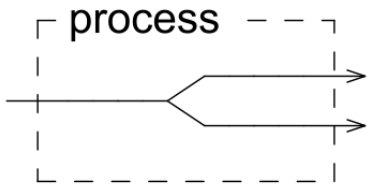
UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
CIEREC - EA 3068

Synthèse avec MaxMSP et FAUST

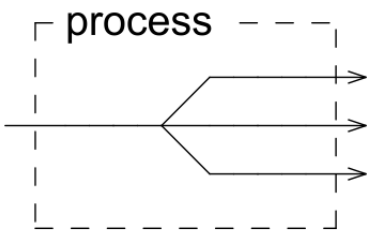
Séance de TD n°1

Ecrivez le code Faust correspondant aux diagrammes indiqués.
Testez le programme dans MaxMSP

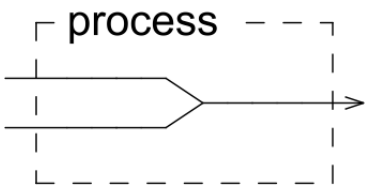
Exercice 1



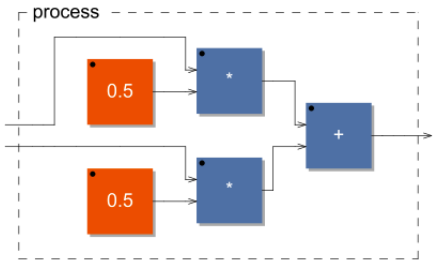
Exercice 2



Exercice 3



Exercice 4



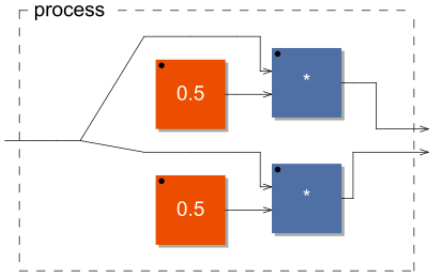
Exercice 5

Faire un panoramique avec une variable.
Le paramètre « panoramic » est introduit dans le code par la commande hslider :

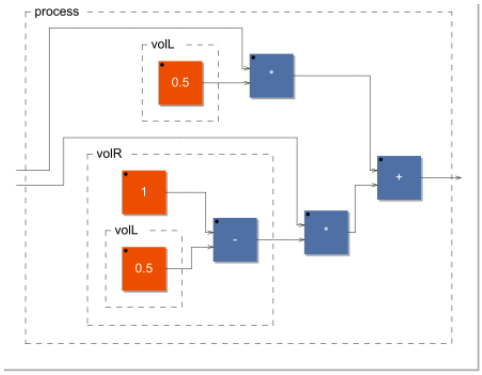
```
pan = hslider("panoramic",0.5,0.0,1.0,0.01);
```

Cela crée dans Faust une variable « pan » qui a pour valeur par défaut 0.5, comme valeur minimum 0, valeur maximum 1 et qui peut varier par pas de 0.01.

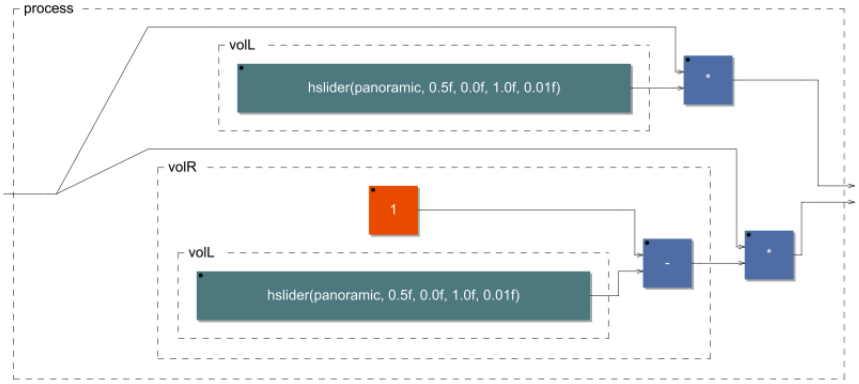
a) Sans slider, sans variables



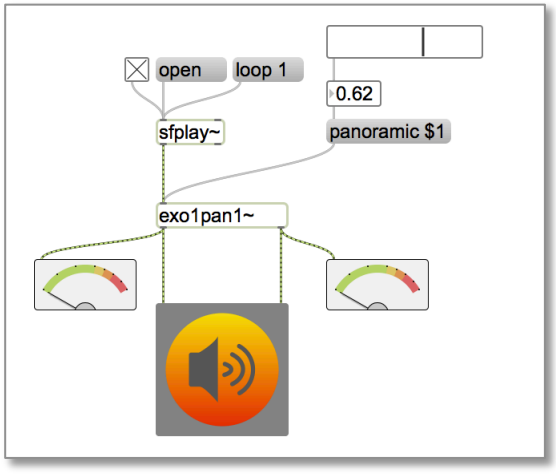
b) Sans slider, avec variables



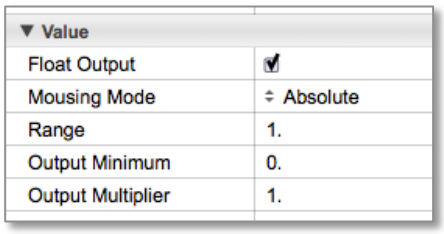
c) Avec slider et variables



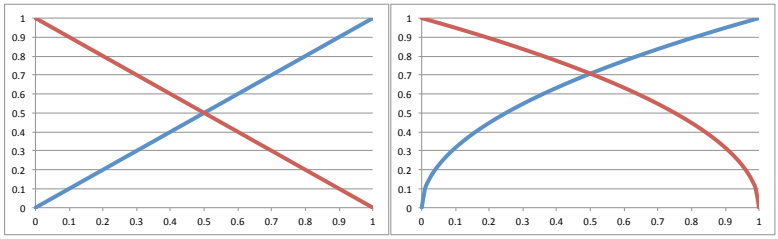
Compilez l'objet pour MaxMSP et insérez le dans un patch comme indiqué dans la figure ci-dessous :



Remarque : pour le curseur (*slider*) dans Max/MSP, aller dans l'inspector (Menu Object), demander « Float Ouput » et étendue (Range) = 1.



d) Pour faire un panoramique, il ne faut pas utiliser une fonction linéaire (une ligne droite) pour monter le volume dans un haut-parleur et le baisser dans un autre, mais une racine carrée.



