Chapter 1

Une introduction aux sons en Pharo

Dans cet article nous allons vous présenter une facette de Pharo qu'on n'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 dernière version 30 de Pharo à l'adresse: http://www.pharo.org/.Le plus simple restant d'exécuter le script de téléchargement qui suit dans un shell.

```
curl get.pharo.org/30+vmLatest | bash
```

ou si curl n'est pas disponible:

```
wget −O− get.pharo.org/30+vmLatest | bash
```

D'autres scripts de téléchargement sont disponibles à l'adresse http://get.pharo.org.

Une fois obtenue vous devez charger le package PharoSound en utilisant

- soit le configurationBrowser (comme le montre la Figure 1.1) qui se trouve dans le menu Tools. Entrer `Sound` dans la zone de texte pour sélectionner le package, puis presser le bouton 'Install Stable Version'.

- soit Gofer le gestionnaire de packages:

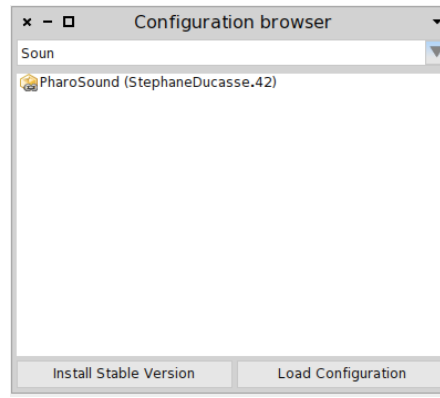

Figure 1.1: Configuration browser: un gestionnaire de packages.

```
Gofer new
   smalltalkhubUser: 'PharoExtras' project: 'Sound';
   package: 'ConfigurationOfPharoSound';
   load.
(#ConfigurationOfPharoSound asClass project version: #stable) load
```

1.1 Vérifications

Avant de se familiariser avec les différentes méthodes utiles, nous allons nous 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 en sélectionnant l'expression puis en utilisant l'item 'Do it' du menu. Nous vous expliquerons par la suite ce dont il s'agit. Notez que dans un workspace pour exécuter une ligne il suffit d'y positionner le curseur de la souris et de presser alt-d (cela equivaut à sélectionner la ligne et utiliser le menu). Pour sélectionner un bloc d'instructions rapidement placez la souris au tout début ou à la fin du block et cliquez, cela a pour effet de tout sélectionner.

```
FMSound organ1 play.
```

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

- Le volume sonore de votre OS n'est pas correctement ajusté ou en mode muet

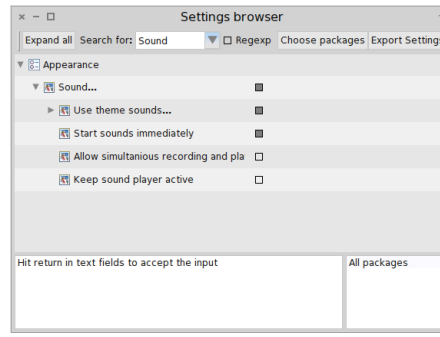

Figure 1.2: Settings.

- Vérifier que le son est permis dans votre image Pharo. Pour cela ouvrez le Setting Browser depuis le menu Tools, tapez sound dans la barre et cliquez sur le radio bouton. Vous devez obtenir un système proche de l'image montrée par la figure 1.2.

- Le plugin sonore n'est pas installé (Linux).

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

Aussi mais il faut que cela fonctionne je vais ouvrir un bug

1. Télécharger la dernière version de Squeak 4.4
2. Décompresser l'archive dans le répertoire de votre choix.
3. Copier le fichier se trouvant dans `Contents/Linux-i686/lib/squeak/4.4.7-2357/so.vm-sound-pulse` dans `bin/vm-sound-pulse`.
4. Changer les droits du plugins avec `chmod +x bin/vm-sound-pulse`.

est-ce que sous les autres OS le son est actif d'origine ?

cela depend de l'image et pas des OS. Maintenant le plugin sound sur linux n'est plus a jour et on va le fixer.

- Le process sonore interne ne fonctionne plus. Il faut l'arrêter puis le relancer en évaluant les expressions suivantes:

```
SoundPlayer stopPlayerProcess.
SoundPlayer startUp
```

1.2 Bases de la synthèse FM par l'exemple

Maintenant que nous nous sommes assurés que le son était fonctionnel, 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 duration: 1.0 loudness: 0.5.
      ^ snd
```

Etudions cette méthode:

Tout d'abord, elle crée une nouvelle instance de la classe `FMSound`.

