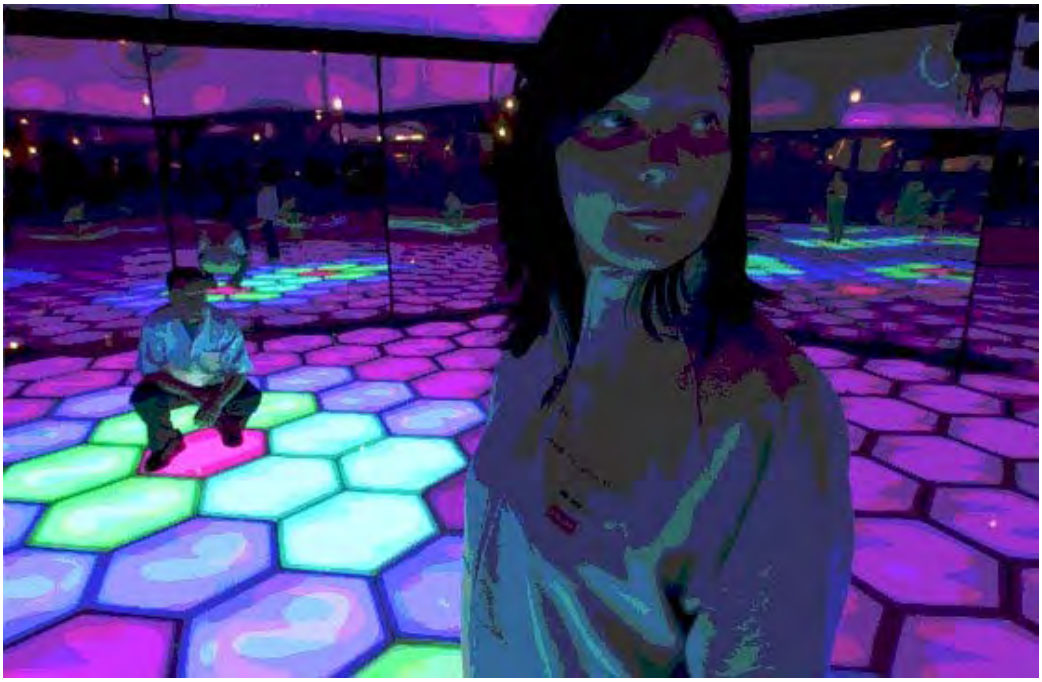
Using a luminous-tactile floor and auditory signal processing to make an automatic disco

Project Report



Student: Ruihua Jin (rjin@student.ethz.ch)

Supervisors: Tobi Delbruck, Kynan Eng

Institute of Neuroinformatics

University and ETH Zurich, Switzerland

November 2006 – January 2007

Table of contents

Overview

The luminous tactile floor developed for INI's Expo.02 project is ideal for building "the mother of all disco floors." The information retrieved by using auditory signal processing methods, together with the tactile information that comes from the floor's load sensors will be used to drive the floor's luminous output in order to produce a compelling and automatic dance floor that senses both the players and the music being played.

I. Original goal of auditory signal processing

The original goal of auditory signal processing was to find processing methods for automatic beat tracking and extracting the style of music. The tools used were low-pass filters and FFT. However, after a long try the author came to the conclusion that beat tracking is not trivial at all as it maybe seems to be. One main reason is explained in [1]:

> *Rule-based approaches have never been applied to audio and have solely been used to code sensible but simple music theoretic rules in order to model music psychology expectations. The reason that they have never been used on audio signals is possibly because they are not easily expanded to cope with erroneous data and hence would perform poorly on the inexact data produced by onset detection algorithms.*

During experiments for beat tracking the author encountered several difficulties:

- Many music pieces have beat played in the background, and in the foreground there is somebody singing with much power. It is rather difficult to distinguish these two tracks computationally only based on the bytes retrieved from microphone.
- When after a period of mild melody the music is *gradually* getting more powerful, then onset detection does not work well, since the current sound power does not clearly differentiate itself from the previous one.
- When following a strong beat the sound switches between up and down rapidly, then beat tracking is also difficult here.

More successful methods for beat tracking are multiple agent approaches and probabilistic, model-based methods. But to implement one of them would surely go beyond the scope of a term project.

II. Secondary goal of auditory signal processing

After the author failed to achieve the original goal of auditory signal processing – extracting the beat and the style of music, she turned to accomplish the secondary goal: using sound energy to follow music (see below).

III. Floor’s luminous output

Five effects were implemented to give a rich luminous output of the floor.

Scope of the work

I. Auditory signal processing

The purpose of auditory signal processing is to determine the sound energy level by comparing the current sound energy with the previous local average of the sound energy, the relative value from the comparison indicates whether the sound energy has been increased or decreased. The audio signals are retrieved one buffer after another using a microphone, each buffer contains 1024 samples and represents one unit of signal processing. The sample rate is 8 kHz, and the sample size is 2 bytes.

Computation of magnitude of the audio signal

To compute the magnitude of the audio signal, the signal data are first fed into a low-pass filter which passes 0 – 1 kHz frequencies. After that the data are analysed using Fourier transform which returns a set of complex numbers, for each complex number

$$x = a + ib$$

the magnitude is equal to

$$\sqrt{a^2 + b^2}$$

Computation of sound energy of the audio signal

For each signal data buffer the magnitudes are first computed, then based on these new magnitude values and the old magnitude values the sound energy is computed as follows:

$$E = \sum_{i=s}^{n-1} pm[i]^2 + \sum_{i=0}^{n-1} cm[i]^2$$

where `pm` is the magnitude array of the previous data buffer, `cm` is the magnitude array of the current data buffer, `s` is the starting index for the previous magnitude array, and `n` is the length of the magnitude array.