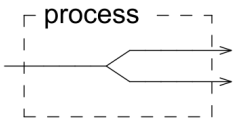
Quand la valeur du panoramique varie de pan=0 (son à gauche) à pan=1 (son à droite), on multiplie respectivement le volume du son dans les haut parleurs gauche et droite par $\sqrt{p}$ *et* $\sqrt{1-p}$.

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
CIEREC - EA 3068

Solutions au TD1

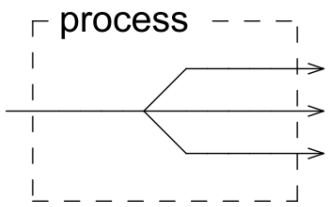
Exercice 1

```
process = _<:_,_;
```



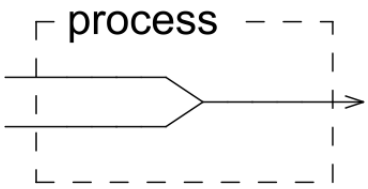
Exercice 2

```
process = _<:_,_,_;
```



Exercice 3

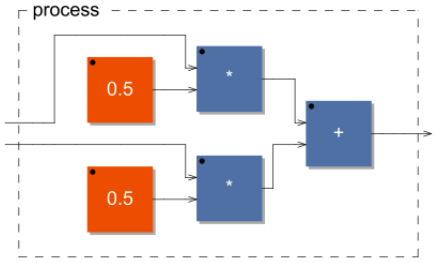
```
// process = _,_:+;
process = +;
```



Exercice 4

Solution 1

```
// mixage
mix = _ * 0.5 + _ * 0.5;
process = mix;
```



remarques :

```
mix = _ * 0.5 + _ * 0.5;
```

peut aussi s'écrire

```
mix = _ * 0.5 , _ * 0.5 : +;
```

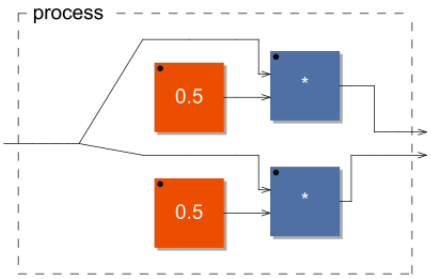
ou

```
mix = * (0.5) , * (0.5) : +;
```

Exercice 5

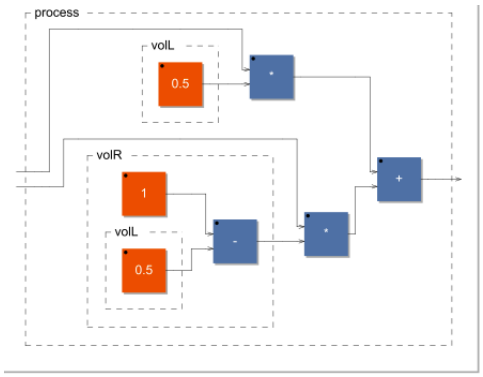
a)

```
pan = _ <: * (0.5) , * (0.5) ;
process = pan;
```

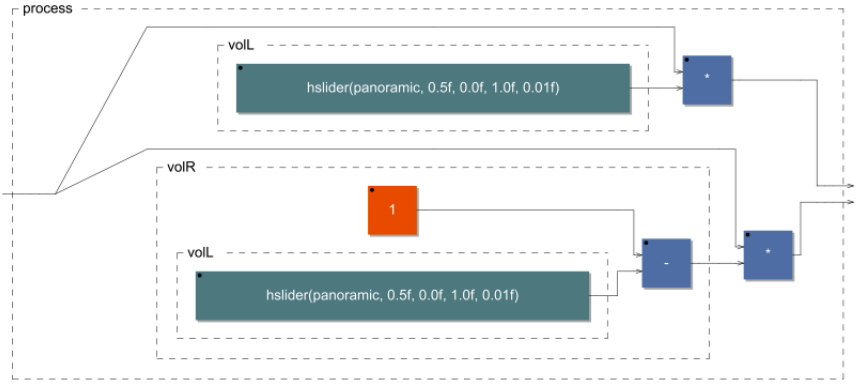


b)

```
// pan1
//pan = hslider("panoramic",0.5,0.0,1.0,0.01);
p = 0.5;
volL = p;
volR = 1-p;
pan = _ <: * (volL) , * (volR) ;
process = pan;
```



c)



```
// pan1 linéaire
p = hslider("panoramic",0.5,0.0,1.0,0.01);
volR = p;
volL = 1-p;
pan = _ <: * (volL) , * (volR) ;
process = pan;

// pan1 racine
p = hslider("panoramic",0.5,0.0,1.0,0.01);
volR = p:sqrt;
volL = 1-p:sqrt;
pan = _ <: * (volL) , * (volR) ;
process = pan;
```

ou

```
process = <: *(1 - c : sqrt), *(c : sqrt);
```

option : utilisation de la fonction *smooth* pour ne pas avoir de clicks quand on change une valeur

```
import("filter.lib") ;
// pan1
p = hslider("panoramic",0.5,0.0,1.0,0.01):smooth(0.999);
volR = p:sqrt;
volL = 1-p:sqrt;
pan = _ <: * (volL) , * (volR) ;
process = pan;
```

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
CIEREC - EA 3068

Le langage FAUST

Les opérateurs de FAUST

Syntax	Pri.	Assoc.	Description
expression ~ *expression*	4	left	recursive composition
expression , *expression*	3	right	parallel composition
expression : *expression*	2	right	sequential composition
expression <: *expression*	1	right	split composition
expression :> *expression*	1	right	merge composition

