Daniel Gea Palenzuela
up202204419

The Metamorphosis

Audio-reactive performance based on the book “The Metamorphosis” by Kafka.

This project is based on Processing + Pure data + Touchdesigner.

It will consist of 4 different soundscapes made with Pure Data, controlled by Processing and live visuals reacting to the audio made on Touchdesigner.

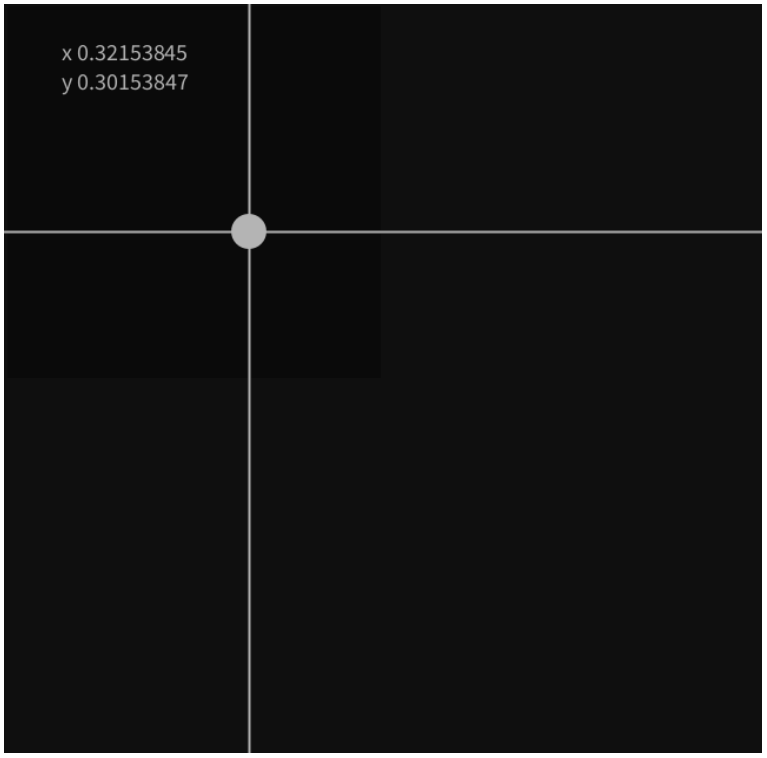
Each soundscape will represent a phase of the metamorphosis, therefore is important that we can progressively switch from one to another without breaking the dynamic of the performance. I will explain everything in the following points.

1. PROCESSING

2D Slider

For the project, one of the main challenges I had was to control as many values as possible with the least amount of controllers (sliders, buttons, etc). Therefore, I explored the idea of using a 2D slider as a master controller for the whole performance.

I decided to use processing to design the 2D slider. I used basic shapes and tracked the mouse position to get the values needed to start working with pure data. I will not go deep into how I developed the slider, I will only mention the main strategies were to define conditions for the limits of every chapter and some other mathematical operations to map the values exactly how I required.

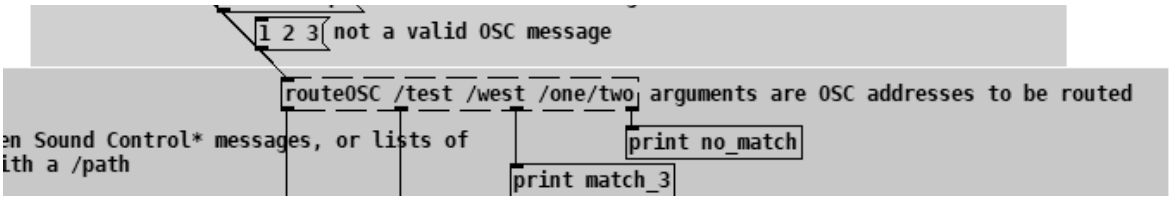


2D Slider made with Processing

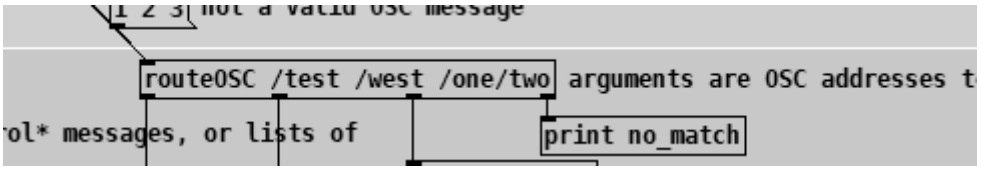
OSC - Processing to Pure Data

Some of the libraries I was looking at were oscP5, controlP5, mrPeach and OSC library (found in the browser of PD). For processing, the first 2 libraries were important to start sending values with OSC.

