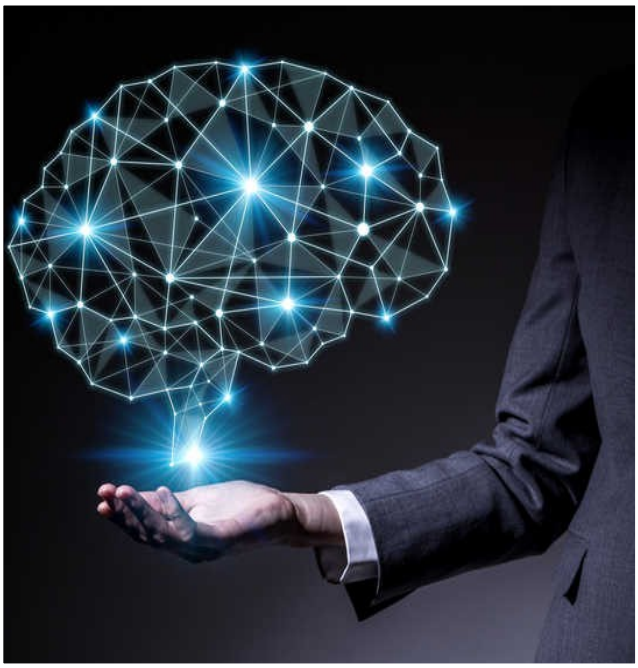
Mykhaylo DUDYAK
Narek ARAKELYAN
Hanna PODZHAROVA
Steven NJIKE



PROJET DEEP LEARNING

Création d'un jeu vidéo basé sur la reconnaissance vocale

Année universitaire 2018/2019

Institut du Risque & de l'Assurance
Le Mans Université

Sommaire

Remerciements

Nous remercions toutes les personnes qui ont prêté leur voix pour la réalisation de ce projet ; nos familles, amis et camarades, de la France au Cameroun en passant par l'Arménie, l'Ukraine, la Chine, le Nigéria et le Canada.

Introduction

Le Deep Learning ou apprentissage profond, est une dérivée du Machine Learning.

Le Machine Learning ou apprentissage automatique ou apprentissage statistique, est un sous ensemble de l'Intelligence artificielle. Il est défini en 1959 par Arthur Samuel, l'un des pionniers dans ce domaine, comme « le champ d'étude qui donne aux ordinateurs la capacité d'apprendre sans être explicitement programmés à apprendre ». L'objectif est de développer un moteur de prédictions à partir de données. Les différentes techniques de Machine Learning ont ainsi été développées pour créer des algorithmes capables d'apprendre et de s'améliorer de manière autonome.

Le Deep Learning constitue une classe d'algorithmes d'apprentissage automatique qui utilisent différentes couches d'unité de traitement non linéaires pour l'extraction et la transformation des caractéristiques. Ils fonctionnent avec un apprentissage à plusieurs niveaux de détails ou de représentations de données.

Pour la réalisation de ce projet nous cherchons à résoudre un problème de classification multiple de fichiers audio répartis dans 8 classes prédéfinies :

- Bonjour,
- Au revoir,
- A gauche,
- A droite,
- En haut,
- En bas,
- Oui,
- Non.

Une compétition de ce type, existe déjà sur Kaggle.com (https://www.kaggle.com/c/tensorflow-speech-recognition-challenge). Cependant, cette dernière est basée sur des commandes vocales en anglais, alors que nous avons choisi de baser notre reconnaissance vocale sur des mots en français. Ainsi, n'ayant pas pu trouver une base de données de mots en français, nous avons décidé de créer notre propre base de données.

Nous nous sommes ensuite posés plusieurs questions :

- Comment obtenir les mots souhaités pour constituer notre base de données ?

- Comment faire si notre base de données se révèle insuffisante ? Comment enrichir notre base de données ?
- Quel modèle choisir pour notre programme ?
- Le modèle choisi est-il suffisamment généralisé ?
- Comment incorporer notre modèle dans un jeu ?

Ce projet repose sur une base de fichiers audio en format Wavefront(WAV) répartis en 8 classes. Chacune de ces classes est constituée d'environ 130 fichiers.

Afin de détailler la réalisation de notre projet, nous allons tout d'abord revenir sur la constitution de notre base de données. Nous présenterons ensuite les raisons qui nous ont conduit à choisir le modèle sélectionné. Nous décrirons également ce modèle ainsi que ses performances. Aussi, nous nous arrêterons sur l'utilisation de ce modèle pour la création d'un jeu vidéo. Enfin, nous aborderons les améliorations envisagées pour notre modèle.

I. Constitution de la base de données

1.1. Collecte et traitement des données

Après plusieurs recherches de données pour la classification de mots français uniques, nous avons décidé de construire notre propre base de données.

Ainsi, nous avons récolté des messages vocaux, majoritairement via Messenger et WhatsApp, en sollicitant notre entourage. Au total, plus de 40 personnes ont répondus à notre demande. Chacune d'entre elle a répété en moyenne 3 fois les mots représentant les classes définies plus haut.

Nous avons ensuite traité nos fichiers avec le logiciel libre Audacity. Dans un premier temps, nous avons découpé les fichiers afin d'obtenir un seul mot par fichier. Dans un second temps nous avons procédé au nettoyage des fichiers, en utilisant l'effet « Noise Reduction », qui permet de retirer les bruits de fond. Enfin, nous avons converti les divers formats de ces fichiers en AAC, MP4, M4A, OGG en WAV.

1.2. Enrichissement de la base de données

Contraints par le temps et dans l'incapacité de collecter des milliers d'enregistrements, nous avons fait le choix d'enrichir la base de données en ajoutant des transformations des enregistrements audio à notre disposition.

Lors de nos recherches nous sommes tombés sur un article nommé « Audio Data Augmentation », écrit par Ali Buğra Kanburoğlu, où les trois méthodes suivantes sont mentionnées :

- Ajout du bruit blanc : Cette méthode ne saurait nous être très utile car les données seront, soit trop corrélées, soit nous perdrons en qualité en ajoutant un bruit trop important. De plus, nous prenons le risque que le modèle considère la présence du bruit comme un facteur pertinent pour la classification. Nous pourrions également rencontrer des problèmes pour les petites parties de phrase d'une moindre amplitude que le reste (à titre d'exemple le son du « t » dans « à droite » est souvent de faible intensité et risque d'être masqué par le bruit).

- Décalage de l'audio : Cette méthode n'a aucun intérêt ici car nous coupons le silence en début et en fin de phrase.

- Élongation de l'audio : Cette troisième méthode est intéressante, mais elle nécessite quelques améliorations. Etant donné que nous normalisons la durée de l'enregistrement audio, la méthode n'est pas très pertinente en soi.

  En revanche on pourrait utiliser cette approche pour moduler la vitesse de la voix sur certaines parties de l'audio tout en gardant la même durée d'enregistrement.

C'est ainsi que nous avons décidé de créer un algorithme de redistribution de la vitesse du son.

Nous sommes partis de l'idée simple que la durée de l'audio c'est sa vitesse multipliée par le temps :

$$D = V \cdot T$$

Si on considère que la vitesse dépend du temps, nous obtenons :

$$D = \int_0^T V\,(t)dt$$

Normalement la vitesse est constante, $V(t) = V_0$, mais nous allons introduire une fonction de perturbation $p(t)$ pour la modifier :

$$V(t) = V_0 + p(t)$$

$$D = \int_0^T V\,(t)dt = \int_0^T \big(V_0 + p(t)\big)\,dt = V_0 \cdot T + \int_0^T p\,(t)dt = D + \int_0^T p\,(t)dt$$

Pour introduire les perturbations nous avons besoin de générer des fonctions intégrables $p(t)$ pour lesquelles on a :

$$\int_0^T p\,(t)dt = 0$$

Cependant, les enregistrements audio ne sont pas continus et nous ne disposons d'aucune fonction qui nous permette de modifier la durée d'un audio par l'intermédiaire d'un facteur constant.

Nous avons donc besoin de discrétiser la fonction $p(t)$ (dans la suite on va considérer l'intervalle $[0,1]$ sans perte de généralité, puisqu'on peut toujours mettre la fonction à l'échelle nécessaire) :

$$\sum_{i=1}^{n} \int_{i-1}^{i} p\left(\frac{x}{n}\right) dx = 0$$

d'où les coefficients d'échelle :

$$p_i = \int_{i-1}^{i} p\left(\frac{x}{n}\right) dx$$

Ici, nous rencontrons la première difficulté. En effet, il n'est pas évident de trouver des fonctions dont l'intégrale est égale à zéro. Heureusement, cela est facilement généralisable :

Considérons par exemple une fonction $g(x)$ intégrable, pour obtenir une fonction dont l'intégrale est nulle il suffit d'enlever la valeur de l'intégrale de cette même fonction :

$$p(x) = g(x) - \int_0^1 g\,(t)dt$$

$$p_i = \int_{i-1}^{i} \left(g\left(\frac{x}{n}\right) - \int_0^1 g\,(t)dt \right) dx = \int_{i-1}^{i} g\left(\frac{x}{n}\right) dx - \int_0^1 g\,(x)dx$$

$$= \int_0^1 \left(g\left(\frac{i-1+x}{n}\right) - g(x) \right) dx$$

Adaptons l'indice pour Python :

$$p[j] = \int_0^1 \left(g\left(\frac{j+x}{n}\right) - g(x) \right) dx, \forall j = \overline{0, n-1}$$

Nous rencontrons une deuxième difficulté. Afin d'éviter d'utiliser des intégrales dans le code Python, nous allons considérer qu'on connaît la primitive de g :

$$\int g\,(x)dx = G(x) + C$$

Par la formule de Newton-Leibnitz on obtient :

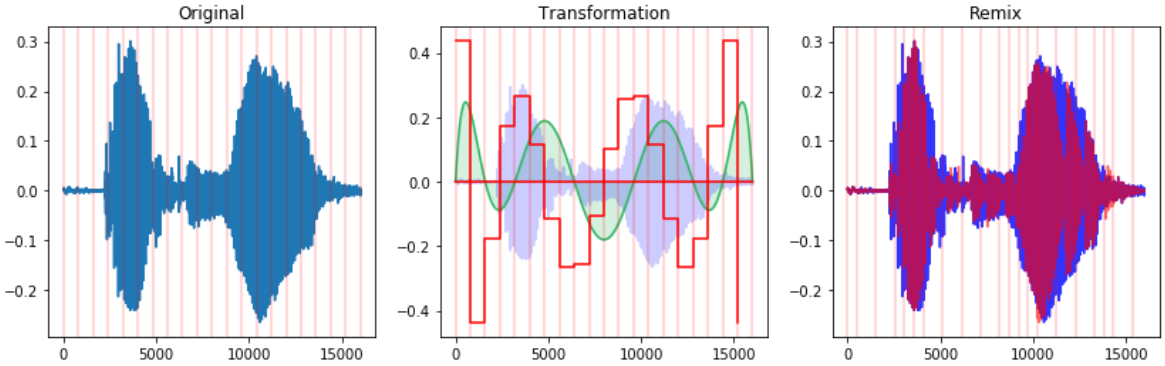
$$p[j] = n\left(G\left(\frac{j+1}{n}\right) - G\left(\frac{j}{n}\right) \right) - G(1) + G(0)$$

On se retrouve ainsi avec un problème beaucoup moins complexe : la génération de fonctions G dérivables quelconques.

La solution la plus évidente est la génération de polynômes à partir de leurs racines en utilisant la bibliothèque ***numpy***.

Les racines des polynômes sont distribuées aléatoirement sur $[0,1]$.