The overlap between successive data buffers has been proved to be necessary in the practice [1], and in the current implementation the overlap size is 70%.

Determination of sound energy level

In the current implementation there are four sound energy levels from level 0 to level 3. Whether the sound energy level is increased or decreased depends on the following criteria:

- The sound energy level is increased by 1 if

$$(E_c \geq m_1 E_p) \; and \; (E_c + E_p \geq m_2 E_{avg})$$

where E_c is the current sound energy, E_p is the previous sound energy, E_{avg} is the sound energy average of the last 10 sound energy values exclusive the previous and the current sound energy values. Since the sample rate is 8 kHz and one data buffer contains 1024 samples, E_{avg} corresponds to the local 1.28-second average.

The author has tried different combinations of the values of m_1 and m_2, and has found out that with

$$m_1 = 1.2 \; and \; m_2 = 2.5$$

the sound energy level determination performs well.

- The sound energy level is decreased by 1 if

$$(E_c \leq m_3 E_p) \; and \; (E_c + E_p \leq m_4 E_{avg})$$

The author has tried different combinations of the values of m_3 and m_4, and has found out that with

$$m_3 = 0.8 \; and \; m_4 = 1.5$$

the sound energy level determination performs well.

- Otherwise, the sound energy level remains unchanged.

Computation of modulation spectrum of the audio signal

The computation of modulation spectrum of the audio signal is based on three equations 1, 2 and 3 in [3], and its implementation follows the MATLAB implementation available in [4]. The steps of computing the modulation spectrum of one data buffer are as follows:

Step 1: low-pass filter the data, which passes 0 – 1 kHz frequencies;

Step 2: frame the data with a 50% overlap in time:

one hop length is 10 in ms, which corresponds to

$$10 \cdot 8 \text{ kHz}/1000 = 80 \text{ samples},$$

so with a 50% overlap each frame contains 160 samples, which results in a 160 x 14 matrix for a 1024-sample-buffer, one frame per column;

Step 3: window the data by a Hanning windowing function;

Step 4: transform the windowed data by either a modified discrete cosine transform (MDCT) or a modified discrete sine transform (MDST). Odd columns are transformed by MDST, and even columns are transformed by MDCT. The transformation matrix is of size 80 x 160, so the result of a transform is an 80 x 7 matrix.

Step 5: combine the two adjacent MDCT and MDST into a single complex transform, and compute the magnitude matrix based on this complex matrix. The final magnitude matrix is of size 80 x 7, each column represents one time unit and each row represents one frequency, the values in the matrix indicate the power of frequencies at some specific time.

II. Floor's luminous output

The floor’s luminous output is affected by two factors: the sound energy level and the loads of the individual tiles. The sound energy level determines which tiles are lit up and which are not. When the sound energy is getting stronger, the number of lighting tiles will be increased; otherwise, the number of lighting tiles will be decreased. On the other hand, the colors of the lighting tiles are determined by the tiles’ loads.

Determination of lighting tiles

There are five effects implemented: CYCLE, EXPANSION, GEOMETRY, RANDOM and TRACKER.

In the CYCLE effect the floor makes the impression that the stronger the sound energy, the more tiles are moving (Figure 1). The CYCLE effect is good for music pieces which are in a rapid rhythm.

