

Chapter 1

Intro

Dans cet article je vais vous présenter une facette de Pharo qu'on a pas l'habitude de voir, un domaine pour lequel il peut surprendre. Ce domaine c'est la synthèse sonore. Dès qu'on a besoin de sonoriser un peu notre tout nouveau programme on pense soit à du mp3 pour la musique ou des sons digitalisés pour les bruitages. Dans Pharo on dispose d'outils et bibliothèques permettant créer de toute pièce du son synthétisé pour nos productions. Nous allons vous montrer comment on fait cela en quelques lignes de code seulement.

Tout d'abord récupérer et installer la version XXX de Pharo à l'adresse:
<http://www.pharo.org/YYY>

rajouter l'adresse de téléchargement

Chapter 2

Bases de la synthèse FM par l'exemple

Avant de se familiariser avec les différentes méthodes utiles on va s'assurer que notre environnement puisse jouer un son convenablement. Pour cela on va ouvrir un workspace (depuis le menu Tools) et évaluer la ligne de code suivante. Nous vous expliquerons par la suite ce dont il s'agit.

```
FMSound organ1 play.
```

Si aucun son n'est produit vous pouvez vérifier ces premiers points

Indiquer la marche à suivre ou une page internet l'indiquant ?

- Le volume sonore
- Le plugin sonore n'est pas installé.
- Le process sonore ne fonctionne plus. Il faut l'arrêter puis le relancer en évaluant

```
SoundPlayer stopPlayerProcess  
SoundPlayer startUp
```

Maintenant que le son est restauré nous allons découvrir comment celui ci est généré. Prenons le code de la méthode qui définit l'instrument **organ1**:

```
FMSound class>>organ1  
"FMSound organ1 play"  
"(FMSound majorScaleOn: FMSound organ1) play"  
  
| sound |
```

```

sound := self new.
sound addEnvelope: (VolumeEnvelope
    points: { 0@0 . 60@1.0 . 110@0.8 . 200@1.0 . 250@0.0}
    loopStart: 2 loopEnd: 4).
sound setPitch: 440.0 dur: 1.0 loudness: 0.5.
^ snd

```

Tout d'abord, créons une nouvelle instance de la classe **FMSound**

```

sound := self new.

```

Nous définissons ensuite un ensemble de points afin de créer une enveloppe de volume. Pour chaque point la première donnée est un nombre en milliseconde, la seconde est le volume exprimé de 0 à 1 (1 étant égal à 100% du volume). Notez qu'utiliser un point est un abus et que le programmeur aurait dû utiliser une paire.

```

sound addEnvelope: (VolumeEnvelope
    points: { 0@0 . 60@1.0 . 110@0.8 . 200@1.0 . 250@0.0}
    loopStart: 2 loopEnd: 4).

```

la phrase ne va pas du tout, a reformuler

L'enveloppe est définie d'après un ensemble de points et d'une boucle. Elle peut aussi bien reprendre l'ensemble des points ou seulement boucler sur 1 seul point. On peut ainsi jouer avec le volume d'un son pour qu'il s'atténue, ou pour donner un effet d'écho par exemple. On ajoute cette enveloppe au son avec une boucle sur l'indice 2 et 4. Si on regarde les valeurs on voit que ça oscille entre 0.8 et 1.

La méthode retourne le son avec comme paramètre la note (ici exprimée en Hz, voir tableau 10.1), la durée en seconde et un volume global de la note. Ce volume sera ensuite modulé grâce à l'enveloppe qui est définie.

```

^ sound setPitch: 440.0 dur: 1.0 loudness: 0.5

```

Chapter 3

Comment faire pour jouer notre propre son ?

Nous pouvons jouer un LA de cette manière

```
(FMSound new setPitch: 440 dur: 1.0 loudness: 1) play
```

Le son est joué à une fréquence de 440 Hz pendant 1 seconde.

La classe **FMSound** permet la génération de sons sinusoïdaux. Ça donne un son assez doux et régulier. La figure 3.1 ci dessous montre à quoi ressemble la courbe.

Pourquoi la figure n'est pas juste en dessous du paragraphe ???