Les opérateurs algébriques

Syntax	Type	Description
n	$\mathbb{S}^0 \rightarrow \mathbb{S}^1$	integer number: $y(t) = n$
$n.m$	$\mathbb{S}^0 \rightarrow \mathbb{S}^1$	floating point number: $y(t) = n.m$
_	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	identity function: $y(t) = x(t)$
!	$\mathbb{S}^1 \rightarrow \mathbb{S}^0$	cut function: $\forall x \in \mathbb{S}, (x) \rightarrow ()$
int	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	cast into an int signal: $y(t) = (int)x(t)$
float	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	cast into an float signal: $y(t) = (float)x(t)$
+	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	addition: $y(t) = x_1(t) + x_2(t)$
-	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	substraction: $y(t) = x_1(t) - x_2(t)$
*	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	multiplication: $y(t) = x_1(t) * x_2(t)$
/	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	division: $y(t) = x_1(t)/x_2(t)$
%	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	modulo: $y(t) = x_1(t)\%x_2(t)$
&	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical AND: $y(t) = x_1(t)\&x_2(t)$
|	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical OR: $y(t) = x_1(t)\|x_2(t)$
^	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical XOR: $y(t) = x_1(t) \wedge x_2(t)$
<<	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	arith. shift left: $y(t) = x_1(t) << x_2(t)$
>>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	arith. shift right: $y(t) = x_1(t) >> x_2(t)$
<	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	less than: $y(t) = x_1(t) < x_2(t)$
<=	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	less or equal: $y(t) = x_1(t) <= x_2(t)$
>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	greater than: $y(t) = x_1(t) > x_2(t)$
>=	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	greater or equal: $y(t) = x_1(t) >= x_2(t)$
==	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	equal: $y(t) = x_1(t) == x_2(t)$
!=	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	different: $y(t) = x_1(t)! = x_2(t)$

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
CIEREC - EA 3068

Les fonctions mathématiques

Syntax	Type	Description
acos	$\mathbb{S}^1 \to \mathbb{S}^1$	arc cosine: $y(t) = \text{acosf}(x(t))$
asin	$\mathbb{S}^1 \to \mathbb{S}^1$	arc sine: $y(t) = \text{asinf}(x(t))$
atan	$\mathbb{S}^1 \to \mathbb{S}^1$	arc tangent: $y(t) = \text{atanf}(x(t))$
atan2	$\mathbb{S}^2 \to \mathbb{S}^1$	arc tangent of 2 signals: $y(t) = \text{atan2f}(x_1(t), x_2(t))$
cos	$\mathbb{S}^1 \to \mathbb{S}^1$	cosine: $y(t) = \text{cosf}(x(t))$
sin	$\mathbb{S}^1 \to \mathbb{S}^1$	sine: $y(t) = \text{sinf}(x(t))$
tan	$\mathbb{S}^1 \to \mathbb{S}^1$	tangent: $y(t) = \text{tanf}(x(t))$
exp	$\mathbb{S}^1 \to \mathbb{S}^1$	base-e exponential: $y(t) = \text{expf}(x(t))$
log	$\mathbb{S}^1 \to \mathbb{S}^1$	base-e logarithm: $y(t) = \text{logf}(x(t))$
log10	$\mathbb{S}^1 \to \mathbb{S}^1$	base-10 logarithm: $y(t) = \text{log10f}(x(t))$
pow	$\mathbb{S}^2 \to \mathbb{S}^1$	power: $y(t) = \text{powf}(x_1(t), x_2(t))$
sqrt	$\mathbb{S}^1 \to \mathbb{S}^1$	square root: $y(t) = \text{sqrtf}(x(t))$
abs	$\mathbb{S}^1 \to \mathbb{S}^1$	absolute value (int): $y(t) = \text{abs}(x(t))$
		absolute value (float): $y(t) = \text{fabsf}(x(t))$
min	$\mathbb{S}^2 \to \mathbb{S}^1$	minimum: $y(t) = \min(x_1(t), x_2(t))$
max	$\mathbb{S}^2 \to \mathbb{S}^1$	maximum: $y(t) = \max(x_1(t), x_2(t))$
fmod	$\mathbb{S}^2 \to \mathbb{S}^1$	float modulo: $y(t) = \text{fmodf}(x_1(t), x_2(t))$
remainder	$\mathbb{S}^2 \to \mathbb{S}^1$	float remainder: $y(t) = \text{remainderf}(x_1(t), x_2(t))$
floor	$\mathbb{S}^1 \to \mathbb{S}^1$	largest int $\leq$: $y(t) = \text{floorf}(x(t))$
ceil	$\mathbb{S}^1 \to \mathbb{S}^1$	smallest int $\geq$: $y(t) = \text{ceilf}(x(t))$
rint	$\mathbb{S}^1 \to \mathbb{S}^1$	closest int: $y(t) = \text{rintf}(x(t))$

Les fonctions pour l'interface

Syntax	Example
button(*str*)	button("play")
checkbox(*str*)	checkbox("mute")
vslider(*str*,*cur*,*min*,*max*,*step*)	vslider("vol",50,0,100,1)
hslider(*str*,*cur*,*min*,*max*,*step*)	hslider("vol",0.5,0,1,0.01)
nentry(*str*,*cur*,*min*,*max*,*step*)	nentry("freq",440,0,8000,1)
vgroup(*str*,*block-diagram*)	vgroup("reverb", ...)
hgroup(*str*,*block-diagram*)	hgroup("mixer", ...)
tgroup(*str*,*block-diagram*)	vgroup("parametric", ...)
vbargraph(*str*,*min*,*max*)	vbargraph("input",0,100)
hbargraph(*str*,*min*,*max*)	hbargraph("signal",0,1.0)
attach	attach(x, vumeter(x))

Quelques applications

Le panoramique stéréophonique

```
1 //-----------------------------------------------
2 //       The Second Simplest Panner
3 //-----------------------------------------------
4
5 c = hslider("pan", 0.5, 0, 1, 0.01);
6 process = _ <: *((1-c) : sqrt), *((c) : sqrt);
```

```
process = _ <: *(1 - c : sqrt), *(c : sqrt) ;
```

```
process = _ <: *(sqrt(1 - c)), *(sqrt(c)) ;
```

```
1 //-----------------------------------------------
2 //       Angle-Interpolated Panner
3 //-----------------------------------------------
4
5 import("filter.lib") ;
6 t = hslider("interpolation time", 0.001, 0, 0.01, 0.0001) ;
7 c = hslider("pan", 0.5, 0, 1, 0.01) : smooth(tau2pole(t)) ;
8 process = _ <: *(1-c : sqrt), *(c : sqrt) ;
```

```
1 //-----------------------------------------------
2 //       Output-Interpolated Panner
3 //-----------------------------------------------
4
5 import("filter.lib") ;
6 t = hslider("interpolation time", 0.001, 0, 0.01, 0.0001) ;
7 c = hslider("pan", 0.5, 0, 1, 0.01) ;
8 process = _ <: *(1-c : sqrt) : smooth(tau2pole(t)),
9                (*(c : sqrt) : smooth(tau2pole(t))) ;
```