Et voici le résultat (la courbe verte est le polynôme $G(x)$ et la courbe rouge représente les $p[j]$) :

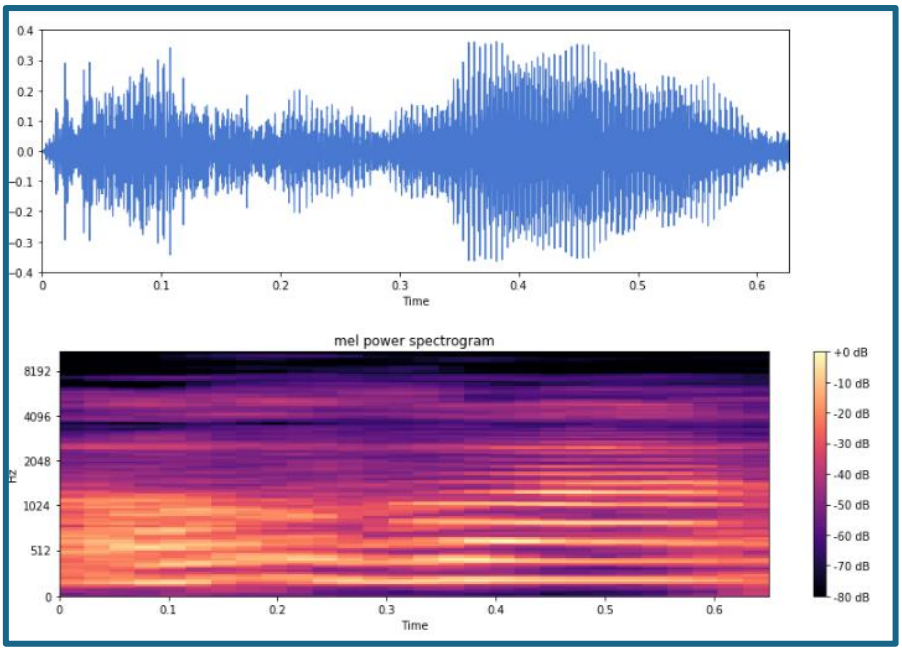


- Remarque : par symétrie en générant un polynôme on peut subtilement en obtenir quatre puisqu'on peut également utiliser $-G(x)$, $G(1-x)$ et $-G(1-x)$.

II. Choix du modèle

Afin de résoudre le problème de classification, nous nous inspirons de l'article intitulé « Building a Dead Simple Speech Recognition Engine using ConvNet in Keras » (https://blog.manash.me/building-a-dead-simple-word-recognition-engine-using-convnet-in-keras-25e72c19c12b), en l'adaptant à nos besoins.

Ici, nous cherchons à convertir des fichiers audio en leur représentation MFCC (Mel-frequency cepstral coefficients) qui est une image.

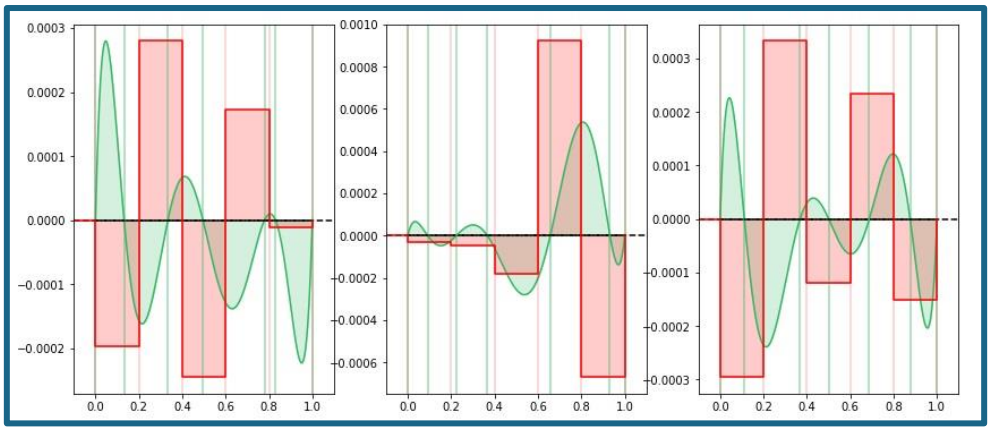


Ainsi, nous nous retrouvons avec un problème plus simple et beaucoup plus étudié : la reconnaissance d'images. Nous trouvons de nombreux modèles déjà faits et nous choisissons les CNN comme une solution fiable et optimale utilisée dans plusieurs domaines de recherche.

Après avoir testé le modèle avec les données augmentées d'une manière excessive (3000+ fichiers par classe) nous avons rencontré un problème de surapprentissage.

En effet, lorsque nous effectuons beaucoup de modifications d'un seul fichier, nous obtenons de fortes corrélations entre les fichiers qui apparaissent dans la base d'entrainement et la base de test. Or, ceci peut nous amener à des conclusions erronées sur la performance du modèle. Nous prendrions ainsi le risque d'avoir un score élevé pour la base d'apprentissage et la base de validation, alors que le modèle n'est en réalité pas généralisable.

Pour pallier à ce risque, nous avons essayé de contrôler au maximum la quantité des polynômes que nous utilisons pour l'augmentation de la base ainsi que leur hétérogénéité. Ainsi, nous nous sommes arrêté à 3 polynômes, ce qui aboutit à la multiplication de la base de données par 12.



Nous obtenons alors un modèle final relativement généralisable et pertinent pour le contrôle du personnage de notre jeu avec par la voix.

III. Description et performance du modèle

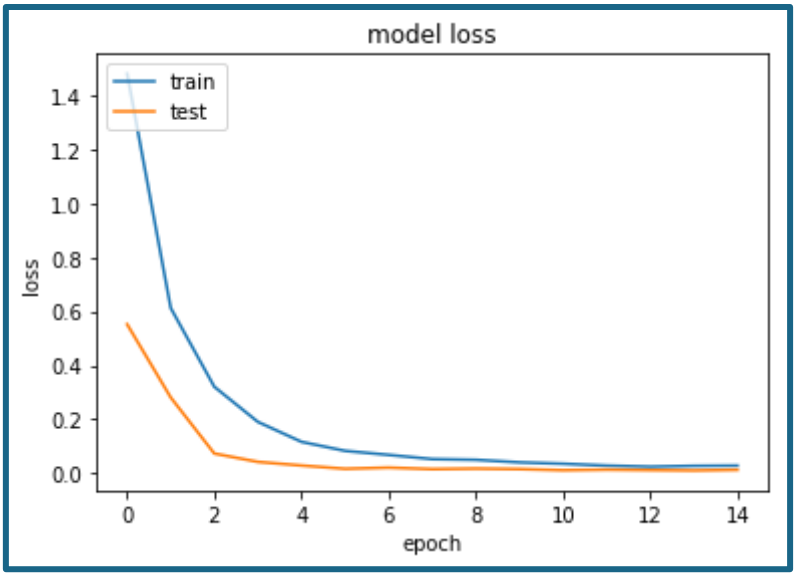
Après avoir testé plusieurs architectures différentes, nous avons sélectionné la plus optimale et lui avons apporté quelques modifications.

Nous avons ainsi choisi d'utiliser le modèle avec 3 couches de convolution qui permettent de diminuer la dimensionnalité; une couche de MaxPooling et 3 couches Fully-Connected suivies par Dropout pour éviter le surapprentissage.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 19, 10, 32)	160
conv2d_5 (Conv2D)	(None, 18, 9, 48)	6192
conv2d_6 (Conv2D)	(None, 17, 8, 120)	23160
max_pooling2d_2 (MaxPooling2	(None, 8, 4, 120)	0
dropout_4 (Dropout)	(None, 8, 4, 120)	0
flatten_2 (Flatten)	(None, 3840)	0
dense_4 (Dense)	(None, 128)	491648
dropout_5 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 64)	8256
dropout_6 (Dropout)	(None, 64)	0
dense_6 (Dense)	(None, 8)	520

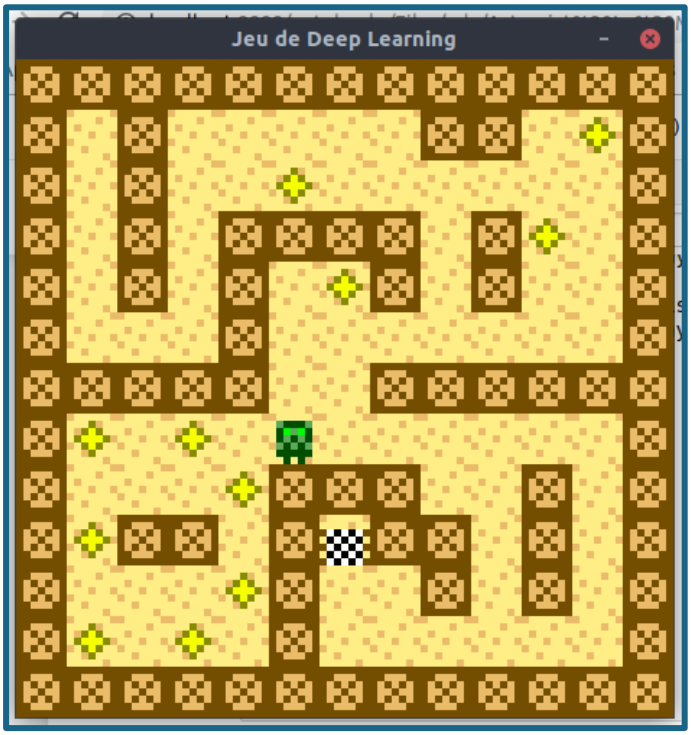
Total params: 529,936
Trainable params: 529,936
Non-trainable params: 0

Quant à la qualité de notre modèle, l'accuracy ou encore le taux de succès, s'élève à 99,62 %. Autrement dit sur 100 essais, le modèle saura prédire le mot correctement 99,62 fois.



IV. Utilisation du modèle : création d'un jeu vidéo

En vue d'exploiter notre modèle, nous avons créé un petit jeu vidéo, dans lequel nous avons un personnage dans un labyrinthe, qui se déplace pour collecter les pièces.



Le contrôle de ce personnage se fait avec la voix. Cependant, nous n'avons pas eu le temps nécessaire pour implémenter la reconnaissance vocale en mode streaming. Nous avons donc réalisé un simple Proof-of-Concept, dans lequel nous prenons des enregistrements aléatoires de la base de données qui correspondent aux flèches pressées. Une fois la direction sélectionnée sur le clavier, le jeu choisit le déplacement du caractère à partir de la prédiction faite par le modèle pour le fichier choisi.

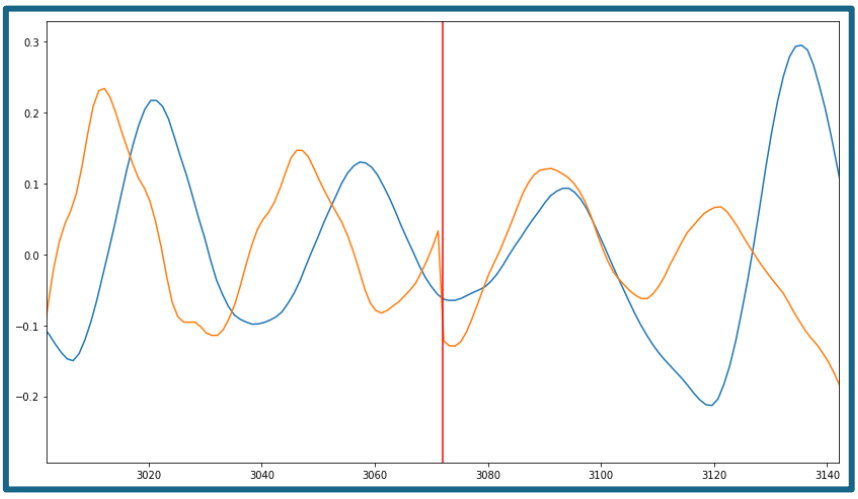
Par exemple si le modèle se trompe et prédit la phrase « à gauche » alors que nous avons pressé la flèche qui indique la direction de droite, le personnage obéira au modèle mais non à la touche que l'utilisateur aura pressée.

