neural networks that perform unsupervised dimension reduction to a predefined ‘map space’ that is chosen as 2D grid in our case. They are especially well suited to our case because we start with a number of songs, each accompanied by a vector of extracted numeric features, and want to obtain a landscape-like structure. This structure resembles our board, consisting of one field per song. The placement should respect similarities and thus be somewhat intuitive without the necessity to introduce high-level tags as e.g. genres. As we build a game and not a music classification system, we have the same requirement as for our minigames: the relation between
ordering and music shall be intuitive, but not too strong, and provide some means to redo it in a different way. The random elements in SOM training enable this: started several times, slightly different maps emerge. However, all of them possess an intuitive ‘meaningfulness’ in the sense that the map could have been designed manually with the feature data of the songs in mind.

The integration of SOM into music management is not novel and was successfully applied in several studies - however not with the target to create music games. MusicMiner [19] is a framework which uses an extension of SOM (Emergent SOM) for the mapping of songs represented by large generic feature vectors (approach similar to [10]) into topographic maps. Another SOM modification, Mnemonic SOM, which enables to produce maps of given recognizable shapes, was applied to organize Mozart works as the silhouette of the composer in [20]. In [21] three-dimensional landscapes for navigation through music collections are generated by SOM and so-called Smoothed Data Histograms.

We provide a rough description of the learning/training process of a SOM, please consult the literature (e.g. [18]) for more details. At first, we set up a square grid of nodes (large enough to contain one node for every available song, but not larger) and initialize it with randomly chosen weights (feature vectors). Then, the following steps are repeated at least 4000 times:

1) A song is chosen at random
2) The best matching unit (BMU) is computed, that is the node with the smallest distance to the songs feature vector
3) For the BMU and the neighboring nodes, we adjust the weights to make them more similar to the chosen songs feature vector. The closer to the BMU, the stronger the modification.

The implementation uses the code provided at http://www.ai-junkie.com/ann/som/som1.html with small modifications and corrections, the learning rate \( \tau \) in the weight \( w \) adjustment in (5) is set to 0.1.

\[
    w_{\text{new}} = w_{\text{old}} + \text{distanceFalloff} \cdot \tau \cdot (\text{features} - w_{\text{old}}) \quad (5)
\]

We consider the following features over which the distance is computed as Euclidean distance:

- root mean square (mean, standard deviation, kurtosis and skewness) of the whole song
- root mean square derivation (mean and standard deviation) of the whole song
- sub-band energy ratio: means of all 4 bands, weighted by the root mean square at the matching point in time
- mean of angles in phase domain

This selection is rather due to experimental testing than theoretical considerations.

After training the SOM, every song is attached to its BMU node. Figure 5 depicts the SOM after training and assigning phase. The assigning phase corrects the problem that several songs may have the same BMU by an iterative process: Over all nodes, the song with the largest distance (error) to its BMU is detected and distributed to a nearby empty node. This process is repeated until all nodes have at most one attached song left. In order to improve the visual appeal of the result and also to enable visual debugging, an independent k-means clustering with \( k = 5 \) is applied to the feature vectors and the nodes are colored according to their cluster. This color scheme is taken over to the board as seen in figure 1. We obtain the hexagonal plane from the square grid of nodes by moving every second column down by half the node size.

Visual inspection of the produced boards shows that the mapping is intuitive. Especially very extraordinary songs that do not match well with any other are placed at the borders, often as islands.

V. PLAYER SATISFACTION: FEATURE USE FOR CONTENT CREATION

At several places in this work we stressed the challenge to use the feature information extracted from music in a way that the relationship is perceivable but not too obvious. We took that into account while designing the board layout method as well as in the minigames. Maybe the relationship is the most visible in the Tap by Tap game. Therefore, we employ this minigame in a preference learning style experiment (let the user choose between two possibilities) in order to find out how strong the perceived relationship is. In principle, similar tests would be possible for the board creation and also the other minigames. However, for the minigames we can embed a user study directly into the game which would not be possible for the board, and due to time restrictions we performed only one study.

The basic idea of the following experiment is to let the users play the same Tap by Tap setting (same song) two times, once with the features extracted from the song, and once with the features from any other song. They shall then rate which of the games better matched the music, which is similar to recognizing if the features belong to the song or not. Note that we do not expect and also do not target a perfect match. This could be achieved e.g. by using only the beats, so that it suffices to compare song tempo with the beats, visualized as
balls. Furthermore, the game currently does not possess any dynamic difficulty adjustment (DDA) mechanism [22], which would be difficult to establish while keeping the multiplayer high score concept. Thus, we are also interested in seeing if different types of users (more or less involved in video games and music) behave similarly or exhibit striking differences. We have to admit that the study is small, meaning that the results should be rather understood as a trend.

a) Pre-experimental planning: By doing some manual testing, the amount of time used for each song was adjusted to 1 minute, and the number of songs presented to each user to 3. Note that each song is partly played two times (with different features sets), and that the length of the whole experiment should be significantly smaller than 10 minutes.

b) Setup: From a pool of 82 songs of very different genres (pop, rock, ethno, avantgarde), 3 songs are chosen randomly for every user, and 3 additional feature sets are also randomly determined. Prior to starting the game, the user is asked by an interviewer (who may help to explain the game in a test run if necessary) if she/he plays videogames often and if she/he deals a lot with music. No other demographic information is recorded.