```
sound := self new.
```

Elle définit ensuite un ensemble de points afin de créer une enveloppe de volume. Pour chaque point la première donnée exprime une durée en milliseconde, la seconde un volume exprimé de 0 à 1 (1 étant égal à 100% du volume).

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

Nous remarquons qu'une enveloppe se caractérise d'après un ensemble de points et d'une boucle. La méthode associée est `points:loopStart:loopEnd:`. Une boucle peut aussi bien utiliser l'ensemble des points définis ou 1 seul.

Par exemple ici on ajoute au son une enveloppe avec une boucle allant des indices 2 à 4 (l'indice 1 correspondant la première valeur du tableau).La séquence suivante est jouée pour des valeurs oscillant entre 0.8 et 1. Le volume du son semble vibrer.

```
60@1.0 . 110@0.8 . 200@1.0
```

Si nous avions voulu maintenir le volume à un certain niveau, nous aurions écris l'enveloppe comme ceci

```
sound addEnvelope: (VolumeEnvelope
```

```
        points: { 0@0 . 60@1.0 . 110@0.8 . 200@1.0 . 250@0.0}
        loopStart: 5 loopEnd: 5).
```

Le son se termine avec la valeur `250@0.0` et une boucle maintenant le volume à 0 pendant toute sa durée.

Finalement, la méthode retourne le son avec comme paramètre la note (ici exprimée en Hz, voir tableau **1.2**), 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 duration: 1.0 loudness: 0.5
```

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

1.3 Comment faire pour jouer notre propre son ?

Nous allons vous montrer comment nous allons créer notre propre son. Commençons par jouer un `LA` de la manière suivante:

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

Le son est joué à une fréquence de 440 Hz pendant 1 seconde avec un volume maximum.

La classe `FMSound` permet la génération de sons sinusoïdaux. Ça donne un son assez doux et régulier. La figure 1.3 ci dessous montre à quoi ressemble la courbe.

utile ou pas ? L'écriture pour une seule note est simple, mais cela peut rapidement devenir long, fastidieux et les erreurs difficiles à détecter si nous devons écrire tout une mélodie de cette manière.

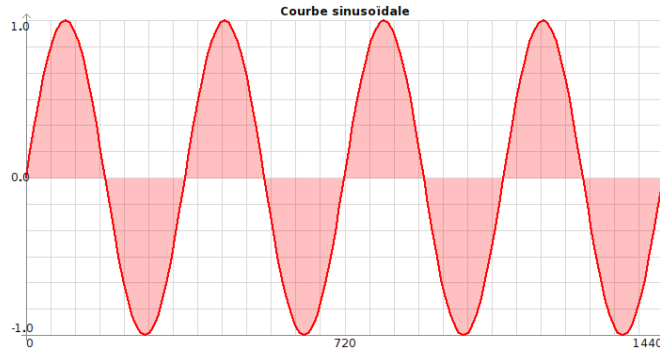

Figure 1.3: Sinusoïde produite.

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 **1.3** va nous indiquer les correspondances pour plus de facilité. Par exemple, le LA est représenté dans la notation anglaise par un 'a'. Pour obtenir le LA de la quatrième octave nous demandons a4.

Le `LA` de tout à l'heure peut s'écrire maintenant sous cette forme

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

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

Le `SOL#` par exemple s'écrit

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

encore une fois passer une chaine ou un nombre est horrible. cela serait bien de faire le menage dans ce code. - XM: Tu verrais quoi à la place ? deux interfaces differentes et commune qui accepte les deux. setPitchForFrequence: aNumber setPitchForNote: setPitch

1.4 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 classe nommée `SequentialSound`. Comme son nom l'indique nous allons pouvoir jouer les sons de manière séquentielle.