En revanche, si la prédiction est bonne, elle correspondra donc bien à la direction pressée.

V. Améliorations à apporter

Nous relevons plusieurs pistes d'amélioration pour notre modèle :

- La qualité du son après la transformation. En effet, dans le processus de répartition de la vitesse des audio, ***librosa*** passe par la FFT. Nous avons alors des "chirps" (voir le saut de la courbe orange sur l'image ci-dessous) aux lieux de collage des morceaux que nous pourrions éviter en faisant une interpolation linéaire ou bien un analogue de l'effet "Repair" de Audacity.



- La quantité de données. Il est nécessaire de remplir la base de données avec beaucoup plus de voix et avec des voix hétérogènes afin que le modèle se généralise mieux.

- Retirer le downsampling dans l'étape de traitement des audio pour pouvoir reconnaître plus de phrases dans le futur.

- Ajouter des "sanity checks" pour ne prendre en compte que des probabilités prédites, pour les réponses qui sont pertinentes en fonction de l’avancée du personnage dans le jeu (par exemple, ne pas regarder les "oui", "non" lorsque nous attendons des directions de déplacement).

Conclusion

En conclusion, nous pouvons dire que notre modèle fonctionne convenablement. En effet, grâce la création de notre jeu vidéo, nous avons pu nous assurer que ce dernier était exploitable.

Nous soulignons cependant que l'enrichissement de la base de données est indispensable en vue d'une utilisation optimale de notre modèle.

Quant à notre jeu vidéo, ce dernier reste inachevé tout comme la reconnaissance d'un flux audio. C'est pourquoi nous avons réalisé un simple Proof-of-Concept

Questions complémentaires

- Prolongements / perspectives dans un environnement industriel ?

Nous avons créé un jeux vidéo pour lequel le personnage se déplace par commandes vocales, en français. Cependant, en ajoutant des commandes vocales supplémentaires, notre modèle pourrait être utilisé pour commander un fauteuil roulant, dans le cas d'une personne en situation de handicap qui ne pourrait user que de la parole pour commander sa direction.

- Le modèle est-il exploitable avec ses performances ?

En augmentant la base de données, le modèle aura besoin de plus de temps pour apprendre. Cependant, une fois le modèle entraîné, il pourra faire des prédictions très rapidement.

- Est-il exploitable seul ou associé à autre chose ?

Notre modèle n'est exploitable que s'il est associé à autre chose, en l'occurrence ici nous l'utilisons dans un jeu vidéo. En effet, la classification des commandes vocales seule, ne sert à rien.

- Quelle est sa durée de vie ? Ses performances vont-elles se dégrader dans le temps ? Peut-on le détecter ?

Notre modèle n'est pas limité dans le temps. Ses performances ne sont pas amenées à se dégrader.

- Le modèle peut-il passer à l'échelle et être entraîné sur des jeux 10, 100, 1000 fois plus grand ?

Si le nombre de données devient trop important pour notre modèle, nous pouvons diminuer le niveau dropout. Cela permettra d'activer de nouveaux neurones et donc d'exploiter encore plus de données.

- Doit-il être maintenu, rafraîchi, à quel coût ?

Notre modèle ne nécessite pas de maintenance, ni de rafraîchissement. Il continuera à être exploitable en l'état.

- Lorsque l’apprentissage est long et coûteux, l’estimation d’un nouveau modèle pourrait-elle être faite à partir de la précédente ou non ? (Nouvel apprentissage ou simple mise à jour)

Nous pourrions par exemple fixer les premières couches du modèle et effectuer un « transfert-learning » avec une nouvelle base de données en réactualisant les dernières couches du modèle.

- Si vous deviez vendre ce modèle de deep learning, quel modèle économique choisiriez-vous ?(une application smartphone, le modèle seulement…)

Nous avons fait le choix d’exploiter ce modèle en vue de la création d’un jeu vidéo. Ce dernier pourrait être commercialisé sous forme d’application smartphone. Aussi comme mentionné précédemment, le modèle pourrait très bien être exploité pour une autre technologie (fauteuil roulant électrique).

- Lorsque le modèle se trompe, se trompe-t-il de beaucoup ? Quel serait le coût de l’erreur ? Peut-on le réduire ?

Lorsque le modèle se trompe (mauvaise détection du mot prononcé), les conséquences sont importantes dans la mesure où cela impact la direction attendue. A titre d’exemple, la commande vocale indique « à droite » mais notre personnage se dirige vers le bas. Néanmoins, dans le cadre de notre jeu, l’impact est relatif. En effet, cela n’empêche pas l’utilisateur de poursuivre sa partie.

Dans le cas de l’utilisation du modèle pour la technologie du fauteuil roulant donnée en exemple précédemment les conséquences d’une erreur seraient beaucoup plus graves. C’est pourquoi il est important d’avoir une base de données de qualité.

Projet Voice Recognition - Premiers Etudes

December 23, 2018

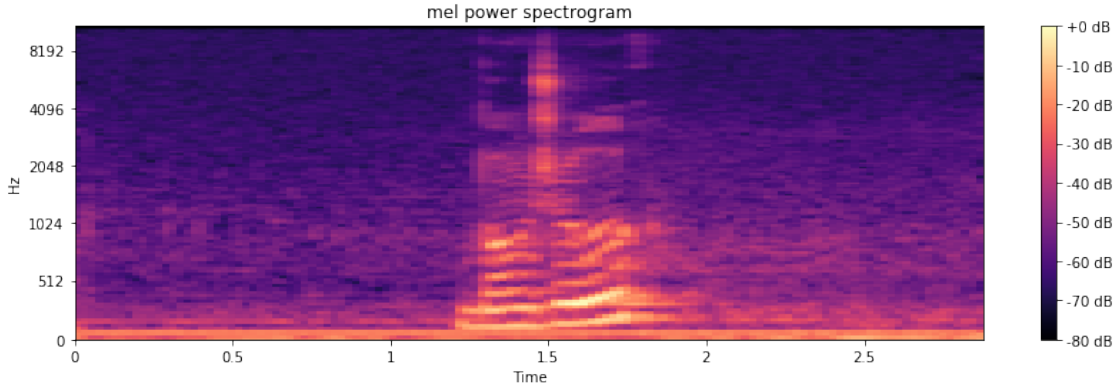
In [1]:
```
import numpy as np
import pandas as pd
# matplotlib for displaying the output
import matplotlib.pyplot as plt
import matplotlib.style as ms
ms.use('seaborn-muted')
%matplotlib inline

# and IPython.display for audio output
import IPython.display as ipd

# Librosa for audio
import librosa
# And the display module for visualization
import librosa.display
```

In [2]:
```
audio_path = "data/audio/bonjour/bonjour_mykhaylo_01.wav"

# or uncomment the line below and point it at your favorite song:
#
# audio_path = '/path/to/your/favorite/song.mp3'

y, sr = librosa.load(audio_path)
```

In [3]:
```
def spectrogram(y, sr):
    # Let's make and display a mel-scaled power (energy-squared) spectrogram
    S = librosa.feature.melspectrogram(y, sr=sr, n_mels=128)

    # Convert to log scale (dB). We'll use the peak power (max) as reference.
    log_S = librosa.power_to_db(S, ref=np.max)

    # Make a new figure
    plt.figure(figsize=(12,4))

    # Display the spectrogram on a mel scale
    # sample rate and hop length parameters are used to render the time axis
    librosa.display.specshow(log_S, sr=sr, x_axis='time', y_axis='mel')
```

```
        # Put a descriptive title on the plot
        plt.title('mel power spectrogram')

        # draw a color bar
        plt.colorbar(format='%+02.0f dB')

        # Make the figure layout compact
        plt.tight_layout()
```