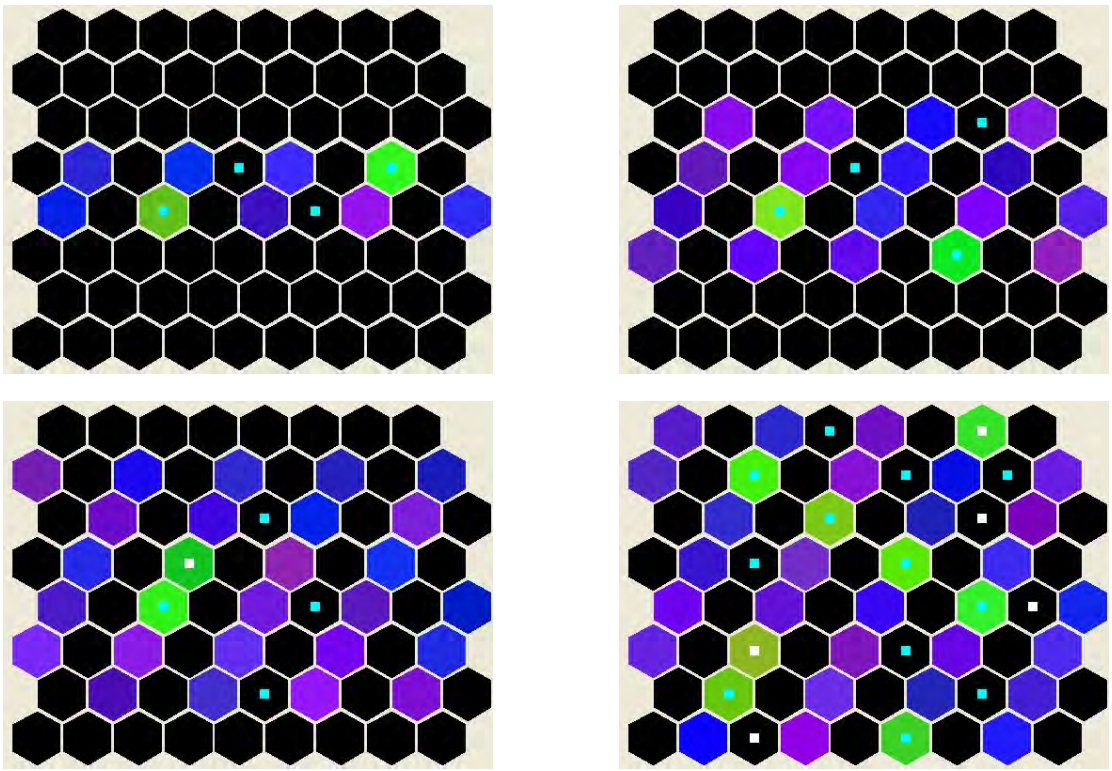


Figure 1. Floor’s luminous output of the CYCLE effect when the sound energy level is 0, 1, 2 or 3. A dot simulates one person on the tile.

In the EXPANSION effect the lighting area in the middle of the floor expands with the sound energy (Figure 2). The EXPANSION effect is suitable for tender music.

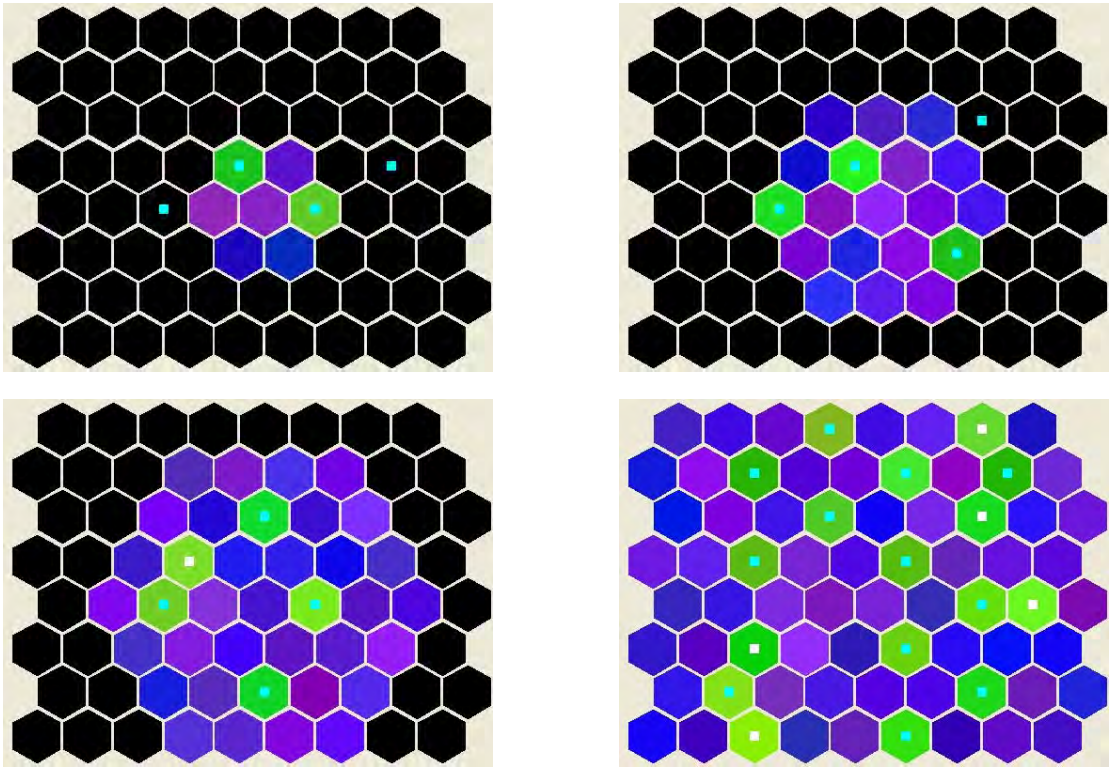


Figure 2. Floor's luminous output of the EXPANSION effect when the sound energy level is 0, 1, 2 or 3. A dot simulates one person on the tile.

In the GEOMETRY effect the stronger the sound energy, the more lighting tiles are in a group (Figure 3). The GEOMETRY effect performs well for different kinds of music pieces.

