Computer Music - Languages and Systems
HMI - Homework #3

Group 11 - BeetleJUCE:
Adriano Farina, Alessandro Gorni, Giuseppe Risitano, Enrico Regondi, Rebecca Superbo

May 31st, 2022

BEATRUN

1 Idea

Beatrun is a multi-sensor music system with cool 80's outrun vibes. The rhythm of the music should follow the heartbeat of the performer, while the pitch of the triggered notes is governed by the distance between the performer's hand and a distance sensor.

2 Implementation

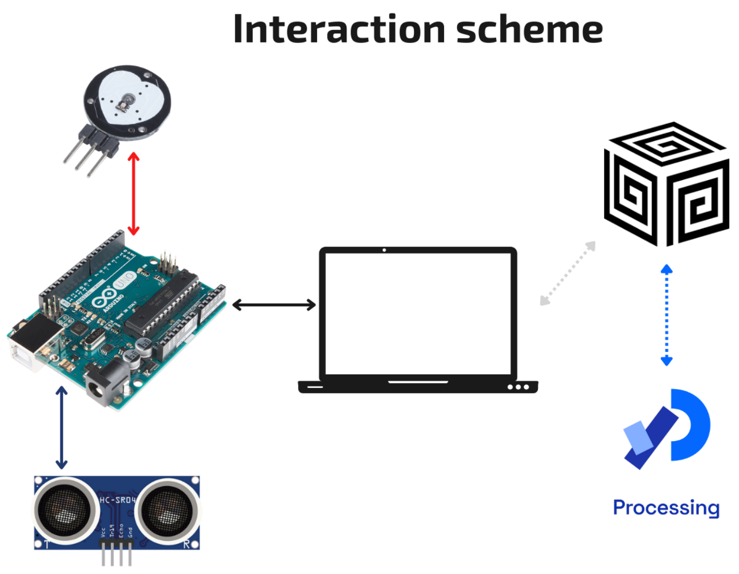

Figure 1: BEATRUN interaction scheme

Beatrun has been created with the joint force of several elements, connected as you can see in Figure 1.

Basically, it all starts with two (fundamental) sensors, the Heart-rate sensor and the Ultrasonic Sensor HC-SR04 connected to an Arduino UNO board. Through the Arduino IDE we are able to acquire the signals from the two sensors, which are sent to a computer via serial port, and decoded by Supercollider.
Supercollider plays a central role in the whole scheme, since it is responsible for the sonification of the signals coming from the Arduino and its sensors. By means of its routine, sound is triggered everytime our heart beats, i.e. following the heartbeat, while the pitch changes with the change of distance between the ultrasonic sensor and the user's hand.
Supercollider then sends Osc messages to Processing, which captures them and transforms it all into the BEATRUN visualization.

2.1 Arduino

Material used:

- Heart-rate sensor: through the library PulseSensorPlayground.h we used the pulsesensor "getBeatsPerMinute()" method that needs time to give a stable estimation of BPM, so initially the sounds may not be triggered in the proper way. It is connected to the analog pin 0 and constantly awaits for a heartbeat signal, which will eventually come after the user has been touching the sensor with their fingertip (or ear lobe).
- Ultrasonic sensor: the distance sensor is even easier to set up thanks to the NewPing.h library, and it simply detects the effective distance between the sensor itself and the hand of the user.

Hardware configuration:

- PulseSensor **-** connected to Arduino **GND**
- PulseSensor + connected to Arduino **5V**
- PulseSensor **S** connected to Arduino **A0**
- HC-SR04 **Vcc** connected to Arduino **5V**
- HC-SR04 **Trig** connected to Arduino **12**
- HC-SR04 **Echo** connected to Arduino **11**
- HC-SR04 **GND** connected to Arduino **GND**

From the two sensors we get synchronously integer values, one for the heart-rate "p" and one for the distance "d", which are eventually sent to SuperCollider. The sequence of serial messages contains both the distance and the heartbeat values, in no particular order. The parsing is done by decoding the ASCII labels that accompany each message.