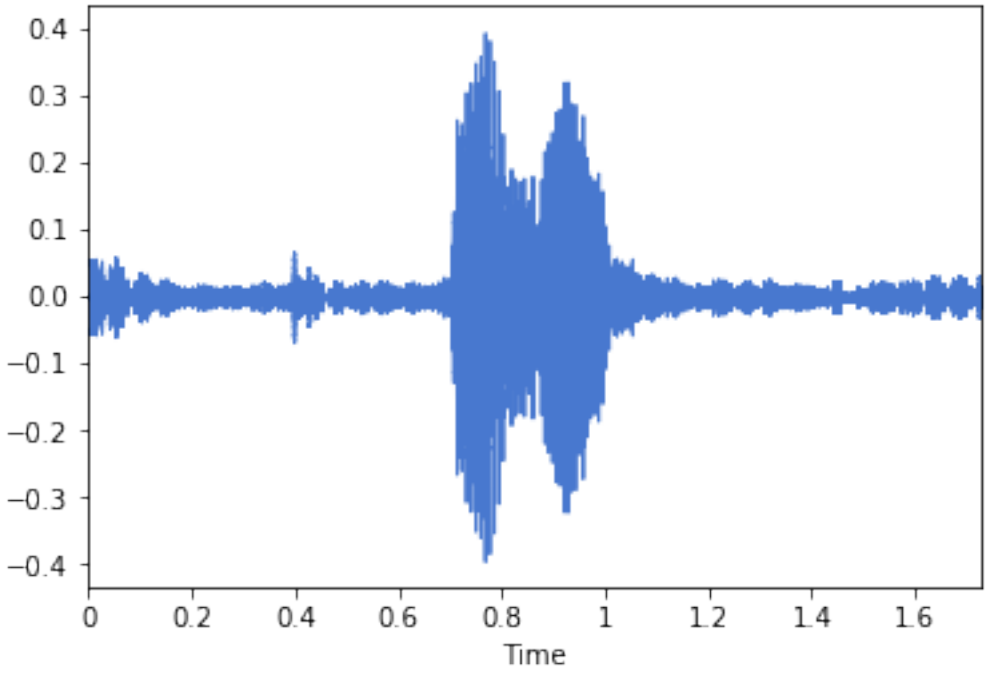
In [4]:
```
spectrogram(y,sr)
```

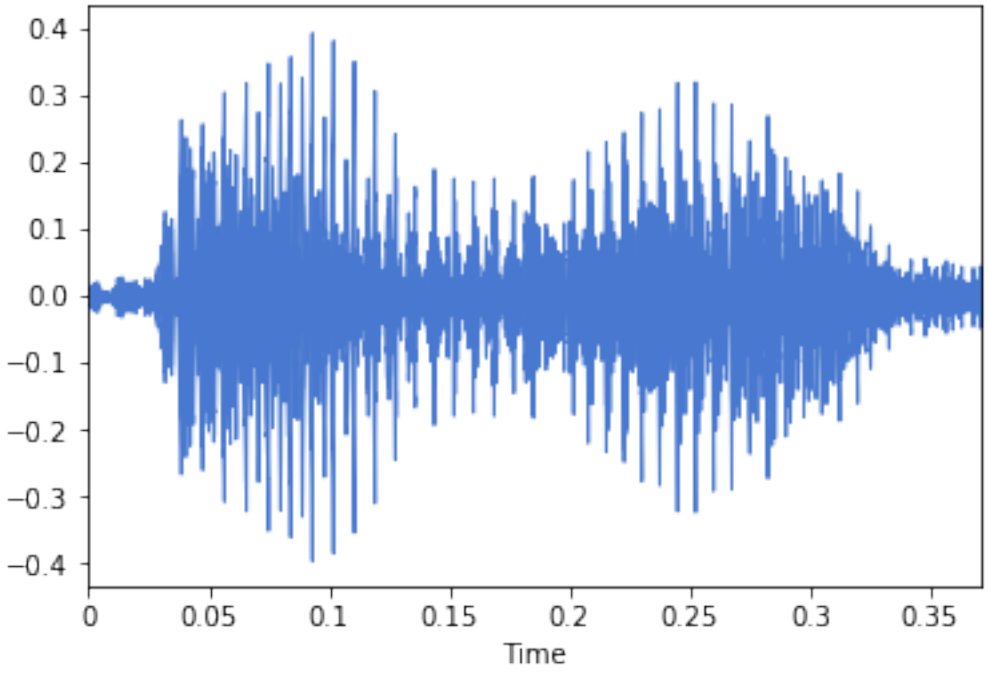


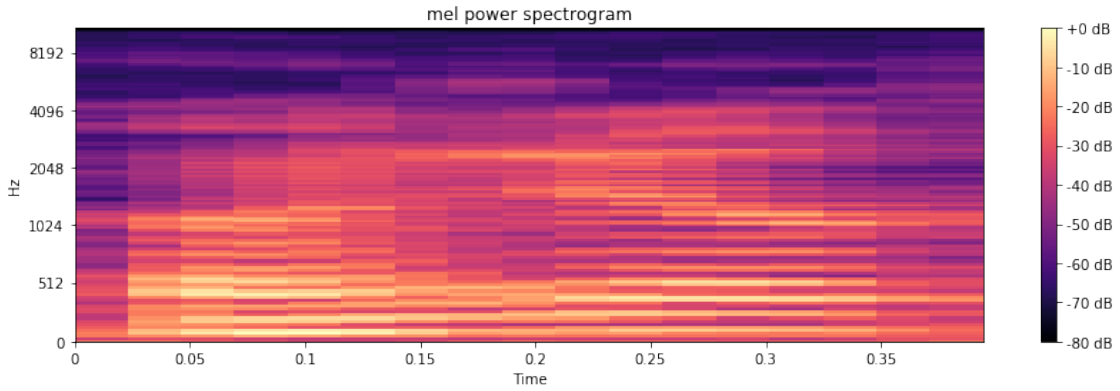
In [5]:
```
def trimClip(y,sr):
    plt.figure(1)
    librosa.display.waveplot(y, sr=sr)
    # Trim the beginning and ending silence
    yt, index = librosa.effects.trim(y, top_db=10)
    # Print the durations
    print(librosa.get_duration(y), librosa.get_duration(yt))
    plt.figure(2)
    librosa.display.waveplot(yt, sr=sr)
    spectrogram(yt,sr)
    return (yt,sr)
```

In [6]:
```
# Load some audio
y, sr = librosa.load("data/audio/bonjour/bonjour_narek_01.wav")
y1, sr1 = trimClip(y,sr)
```

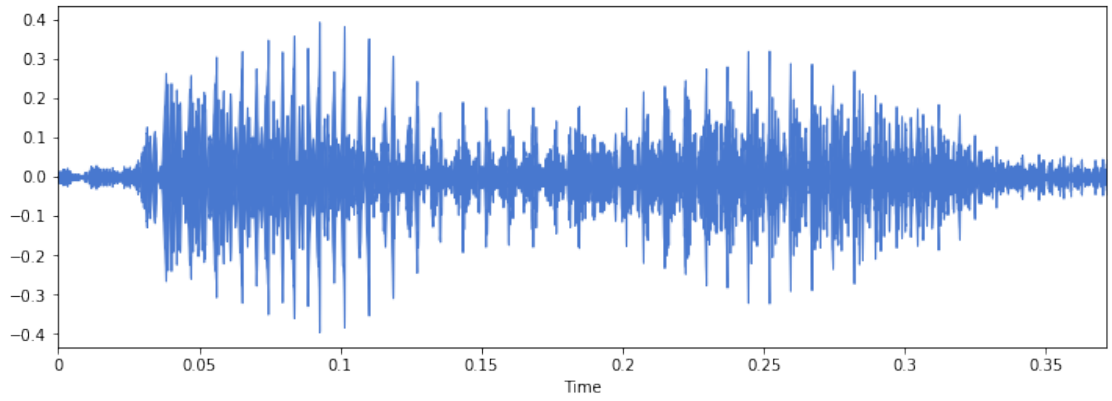
```
1.7298866213151927 0.37151927437641724
```

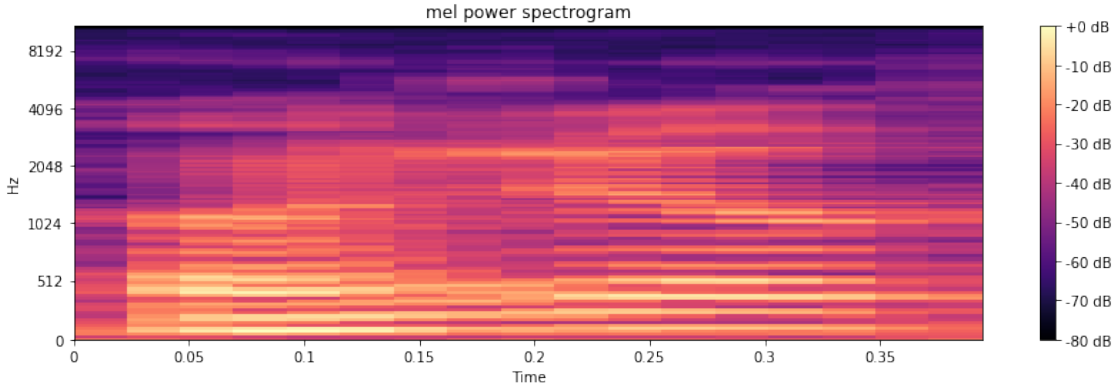

0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
Time


0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
Time




```
In [7]: def trimClip(y,sr):
            yt, index = librosa.effects.trim(y, top_db=11)
            return (yt,sr)

In [8]: def show(y, sr):
            plt.figure(figsize=(12,4))
            librosa.display.waveplot(y, sr=sr)
            spectrogram(y, sr)
            ipd.display(ipd.Audio(y, rate=sr))

In [9]: show(y1,sr)

<IPython.lib.display.Audio object>
```




mel power spectrogram
8192
4096
2048
1024
512
0
Hz
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
Time
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB

In [ ]:

Data Augmentation

December 23, 2018

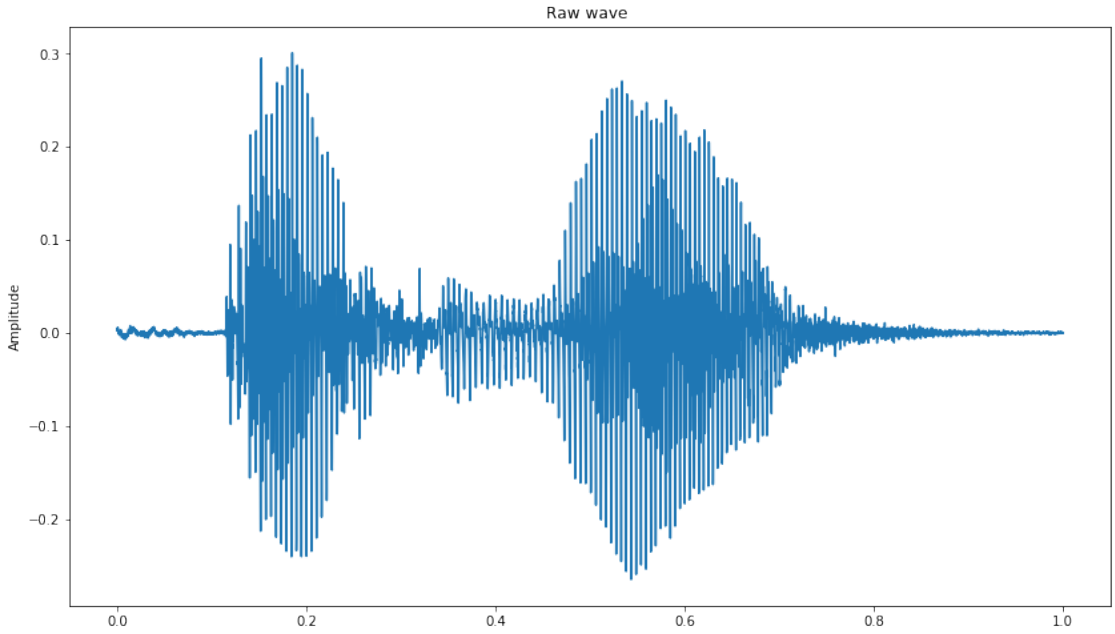
In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-py
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the fi

from subprocess import check_output
print(check_output(["ls", "data/audio/"]).decode("utf8"))

# Any results you write to the current directory are saved as output.
```

```
aurevoir
bas
bonjour
droite
gauche
haut
non
oui
```

In [2]:
```
#Import stuff

import numpy as np
import random
import itertools
import librosa
import IPython.display as ipd
import matplotlib.pyplot as plt
from matplotlib.colors import to_rgba