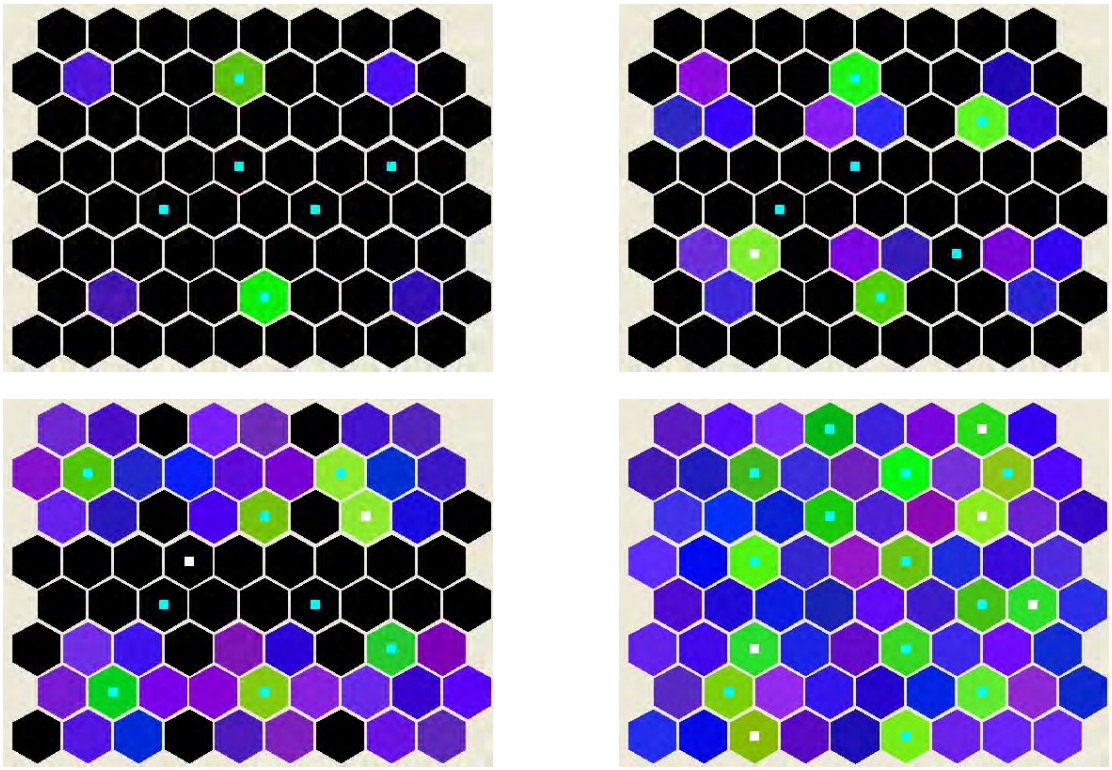


Figure 3. Floor's luminous output of the GEOMETRY effect when the sound energy is 0, 1, 2 or 3. A dot simulates one person on the tile.

In the RANDOM effect the number of lighting tiles increases with the sound energy level and where these tiles locate is determined at random (Figure 4). The RANDOM effect does not indicate the sound energy very well, since the lighting tiles are spreading around, and it takes effort to tell at one sight whether the number of lighting tiles has been increased or decreased.

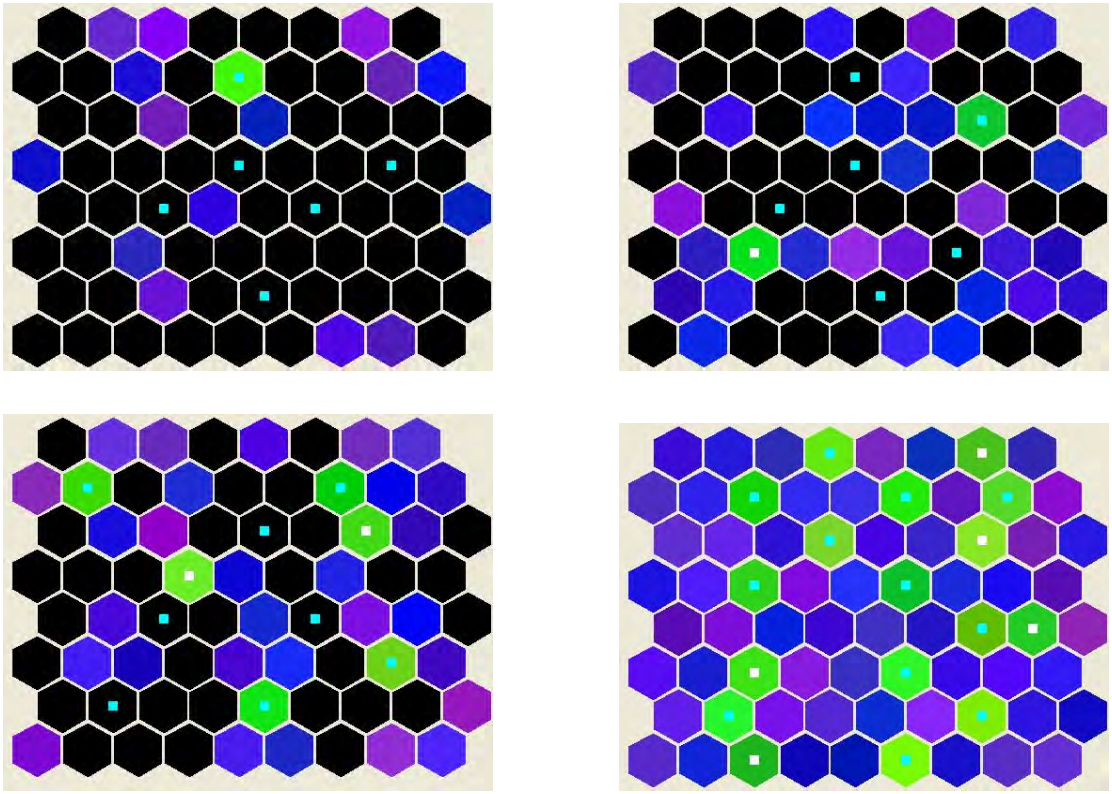


Figure 4. Floor's luminous output of the RANDOM effect when the sound energy is 0, 1, 2 or 3. A dot simulates one person on the tile.