The first OSC syntax that I tried, using mrPeach and routeOSC was working, but not all the time. I had to open and close repetitively until Pd properly read the objects, so in the end, I decided to keep looking for other alternatives. The problem was that to start sending information from processing to PD, I experienced that first, I had to open Pd and check (in help/browser -> external/osc/routeOSC) if the object "routeOSC" was correctly read and created. Then, open the PD project and after that, run the code from processing.

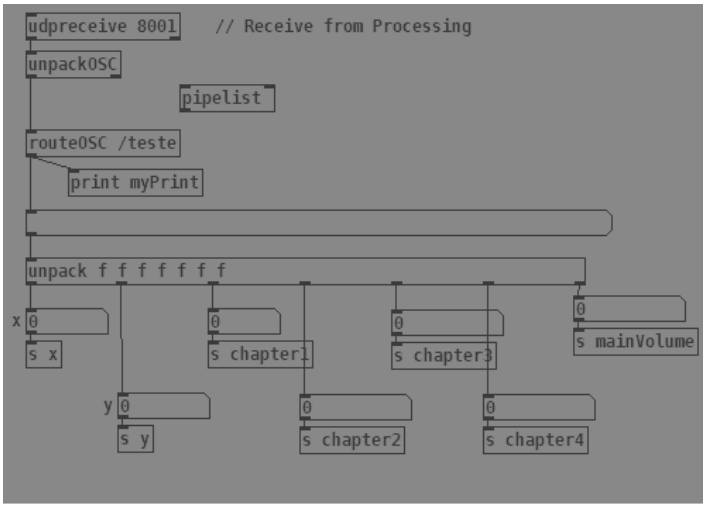


Object routeOSC not created above (NOT USED)



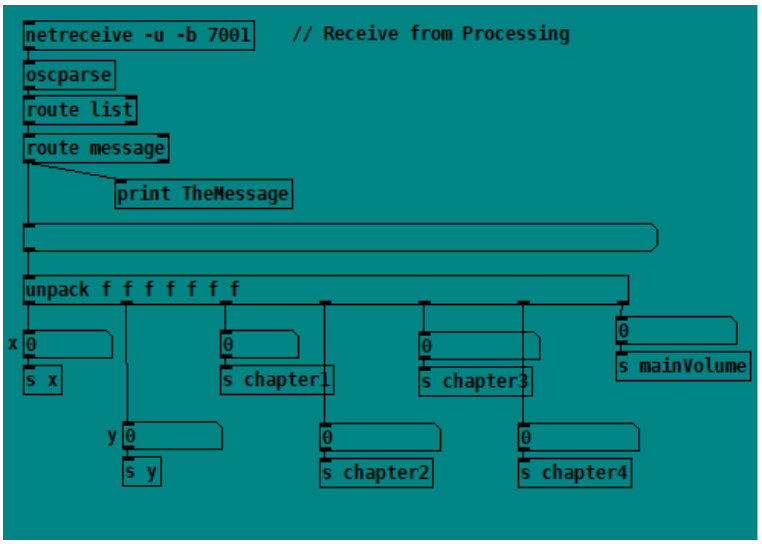
Object routeOSC readed and created in this second image (NOT USED)

And then, I would be reading the message sent by processing.



PD, receiving data from processing and splitting it (NOT USED)

However, this method wasn't working at all and later on, I found this other syntax using *netreceive*, which solved every problem that I had receiving OSC messages.



Netreceive for reading OSC messages, final implementation

2. PURE DATA

Sounds and Chapters

The performance is divided into 4 chapters, each of which represents a phase of the metamorphosis. The first chapter represents the awakening, the interruption of the silence and the sleep. The second chapter represents the negation, the lostness and the beginning of the transformation. The third chapter represents the metamorphosis, where there is no point of return. The 4th chapter represents the acceptance and the new reality, the back to calm with new aspects of the transformation included in this new reality.

Since I have 4 chapters to cover, I decided to recreate 4 soundscapes with their own feeling and rhythm to easily differentiate one from the other. The main challenge was to make them belong to the same project, in other words, make them differentiable but still have the feeling of belonging to the project, as one entire piece.

To achieve this, I based all the performance around **4 main notes** that I play during all the chapters. Sometimes with a different rhythm, reverberation or oscillator but the notes are there providing continuity. In specific these 4 notes are introduced in the first chapter in this exact sequence.