Sample et Hold

```
1 //-----------------------------------------------
2 //          S&H
3 //-----------------------------------------------
4
5 but         = button("Hold!") ;
6 SH(trig,x) = (*(1 - trig) + x * trig) ~ _ ;
7 process     = SH(but) ;
```

SH(trig, x) = (* (1 – trig) + x * trig) ~ _ ;

Retards

@ ~ n'

```
. //-----------------------------------------------
. // @ example
. //-----------------------------------------------
. a =100;
. del(a,x) = x @ (a);
. process = del(a);
```

La dosage de la durée de retard doit être connue par le compilateur lors de la compilation, de sorte que vous ne pouvez utiliser cet objet que si cette valeur est constante ou varie dans une

plage finie connue ; par exemple, il est possible d'utiliser un curseur, parce que les valeurs de curseur ont des limites qui sont fixées dans la définition de curseur :

```
. //-----------------------------------------------
. // @ example 1
. //-----------------------------------------------
.
. deldur = hslider("delaydur", 10000,1,44100,1);
. del (x) = x @ (deldur);
. process = del;
```

Delay recursif

```
. //-----------------------------------------------
. //  @ example 2
. //-----------------------------------------------
. deldur = hslider("delaydur", 10000,1,44100,1);
. fdbck = hslider("feedback ", 0.2,0.0,1.0,0.01);
. recursivdel = (_,_ :> @(deldur)) ~ *(q);
. process = recursivdel;
```

FILTRES

// Comb and Allpass filters
//-------------------------

allpass(dt,fb) = (_,_ <: (*(fb),_:+:@(dt)), -) ~ _ : (!,_);

comb(dt, fb, damp) = (+:@(dt)) ~ (*(1-damp) : (+ ~ *(damp)) : *(fb));

Niveau sonore

$$RMS(x_k) = \sqrt{\frac{\sum_{i=k-n}^{k} x_i^2}{n}}$$

```
. 1  //-----------------------------------------------
. 2  // The RMS example - fixed n
. 3  //-----------------------------------------------
4
. 5  S(n,x) =+(x-x@n)~ ;
. 6  Quad(x) = x * x;
. 7  RMS(n) = Quad : S(n) : /(n) : sqrt;
. 8  process = RMS(1000);
```

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
CIEREC - EA 3068

TD2 synthèse avec Faust et Max

Oscillateur simple sinusoïdal

Synthèse additive

```
// Simple Organ
gate = button ("gate");  //  keyon-keyoff
freq = hslider("freq[unit:Hz]", 440, 20, 20000, 1); //  keyon key
gain = hslider("gain", 0.5, 0, 10, 0.01);  //  keyon velocity process = voice(gate,
gain, freq) * hslider("volume", 0, 0, 1, 0.01);
// Implementation
phasor(f) = f/fconstant(int fSamplingFreq, <math.h>) : (+,1.0:fmod) ~ _ ;
osc(f) = phasor(f) * 6.28318530718 : sin;
timbre(freq)= osc(freq) + 0.5*osc(2.0*freq) + 0.25*osc(3.0*freq);
envelop(gate, gain) = gate * gain : smooth(0.9995) with { smooth(c) = * (1-c) : + ~
* (c) ; } ;
voice(gate, gain, freq) = envelop(gate, gain) * timbre(freq);
```

TD1

a) Faire un patch MaxMSP intégrant cet instrument monophonique

b) Modifier le code pour ajouter une sortie enveloppe.

c) Faire un patch MaxMSP intégrant une version polyphonique de cet instrument avec Mute

Oscillateurs forme d'ondes variées

Introduction aux librairies

effect.lib (midikey2hz ; log2 ; moog_vcf ; moog_vcf_2b ; moog_vcf_2bn ; wah4 ; crybaby ; flanger_mono ; phaser2_mono ; vocoder ; amp_follower ; gate_mono ; compressor_stereo ; jcrev ; satrev ; mono_freeverb ; fdnrev0 ; zita_rev1)

filter.lib (smooth ; rev2 ; resonlp ; lowpass ; highpass ; mth_octave_analyzer ; filterbank ; tf2np ; nlf2 ; tf2s)

hoa.lib ()

instrument.lib (envVibrato ; asr ; bandPass)

math.lib (SR ; PI ; neg ; inv ; fmax ; fabs ; fmin ; if ; count ; take ; interpolate ; selector ;

maxmsp.lib (LPF ; HPF ; BPF ; APF ; line ;

music.lib (delay ; fdelay ; sdelay ; db2linear ; noise ; osc ; osci ; adsr ; spat ; bpf ; bsmooth ; chebychev)

oscillator.lib (**saw2 ; square ; triangle ; pt** ; oscb ; oscs ; oscwc ; pink_noise ; lfnoise)

reduce.lib (maxn ; mainn ; mean ; RMS)

tonestack (tonestack)