The TRACKER effect tracks the players and shows patterns with the players in the middle. The stronger the sound energy, the more lighting tiles are around a player, and active players have more lighting tiles around them than inactive players (Figure 5, 6). To determine whether a player is active or inactive, the program records the player's activities during the last 300 floor update cycles, and assigns different weights to different activities: if a player stays on the same tile, then his activity value remains unchanged, if a player pogos, then his activity value is increased by 1, if a player steps onto another tile, then his activity value is increased by 2, and if a player jumps, then his activity value is increased by 3. If the activity value of a player is equal or larger than 10, then he is considered to be active, otherwise he is considered to be inactive.

When a player jumps or pogos, then his path is shown for 500 ms, with the brightness of the color gradually increasing (Figure 7). The path length and the path color can be set by the user. If the user sets the path length to 0, then no path will be shown.

The most active area is highlighted for 500 ms at the given rate (Figure 7). The highlighting rate and the highlighting color can be set by the user. If the user sets the highlighting rate to 0, then no highlighting will be shown. The activity value of a continuous area is computed as the sum of the activity values of the tiles and the number of persons on the tiles. The activity value of each tile is accumulated between two successive highlighting events, it is increased by 1, if somebody pogos on the tile, it is increased by 2, if somebody steps from another tile onto it, and it is increased by 3, if somebody jumps onto it.

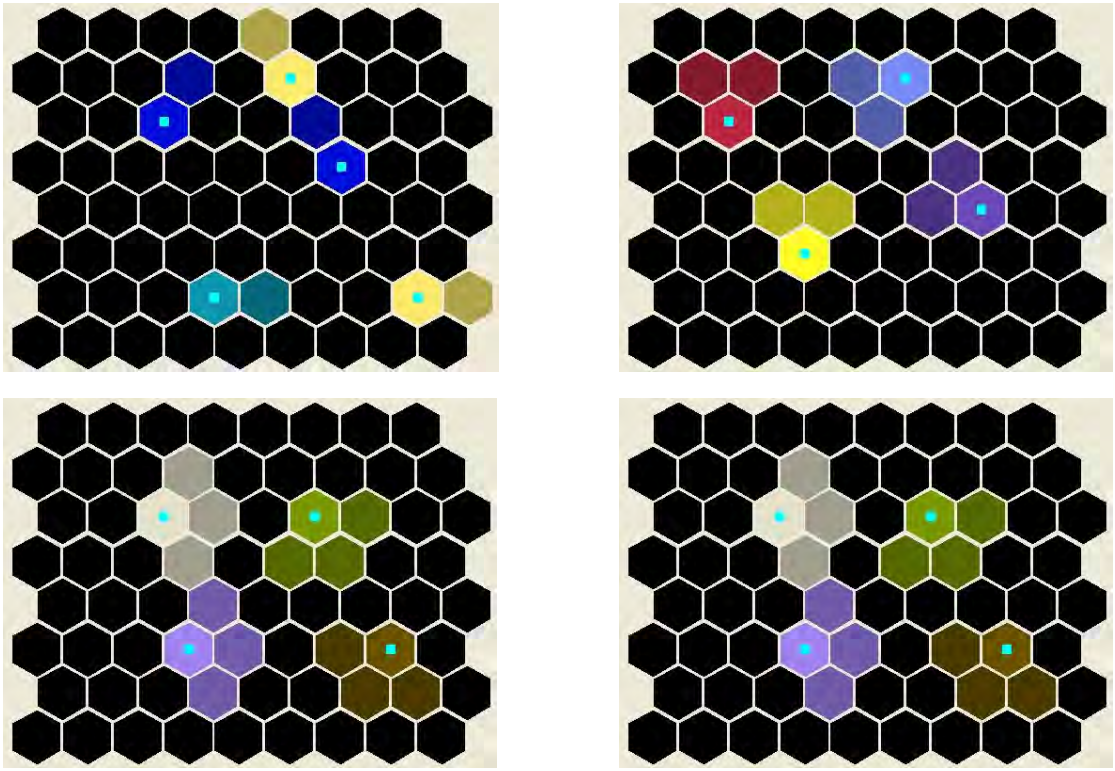


Figure 5. Floor's luminous output of the TRACKER effect
when the sound energy is 0, 1, 2 or 3 and the players are inactive.
A dot simulates one person on the tile.