Four main notes of the melodies

I also tried to add some other notes depending on the scale I was using to recreate a feeling of dissonance or tension that later would be solved with one of those main notes, more exactly in chapters 2 and 4.

These 4 notes may be found with that exact message value or various scales up or down.
(Example: 65 - 24 = 41; 55 - 12 = 43; 74 - 36 = 38; ...).

Furthermore, not only based on these notes I achieved this “continuity”, it was also a subjective work of listening, trial and error to make everything fit in one entire piece.

The main strategy I use is **additive synthesis**, which can be found in the whole project. I used it to combine frequencies in a logical order, based on chords with changes in the major mode, minor mode, augmented or diminished.

I also used **frequency modulation** to give some aesthetics of dissonance and tension and the use of different waveforms to represent the transformation (metamorphosis) going from normal rounded *osc~* to more dissonant *square~* waves.

Moreover, on the 3rd piece, I used **grain synthesis** to imitate the sound of an insect. This specific chapter is the moment where the transformation takes place. By using grain I could have some grain sounds at the beginning of the chapter, and as the volume increases and the closer the peak of the chapter is, the better we listen to the actual audio that is being played with the grain synth technique.

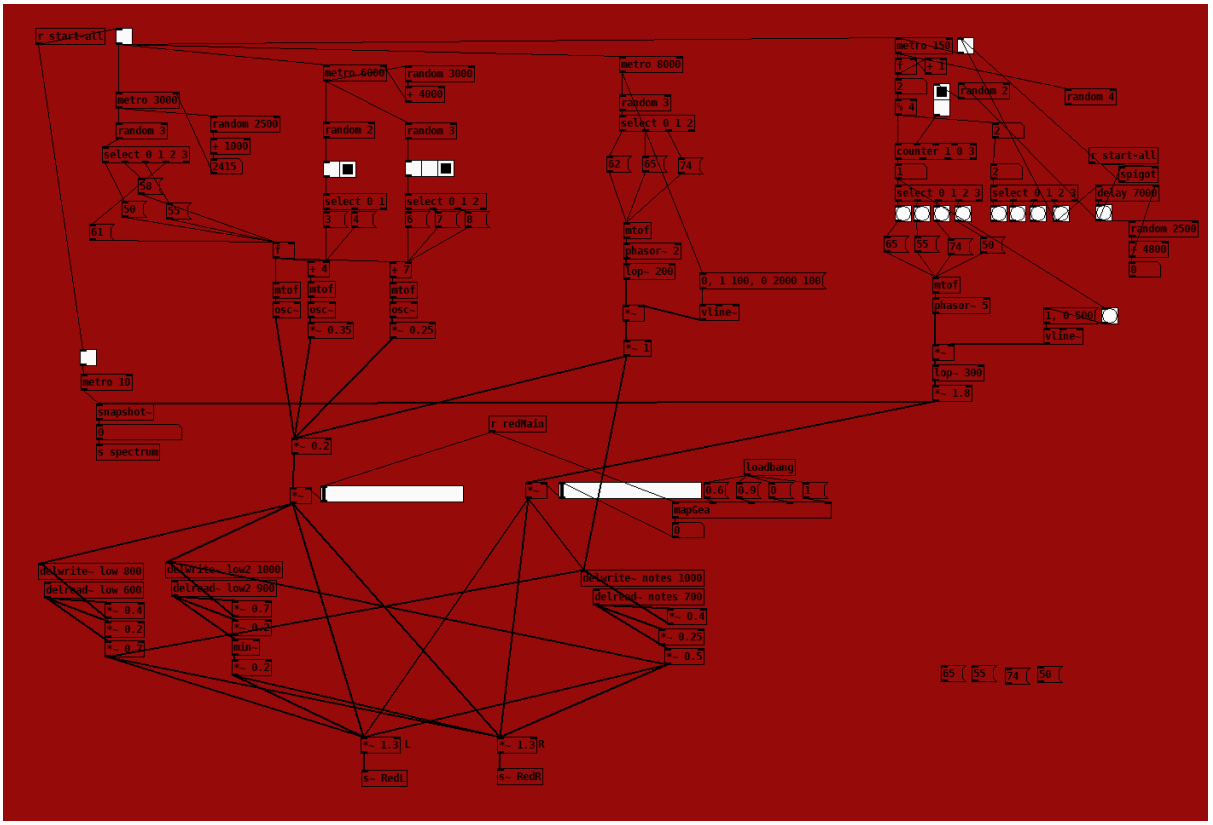
To make it more playful, I used two audio files for grain, one of them would play when the slider is static. The other one will play only when the mouse is moving, and as I move the slider I change the value of the grain duration.