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
CIEREC - EA 3068

Un synthétiseur monophonique

```
import("math.lib");
import("filter.lib");
import("effect.lib");
import("maxmsp.lib");
import("oscillator.lib");

freq = vslider("[1] Frequency [unit:PK] [style:knob] [tooltip: Sawtooth frequency
as a Piano Key (PK) number (A440 = key 49)]",  49,1,88,0.01) : pianokey2hz;

ampdb  = vslider("[2] Amplitude [unit:dB] [style:knob] [tooltip: Sawtooth waveform
amplitude]",  -20,-120,10,0.1);

amp = ampdb : ml.db2linear : fl.smooth(0.999);
// amp = 0.8;

detune1 = 1 + 0.01 *  vslider("[3] Detuning 1 [unit:%%] [style:knob] [tooltip:
Percentange frequency-shift up or down for second oscillator]",  -0.1,-10,10,0.01);
detune2 = 1 + 0.01 * vslider("[4] Detuning 2 [unit:%%] [style:knob] [tooltip:
Percentange frequency-shift up or down for third oscillator]",  +0.1,-10,10,0.01);


portamento = vslider("[5] Portamento [unit:sec] [style:knob] [scale:log] [tooltip:
Portamento (frequency-glide) time-constant in seconds]",  0.1,0.001,10,0.001);

sfreq = freq : fl.smooth(fl.tau2pole(portamento));

tone = (amp/3) * (sawtooth(sfreq) + sawtooth(sfreq*detune1) +
sawtooth(sfreq*detune2));

process = tone;
```

a) Ajouter un filtre dans Faust à cet instrument

b) Faire un patch MaxMSP intégrant cet instrument et faire des presets.

Bruit blanc – rose - Filtres

Dans la bibliothèque « music.lib » fonction noise
Dans la bibliothèque « oscillator.lib » fonction pink_noise

```
import("math.lib");
import("maxmsp.lib");
import("music.lib");
import("oscillator.lib");
import("reduce.lib");
import("filter.lib");
import("effect.lib");


freq = hslider("freqC", 440, 10, 10000, 0.5);
gain = hslider("gain", 0.5, 0.0, 1.0, 0.01);
Q = hslider("Q", 50, 0.5, 10000, 0.5);

process=BPF(noise, freq, 1, Q)*gain;
```

a) Contrôler la fréquence du filtre avec un portamento et le clavier

b) Faire un preset pour simuler le bruit de la mer (bruit rose)

c) Faire un preset pour simuler le bruit du vent (bruit blanc)

Synthèse FM

```
import("math.lib");
import("filter.lib");
import("maxmsp.lib");
import("oscillator.lib");

//----------- params -------------
freq = hslider("freq ", 440, 20, 2000, 1);
gain = hslider("gain", 1, 0, 1, 0.01)  : smooth(0.999);
indexFM = hslider("index", 6, 0, 20, 0.1) : smooth(0.999);
fmfreq = hslider("freqmod", 1, 1, 4, 1);

//----------- oscillos -------
fmfm (freq, amp, transpfm, index) =
        amp * osci(freq + amp * index * freq * transpfm * osci(freq * transpfm)) ;

//------------- process -----------//
process = fmfm (freq, gain,  fmfreq, indexFM);
```

Contrôle de « freq » et « index » par signal et non par messages

```
import("math.lib");
import("filter.lib");
import("maxmsp.lib");
import("oscillator.lib");

//----------- note -------------
gain = hslider("gain", 1, 0, 1, 0.01)  : smooth(0.999);
freqmod = hslider("freqmod", 1.01, 0.1, 20, 0.01);


//----------- oscillos -------
fmfm (freq, amp, transpfm, index) =
        amp * osci(freq + amp * index * freq * transpfm * osci(freq * transpfm)) ;

//------------- process avec deux entrées signal----------//
process(x, y) = fmfm (x, gain,  freqmod, y);
```

a) Faire des presets pour des sons percussifs (harmoniques et inharmoniques)

b) Faire des presets pour des sons lents (nappes)

Modèles physiques (dossier examples/faust-stk/)

Instruments à cordes

Instruments à vent

Percussions métalliques

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
CIEREC - EA 3068

Quelques instruments

Orgue simple (d'après A. Graef LAC 2007)

Organ2EkV.dsp : orgue avec clavier, contrôle par clavier et pad 2D (7/4/16)

```
import("effect.lib");

gate_group(x)    = hgroup("3-gates", x);
gate     = gate_group(checkbox ("[1] gate"));         //     keyon-keyoff
gate_m   = gate_group(checkbox ("[2] gate_min"));     //     keyon-keyoff
gate_M   = gate_group(checkbox ("[3] gate_Maj"));     //     keyon-keyoff

freq     = hslider("[4] freq [unit:Hz]", 440, 40, 5000, 1); // keyon key

gain     = hslider("[5] gain", 0.5, 0, 1, 0.01);            // keyon velocity

timbre(freq)= osc(freq) + 0.5*osc(2.0*freq) + 0.25*osc(3.0*freq);

envelop(gate, gain) = gate * gain : smooth(0.9995)
                with { smooth(c) = * (1-c) : + ~ * (c) ; } ;

voice(gain, freq) = (envelop(gate, gain) : *(timbre (freq)))
                 + (envelop(gate*gate_m, gain*0.5) :
                 *(timbre(freq) + timbre(f3m) + timbre(f5)))
                 + (envelop(gate*gate_M, gain*0.5) :
                 *(timbre(freq) + timbre(f3M) + timbre(f5))) ;

f3m = freq * 6/5 ;
f3M = freq * 5/4 ;
f5 = freq * 1.5 ;

// --- reverb ------

roomSize = hslider("[6]roomSize [multi:1]",0.72,0.01,3,0.01);

instrReverb2 = zita_rev1_stereo(rdel,f1,f2,t60dc,t60m,fsmax)
       with{
       rdel = 20;
       f1 = 200;
       f2 = 6000;
       t60dc = roomSize*3;
       t60m = roomSize*2;
       // fsmax =SR ;
       fsmax = 96000.;
       };

tbar             = 0.1;                    // attack/release time in seconds
gbar             = exp(-1/(SR*tbar));      // corresponding gain factor
envbar           = abs : *(1-gbar) : + ~ *(gbar) : linear2db;
meter(x) = attach(x, envbar(x) : hbargraph("[9]level", -96, 10));

process = vgroup("Synth1 [style:multikeyboard]", voice(gain, freq)  :_ *
hslider("[8]volume [multi:0]", 0, 0, 1, 0.01)<:instrReverb2 :meter, _  );
```