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

```
seq add: (FMSound new setPitch: 'a3' duration: 0.5 loudness: 1).
seq add: (FMSound new setPitch: 'g#4' duration:0.5 loudness: 1).
seq play.
```

Si nous écoutons les sons générés, nous pouvons percevoir un petit clap entre chaques transitions de notes. Afin d'éviter cela nous allons appliquer une enveloppe de volume sur chacune des notes. Ceci est fait comme suit:

```
points := { 0@0 . 10@1 . 400@0.4 . 500@0}.
```

La note jouée est à son maximum au début puis repasse progressivement à 0 au bout de 500ms. L'enveloppe étant définie sur une durée égale à la durée de la note la transition se fait plus naturellement et sonne mieux à l'oreille.

enveloppe qui fait quoi? est ce correct ce que j'ai ecrit et pourquoi 0 pour le premier? Oui ca me paraît correcte. Le zéro correspond au début du son. On est alors au maximum du volume. Par contre je n'ai jamais fais le test de ne pas définir le 0. Peut être est on déjà à 1 ? a creuser.

Voyons ce que donne notre code précédent avec une enveloppe de volume

```
| seq snd1 snd2 snd3 points |
"Declaration des sons"
seq := SequentialSound new.
snd1 := FMSound new.
snd2 := FMSound new.
snd3 := FMSound new.

"Creation du volume pour les instruments"
points := { 0@0 . 10@1 . 400@0.4 . 500@0}.

"Generation des notes"
snd1 setPitch: 'g#4' duration:0.5 loudness: 1.
snd2 setPitch: 'a3' duration:0.5 loudness: 1.
snd3 setPitch: 'g#4' duration:0.5 loudness: 1.

"Application de l'enveloppe aux sons"
snd1 addEnvelope: (VolumeEnvelope points: points loopStart: 4  loopEnd: 4).
snd2 addEnvelope: (VolumeEnvelope points: points loopStart: 4  loopEnd: 4).
snd3 addEnvelope: (VolumeEnvelope points: points loopStart: 4  loopEnd: 4).

"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 maintenant plus agréable à écouter.

1.5 Jouer plusieurs sons simultanément

Jouer des sons en séquence c'est bien mais pas encore suffisant pour donner une âme à notre production. Pour cela nous allons utiliser La classe `MixedSound`. Elle va litéralement mixer plusieurs sons en même temps.

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

"Creation du volume pour les instruments"
points := { 0@0 . 10@1 . 400@0.4 . 500@0}.

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

"Application de l'enveloppe aux sons"
snd1 addEnvelope: (VolumeEnvelope points: points loopStart: 4  loopEnd: 4).
snd2 addEnvelope: (VolumeEnvelope points: points loopStart: 4  loopEnd: 4).

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

mix play.
```

c'est bien la methode add:pan:?

Modifions un peu les valeurs de l'attaque à l'aide de la méthode `setPitch:duration:loudness:` pour obtenir des notes différentes et nous familiariser avec les méthodes. Les enveloppes de volume modifient la courbe du son au cours du temps et peuvent être utilisées pour obtenir des effets particuliers.

Par exemple si nous voulions rajouter un effet d'écho à notre son il faudrait modifier les points de l'enveloppe de volume par

```
points := { 0@1 . 100@0.6  . 200@0 . 300@0.2 . 400@0.1 . 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).
```

Dans notre exemple nous avons 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.

Nous allons réutiliser l'exemple plus haut et lui rajouter de la stéréo.

```
| mix seq1 seq2 seq3 snd1 snd2 snd3 points rest rest2 |
"Declaration des sons"
mix := MixedSound new.
rest := RestSound new.
rest2 := RestSound new.
seq1 := SequentialSound new.
seq2 := SequentialSound new.
seq3 := SequentialSound new.
snd1 := FMSound new.
snd2 := FMSound new.
snd3 := FMSound new.

"Creation du volume pour les instruments"
points := { 0@1 . 400@0.4 . 500@0}.

"Generation des notes"
snd1 setPitch: 'g#4' duration:0.5 loudness: 1.
snd2 setPitch: 'a3' duration:0.5 loudness: 1.
snd3 setPitch: 'g#4' duration:0.5 loudness: 1.
rest duration: 0.5.
rest2 duration: 0.5.

"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"
seq1 add: snd1.
seq2 add: rest; add: snd2.
seq3 add: rest; add: rest2; add: snd3.

mix add: seq1 pan:0.
mix add: seq2 pan:0.5.
mix add: seq3 pan:1.
"On joue le tout"
mix play.
```

Dans ce nouvel exemple nous avons introduit une nouvelle classe de son `RestSound` pour créer uun silence.Elle ne possède qu'un seul message `duration:` exprimé en seconde générant un son plus ou moins long.