A few other techniques that I used were **pannings**, **delays** (for creating feedbacks), **envelopes** and **random timing** to change metronomes.

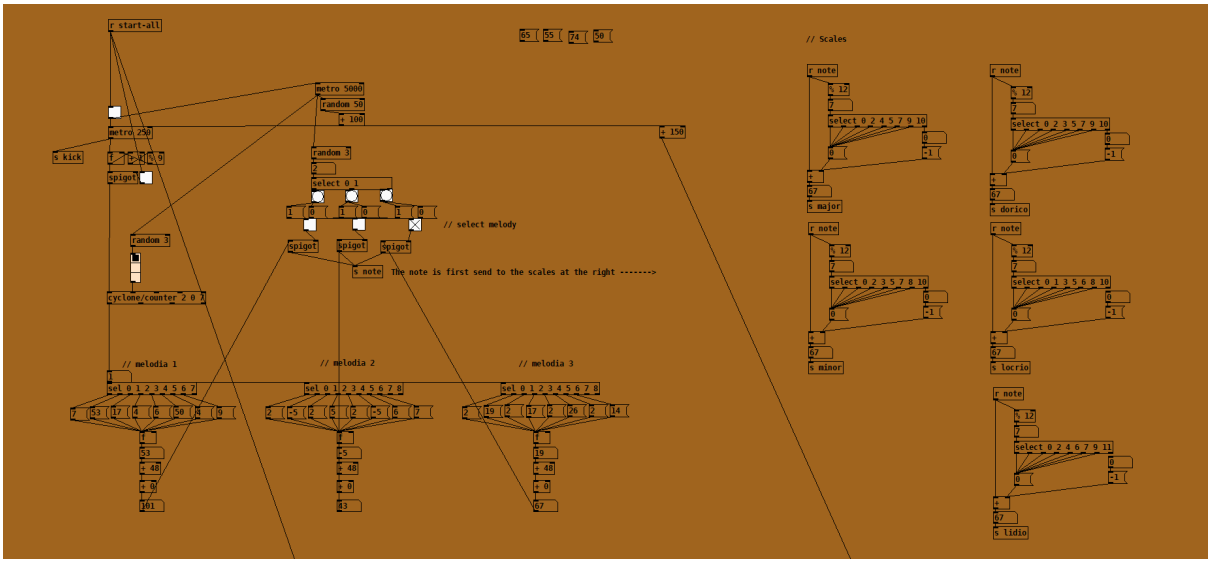
I will show the patch structure of every chapter to give an overview of how it was made:

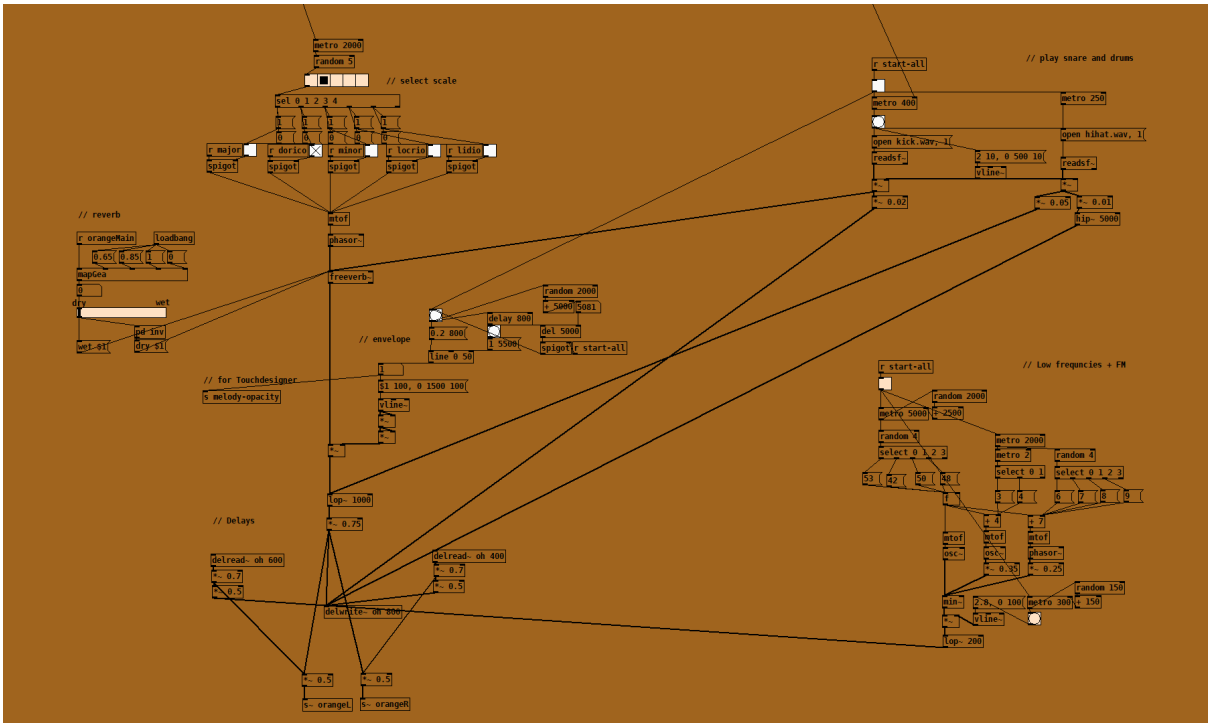
Daniel Gea Palenzuela
up202204419

Chapter1

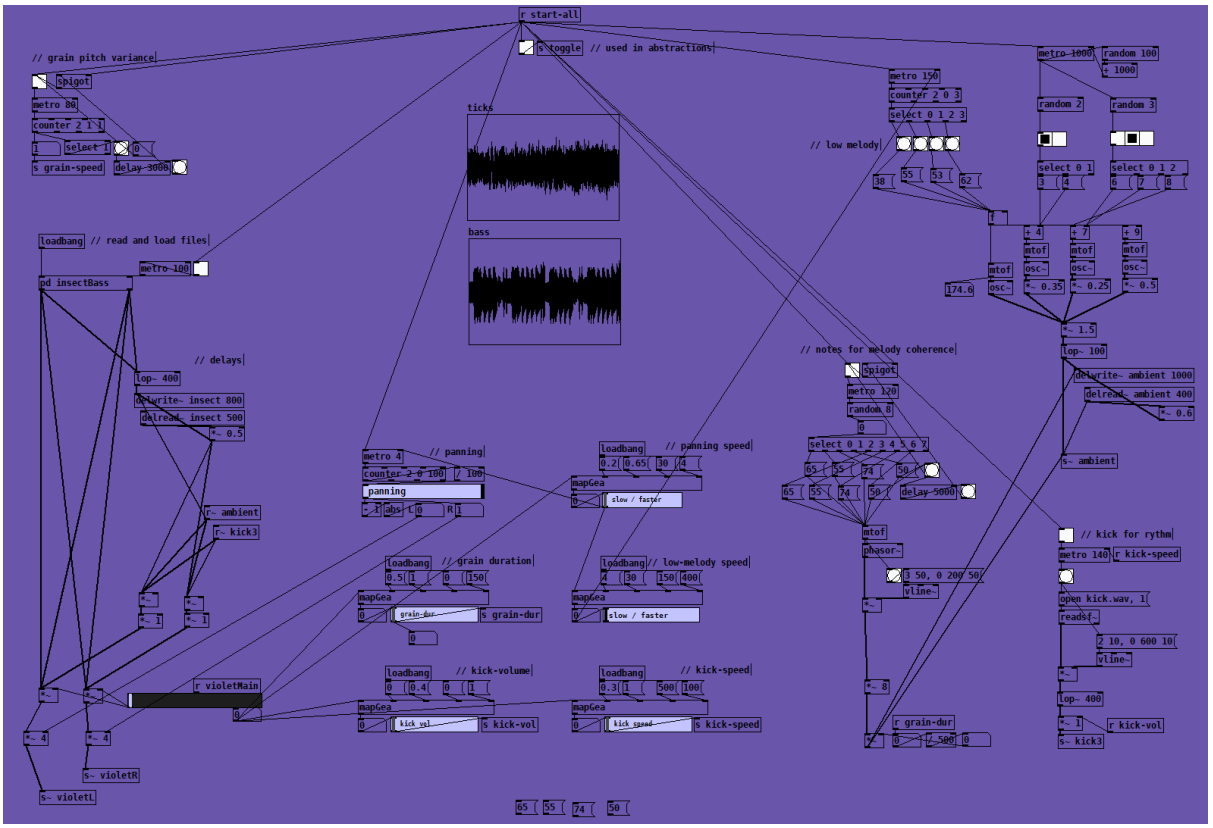


Chapter 2

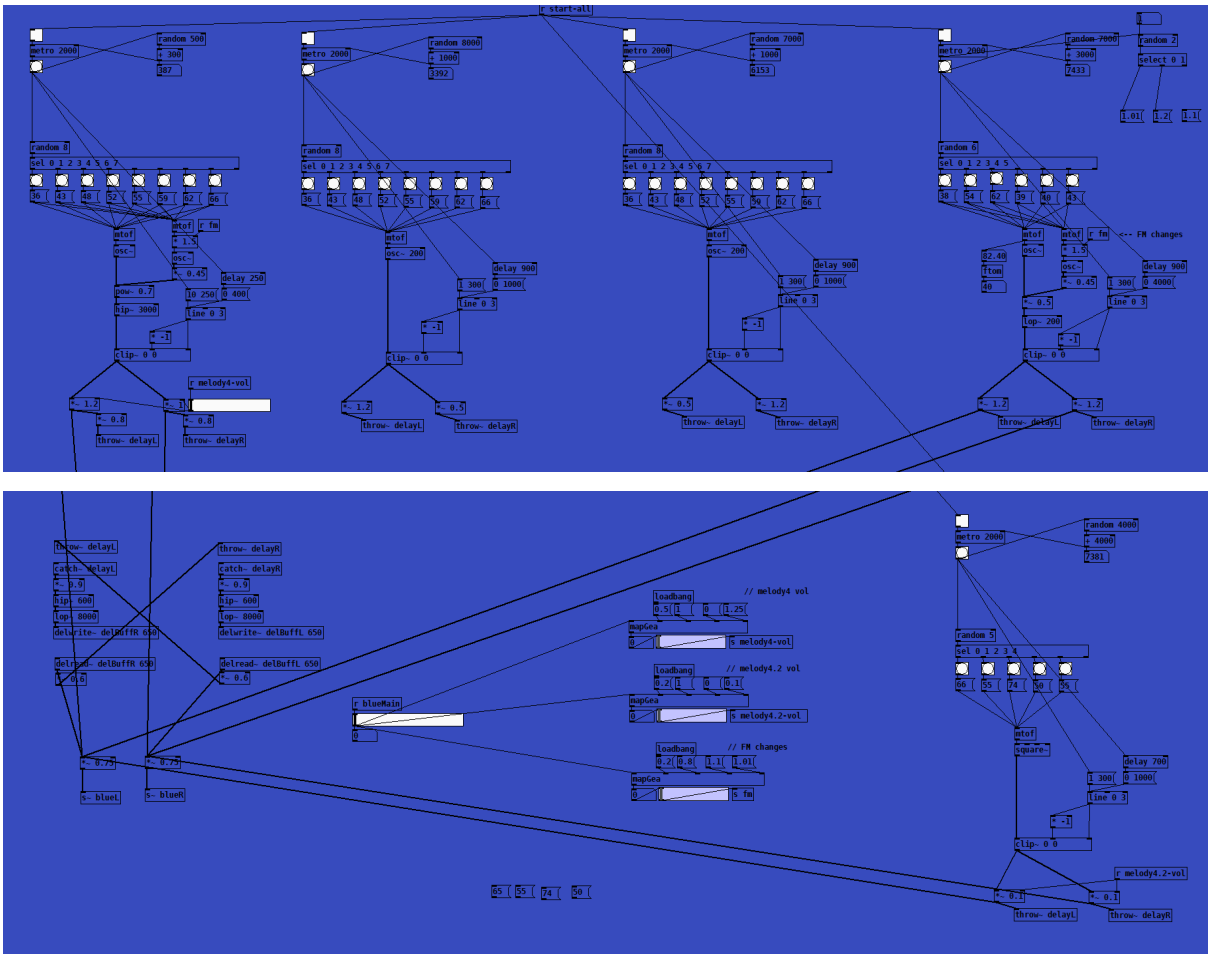

r start-all
65 55 74 50
// Scales
metro 5000
random 50
+ 100
metro 250
+ 150
s kick
random 3
select 0 1
// select melody
spigot
s note
The note is first send to the scales at the right ------->
cyclone/counter 2 0 7
// melodia 1
// melodia 2
// melodia 3
r note
select 0 2 4 5 7 9 10
s major
select 0 2 3 5 7 9 10
s dorico
select 0 2 3 5 7 8 10
s minor
select 0 1 3 5 6 8 10
s locrio
select 0 2 4 6 7 9 11
s lidio


metro 2000
random 5
// select scale
r major
r dorico
r minor
r locrio
r lidio
// reverb
r orangeMain
loadbang
mapGea
dry
wet
wet-$1
inv
dry-$1
mtof
phasor~
freeverb~
// envelope
random 2000
delay 800
del 5000
line~ 0 50
// for Touchdesigner
s melody-opacity
vline~
lop~ 1000
// Delays
delread~ oh 600
delread~ oh 400
delwrite~ oh 800
s~ orangeL
s~ orangeR
r stakt-all
// play snare and drums
metro 400
metro 250
open kick.wav, 1
open hihat.wav, 1
readsf~
hip~ 5000
// Low frequncies + FM
osc~
min~
metro 300
random 150