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
CIEREC - EA 3068

Générateur de bruit continu ou granulaire

WindMgD.dsp : 6 paramètres avec accéléromètres + volume manuel (7/4/16)

```
import("math.lib");
import("maxmsp.lib");
import("music.lib");
import("oscillator.lib");
import("reduce.lib");
import("filter.lib");
import("effect.lib");

moog_vcf_wind =  moog_vcf(res,freq) * 0.5; // fix select normalized

gain = hslider("[1]Gain[acc:0 0 -10 0 10]", 0.2, 0, 1, 0.01):smooth(0.9997);
freq = hslider("[2]Freq[acc:1 0 -10 0 10]", 440, 100, 5000, 1):smooth(0.9997);
res  = hslider("[3]Res[acc:2 0 -10 0 10]", 0.8, 0, 0.995, 0.001):smooth(0.9997);

crounch  = hslider("[4]Crounch[acc:2 0 -10 0 10]", 0.0, 0, 1, 0.01):smooth(0.9997);

//----------------------- NOISEBURST -------------------------

noiseburst = noise : *(gate : trigger(P))
        with {
        upfront(x) = (x-x') > 0;
        decay(n,x) = x - (x>0)/n;
        release(n) = + ~ decay(n);
        trigger(n) = upfront : release(n) : > (0.0) ;      };

P = freq2; // fundamental period in samples
Pmax = 4096; // maximum P (for delay-line allocation)

gate = phasor(1) :-(0.001):pulsar;
gain22 = 1;
freq2 = hslider("[5]Grain Size[acc:2 0 -10 0 10]", 200,100,2205,1);
// la fréquence donne la largeur de bande extraite du bruit blanc

// PHASOR ////////////////////////////
phasor(init) =  (+(float(speed)/float(SR)) : fmod(_,1.0)) ~ *(init);

// PULSAR ////////////////////////////
//Le pulsar permet de créer une 'pulsation' plus ou moins aléatoire (proba).

pulsar = _<:((_<(ratio_env)):@(100))*(proba>(_,abs(noise):latch));
speed = hslider ("[6]Speed[unit:Hz][acc:0 0 -10 0 10]",
10,1,20,0.0001):lowpass(1,1);
ratio_env = 0.5;
fade = (0.5); // min > 0 pour eviter division par 0
proba = hslider ("[7]Probability[unit:%][acc:1 1 -10 0 10]", 70,50,100,1) *
(0.01):lowpass(1,1);
duree_env = 1/(speed: / (ratio_env*(0.25)*fade));

tbar            = 0.1;                                  // attack/release time
in seconds
gbar            = exp(-1/(SR*tbar));      // corresponding gain factor
envbar          = abs : *(1-gbar) : + ~ *(gbar) : linear2db;
meter(x) = attach(x, envbar(x) : hbargraph("level", -96, 10));


process= noise * (1 - crounch) + noiseburst * crounch * 2 :
          *(gain):*(hslider("[8]volume", 0, 0, 1, 0.01)):moog_vcf_wind :
          max(-0.99):min(0.99) : meter <: _,_;
```

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
CIEREC - EA 3068

Synthétiseur avec harmoniseur

syntwavfk6.dsp : synthétiseur avec accéléromètres (14/4/16)

```
import("music.lib");
import("math.lib");
import("filter.lib");
import("effect.lib");
import("maxmsp.lib");
import("oscillator.lib");

freq = hslider("[1] Frequency [unit:PK]", 49,1,88,0.01) : pianokey2hz:
smooth(0.99);

voldb  = hslider("[8] Volume ",  -96,-96,0,0.1);
vol = voldb : db2linear : smooth(0.995);

detune = 1 + 0.01 *  hslider("[2][acc:0 0 -10 0 10] Detune [unit:%%] ",  -0.1,-
0.1,0.9,0.01);

tone =  saw3(freq) + saw3(freq*detune): *(0.4);

// harmofbk
win = 5000;
xfade = 500;
shift = hslider("[4]transpose", 0.01, -12, 12, 0.01);
fbk = hslider("[5] [acc:1 0 -10 0 10] fbk", 0.2, 0, 1, 0.001);
del = hslider("[6]del", 50, 0, maxdel, 0.001): smooth(0.995);

maxdel = 5000 ;

transpose (w, x, s, sig)  =
        fdelay1s(d,sig)*fmin(d/x,1) + fdelay1s(d+w,sig)*(1-fmin(d/x,1))
                with {
                        i = 1 - pow(2, s/12);
                        d = i : (+ : +(w) : fmod(_,w)) ~ _;
                };

pitchshifter = _<:(transpose(win, xfade, shift):*(fbk)),_:+:*(0.5);
// --- reverb ------
roomSize = hslider("[7]roomSize",0.72,0.1,3,0.01);

instrReverb2 = zita_rev1_stereo(rdel,f1,f2,t60dc,t60m,fsmax)
       with{
       rdel = 20;
       f1 = 200;
       f2 = 6000;
       t60dc = roomSize*3;
       t60m = roomSize*2;
       // fsmax =SR ;
       fsmax = 96000.;
       };

tbar            = 0.1;          // attack/release time in seconds
gbar            = exp(-1/(SR*tbar));     // corresponding gain factor
envbar          = abs : *(1-gbar) : + ~ *(gbar) : linear2db;
meter(x) = attach(x, envbar(x) : hbargraph("[9]level", -96, 10));

process = tone : (+ : pitchshifter) ~(*(fbk):delay(maxdel,del)):
        *(vol)<:(instrReverb2, _, _)<:!,*(0.1),_,!:+:
        meter :max(-0.99):min(0.99)<:_,_;
```