Si on veut jouer une seule note l'écriture est simple, mais cela peut rapidement devenir long, fastidieux et les erreurs difficiles à détecter si on doit

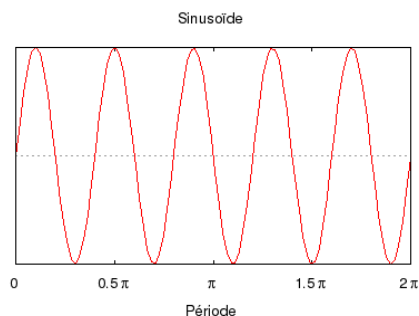


Figure 3.1: Sinusoïde produite.

écrire tout une mélodie de cette manière.

Une chose à savoir, est qu'au lieu de définir une note par sa fréquence nous pouvons l'écrire dans sa notation anglaise. Le tableau **10.2** va nous indiquer les correspondances.

Le LA de tout à l'heure peut s'écrire maintenant comme cela

```
(FMSound new setPitch: 'a4' dur: 1.0 loudness: 1) play
```

Le SOL# par exemple s'écrit

```
(FMSound new setPitch: 'g#4' dur:1.0 loudness: 1) play
```

Chapter 4

Jouer plusieurs sons en séquence

Jouer un son c'est bien, mais en jouer deux c'est mieux. Pour cela nous avons à notre disposition une méthode nommée **SequentialSound**. Comme son nom l'indique nous allons pouvoir jouer les sons qu'on souhaite de manière séquentielle.

```
|seq|  
seq := SequentialSound new.  
seq add: ((FMSound new) setPitch: 'g#4' dur:0.5 loudness: 1).  
seq add: ((FMSound new) setPitch: 'a3' dur:0.5 loudness: 1).  
seq add: ((FMSound new) setPitch: 'g#4' dur:0.5 loudness: 1).  
seq play.
```

Si nous écoutons les sons générés, nous pouvons percevoir un petit clap entre chaque notes. Afin d'éviter cela il faut rajouter à chaque note une enveloppe de volume.

Changeons un peu le code

```
|seq snd1 snd2 snd3 points|  
"Declaration des sons"  
seq := SequentialSound new.  
snd1 := FMSound new.  
snd2 := FMSound new.  
snd3 := FMSound new.  
points := OrderedCollection new.  
  
"Creation du volume pour les instruments"  
points add: 0@1;add: 400@0.4;add: 500@0.  
  
"Generation des notes"
```

```
snd1 setPitch: 'g#4' dur:0.5 loudness: 1.  
snd2 setPitch: 'a3' dur:0.5 loudness: 1.  
snd3 setPitch: 'g#4' dur:0.5 loudness: 1.
```

"Application de l'enveloppe aux sons"

```
snd1 addEnvelope: (VolumeEnvelope points: points loopStart: 3 loopEnd:3 ).  
snd2 addEnvelope: (VolumeEnvelope points: points loopStart: 3 loopEnd:3 ).  
snd3 addEnvelope: (VolumeEnvelope points: points loopStart: 3 loopEnd:3 ).
```

"Les sons les uns a la suite des autres"

```
seq add: snd1.  
seq add: snd2.  
seq add: snd3.
```

"On joue le tout"

```
seq play.
```

La séquence est plus agréable à écouter et les claps de fin ne sont plus audibles. Avec un peu créativité nous allons pouvoir faire quelque chose de ce son.

Chapter 5

Jouer plusieurs sons simultanément

Jouer des sons en séquence c'est bien mais pas encore suffisant pour donner une âme à notre production. Si nous voulons avoir quelque chose qui sonne correctement à nos oreilles il va falloir maintenant rajouter ce que nous appellerons des pistes. La classe **MixedSound** va nous aider à cela.

```
|mix snd1 snd2 points|
mix := MixedSound new.
snd1 := FMSound new.
snd2 := FMSound new.

points := OrderedCollection new.

"Creation du volume pour les instruments"
points add: 0@1;add: 400@0.4;add: 500@0.

"Generation des notes"
snd1 setPitch: 'a4' dur:1 loudness: 1.
snd2 setPitch: 'a3' dur:1 loudness: 1.

"Application de l'enveloppe aux sons"
snd1 addEnvelope: (VolumeEnvelope points: points loopStart: 3 loopEnd:3 ).
snd2 addEnvelope: (VolumeEnvelope points: points loopStart: 3 loopEnd:3 ).

mix add: snd1 pan: 0; add: snd2 pan: 1.

mix play.
```

Le son généré est quelque peu old school. Modifions un peu les valeurs **setPitch** pour obtenir des notes différentes et nous familiariser avec les méth-

odes. Les enveloppes de volume modifient la courbe du son au cours du temps et peuvent être utilisées pour obtenir des effets particuliers.