2.2 SuperCollider

In Supercollider we created a main SynthDef, in charge of the rendering of the sound we had in mind. It is characterized by a rich sound which easily reminds us of the typical 80's sound: it's resonant, enriched by a small amount of noise (PinkNoise), modulated by a LP Pulse, filtered through a chain of two BP filters and a HP one and then passed to the output, after a "pan" operation. We have a set of notes whose "freq" parameter can be driven by the data flowing from the distance sensor: the result is a variation of the pitch perceived.
The set of notes we can interact with is managed by three sub routines, each one of them triggered every $N * 60/BPM$ seconds, with N = number of notes to be triggered and "BPM" the signal from our heart.
We can achieve a polymetric effect in a very simple way: the different rate (depending on the BPM parameter, "value2" in the code) at which we trigger different number of notes, creates the effect by itself .
This mechanism is similar to the concept of the different velocities at which planets run over their orbits: despite this, there are always some points in which their position is aligned, in the same instant. This underlying concept drove us in the creation of the visual part of this work in Processing.
We have also developed another synth, with a dry sound, to add lower frequencies to the whole thing.
The code is organized as follow:

- **SERIAL ROUTINE**: a routine called "getValues" receives serial data from the selected port and parses the ascii values. If a "d" char terminator is encountered the routine stores the raw distance measure in the "val1" variable and masp it into a discretized midi "note". If a "p" char terminator is encountered the routine stores the raw BPM value in the "val2" variable.
- **SYNTH DEFINITIONS**:

  Name: wetSynth

  Main features: output signal is a hpf version of the sum of a signal modulated by LPF Pulse, with some PinkNoise added and a peculiar Envelope defined by the classic adsr parameters.

```
SynthDef(\wetSynth, {
arg freq, amp=0.5, t_gate = 1, dur = 1.5;
var sig, sig2, hpf_sig, out;

sig = Mix.ar(LFPulse.ar(freq * [0.99, 1, 1.01]));
sig2 = PinkNoise.ar(mul: 1.5, add: 0.0);
hpf_sig = HPF.ar(sig+sig2, 200);

out = hpf_sig + BPF.ar(sig+sig2, 500, 5);

a = EnvGen.kr(Env.asr(0.01,1,dur), gate:t_gate);
Out.ar(0, Pan2.ar(out*a*amp, 0, 0.5));
})
```

  Name: drySynth

  Main Features:

  A SinOsc, with frequencies ranging random in a given range, modulated by Env and mapped through a linear mapping with Phasor and LinLin. The final result is spread over the stereo field by using the function Splay.

```
SynthDef(\drySynth, {
arg freq=220, amp=1, t_gate = 1;
var freqBase=freq;
var freqRes=SinOsc.kr(Rand(0,0.2),0).range(freqBase/2,freqBase*2);
var pdbase=Impulse.ar(freqBase);
var pd=Phasor.ar(pdbase,2*pi*freqBase/s.sampleRate,0,2pi);
var pdres=Phasor.ar(pdbase,2*pi*freqRes/s.sampleRate,0,2pi);
var pdi=LinLin.ar((2pi-pd).max(0),0,2pi,0,1);
var snd=Lag.ar(SinOsc.ar(0,pdres)*pdi,1/freqBase).dup;
snd=Splay.ar(snd);
snd=snd*EnvGen.ar(Env.perc(0.005,1.5), gate:t_gate);
Out.ar(0,snd*amp);
})
```

- **MAIN ROUTINES**: Four different asynchronous routines are implemented:
  - controlDist: sends raw distance values (stored in val1) at a fixed rate
  - controlBpm: this simple routine sends a trigger message ("trigger0") to Processing every time a heart pulse happens
  - control1: plays a sequence of notes (synth 1) and sends a trigger message (trigger1) every period calculated with respect to the bpm value (val2)
  - control2: plays a sequence of notes (synth 2) and sends a trigger message (trigger2) every period calculated with respect to the bpm value (val2)

2.3 Processing

The animation in Processing is consistent with the sound style and theme.