La méthode `SequentialSound` possède une limitation. Regardons l'exemple suivant pour voir ce qu'il se passe

```
| seq snd1 snd2 snd3 |
"Declaration des sons"
seq := SequentialSound new.
snd1 := FMSound new.
snd2 := FMSound new.
snd3 := FMSound new.

"Generation des notes"
snd1 setPitch: 'g#4' duration:0.5 loudness: 1.
snd2 setPitch: 'a3' duration:0.5 loudness: 1.
snd3 setPitch: 'g#4' duration:0.5 loudness: 1.

"Les sons les uns a la suite des autres"
"Le rÃl'sultat ne va pas etre celui attendu"
seq add: snd1.
seq add: snd2.
seq add: snd3.
seq add: snd2.

"On joue le tout"
seq play.
```

Normalement nous devrions entendre 4 sons joués à la suite, or il n'y en a que 3. Visiblement nous ne pouvons pas réutiliser un son déjà déclaré. Cette méthode n'est pas adaptée pour ce que nous souhaitons faire. Pour cela il existe une autre classe qui va répondre à notre besoin: `QueueSound`

Remplaçons la ligne `seq := SequentialSound` par `seq := QueueSound` et ecoutons de nouveau notre exemple. Nous entendons bien nos 4 sons. Le tour est joué !

1.6 Les enveloppes

Nous avons introduit une première enveloppe qui agit sur le volume d'un son. Il y en a une autre qui change la note ou plutôt sa fréquence pendant qu'elle est jouée, c'est ce qu'on appelle l'attaque de la note, le pitch en anglais.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 sera haute. Si on indique un chiffre négatif, la note descendra en dessous de la note de départ. Nous définissons donc une enveloppe mais cette fois-ci une enveloppe d'attaque en utilisant la classe `PitchEnvelope`.

```
| snd p |
p := { 0@0 . 100@0.1 . 200@0.2 . 300@(−0.5) . 400@0.7 . 800@0.8 . 1000@1 }.
snd := FMSound new.
```

```
snd addEnvelope: (PitchEnvelope points: p loopStart: 7  loopEnd:7).
(snd setPitch: 'a4' duration: 2 loudness: 0.5) play.
```

La transition entre les différentes fréquences se fait en douceur et sans cassures. Nous allons rajouter à cela une enveloppe de volume pour produire un son plus intéressant.

```
| snd p p2 |
p := { 0@0 . 100@0.1 . 200@0.2 . 300@(−0.5) . 400@0.7 . 800@0.8 . 1000@1 }.
p2 := { 0@0 . 100@0.1 . 200@0.2 . 300@(−0.5) . 400@0.7 . 800@0.8 . 1000@0 }.
snd := FMSound new.
snd addEnvelope: (PitchEnvelope points: p loopStart: 7  loopEnd: 7).
snd addEnvelope: (VolumeEnvelope points: p2 loopStart: 7  loopEnd: 7).
(snd setPitch: 'a4' duration: 2 loudness: 0.5) play.
```

Le volume est dans notre exemple très variable, l'ensemble des points ont été définis manuellement. Imaginons maintenant que nous voulions le réduire progressivement jusqu'à 0. Nous aurions à définir un certains nombre de points afin pour produire une courbe décroissante. C'est faisable, fastidieux et surtout la courbe ne produirait pas l'effet souhaité.Au lieu de la définir points par points nous allons utiliser une méthode appelée `exponentialDecay:` de la classe `VolumeEnvelope` pour le volume ou la classe `PitchEnvelope` pour le pitch. La courbe sera de type exponentielle partant du volume le plus élevé 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.
```

Nous obtenons le résultat suivant

```
{(0@0.0). (10@1.0). (20@0.2). (30@0.04000000000000001). (50@0.0)}
```

Plus la valeur est grande plus le nombre de valeurs sera important.

Nous pouvons visualiser à quoi ressemble notre envelope en ouvrant le morph `EnvelopeEditorMorph` comme ceci