%matplotlib inline
```

In [3]:
```
def load_audio_file(file_path):
    input_length = 16000
    data = librosa.core.load(file_path)[0]
    return data
def plot_time_series(data):
    fig = plt.figure(figsize=(14, 8))
    plt.title('Raw wave ')
    plt.ylabel('Amplitude')
    plt.plot(np.linspace(0, 1, len(data)), data)
    plt.show()
```

In [4]:
```
data = load_audio_file("data/audio/aurevoir/aurevoir_mathilde_01.wav")
plot_time_series(data)
```



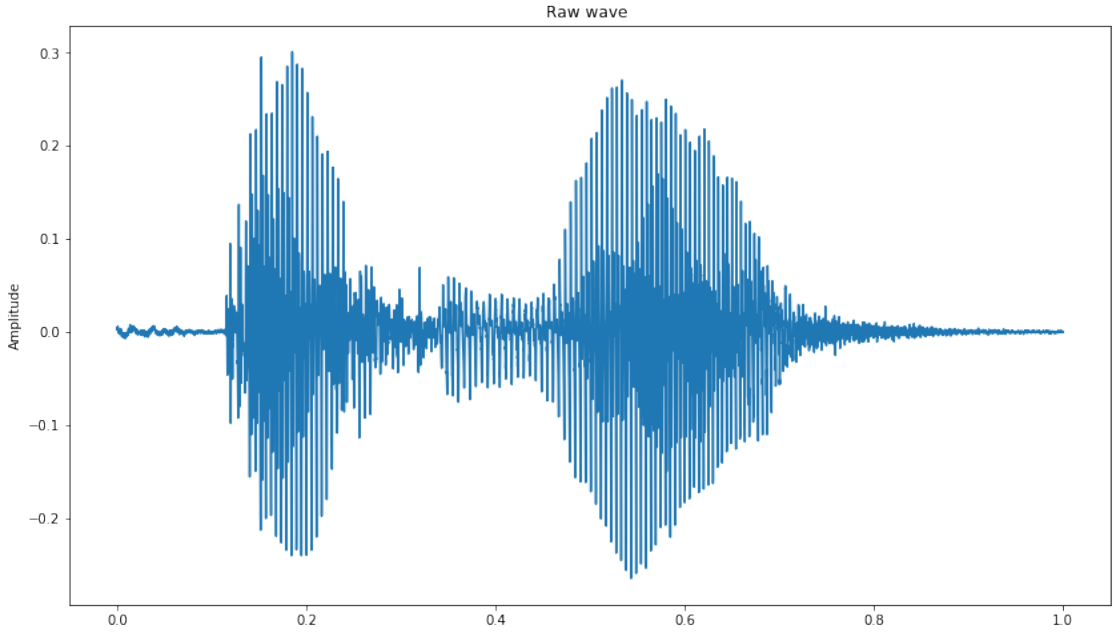
In [5]:
```
#Hear it !
ipd.Audio(data, rate=22050)
```

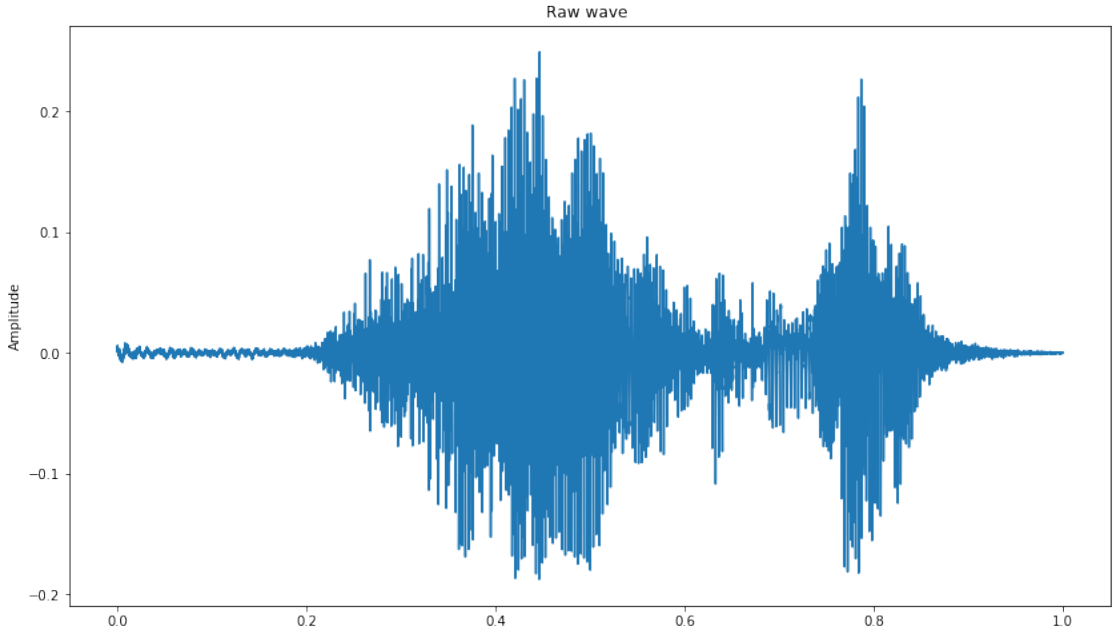
Out[5]:
```
<IPython.lib.display.Audio object>
```

In [6]:
```
alpha = 0.25
k = 0.25
p = (1-k*alpha)/(1-alpha)
y1 = np.concatenate([
    librosa.effects.time_stretch(data[0:int(len(data)*alpha)],k),
    librosa.effects.time_stretch(data[int(len(data)*alpha):],p)
])
print("alpha=%.3f, k=%.3f, p=%.3f" % (alpha, k, p))
```

```
        plot_time_series(data)
        plot_time_series(y1)
        ipd.Audio(y1, rate=22050)
```

```
alpha=0.250, k=0.250, p=1.250
```



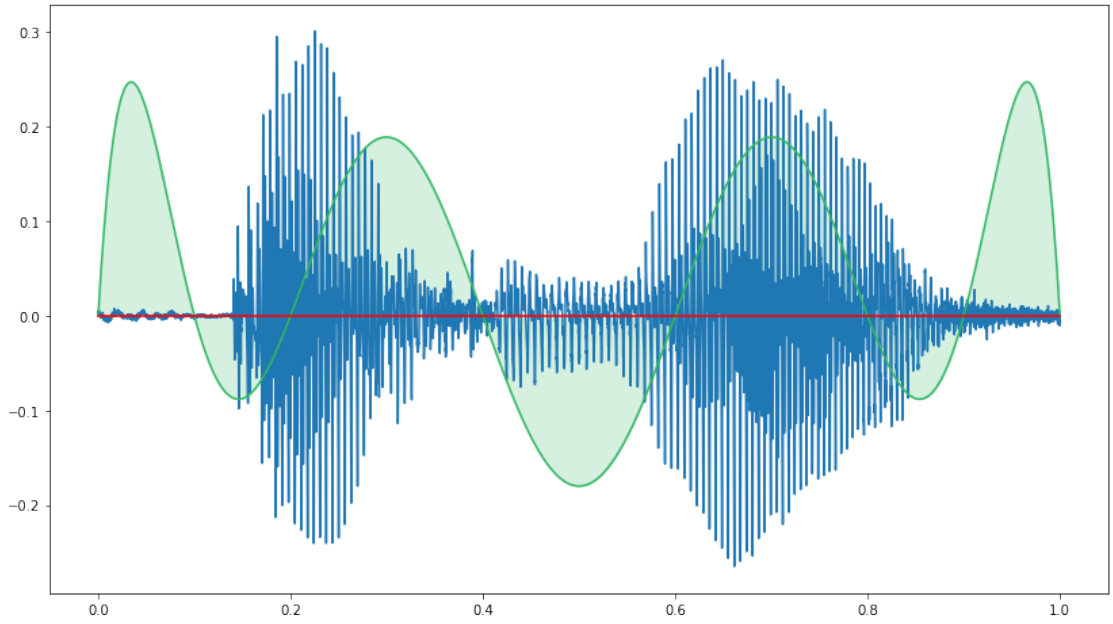


Out[6]: <IPython.lib.display.Audio object>

In [7]:
```
G = lambda x: -x*(x-0.1)*(x-0.2)*(x-0.4)*(x-0.6)*(x-0.8)*(x-0.9)*(x-1.0)
```

In [8]:
```
scale = 5000
plt.figure(figsize=(14, 8))
plt.plot(np.linspace(0,1,16000),data[:16000])
xs = np.linspace(0,1,1000)
plt.plot(xs,G(xs)*scale, color="xkcd:cool green")
plt.plot(xs,np.zeros(len(xs)), "r")
plt.fill_between(xs,G(xs)*scale,color=to_rgba("xkcd:cool green",0.2))
```

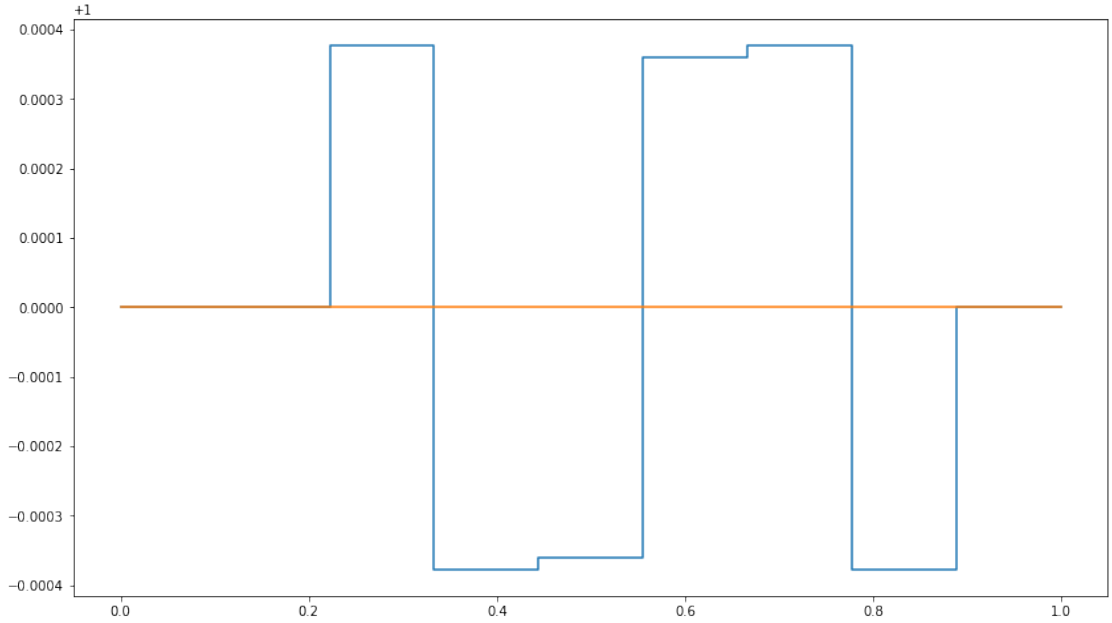
Out[8]: <matplotlib.collections.PolyCollection at 0x7eff0b6bb940>



In [9]:
```
stretch_coef = lambda G,n,amp: lambda j: (n*(G((j+1)/n)-G(j/n))-G(1)+G(0))*amp
```

In [10]:
```
n = 10
amplifier = 1
p = lambda i: 1+stretch_coef(G,n,amplifier)(i)
plt.figure(figsize=(14, 8))
xs = np.linspace(0,1,n)
ys = np.array(list(map(p, range(n))))
plt.step(xs, ys, where="post")
plt.plot(xs, np.zeros(n)+1)
```

Out[10]: [<matplotlib.lines.Line2D at 0x7eff0b60bc88>]

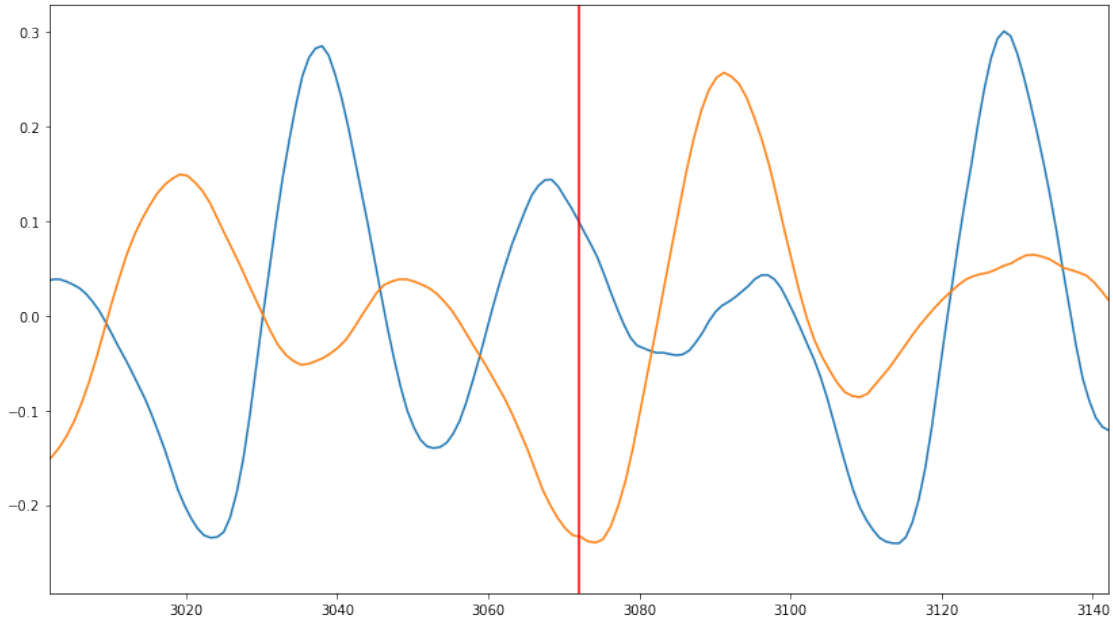