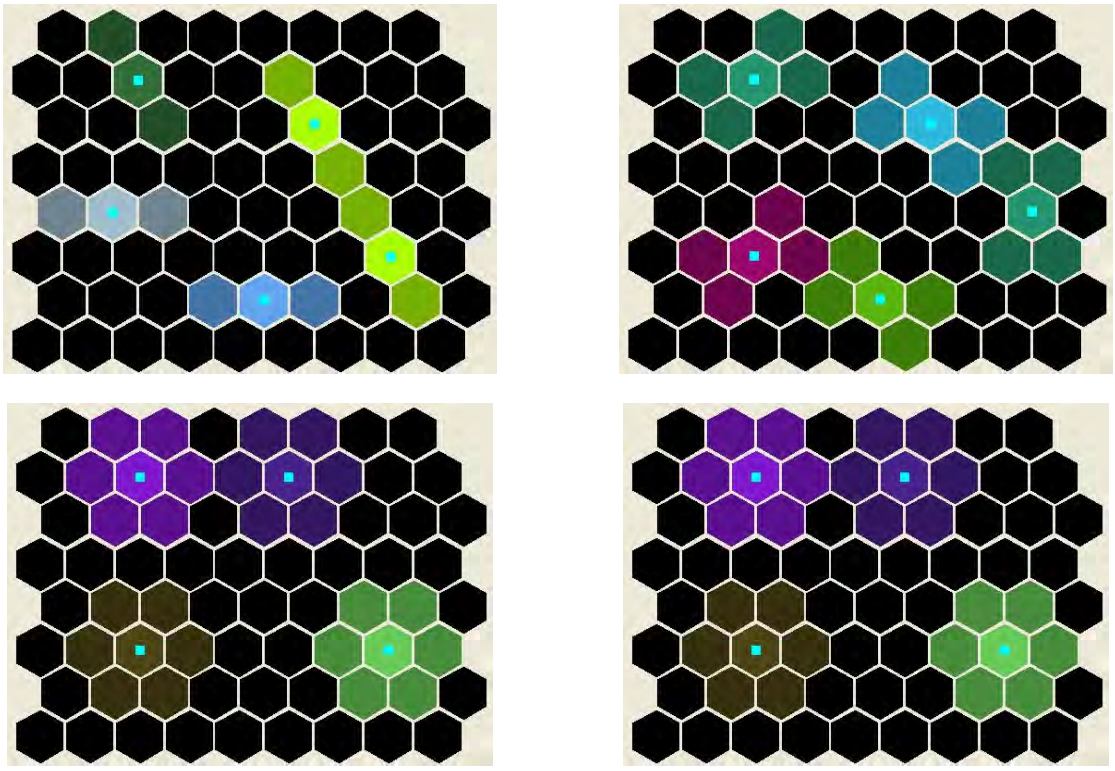


Figure 6. Floor's luminous output of the TRACKER effect
when the sound energy is 0, 1, 2 or 3 and the players are active.
A dot simulates one person on the tile.

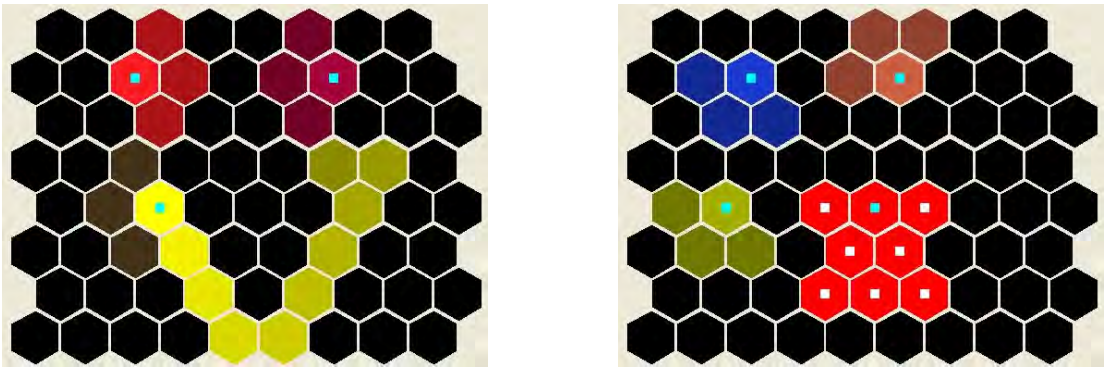


Figure 7. Floor's luminous output of the TRACKER effect.
Left: when a player jumps or pogos, then his path is shown (here yellow tiles).
Right: The most active area is highlighted at the given rate (here red tiles).
A dot simulates one person on the tile.

Determination of tiles' colors

For the CYCLE, EXPANSION, GEOMETRY and RANDOM effects the colors of the individual tiles depend on the tiles' loads. The user can set thresholds for light load and heavy load, and the user can also set red, green or blue colors for unloaded tiles, lightly loaded tiles and heavily loaded tiles. Concretely, the colors are generated as follows:

- If "Red" is chosen, then the color has
  red = random(180, 255), green = random(0, 150), blue = random(0, 50);
- If "Green" is chosen, then the color has
  green = random(180, 255), red = random(0, 150), blue = (0, 50);
- If "Blue" is chosen, then the color has
  blue = random(180, 255), red = random(0, 150), green = random(0, 50).

In the TRACKER effect each player has his own color which is generated at random when the sound energy level changes: red = random(0, 255), green = random(0, 255), blue = random(0, 255). The tile on which the player stands on has the brightness value 1, and other adjacent lighting tiles of the player has the brightness value 0.7.

III. Game GUI

The GUI of the game offers the user the possibility to choose the effect he wants. There are six entries in the upper left combo box: CYCLE, EXPANSION, GEOMETRY, RANDOM, TRACKER and ALL. The former five are the names of the effects, and the last one “ALL” results in that all the effects are played one by one, each within a certain duration which can be set using the slider beside.

The GUI also has some Swing components for the user to set thresholds of light and heavy loads, colors for unloaded, lightly and heavily loaded tiles, the path length and the path color, the highlighting rate and the highlighting color.

The GUI also shows the magnitude and the modulation spectrum of the audio signal. In the magnitude plot, the horizontal axis is the time, and the vertical axis is the magnitude. In the modulation spectrum plot, the horizontal axis is the time, and the vertical axis is the frequency, the more power has the frequency, the darker is the corresponding bar.