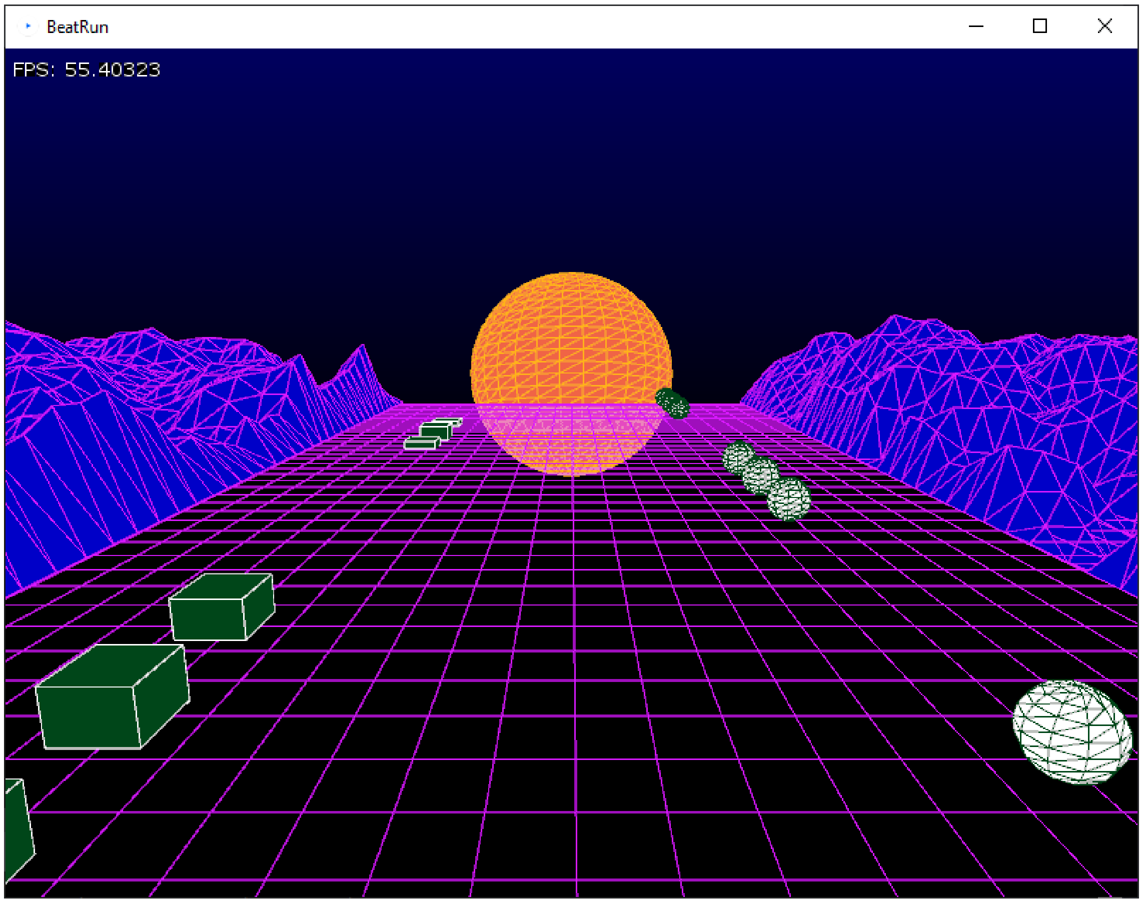

Figure 2: Processing visuals

We can identify three different groups of animations:

- Independent animations. They concern the central grid and the side mountains. They have a fixed speed and do not provide any user interaction, they are only functional to the theme.

  The grid gives the illusion of movement thanks to an intelligent use of its translation in space. The side mountains are created through a grid too, to which a Perlin noise was generated procedurally.

- Sun. The central sun is a sphere that pulses every time a heart beat happens (the trigger is the variable "p", assuming 0 and 1 as values). The animation lasts for just one frame, but it is quite adequate in reference to persistence of vision: a red halo appears clearly visible around the object.

- Animation of the polyrhythm. This is the heart of the animation. On the left appear some Cube objects triggered by the first synth, with height that varies according to the pitch provided. On the right instead appear some Ball objects, triggered by the second synth.

File structure: the main file is called "BeatRun" and needs two other files, called "Ball" and "Cube". These files contain the homonyms classes definitions, useful to instantiate objects of the same type but having different attributes.

3 Conclusions and future work

The aim of the project was that of creating something exploiting the potential of Arduino, Supercollider and Processing. We immediately came up with the idea of using an heart-rate sensor and then after a few sessions of brainstorming and after hearing the sound of the SynthDef prototype we defined in SuperCollider, we decide to bring it all to the 80's and eventually created BEATRUN.
The original idea was to have two performers, and each of their heart-beats would combine into a polymeter. However, the distance between us forced us to give up on this part of the idea, which actually fits quite well thematically.
We are very happy with the achieved result, in particular with the possibility of connecting ourselves (literally) with the application and to become the central engine of it.
Finally, we hope that using it you will experiment the same feeling of really belonging with BEATRUN.
A further upgrade can affect the number of sensors, in fact, given the large number of very affordable sensors we can indeed be able transform almost every action or situation into a signal that can be sent to Arduino and Supercollider and become part of the world of BEATRUN.