```
EnvelopeEditorMorph new openInWorld
```

Sélectionnez l'ensemble de la ligne et faites **clic droit**/**Do it**. Une fenêtre apparaît comme celle présentée en figure 1.4.

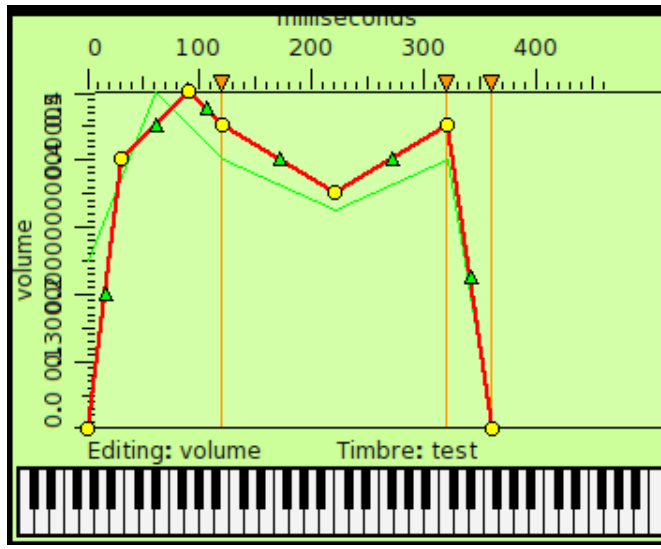

Figure 1.4: Editeur d'enveloppe.

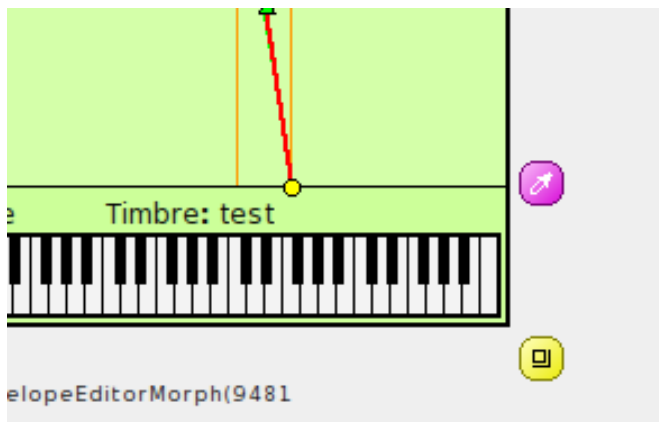

Figure 1.5: Ancre.

Vous pouvez l'agrandir en la sélectionnant avec **shit + clic droit** puis en déplaçant l'ancre en bas à droite de la fenêtre comme sur la figure 1.5

Le morph expose les courbes de l'instrument et nous permet de les modifier en déplaçant les points le composant. Vous pouvez passer d'une courbe à l'autre soit en la sélectionnant directement dans la fenêtre, soit en la choisissant dans le menu `editing` en bas à gauche du morph.

En abscisse nous avons le temps en milisecondes. L'ordonnée se modifie en fonction de la courbe choisie. Nous avons également 3 délimitations représentées par les axes verticaux de couleur orange. Les deux plus à gauche sont utilisées pour la boucle lorsque la note est tenue, ce sont les paramètres `loopStart:loopEnd:`. La plus à droite c'est lorsque la note est relachée. Nous pouvons également les déplacer en fonction de nos besoins en utilisant les triangles orange au dessus de la courbe des abscisses. A noter qu'il n'est pas possible d'avoir sur le morph une boucle d'une durée inférieur à 50ms, comme il n'est pas possible d'avoir une valeur de début de boucle différente de la valeur de fin. Ce qui parraît logique si nous souhaitons obtenir une boucle harmonieuse.

Pour rajouter un point sur une partie de la courbe il suffit de cliquer sur les petits triangles verts pour faire apparaître le point correspondant. Ce point peut être ensuite déplacer pour être positionné à la valeur souhaitée.

peut on supprimer un point une fois qu'il a été créé ? en même temps vu que le morph est là uniquement pour tester, est ce vraiment utile ?

Je vous invite à tester toutes ces possibilités et à écouter le résultat obtenu grace au clavier de piano disponible.

Il faudra rajouter dans le morph d'autres timbres ou du moins faire en sorte qu'il n'y ait pas d'erreur si on veut en sélectionner un autre.

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 joués en même temps.

Nous avons entre nos mains pratiquement tous les outils pour écrire notre première mélodie, que disons-nous, notre première super production !

1.7 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 constituée d'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 d'après sa fréquence 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 être ammené à saturer mais cela peut se révéler utile dans certains cas pour donner un effet.

Voici un exemple de déclaration d'une séquence

```
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 aiment le début de l'air des jeux vidéos. Nous allons définir la mélodie d'introduction du jeu pacman.