Remplacez les valeurs des points par

```
points add: 0@1;add: 100@0.6; add: 200@0; add: 300@0.6;add: 400@0.2;add: 500@0.
```

et la déclaration des enveloppes par

```
snd1 addEnvelope: (VolumeEnvelope points: points loopStart: 6 loopEnd:6 ).  
snd2 addEnvelope: (VolumeEnvelope points: points loopStart: 6 loopEnd:6 ).
```

Le son produit comme un écho. La variation entre les valeurs est douce et continue. Un calcul est fait entre chacune d'entre elles pour éviter les cassures disgracieuse à l'oreille. Vous pouvez vous avec les nouvelles valeurs ci dessous.

```
points add: 0@1;add: 100@0.6; add: 200@0; add: 201@0.6;add: 400@0.2;add: 500@0.
```

La cassure est nettement plus audible.

Nous allons également utiliser le message **pan:** qui va positionner le son en stéréo. Une valeur de **0** pour gauche et **1** pour droite. Une valeur de **0.5** place le son au milieu.

Chapter 6

Les enveloppes

L'enveloppe modifie le son au cours du temps. Nous venons de le voir avec le volume, mais il est également possible de faire varier la note pendant qu'elle est jouée, c'est ce qu'on appelle le pitch. De la même manière que pour le volume nous allons définir un ensemble de points qui indiqueront le moment du changement de la fréquence du son. Plus la valeur est haute, plus la note le sera. Si on indique un chiffre négatif, la note descendra en dessous de la note de départ.

```
|snd p|  
p := OrderedCollection new.  
p add: 0@0; add: 100@0.1; add: 200@0.2; add: 300@(-0.5);  
  add: 400@0.7; add: 800@0.8; add: 1000@1.  
snd := FMSSound new.  
snd addEnvelope: (PitchEnvelope points: p loopStart:7 loopEnd:7 ).  
(snd setPitch: 'a4' dur: 2 loudness: 0.5) play.
```

La variation se fait en douceur, sans cassures. En couplant cela à une enveloppe de volume on peut commencer à produire un son intéressant.

```
|snd p p2|  
p := OrderedCollection new.  
p2 := OrderedCollection new.  
p add: 0@0; add: 100@0.1; add: 200@0.2; add: 300@(-0.5);  
  add: 400@0.7; add: 800@0.8; add: 1000@1.  
p2 add: 0@0; add: 100@0.1; add: 200@0.2; add: 300@(-0.5);  
  add: 400@0.7; add: 800@0.8; add: 1000@0.  
snd := FMSSound new.  
snd addEnvelope: (PitchEnvelope points: p loopStart:7 loopEnd:7 ).  
snd addEnvelope: (VolumeEnvelope points: p2 loopStart:7 loopEnd:7 ).  
(snd setPitch: 'a4' dur: 2 loudness: 0.5) play.
```

Au lieu de définir l'enveloppe points par points nous allons utiliser une méthode appelée **exponentialDecay**: de la classe **VolumeEnvelope** pour le volume ou **PitchEnvelope** pour le pitch. La courbe sera de type exponentielle partant du volume le plus haut pour arriver en douceur à 0.

La définition du volume devient alors ceci

```
snd addEnvelope: (VolumeEnvelope exponentialDecay: 0.9).
```

La valeur doit être impérativement plus grande que 0 et plus petite que 1. Pour connaître l'ensemble des points générés il suffit de faire un **clic droit/Print it** sur la ligne de code suivante

```
(VolumeEnvelope exponentialDecay: 0.2) points.
```

Le résultat est le suivant

```
{{(0@0.0). (10@1.0). (20@0.2). (30@0.040000000000000001). (50@0.0)}}
```

Les enveloppes viennent compléter nos connaissances sur la génération du son. Nous savons donc jouer un son, plusieurs sons les uns après les autres ou encore plusieurs sons joués en même temps.

On a entre nos mains pratiquement tous les outils pour écrire notre première mélodie, que dis-je, notre première super production !

Chapter 7

Jouer une mélodie

A quoi peut bien ressembler de la musique produite dans Pharo ? Evaluons la ligne de code suivante dans un workspace et écoutons.

```
FMSound bachFugue play
```

La musique est une séquence de notes. Nous savons maintenant comment écrire une telle séquence mais pour une mélodie entière ce n'est pas adapté. Il est nécessaire de faire autrement.