```
In [11]: def stretch(y,G,n,amplifier):
             spl = np.array_split(y, n)
             p = lambda i: 1+stretch_coef(G,n,amplifier)(i)
             res = [
                 librosa.effects.time_stretch(spl[i],p(i))
                 for i in range(n)
             ]
             print([len(x) for x in res])
             return np.concatenate(res)

In [12]: #ici on voit les "chirps"
         G1 = lambda x: -np.sin(x)

         n = 10

         print(len(data))
         y1 = stretch(data,G,n,1)
         print(len(y1))
         y2 = stretch(data,G1,n,1)
         print(len(y2))

         plt.figure(figsize=(14, 8))
         plt.plot(np.linspace(0,len(y1),len(data)),data)
         #plt.plot(np.linspace(0,1,len(y2)),y2)
         plt.plot(np.linspace(0,len(y1),len(y1)),y1)
         #plt.plot(np.linspace(0,1,len(y2)),y2)
```

```
        delta = 70
        for i in np.cumsum([1536, 1536, 1536, 2048, 2048, 1536, 1536, 2048, 1536, 1536]):
            plt.axvline(x=i,color="red")
        x0 = 1536*2
        plt.xlim(x0-delta, x0+delta)

        ipd.Audio(y1, rate=22050)
```

```
19472
[1536, 1536, 1536, 2048, 2048, 1536, 1536, 2048, 1536, 1536]
16896
[2048, 2048, 2048, 2048, 2048, 2048, 1536, 1536, 1536, 1536]
18432
```

Out[12]: <IPython.lib.display.Audio object>



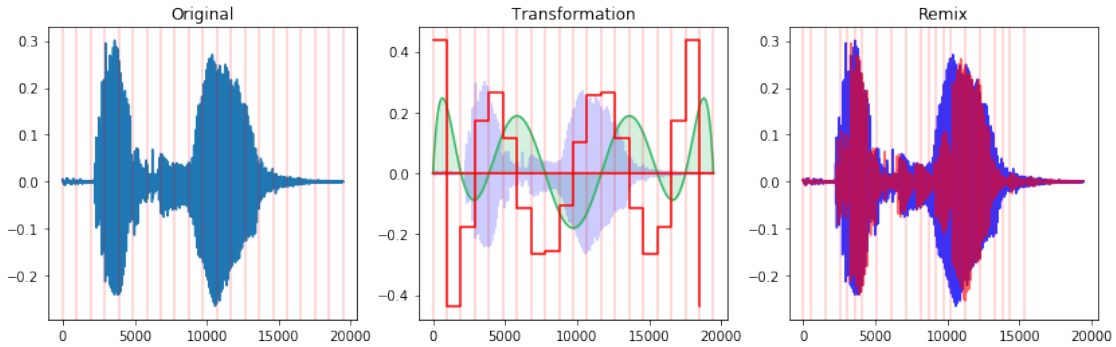

```
In [13]: def time_transform(y, G, n, amplifier, visualize=False, sr=22050, cscale = 8000, dscal
             """Transform the time with 1+G'(x) stretch coefficients

             input:
                 y:          the input signal
                 G:          integral of the transformation function
                              (must be differentiable)
                 n:          number of intervals for discretization
                 amplifier: time distortion intensity amplifier
                 visualize: Original/Transformation/Remix graphs
```

```
                and Original/Remix audio widgets
    sr:         sample rate
    cscale:     G(x) plot scale
    dscale:     discretized G'(x) plot scale

output:
    the remixed sequence
"""
stretch_coef = lambda G,n,amp: lambda j: (n*(G((j+1)/n)-G(j/n))-G(1)+G(0))*amp
p = lambda i: 1+stretch_coef(G,n,amplifier)(i)

if visualize:
    ys = np.array(list(map(p, range(n))))
    plt.figure(figsize=(14, 4))
    ax = plt.subplot(1, 3, 1)
    ax.set_title("Original")
    xs = np.linspace(0,len(y),len(y))
    plt.plot(xs,y)
    for i in range(n+1):
        plt.axvline(x=i/n*len(y),color=to_rgba("red",0.2))
    ax = plt.subplot(1, 3, 2)
    ax.set_title("Transformation")

    plt.plot(xs,y,color=to_rgba("blue",0.2))
    plt.plot(xs,G(xs/len(y))*scale, color="xkcd:cool green")
    plt.plot(xs,np.zeros(len(xs)), "r")
    plt.fill_between(xs,G(xs/len(y))*scale,color=to_rgba("xkcd:cool green",0.2))
    plt.step(np.arange(n)/n*len(y), (ys-1)*dscale, where="post", color="red")
    for i in range(n+1):
        plt.axvline(x=i/n*len(y),color=to_rgba("red",0.2))

spl = np.array_split(y, n)
res = [
    librosa.effects.time_stretch(spl[i],p(i))
    for i in range(n)
]
y1 = np.concatenate(res)

if visualize:
    ax = plt.subplot(1, 3, 3)
    ax.set_title("Remix")
    plt.plot(xs,y,color=to_rgba("blue",0.8))
    xs = np.linspace(0,len(y),len(y1))
    plt.plot(xs,y1,color=to_rgba("red",0.6))
    lens = [len(x) for x in res]
    for i in np.concatenate([np.array([0]),np.cumsum(lens)]):
        plt.axvline(x=i,color=to_rgba("red",0.2))
```

```
            print("Original audio:")
            ipd.display(ipd.Audio(y, rate=sr))
            print("Remixed audio:")
            ipd.display(ipd.Audio(y1, rate=sr))

        # scale back in case something went wrong

        return librosa.effects.time_stretch(y1, len(y1)/len(y))
```

```
In [14]: mixed = time_transform(data, G, 20, 1, visualize=True)
         librosa.output.write_wav("test.wav", mixed, 22050, norm=True)
```

```
Original audio:


<IPython.lib.display.Audio object>


Remixed audio:


<IPython.lib.display.Audio object>
```



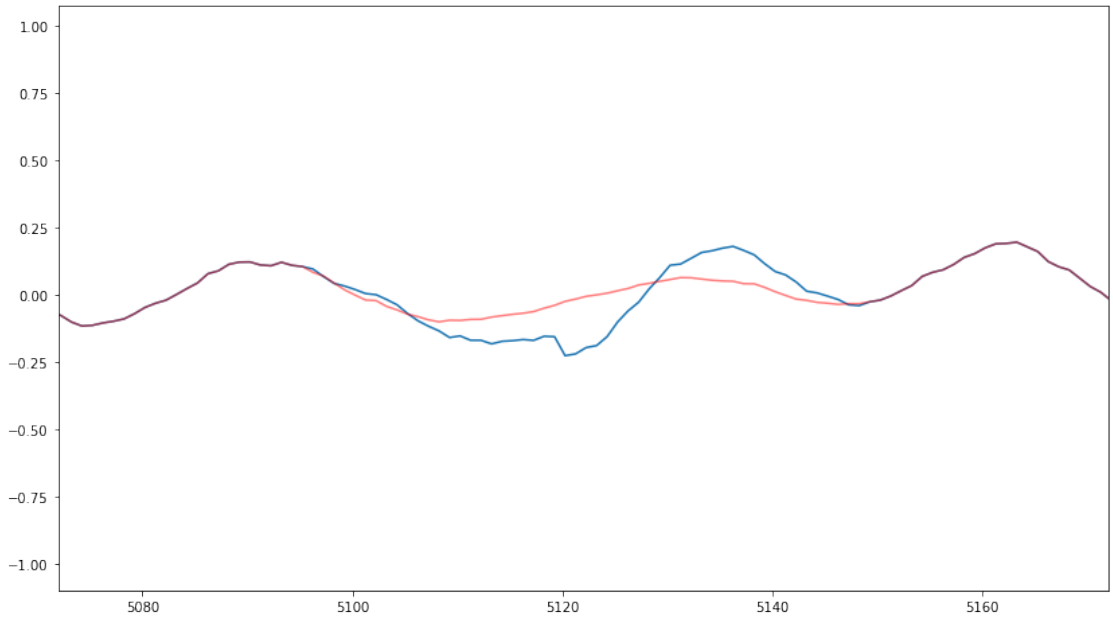

```
In [15]: mixed = load_audio_file("test.wav")
         mix_edited = load_audio_file("test1.wav")
```

Repair Audacity source - need to look here for click removal
Resampling with SciPy - or here

```
In [16]: plt.figure(figsize=(14, 8))
         xs = np.linspace(0,len(mixed),len(mixed))
         plt.plot(xs,mixed)

         xs = np.linspace(0,len(mix_edited),len(mix_edited))
         plt.plot(xs,mix_edited, color=to_rgba("red", 0.5))
```

```
x0 = 5122
delta = 50
plt.xlim(x0-delta, x0+delta)
```

```
Out[16]: (5072, 5172)
```



1 Random splitting

```
In [17]: def ransplit(n, v=1):
             lens = (np.random.randn(n+1)/n*v+1)/n
             cums = np.cumsum(lens)-lens[0]
             return cums / max(cums)
```

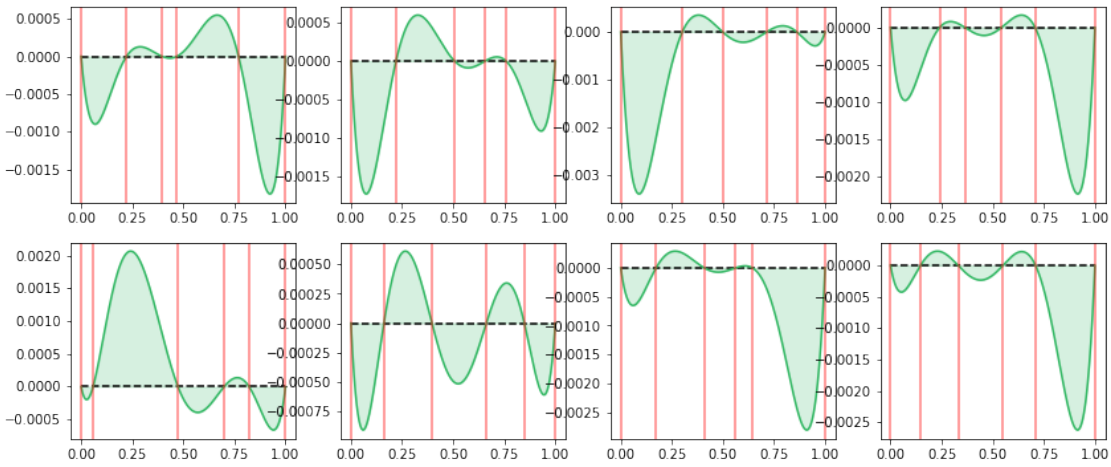
```
In [18]: ransplit(10)
```

```
Out[18]: array([0.        , 0.09664884, 0.19386765, 0.30776238, 0.38612253,
                0.49280952, 0.60983617, 0.694923  , 0.80243466, 0.9029496 ,
                1.        ])
```

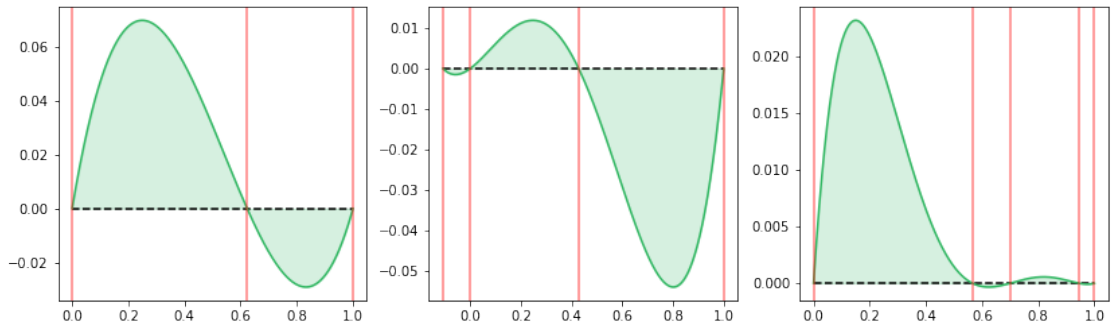
```
In [19]: def fill_plot(xs, ys, color="xkcd:cool green", fill_opacity=0.2):
             plt.plot(xs, ys, color=color)
             plt.fill_between(xs, ys, color=to_rgba(color,fill_opacity))

         def split_plot(xs, color="red", opacity=1):
             for x in xs:
                 plt.axvline(x=x,color=to_rgba(color,opacity))
```