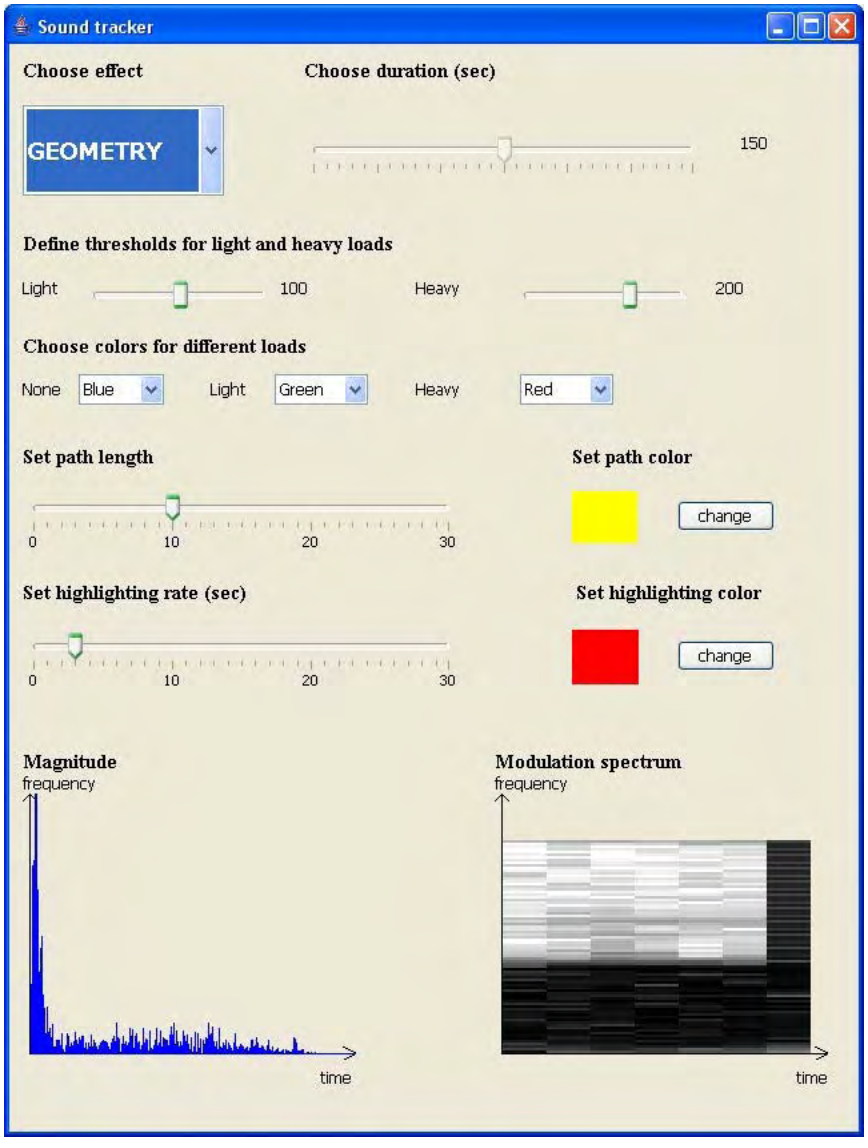


Figure 8. Sound tracker GUI.

Implementation issues

The current project named Sound Tracker is located in the `ada.games.soundtracker` package. There are nine classes in the package: `SoundTracker`, `SampleAudioAnalyser`, `MagnitudePlot`, `ModulationSpectrumPlot`, `SoundTrackerEffect`, `SoundPlayerTracker`, `Player`, `SoundTrackerGUI` and `SoundTrackerConstants`.