Chapter 3


r start-all
s toggle
// used in abstractions
// grain pitch variance
spigot
metro 80
counter 2 1 1
select 1
s grain-speed
delay 3000
ticks
bass
loadbang
// read and load files
metro 100
pd insectBass
// delays
lop~ 400
delwrite~ insect 800
delread~ insect 500
*~ 0.5
r~ ambient
r~ kick3
r violetMain
s~ violetR
s~ violetL
metro 4
// panning
counter 2 0 100
/ 100
panning
loadbang
// panning speed
0.2 0.65 30 4
mapGea
slow / faster
// grain duration
0.5 1 0 150
grain-dur
s grain-dur
// low-melody speed
4 30 150 400
// kick-volume
0 0.4 0 1
kick vol
s kick-vol
// kick-speed
0.3 1 500 100
kick speed
s kick-speed
metro 150
counter 2 0 3
select 0 1 2 3
// low melody
38 55 53 62
// notes for melody coherence
metro 120
random 8
select 0 1 2 3 4 5 6 7
65 55 74 50
delay 5000
mtof
phasor~
3 50, 0 200 50
vline~
r grain-dur
/ 500
s~ kick3
metro 1000
random 100
+ 1000
random 2
random 3
select 0 1
select 0 1 2
+ 4
+ 7
+ 9
174.0
*~ 0.35
*~ 0.25
*~ 0.5
*~ 1.5
lop~ 100
delwrite~ ambient 1000
delread~ ambient 400
*~ 0.6
s~ ambient
// kick for rythm
metro 140
r kick-speed
open kick.wav, 1
readsf~
2 10, 0 600 10
vline~
lop~ 400
r kick-vol
s~ kick3

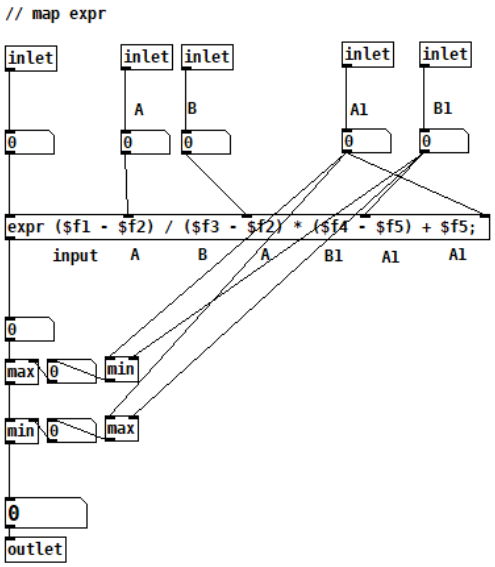
Chapter 4


r start-all
metro 2000
random 500
+ 300
387
random 8
sel 0 1 2 3 4 5 6 7
36 43 48 52 55 59 62 66
mtof
r fm
osc~
+ 1.5
*~ 0.45
delay 250
pow~ 0.7
lo 250
0 400
hip~ 3000
line 0 3
* -1
clip~ 0 0
r melody4-vol
*~ 1.2
*~ 0.8
throw~ delayL
throw~ delayR
random 8000
+ 1000
3392
osc~ 200
delay 900
l 300
0 1000
*~ 0.5
random 7000
+ 1000
6153
random 6
sel 0 1 2 3 4 5
38 54 62 39 40 43
<--- FM changes
*~ 0.4
random 7000
+ 3000
7433
select 0 1
1.01 1.2 1.1
0 4000
lop~ 200
throw~ delayL
throw~ delayR
catch~ delayL
catch~ delayR
*~ 0.9
hip~ 600
lop~ 8000
delwrite~ delBuffR 650
delwrite~ delBuffL 650
delread~ delBuffR 650
delread~ delBuffL 650
*~ 0.6
*~ 0.75
s~ bluel
s~ blueR
r blueMain
loadbang
0.5 1 0 1.25
// melody4 vol
mapGea
s melody4-vol
0.2 1 0 0.1
// melody4.2 vol
s melody4.2-vol
0.2 0.8 1.1 1.01
// FM changes
s fm
random 4000
+ 4000
7381
random 5
sel 0 1 2 3 4
66 55 74 50 55
mtof
square~
delay 700
line 0 3
clip~ 0 0
r melody4.2-vol
*~ 0.1

Mapping values

The whole project is mainly based on the strategy of mapping values, from processing to PD, internally in PD for sounds and other features and from PD to Touchdesigner.