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
CIEREC - EA 3068

Synthétiseur avec Sample&Hold

Synth12c8.dsp : synthétiseur androïd avec clavier et pad 2D + accéléromètres (19/1/15)

```
import("math.lib");
import("filter.lib");
import("maxmsp.lib");
import("oscillator.lib");

//----------- note -------------
gate = checkbox("gate");
gate2 = gate : \(x).(x>x');
freq = hslider("freq ", 440, 20, 2000, 1);
gain = hslider("gain", 1, 0, 1, 0.01)  : smooth(0.999);

swbruit = checkbox ("swbruit") ;

freq1 = freq * vibrato * SampleHold ;

//-----------------------------------------------
//                       osc(freq) : Sinusoidal Oscillator
//-----------------------------------------------
xxx = environment {
        tablesize       = 1 << 16;
        samplingfreq    = SR;

        time            = (+(1)~_ ) - 1;                          // 0,1,2,3,...
        sinwaveform     = float(time)*(2.0*PI)/float(tablesize) : sin;

        decimal(x)      = x - floor(x);
        phase(trig, freq)       = freq/float(samplingfreq) : (+ : decimal :
if(trig>0, 0, _)) ~ _ : *(float(tablesize));
        osc(trig, freq)  = rdtable(tablesize, sinwaveform, int(phase(trig,freq)) );
        osci(trig, freq) = s1 + d * (s2 - s1)
                       with {
                               i = int(phase(trig, freq));
                               d = decimal(phase(trig, freq));
                               s1 = rdtable(tablesize+1,sinwaveform,i);
                               s2 = rdtable(tablesize+1,sinwaveform,i+1);};
};


//**** definit un indicer qui va permettre de lire une table ****
// parametres : dur = duree en echantillons (minimun 1)
//                               trig = quand > 0 declenche le cycle
// sortie : valeur variant de 0 a 1.0 en dur echantillons, reste a 1.0 ensuite
// NOTE : la fonction est protegee contre les durees nulles et les durees qui
changent en cours de route

indicer(dur, trig) = I ~ (_,_) : /
        with {
                       I(p, d) = if(trig>0.0, 0.0, min(d, p+1.0)),
                                       max(1.0, if(trig>0.0, dur, d));
        };

ifunctTab(bpf, low, high, dur, trig) = s1 * (1.0 - d) + s2 *d : *(high - low) + low
        with {
                       phase = 100.0 * indicer(dur, trig) ;
                       i = int(phase) ;
                       d = decimal(phase);
                       s1 = rdtable(101, bpf(time), i);
```