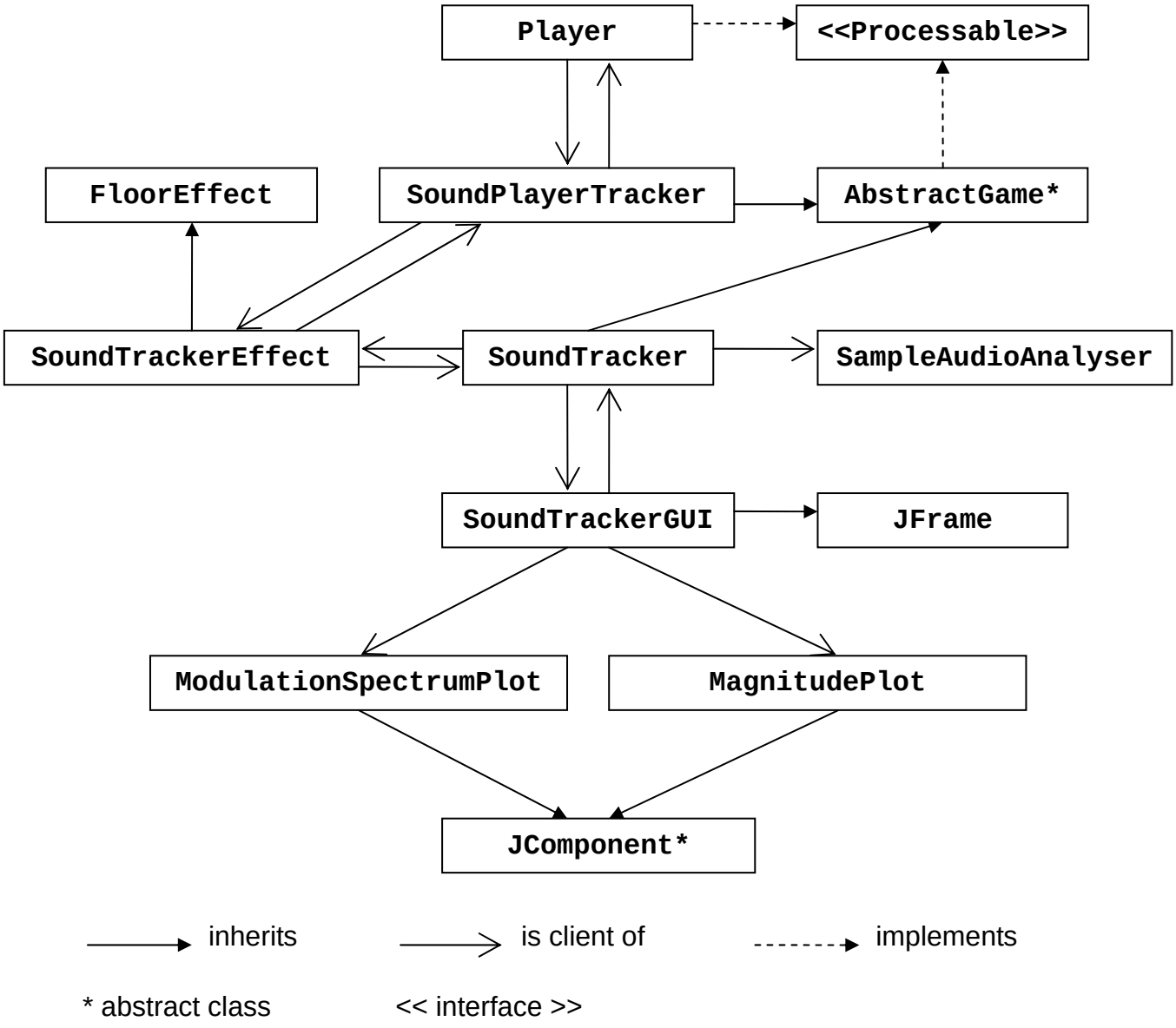


Figure 9. Relationship between classes.

The `SoundTracker` class is the key class of the game. It inherits the `AbstractGame` class, and implements the `process()` method which updates the game during a floor update cycle, the class also overwrites `startGame()` and `stopGame()` methods which are called to start resp. to stop the game.

When a `SoundTracker` object is initialized, an audio line is opened to the microphone, and then the audio signal data can be obtained from the line continuously, where the reading process runs as a separate thread. Each time when a data buffer is read in, the `SoundTracker` object passes this data buffer to a `SampleAudioAnalyser` which analyses the data and then returns the magnitude, the sound energy and the modulation spectrum of the data. The `SoundTracker` object then evaluates the sound energy and determines whether the sound energy level is to be increased or decreased. It also asks a `SoundTrackerGUI` to plot the magnitude and the modulation spectrum.

The `SoundTracker` class serves also as a controller between `SoundTrackerEffect` and `SoundTrackerGUI`. As soon as the user changes the settings of the game, the `SoundTracker` informs the `SoundTrackerEffect` to update the floor's luminous output accordingly.

The `MagnitudePlot` and the `ModulationSpectrumPlot` objects are components in a `SoundTrackerGUI`. The former plots the magnitude of the audio signal data, and the latter plots the modulation spectrum of the audio signal data.

The `SoundTrackerEffect` class updates the floor's luminous output according to the current effect, the current sound energy level and the tiles' loads. And it also offers the methods for setting the sound energy level, the effect, the effect duration, the thresholds of light and heavy loads, the colors for unloaded, lightly and heavily loaded tiles, the path length, the path color, the highlighting rate and the highlighting color.

The `SoundPlayerTracker` class is the key class for the TRACKER effect. It extends the `AbstractGame` class. It keeps track of the players and calls each player's `process()` methods to update the player's state, it also updates the activity values of the tiles during each floor update cycle and highlights the most active area at the given rate.

The `Player` class implements the `Processable` interface. It represents one player on the floor. This class records the activities of the player, determines whether the player is active or inactive, makes pattern for the player, and shows the path if the player jumps or pogos.

The last class which is not shown in Figure 9 is the `SoundTrackerConstants`, it has some final static fields for default values of the settings of the game.

Results

When music is being played, the lighting tiles of the floor form different patterns according to the sound energy, and each lighting tile gets its individual color. So the floor is colorful and has its own way of tracking the music.

There is one problem with the TRACKER effect: when two players stand on two neighboring tiles, then only one player can be detected and tracked, the other one cannot be detected by the program so that he cannot get effects around him.

References

[1] A. Klapuri and M. Davy. Signal Processing Methods for Music Transcription. Springer-Verlag, Juni 2006.

[2] S. Greenberg and B. Kingsbury. The Modulation Spectrogram: In Pursuit of an Invariant Representation of Speech. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1997.

[3] M. Vinton and L. Atlas. A Scalable and Progressive Audio Codec. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2001.

[4] M. Athineos. Modulation-Spectrum Audio Coding in MATLAB; Online at: http://www.ee.columbia.edu/~marios/modspec/modcodec.html, consulted in November – Dezember 2006.

[5] The Java™ Tutorials, Trail: Sound; Online at: http://java.sun.com/docs/books/tutorial/sound/index.html, consulted in November 2006.