Définissons un nouvel élément

Commençons par définir un nouvel instrument. Afin de ne pas compliquer notre exercice, nous allons utiliser la classe `FMSound` dans laquelle nous allons définir quelques nouvelles méthodes. Notez que pour être plus propre et modulaire, nous devrions définir notre propre package et classes.

Pour définir une méthode nous allons utiliser un navigateur de classe: Pour cela

- ouvrez le SystemBrowser depuis le menu Tools
- trouvez la classe `FMSound` en utilisant l'item Find Class du menu de la liste la plus à gauche (la liste des packages)

Vous devez obtenir un navigateur comme celui de la figure 1.6

Comme les méthodes que nous définissons sont des méthodes qui créent des instances de `FMSound` par opposition à des méthodes qui agissent sur des instances de la classe `FMSound` nous allons les définir sur le coté classe de la classe `FMSound`. Pour cela:

- cliquez sur le radio bouton 'Class side'
- à l'aide du menu de la troisième liste en partant de la gauche (liste des protocoles de méthodes - il s'agit de simple classeurs de rangement), ajoutez un protocole nommé par exemple 'new instruments'.
- sélectionner le nouveau protocole puis ajouter la méthode `fm1` suivante. Pour ajouter une méthode, tapez le corps de la méthode puis utiliser l'entrée 'Accept' qui compile la méthode.

```
fm1
   | snd |
   snd := self new.
```

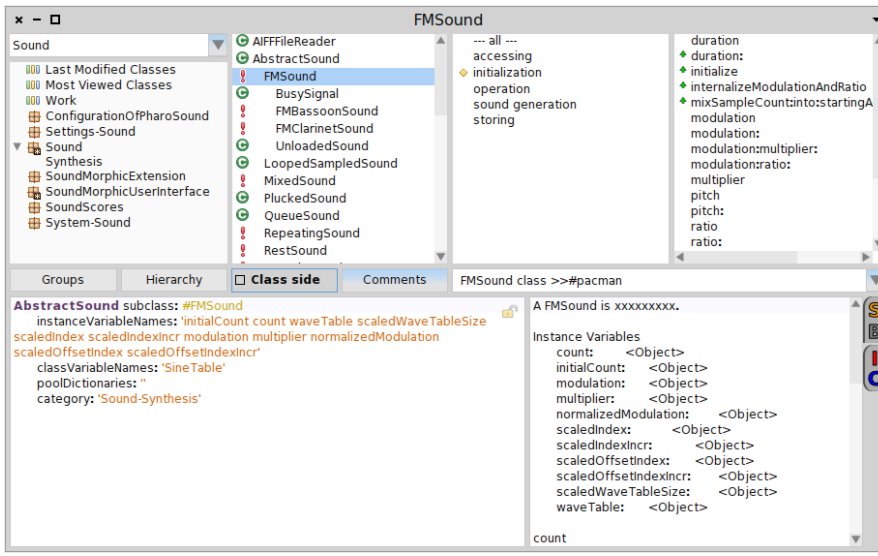

Figure 1.6: System browser: un navigateur de classes.

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

Vous devez obtenir une situation similaire à celle présentée par la figure 1.7

Vérifiez que votre nouvel instrument fonctionne en executant `FMSound fm1 play`.

Nous allons définir un autre instrument en copiant celui-ci afin de vous permettre d'expériementer.

```
fm2
    | snd |
    snd := self new.
    snd addEnvelope: (VolumeEnvelope points: { 0@1 . 300@0 }
                                loopStart: 2
                                loopEnd: 2).
    snd modulation: 1 ratio: 2.
    ^ snd
```

Déclarons maintenant, la méthode permettant de mixer l'ensemble des pistes qui vont être créées à l'aide des méthodes `pacmanV10:` et