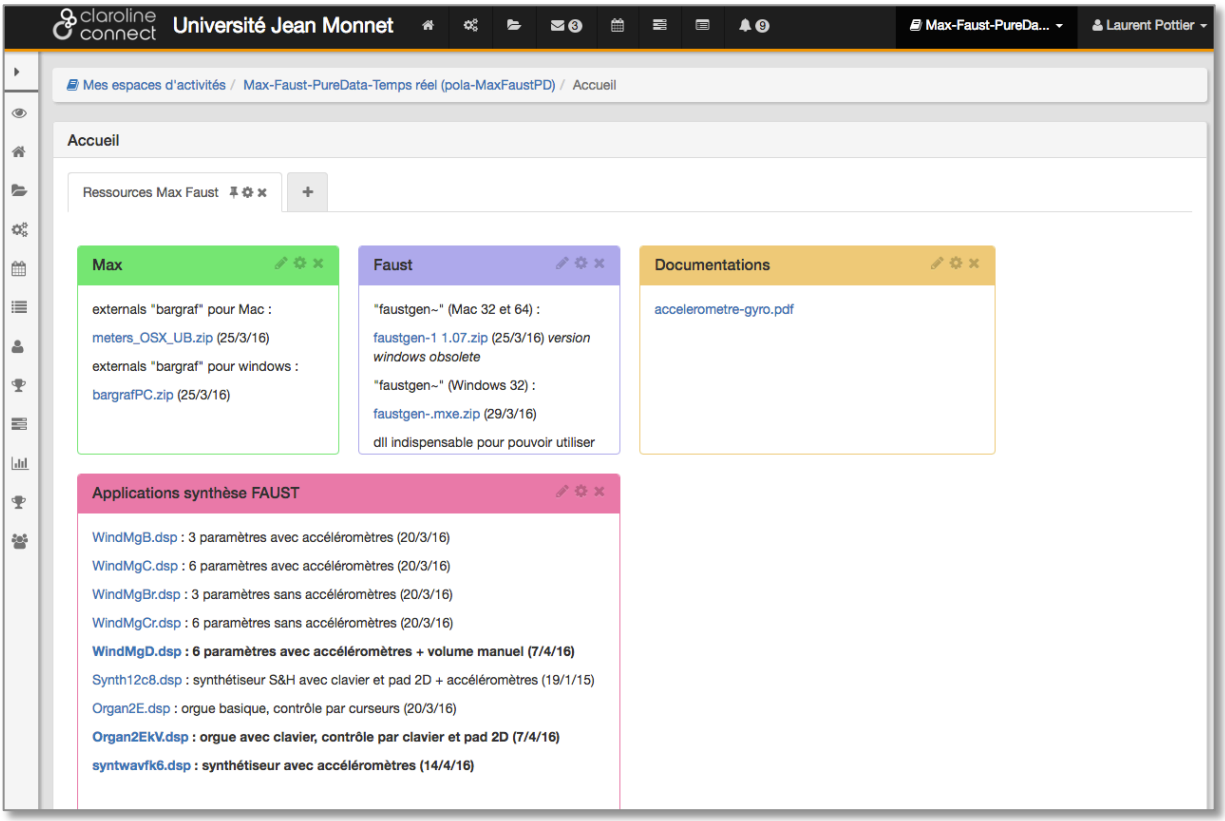
```
                        s2 = rdtable(101, bpf(time), i+1);
                };

bpf9  = bpf.start(0,1.0) : bpf.point(25,0.25) : bpf.point(50,0.1) :
bpf.point(75,0.05) : bpf.end(100,0);
bpf10 = bpf.start(0,0.5) : bpf.point(20,1.0)  : bpf.point(50,0.1) :
bpf.point(75,0.02) : bpf.end(100,0);
bpf11 = bpf.start(0,0.0) : bpf.point(10,0.02) : bpf.point(15,0.1) :
bpf.point(25,0.5)  : bpf.point(30,0.95) : bpf.point(33,1) : bpf.point(85,0.95) :
bpf.point(90,0.9) : bpf.point(95,0.05) : bpf.end(100,0);
bpf12 = bpf.start(0,0.0) : bpf.point(10,0.2)  : bpf.point(15,0.10):
bpf.point(25,0.5)  : bpf.point(30,0.95) : bpf.point(33,1.0) : bpf.end(100,0);
bpf13 = bpf.start(0,0.0) : bpf.point(13,0.5)  : bpf.point(30,1.0) :
bpf.point(60,0.2)  : bpf.point(75,0.04) : bpf.end(100,0);
bpf14 = bpf.start(0,1.0) : bpf.point(12,0.75) : bpf.point(25,1.0) :
bpf.point(50,0.25) : bpf.point(75,0.05) : bpf.end(100,0);
bpf15 = bpf.start(0,1.0) : bpf.point(5,0.1)   : bpf.point(10,0.02) :
bpf.point(25,0.0) : bpf.end(100,0);
bpf16 = bpf.start(0,0.0) : bpf.point(15,0.1) : bpf.point(30,0.50) :
bpf.point(40,0.95) : bpf.point(50,1.0) : bpf.point(60,0.96) : bpf.point(70,0.5) :
bpf.point(85,0.1) : bpf.end(100,0);
bpf17 = bpf.start(0,1.0) : bpf.point(11,0.5) : bpf.point(25,0.25) :
bpf.point(50,0.1) : bpf.point(75,0.05) : bpf.end(100,0);


env(trig, dur, bpfxf, low, high) =
        if (bpfx == 9,  ifunctTab(bpf9,  low, high, dur, trig),
        if (bpfx == 10, ifunctTab(bpf10, low, high, dur, trig),
        if (bpfx == 11, ifunctTab(bpf11, low, high, dur, trig),
        if (bpfx == 12, ifunctTab(bpf12, low, high, dur, trig),
        if (bpfx == 13, ifunctTab(bpf13, low, high, dur, trig),
        if (bpfx == 14, ifunctTab(bpf14, low, high, dur, trig),
        if (bpfx == 15, ifunctTab(bpf15, low, high, dur, trig),
        if (bpfx == 16, ifunctTab(bpf16, low, high, dur, trig),
                                        ifunctTab(bpf17, low, high, dur, trig)
        )))))))) with { bpfx = int(bpfxf); };

swenv = checkbox("swenv");
indexFM = hslider("h:FM/index [multi:1]", 6, 0, 20, 0.1) : smooth(0.999);
dur = 3000 * attack / 100. ;
nbpf = 9;
envelop(swenv) = select3(swenv, env1, env2);

env1 = env(gate2, nsamples, nbpf, 0, 1)
        with {
                nsamples = max(10, dur * SR / 1000.0) ;
        }        ;

//------------ enveloppe tenue --------

attack = hslider("v:General Parameters/att4", 100, 1, 2000, 1) ;
release = attack * 2 ;

durr = select2(gate, release, attack);
env2= line(gate, durr) ;


//----------- oscillos -------

fmfm2 (freq, amp, transpfm, index, transp) =
        env2 * amp *
        osci(freq * transp +
                env2 * amp * index *
                        freq * transpfm * transp * osci(freq * transpfm * transp))
;
```

```
fmfm1 (trig, freq, amp, transpfm, index, transp) =
        env1 * amp *
        xxx.osci(trig, freq * transp +
                env1 * amp * index *
                        freq * transpfm * transp * xxx.osci(trig, freq * transpfm
* transp)) ;

synthfm(swenv) = select2(swenv, fmfm1 (gate2, freq1, gain, fmfreq, indexFM, 1.0) ,
        fmfm2 (freq1, gain,  fmfreq, indexFM, 1.0));

noiseenv(swenv) = select2(swenv, noise * env1 * gain , noise * env2 * gain);

//--------- synth -----------
freqmod = hslider("h:FM/freqmod", 1, 1, 4, 1);
fmfreq = sqrt(freqmod):+(0.001);

synth = select2(swbruit, synthfm(swenv), noiseenv(swenv));
// synth = synthfm(swenv);

//-------------- vibrato ---------------//
ampvib = hslider("ampvib", 0.01, 0, 0.1, 0.001);
freqvib = 5.6;

vibrato =  1 + (ampvib * osci(freqvib));

//------------- sample&hold -----------//
cocoX  = hslider("h:SHold/cocoX [style:knob]", 400, 1 , 1000, 0.1);
cocoY = hslider("h:SHold/cocoY [style:knob]", 10, 1, 50, 0.1);
ampSH = hslider("h:SHold/ampSH [style:knob]", 0.1, 0, 0.9, 0.01) ;

trigSH  = pulse(SR/cocoY) ;
SampAndHold (x, trig) = (* (1 - trig) + x * trig) ~ _ ;
SampleHold = 1 + (ampSH * SampAndHold(triangle(cocoX), trigSH));

//------------- filtres -----------//
f1 = hslider("v:filtre/f1 [multi:0]" , 500, 20, 4000, 0.01) : smooth(0.999);
f2 = 2 * f1 ;
gain01 = 0,8;
gain02 = 0.8;
Q = 1;

filtre1(x) = BPF(x, f1, gain01, Q) ;
filtre2(x) = BPF(x, f2, gain02, Q) ;

filtres(x) = filtre1(x), filtre2(x) ;

//------------- delay -----------//
mixdel = hslider("mix echo", 0.2, 0, 1, 0.01) ;
mixwet = 1.0 - mixdel;
deldur1 = 0.250 * SR ;
deldur2 = 0.180 * SR ;
echos = _<:((_,_:> delay1s (deldur1)) ~ *(0.3)), ((_,_:> delay1s (deldur2)) ~
*(0.4));

tbar            = 0.1;                                  // attack/release time
in seconds
gbar            = exp(-1/(SR*tbar));      // corresponding gain factor
envbar          = abs : *(1-gbar) : + ~ *(gbar) : linear2db;

left_meter(x)    = attach(x, envbar(x) : hbargraph("left", -96, 10));
right_meter(x)   = attach(x, envbar(x) : hbargraph("right", -96, 10));
//------------- process -----------//
process = vgroup("Synth1 [style:multikeyboard]", synth : filtres <: *(mixdel),
*(mixwet) :  echos, (_<:_,_) :> (_,_):vgroup("Vu
metres",(left_meter,right_meter)));
```

Annexes

Site ENT de l'UJM

- Cours Max-Faust-PureData-Temps réel (pola-MaxFaustPD)



- Cours Faust > Applications Androïd

UNIVERSITÉ JEAN MONNET
SAINT-ÉTIENNE
CIEREC - EA 3068