```
def poly_plot(r_or_G, color="xkcd:cool green", split_color="red"):
    if type(r_or_G)==dict:
        roots = r_or_G["roots"]
    else:
        roots = r_or_G
    xs = np.linspace(np.min(roots),np.max(roots),1000)
    plt.plot(xs, np.zeros(1000),color="black", linestyle="dashed")
    G = np.poly1d(roots,r=True)
    fill_plot(xs, G(xs), color=color)
    split_plot(roots, color=split_color, opacity=0.5)

def G_gen(n,e):
    roots = ransplit(n,e)
    return {
        "roots": roots,
        "G": np.poly1d(roots,r=True)
    }

def grid_plot(f, rows, cols, figsize=(14,6)):
    plt.figure(figsize=figsize)
    for i in range(rows):
        for j in range(cols):
            plt.subplot(rows, cols, i*cols+j+1)
            f(i*cols+j)
```

In [20]:
```
n = 5
e = 2

Gs = [G_gen(n,e) for x in range(8)]
grid_plot(lambda i: poly_plot(Gs[i]), 2, 4)
```



```
In [21]: grid_plot(lambda i: [
             lambda: poly_plot(G_gen(2,2)),
             lambda: poly_plot(G_gen(3,2)),
             lambda: poly_plot(G_gen(4,2))
         ][i](), 1, 3, figsize=(14,4))
```

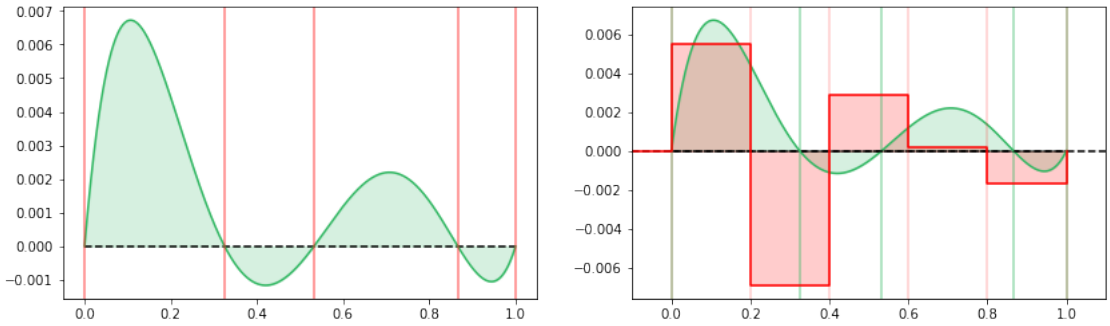


2 Utilisons le formule

```
In [22]: stretch_coef = lambda G,n,amp: lambda j: (n*(G((j+1)/n)-G(j/n))-G(1)+G(0))*amp
```

```
In [23]: def p_plot(Gs, n, amplifier):
             poly_plot(Gs,split_color="xkcd:cool green")
             plt.axhline(y=0, color="black", linestyle="dashed")
             ys = np.array(list(map(stretch_coef(Gs["G"],n,amplifier), range(n))))
             plt.step((np.arange(n+2)-1)/n, np.concatenate([[0],ys,[0]])*0.25, where="post", co
             plt.fill_between(
                 (np.arange(n+2)-1)/n,
                 np.concatenate([[0],ys,[0]])*0.25,
                 np.zeros(n+2),
                 step="post", color=to_rgba("red",0.2))
             for i in range(n+3):
                 plt.axvline(x=i/n,color=to_rgba("red",0.2))
             plt.xlim(-0.1,1.1)
```

```
In [24]: Gs = G_gen(4,2)
         n = 5
         amplifier = 1
         #p = lambda i: 1+stretch_coef(Gs["G"],n,amplifier)(i)

         grid_plot(lambda i: [
             lambda: poly_plot(Gs),
             lambda: p_plot(Gs, n, amplifier)
         ][i](), 1, 2, figsize=(14,4))
```




```
In [25]: mixed1 = time_transform(data, Gs["G"], 5, 30, visualize=True)
         mixed2 = time_transform(data, lambda x: -Gs["G"](x), 5, 30, visualize=True)
```

```
Original audio:
```

```
<IPython.lib.display.Audio object>
```

```
Remixed audio:
```

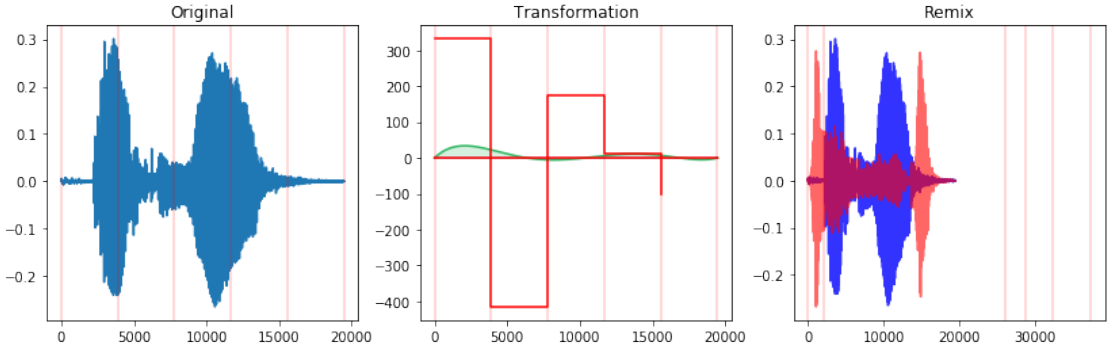
```
<IPython.lib.display.Audio object>
```

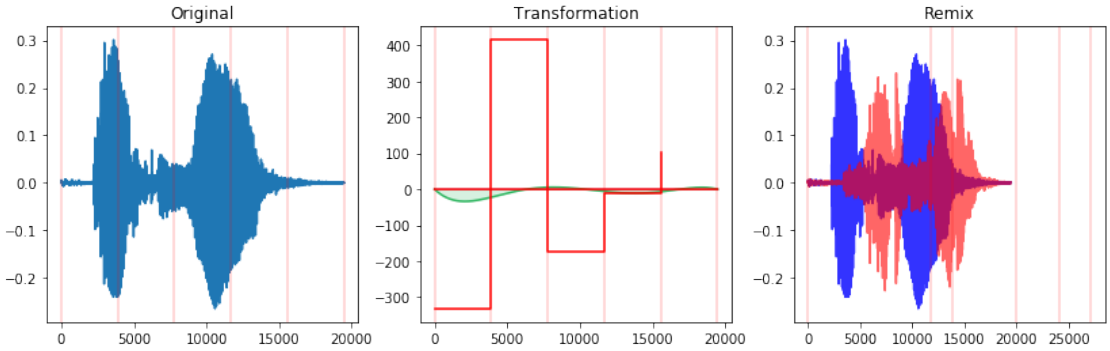
```
Original audio:
```

```
<IPython.lib.display.Audio object>
```

```
Remixed audio:
```

```
<IPython.lib.display.Audio object>
```



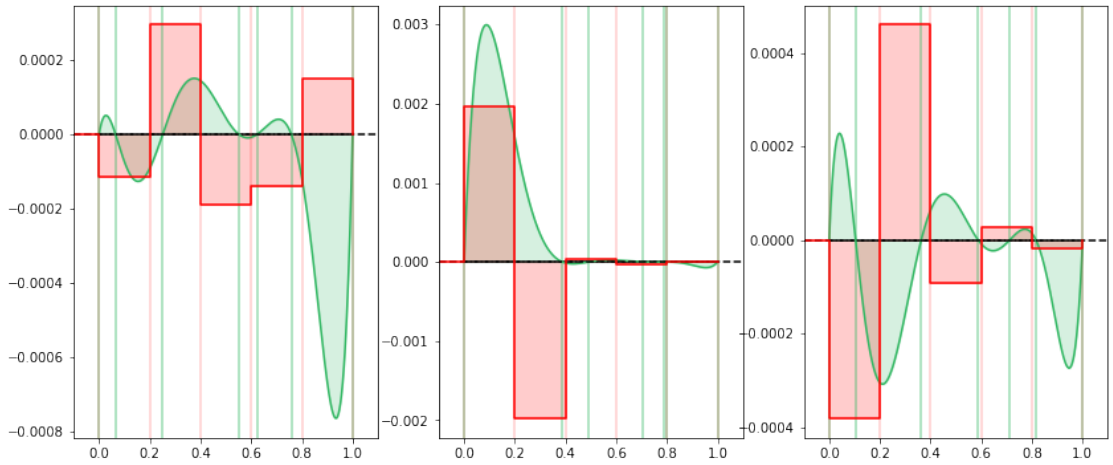


In [26]:
```
n = 5
amplifier = 15
Gs = [G_gen(6,3) for x in range(3)]
```

In [27]:
```
#si ça plante il faut relancer la cellule d'avant
grid_plot(lambda i: p_plot(Gs[i],n,1), 1, 3)

mix = lambda data, G: time_transform(data, G["G"], n, amplifier, visualize=False)
negmix = lambda data, G: time_transform(data, lambda x: -G["G"](x), n, amplifier, visu
revmix = lambda data, G: time_transform(data, lambda x: G["G"](1-x), n, amplifier, vis
negrevmix = lambda data, G: time_transform(data, lambda x: -G["G"](1-x), n, amplifier

mixes = [mix(data,G) for G in Gs] + \
    [negmix(data,G) for G in Gs] + \
    [revmix(data,G) for G in Gs] + \
    [negrevmix(data,G) for G in Gs]

res = np.concatenate(mixes)
print("Generated %d mixes. For 30 tracks this will be %d" % (len(mixes), len(mixes)*3

ipd.Audio(res,rate=22050)
```

```
Generated 12 mixes. For 30 tracks this will be 360
```

Out[27]:
```
<IPython.lib.display.Audio object>
```




```
In [28]: def allmixes(data, Gs, n, amplifier):
             mix = lambda data, G: time_transform(data, G["G"], n, amplifier, visualize=False)
             negmix = lambda data, G: time_transform(data, lambda x: -G["G"](x), n, amplifier,
             revmix = lambda data, G: time_transform(data, lambda x: G["G"](1-x), n, amplifier
             negrevmix = lambda data, G: time_transform(data, lambda x: -G["G"](1-x), n, ampli

             return [mix(data,G) for G in Gs] + \
                 [negmix(data,G) for G in Gs] + \
                 [revmix(data,G) for G in Gs] + \
                 [negrevmix(data,G) for G in Gs]

In [29]: labels = check_output(["ls", "data/audio/"]).decode("utf-8").split("\n")[:-1]
         labels

Out[29]: ['aurevoir', 'bas', 'bonjour', 'droite', 'gauche', 'haut', 'non', 'oui']

In [30]: def trimClip(y):
             yt, index = librosa.effects.trim(y, top_db=11)
             return (yt)