Since I did not find an object to easily map values in PD, I created a patch called “mapGea” which takes 5 parameters.

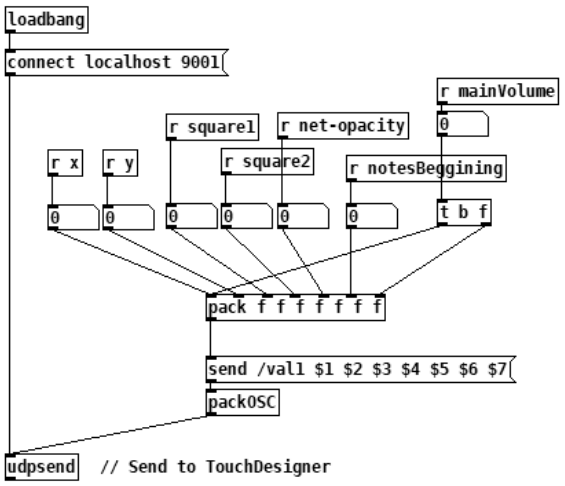


Map expression created in a separate patch

Since for me it was important to never go over the maximum or the minimum, I created a little check for those max and min to always keep the value in between those inputs.
This patch is widely used in the whole performance to make smooth changes and activate or deactivate features.

3. TOUCHDESIGNER

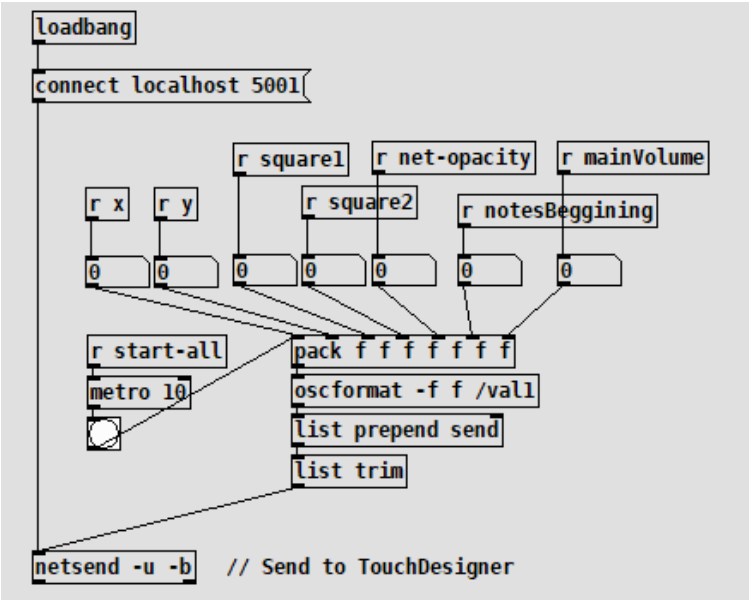
Sending values from PD to Touchdesigner. Again, I used OSC messages to send values to Touchdesigner.



Send a message to Touchdesigner via OSC (NOT USED)

Nevertheless, even though this was working from the very beginning without errors, on the 1st of January, it stopped sending messages. I still don't know the reasons, I simply know that I had to use a different syntax and library to make this work again.

For this reason, I found another code which "defines" better the message to be sent, this is the format and the prepend and finally uses *netsend*.



Sending OSC messages with netsend, final implementation

Once I was receiving values from PD in TD the challenge was to create visuals that could portray the essence of the sound in visual terms. I will not elaborate much on this since it is not the scope of this course but I based my design decisions on features like **textures**, **repetitive patterns** and **feedback**, which I think is the basic structure of the sounds in PD.

Also, these elements mentioned above represent characteristics of insects, like the net from a spider with the second chapter. The fast beating of the heart of an insect from chapter 3 represented with fast square visuals that go on and off. Or the chapter 4 with its visuals styled like multiple mirrors a bit blurry simulating the vision of insects like flies.

To make the visuals react to the sounds, I referenced the values received via OSC to parameters in Touchdesigner nodes, using sliders and math chops.

For one specific filter (chapter 3, the transformation), I used this already-made node found on the internet.
https://github.com/FollowTheDarkside/td-quadtree-filter
However, it was needed to combine it with other nodes to make it work properly. I reached the final result by experimentation, as in every visual design for all the chapters.