Every user is then in random order presented with the Tap by Tap game for 3 songs, and for every song the first minute is played two times, once with the correct features, once with one of the additional features sets. After playing each set of two minigames, the user is asked which one matched the music better, and after running all 3 × 2 short minigames, the results are displayed (“yes” for selecting the right feature set, “no” for selecting the wrong one).

c) Task: We want to see a clearly visible difference from a random distribution (yes and no occuring almost equally often). However, it is not favourable to obtain too extreme results (all “yes”). As far as this is possible, we would like to apply binomial tests to the results, however, this is hardly meaningful for less than 30 samples, so that we restrict the use of tests to this case. In the ideal case, the result would be statistically significant but still far from 100% “yes”.

d) Results/Visualization: The total number of players in the study was 20, their distribution and the results are displayed in table I. Significance testing on the single groups is not feasible due to their size, but for the whole data set (39×39×39×60 answers), the p-value of the binomial test (with probability \( p = 0.5 \)) is 0.02734. This means that the likelihood for obtaining this result if choices are completely random is below 3%.

e) Observations: As the study is small, we have to be very careful with interpreting the result. However, we can observe that either dealing a lot with music or playing video games a lot suffices as qualification to recognize a difference in the feature sets most of the time. From the result-per-song distribution (not shown here), we can see that there is also a dependency on the song itself. For some, the difference seems easy to see, for others it is very hard to tell. Especially for the very experimental music, the right features could rarely be detected. The test persons also reported that the audio feedback they got while tapping (a constant sound that does not adapt to the song or the accuracy of the tapping) is very annoying, which most likely has an influence on the study results. Generally, people were happier with the game when they liked the song.

f) Discussion: The study (however small) shows that for the Tap by Tap game we have achieved what we wanted: The matching of features to songs is obviously not trivial, but most users had a success rate of around 2/3. We can also state that for the most important user groups (plays games and/or deals much with music), the difficulty setting seems to be acceptable, whether for the last group it is not. However, it is questionable if the last group would play a music game by their own initiative, so we may ignore this problem for the time being. Furthermore, it would be very interesting to see the results of a much larger study, possibly also taking in the personal music preferences of the users (e.g. by means of using the board to select songs).

V. CONCLUSIONS AND OUTLOOK

In our work we presented a novel multiplayer mobile music game. For both levels - the board and the minigames - we integrated procedural content generation based on different low-level and high-level audio characteristics. The framework structure was built in a generic way, so that the clear distinction between the two levels is given and it is possible to port the code easily for different operating systems, as has been done already for BlackBerryOS and Android. Several experiments were performed to select and analyze the convenient features for the SOM input / board generation as well as for the concrete minigames. The user experiment showed that the filtered rhythmic event in Tap by Tap was a good choice: it was neither too hard to recognize rhythmic relationship nor too simple to achieve the highest possible score within a few tries (and get bored at the same time).

However a lot of implementation and experimental ideas have to be deferred to future work. Beyond several evident steps (extended testing, development of new minigames and integration of further music-related features etc.), further promising approaches can be considered. DDA shall be applied for the increasing complexity during the minigame process depending on user success, although we would then need a mechanism to weight scores so that they can be compared for

<table>
<thead>
<tr>
<th>plays games</th>
<th>knows music well</th>
<th>users</th>
<th>correct</th>
<th>wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>yes</td>
<td>7</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>7</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>3</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>all</td>
<td>all</td>
<td>20</td>
<td>39</td>
<td>21</td>
</tr>
</tbody>
</table>

TABLE I

User study results for the Tap by Tap minigame according to user groups and (last line) for the aggregated set; last 2 columns give rightly/wrongly recognized features.
the high score table. Computational Intelligence methods and more specifically, Evolutionary Algorithms, appear suitable to be involved in the balancing, for example for feature selection and or enemy properties. In order to test whether users recognised the correlation between songs and game features in a statistically significant way, we would need to run larger studies. It is also thinkable to integrate features from other sources such as organizing the song map with the help of last.fm user genre tags. This would make it possible to create certain regions like ‘electronic swamp’ or ‘heavy metal woods’. For further long term satisfaction it is possible to integrate ‘magic’ items into the board available only after the player attains some level, e.g. the achievement of the highest score for ten minigames.

ACKNOWLEDGMENTS

This work was partially supported by the Klaus Tschira Foundation. The authors would like to thank Research in Motion (RIM) for hardware support and the reviewers for several important hints on how to improve the paper.

REFERENCES