Nous allons décrire l'ensemble des notes dans un tableau. Le format est le suivant

```
(note duree volume)
```

La note peut être écrite sous sa forme littérale ou en hertz. La durée s'exprime en millisecondes. Le volume quant à lui est une valeur entre 0 et 1000. Au delà de 1000 le son peut saturer mais cela peut se révéler utile dans certain cas pour donner un effet.

Voici un exemple de déclaration

```
notes := #((a3 200 150) (a4 100 150) (g3 150 150))
```

Nous allons écrire notre premier morceau avec un air qui devrait rappeler des bons souvenirs à tout ceux qui ont connu le début des jeux vidéos.

Déclarerons la classe permettant de mixer l'ensemble des pistes.

```
FMSound class>>pacman
```

```
"comment stating purpose of message"
```

```
^ MixedSound new
```

```
add: (self pacmanV1On: self fm1) pan: 0.5;
```

```
add: (self pacmanV2On: self fm2) pan: 0.5.
```

Les 2 méthodes suivantes contiennent l'ensemble des notes à jouer avec le triplet note, durée et volume.

```
FMSound class>>pacmanV1On: aSound
```

```
|notes|
notes := #(
  (b4 0.144 150) (b5 0.144 150) ('f#5' 0.144 150) ('d#5' 0.144 150)
  (b5 0.072 150) ('f#5' 0.144 150) ('d#5' 0.288 150) (c5 0.144 150)
  (c6 0.144 150) (g5 0.144 150) (e5 0.144 150) (c6 0.072 150)
  (g5 0.216 150) (e5 0.288 150) (b4 0.144 150) (b5 0.144 150)
  ('f#5' 0.144 150) ('d#5' 0.144 150) (b5 0.072 150) ('f#5' 0.216 150)
  ('d#5' 0.288 150) ('d#5' 0.072 150) (e5 0.072 150) (f5 0.144 150)
  (f5 0.072 150) ('f#5' 0.072 150) (g5 0.144 150) (g5 0.072 150)
  ('g#5' 0.072 150) (a5 0.144 150) (b5 0.216 150)).

^self noteSequenceOn: aSound from: notes.
```

La méthode suivante va produire une mélodie avec des notes assez basse. Lorsqu'un son sinusoïdale possède une fréquence basse, le volume de celui ci semble faible. Pour que cette piste soit audible on va augmenter le son de chacune des notes.

```
FMSound class>>pacmanV2On: aSound
```

```
|notes|
notes := #(
  (b1 0.432 500) (b2 0.144 500) (b1 0.432 500) (b2 0.144 500) (c2 0.432 500)
  (c3 0.144 500) (c2 0.432 500) (c3 0.144 500) (b1 0.432 500) (b2 0.144 500)
  (b1 0.432 500) (b2 0.144 500) ('f#2' 0.288 500) ('g#2' 0.288 500)
  ('a#2' 0.288 500) (b2 0.288 500)).

^self noteSequenceOn: aSound from: notes.
```

Les 2 méthodes suivantes décrivent les instruments qu'on va utiliser pour jouer la mélodie.

```
FMSound class>>fm1
```

```
|snd|

snd := self new.

snd addEnvelope: (VolumeEnvelope
  points: { 0@1 . 300@0 }
  loopStart: 2
  loopEnd: 2
  ).
snd modulation: 1 ratio: 2.

^snd
```

```
FMSound class>>fm2
|snd|

snd := self new.

snd addEnvelope: (VolumeEnvelope
  points: { 0@1 . 300@0 }
  loopStart: 2
  loopEnd: 2
  ).
snd modulation: 1 ratio: 2.

^snd
```

revue de code en cours

Les 2 classes définissant les instruments sont rigoureusement les mêmes, mais je vous invite une fois de plus à modifier les valeurs de l'enveloppe ou de la modulation et d'écouter le résultat.

Chapter 8

Jouer un sample à partir d'un fichier ou directement depuis la mémoire

La sonorisation d'une production ne passe pas essentiellement par de la synthèse sonore, parfois il est nécessaire de faire appel à des bruitages ou des voix enregistrées.