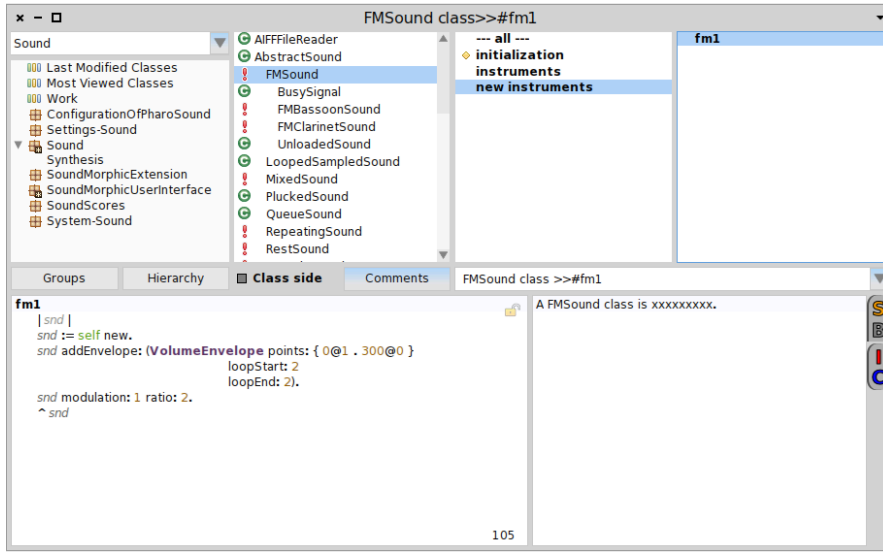

Figure 1.7: System browser sur une méthode de classe.

`pacmanV20:.`

```
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 deux 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 `noteSequenceOn:from:` va générer une séquence de notes (par exemple SequentialSound) à partir d'un tableau de notes pour un instrument donné. Elle va utiliser comme vu précédemment le triplet note, durée, volume.

```
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.
```

Si votre mélodie utilise des notes à fréquence basse, le volume de celle ci sera difficilement audible. Vous pouvez alors pour y remédier augementer le volume des notes. Ici nous sommes passé à un volume de 500, contrairement à la précédente mélodie où nous étions resté à 150.

Voila il ne reste plus qu'à exécuter la mélodie: `FMSound pacman play`.

1.8 Jouer un sample

rajouter le téléchargement des fichiers depuis GIT ?

La sonorisation d'une production ne passe pas essentiellement par de la synthèse sonore, parfois il est nécessaire de faire appel à des sons digitalisés reproduisant des bruitages, des voix ou des musiques entières. Nous allons vous montrer comment jouer un sample à partir d'un fichier ou directement depuis la mémoire.

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 en tâche de fond sans bloquer le reste de notre environnement. Nous pouvons par exemple utiliser cette méthode pour sonoriser une action ponctuelle ou une alerte.

Si notre but était de concevoir un jeu, le chargement du son directement depuis le disque à des moments clés risquerait de ralentir les animations voir de le rendre injouable. Afin d'éviter ces petits désagréments les sons seront chargés en mémoire et joués lorsque ce sera nécessaire pour gagner en temps de réponse.

Pharo met à notre disposition une banque de sons que nous pouvons alimenter suivant nos besoins. Pour rajouter un son à cette librairie nous utiliserons le message `addLibrarySoundNamed:samples:samplingRate:`.

on devrait ajouter une methode `addLibrarySoundNamed:` qui appelle addLibrary avec samples et originalSamplingRate

à corriger après la revue

rajouter un exemple pour lister le contenu de la librairie si on peut ... mais oui on peut :)

Dans un workspace copiez le code suivant et évaluez-le pour rajouter le sample dans la librairie

```
| 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.
```

Maintenant nous pouvons jouer le sample à la fréquence et à la durée souhaitée

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

estil possible de le jouer a la vitesse ou il a ete enregistre? il me semble que oui avec originalSamplingRate. Il faut que je fasse le test sur un sample plus significatif.

1.9 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.
```

on doit ameliorer fromWaveFileNamed: pour lui passer un fileSystem FileSystem disc workingDirectory / 'ocean.wav'

Maintenant que tous nos sons sont chargés nous allons les jouer en séquence grâce à la classe `QueueSound` vue précédemment. Ils sont joués les uns après les autres à la fréquence souhaitée.

```
| 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.
```

1.10 Conclusion

Nous vous avons montré une petite partie des possibilités que Pharo offre pour la création et manipulations des sons. Nous espérons vous avoir donner envie d'experimenter.