Essayons dans un premier temps de jouer un son à partir d'un fichier au format wav

```
(SampledSound fromWaveFileNamed: '/home/messner/sound.wav') play
```

Le son est joué directement depuis le fichier, cependant si nous faisons un jeu, les chargements risqueraient de saccader les animations et rendre le jeu injouable. Afin d'éviter ces petits désagréments les sons seront chargés en mémoire et joués lorsque ce sera nécessaire.

Pour cela on va le rajouter à notre librairie de sons

```
|spl spl2|
```

```
"Chargement du son a partir d'un fichier"
```

```
spl := SampledSound fromWaveFileNamed: '/home/messner/guitar.wav'.
```

```
"Ajout du son dans la librairie pour le jouer plus tard"
```

```
SampledSound addLibrarySoundNamed: 'guitar'
```

```
  samples: spl samples
```

```
  samplingRate: spl originalSamplingRate.
```

le "à" n'est pas correctement interprété dans un bloc. Pkoi ???

Il nous reste plus qu'à le jouer à la fréquence et à la durée souhaitée

18 *Jouer un sample à partir d'un fichier ou directement depuis la mémoire*

```
|spl spl2|  
spl := SampledSound soundNamed: 'guitar'.  
spl2 := (SampledSound samples: spl samples samplingRate: 2000).  
spl2 duration: 0.1.  
spl2 play.
```

Chapter 9

Un dernier petit effort

Nous allons clore cet article sur une dernière méthode permettant de jouer plusieurs sons digitalisés en séquence.

Dans un premier temps nous allons charger en mémoire les différents samples qui seront utilisés.

```
|spl spl1 spl2|

"Chargement des sons"
spl := SampledSound fromWaveFileNamed: '/home/messner/ocean.wav'.
spl1 := SampledSound fromWaveFileNamed: '/home/messner/mouette.wav'.
spl2 := SampledSound fromWaveFileNamed: '/home/messner/bateau.wav'.

"Ajout des sons dans la librairie pour les jouer plus tard"
SampledSound addLibrarySoundNamed: 'ocean'
    samples: spl samples
    samplingRate: spl originalSamplingRate.
SampledSound addLibrarySoundNamed: 'mouette'
    samples: spl1 samples
    samplingRate: spl1 originalSamplingRate.
SampledSound addLibrarySoundNamed: 'bateau'
    samples: spl2 samples
    samplingRate: spl2 originalSamplingRate.
```

Maintenant que tout nos sons sont chargés nous allons les jouer en séquence grâce à la méthode **QueueSound**.

```
|mix q1 q2 spl1 spl2 spl3|

q1 := QueueSound new.
q2 := QueueSound new.
mix := MixedSound new.
spl1 := (SampledSound samples: (SampledSound soundNamed: 'ocean')
```

```
        samples samplingRate: 22000).
spl1 duration: 20.
spl2 := (SampledSound samples: (SampledSound soundNamed: 'mouette')
        samples samplingRate: 22000).
spl2 duration: 10.
spl3 := (SampledSound samples: (SampledSound soundNamed: 'bateau')
        samples samplingRate: 22000).
spl3 duration: 5.

q1 add:spl2;add:spl2.
q2 add:spl3;add:spl3.
mix add: q1 pan: 0 volume: 0.100;
    add: q2 pan: 1 volume: 0.5;
    add: spl1 pan: 0.5 volume: 1.
mix play.
```

Chapter 10

Annexes

10.1 Tableau des fréquences

Note	Octave								
	-1	0	1	2	3	4	5	6	7
C	16.3	32.7	65	131	262	523	1046.5	2093	4186
C#	17.3	34.6	69	139	277	554	1109	2217	4435
D	18.3	36.7	74	147	294	587	1175	2349	4698
D#	19.4	38.9	78	156	311	622	1244.5	2489	4978
E	20.5	41.2	83	165	330	659	1318.5	2637	5274
F	21.8	43.6	87	175	349	698.5	1397	2794	5588
F#	23.1	46.2	92.5	185	370	740	1480	2960	5920
G	24.5	49.0	98	196	392	784	1568	3136	6272
G#	26.0	51.9	104	208	415	831	1661	3322	6645
A	27.5	55.0	110	220	440	880	1760	3520	7040
A#	29.1	58.0	117	233	466	932	1865	3729	7458
B	30.8	62.0	123	247	494	988	1975	3951	7902

10.2 Notation anglaise

Anglais	C	C#	D	D#	E	F	F#	G	G#	A	A#	H
Français	Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si