In [ ]: for label in labels:
            print("Processing "+label+":")
            filenames = check_output(["ls", "data/audio/"+label+"/"]).decode("utf-8").split("\
            for fname in filenames:
                print("> Processing "+fname[:-4]+":")
                data = trimClip(librosa.util.normalize(load_audio_file("data/audio/"+label+"/"
                librosa.output.write_wav("mixes/"+label+"/"+fname[:-4]+"_00.wav", data, 22050,
                mixes = allmixes(data, Gs, 5, 15)
                for i in range(len(mixes)):
                    print(">> Processing mix #"+str(i+1))
                    librosa.output.write_wav(
```

```
                    "mixes/"+label+"/"+fname[:-4]+"_"+"{0:02d}".format(i+1)+".wav",
                    trimClip(mixes[i]), 22050, norm=True
                )
```

In [32]:
```
def preprocess(file):
    librosa.output.write_wav(
        file,
        trimClip(librosa.util.normalize(load_audio_file(file))),
        22050,
        norm=True
    )
```

In [33]:
```
preprocess("data/test/steven_test-01.wav")
preprocess("data/test/steven_test-02.wav")
preprocess("data/test/test-01.wav")
preprocess("data/test/test-02.wav")
preprocess("data/test/test-03.wav")
preprocess("data/test/test-04.wav")
```

In [ ]:

Modele

December 23, 2018

In [1]:
```
%load_ext autoreload
%autoreload 2

from preprocess import *
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.utils import to_categorical
```

```
Using TensorFlow backend.
```

In [2]:
```
# Second dimension of the feature is dim2
feature_dim_2 = 11
```

In [3]:
```
# Save data to array file first
save_data_to_array(max_len=feature_dim_2)
```

```
Saving vectors of label - 'bas': 100%|| 1677/1677 [01:24<00:00, 19.92it/s]
Saving vectors of label - 'droite': 100%|| 1703/1703 [01:25<00:00, 20.22it/s]
Saving vectors of label - 'aurevoir': 100%|| 1690/1690 [01:33<00:00, 17.80it/s]
Saving vectors of label - 'bonjour': 100%|| 1794/1794 [01:42<00:00, 17.55it/s]
Saving vectors of label - 'non': 100%|| 1599/1599 [01:21<00:00, 19.71it/s]
Saving vectors of label - 'haut': 100%|| 1690/1690 [01:24<00:00, 19.92it/s]
Saving vectors of label - 'gauche': 100%|| 1729/1729 [01:27<00:00, 19.93it/s]
Saving vectors of label - 'oui': 100%|| 1651/1651 [01:21<00:00, 20.17it/s]
```

In [4]:
```
# # Loading train set and test set
X_train, X_test, y_train, y_test = get_train_test()

# # Feature dimension
feature_dim_1 = 20
channel = 1
epochs = 15
batch_size = 100
verbose = 1
num_classes = 8
```

```
        # Reshaping to perform 2D convolution
        X_train = X_train.reshape(X_train.shape[0], feature_dim_1, feature_dim_2, channel)
        X_test = X_test.reshape(X_test.shape[0], feature_dim_1, feature_dim_2, channel)

        y_train_hot = to_categorical(y_train)
        y_test_hot = to_categorical(y_test)
```

```
In [5]: def get_model():
            model = Sequential()
            model.add(Conv2D(32, kernel_size=(2, 2), activation='relu', input_shape=(feature_di
            model.add(Conv2D(48, kernel_size=(2, 2), activation='relu'))
            model.add(Conv2D(120, kernel_size=(2, 2), activation='relu'))
            model.add(MaxPooling2D(pool_size=(2, 2)))
            model.add(Dropout(0.25))
            model.add(Flatten())
            model.add(Dense(128, activation='relu'))
            model.add(Dropout(0.25))
            model.add(Dense(64, activation='relu'))
            model.add(Dropout(0.4))
            model.add(Dense(num_classes, activation='softmax'))
            model.compile(loss=keras.losses.categorical_crossentropy,
                          optimizer=keras.optimizers.Adadelta(),
                          metrics=['accuracy'])
            return model

        # Predicts one sample
        def predict(filepath, model):
            sample = wav2mfcc(filepath)
            sample_reshaped = sample.reshape(1, feature_dim_1, feature_dim_2, channel)
            return get_labels()[0][
                    np.argmax(model.predict(sample_reshaped))
            ]
```

1 Building The Model Then Training it

```
In [6]: model = get_model()
        Model1 = model.fit(
            X_train, y_train_hot,
            batch_size=batch_size,
            epochs=epochs,
            verbose=verbose,
            validation_data=(X_test, y_test_hot)
        )
```

```
Train on 8119 samples, validate on 5414 samples
Epoch 1/15
8119/8119 [==============================] - 7s 884us/step - loss: 1.5159 - acc: 0.4860 - val_l
```

```
Epoch 2/15
8119/8119 [==============================] - 8s 1ms/step - loss: 0.5893 - acc: 0.8090 - val_los
Epoch 3/15
8119/8119 [==============================] - 10s 1ms/step - loss: 0.2944 - acc: 0.9090 - val_lo
Epoch 4/15
8119/8119 [==============================] - 9s 1ms/step - loss: 0.1638 - acc: 0.9528 - val_los
Epoch 5/15
8119/8119 [==============================] - 8s 945us/step - loss: 0.1096 - acc: 0.9665 - val_
Epoch 6/15
8119/8119 [==============================] - 8s 964us/step - loss: 0.0828 - acc: 0.9722 - val_
Epoch 7/15
8119/8119 [==============================] - 9s 1ms/step - loss: 0.0618 - acc: 0.9808 - val_los
Epoch 8/15
8119/8119 [==============================] - 8s 931us/step - loss: 0.0568 - acc: 0.9835 - val_
Epoch 9/15
8119/8119 [==============================] - 8s 927us/step - loss: 0.0393 - acc: 0.9867 - val_
Epoch 10/15
8119/8119 [==============================] - 7s 897us/step - loss: 0.0347 - acc: 0.9903 - val_
Epoch 11/15
8119/8119 [==============================] - 8s 973us/step - loss: 0.0327 - acc: 0.9897 - val_
Epoch 12/15
8119/8119 [==============================] - 7s 883us/step - loss: 0.0308 - acc: 0.9911 - val_
Epoch 13/15
8119/8119 [==============================] - 7s 868us/step - loss: 0.0253 - acc: 0.9911 - val_
Epoch 14/15
8119/8119 [==============================] - 8s 952us/step - loss: 0.0237 - acc: 0.9917 - val_
Epoch 15/15
8119/8119 [==============================] - 9s 1ms/step - loss: 0.0184 - acc: 0.9940 - val_los
```

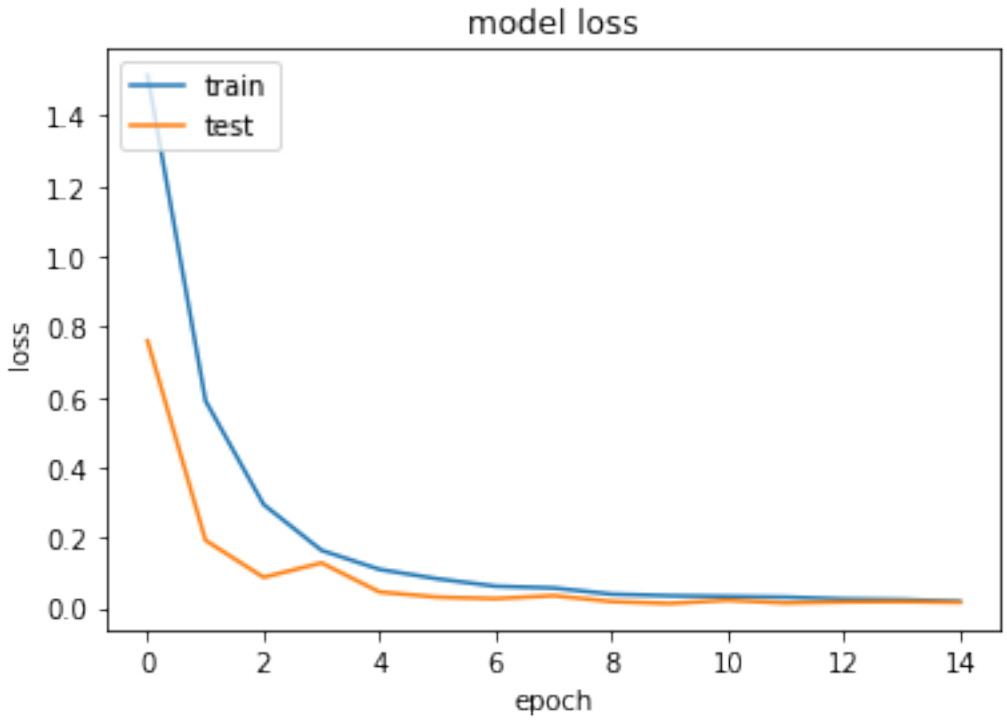
In [7]: `model.summary()`

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 19, 10, 32)        160
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 18, 9, 48)         6192
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 17, 8, 120)        23160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 4, 120)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 4, 120)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3840)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               491648
```

```
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 520
=================================================================
Total params: 529,936
Trainable params: 529,936
Non-trainable params: 0
_________________________________________________________________
```

```
In [8]: import matplotlib.pyplot as plt
        %matplotlib inline
```

```
In [9]: plt.plot(Model1.history['loss'])
        plt.plot(Model1.history['val_loss'])
        plt.title('model loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['train','test'], loc='upper left')
        plt.show()
```



1.1 Prediction

```
In [10]: print(predict('mixes/droite/droite_mathilde_01_09.wav', model=model))
```

```
droite
```

```
In [11]: print(predict('data/test/test-01.wav', model=model)) #bonjour
         print(predict('data/test/test-02.wav', model=model)) #au revoir
         print(predict('data/test/test-03.wav', model=model)) #a gauche
         print(predict('data/test/test-04.wav', model=model)) #a droite
         print(predict('data/test/steven_test-01.wav', model=model)) #oui
         print(predict('data/test/steven_test-02.wav', model=model)) #oui
```

```
bonjour
aurevoir
gauche
droite
oui
oui
```

```
In [12]: model.save("model.h5")
```

```
In [ ]:
```

Game

December 23, 2018

In [1]:
```
from preprocess import *
from subprocess import check_output

import keras
import numpy as np
import librosa
```

```
Using TensorFlow backend.
```

In [2]:
```
def load_audio_file(file_path):
    input_length = 16000
    data = librosa.core.load(file_path)[0]
    return data
```

In [3]:
```
model = keras.models.load_model("model.h5")
```

In [4]:
```
# # Feature dimension
feature_dim_1 = 20
feature_dim_2 = 11
channel = 1
epochs = 15
batch_size = 100
verbose = 1
num_classes = 8

def files(label):
    return check_output(["ls", "mixes/"+label+"/"]).decode("utf-8").split("\n")[:-1]

def trimClip(y):
    yt, index = librosa.effects.trim(y, top_db=11)
    return (yt)

def preproc(file_path, max_len=11):
    wave, sr = librosa.load(file_path, mono=True, sr=None)
    wave = trimClip(librosa.util.normalize(wave))
    wave = wave[::3]
    mfcc = librosa.feature.mfcc(wave, sr=16000)
```

```
            # If maximum length exceeds mfcc lengths then pad the remaining ones
            if (max_len > mfcc.shape[1]):
                pad_width = max_len - mfcc.shape[1]
                mfcc = np.pad(mfcc, pad_width=((0, 0), (0, pad_width)), mode='constant')

            # Else cutoff the remaining parts
            else:
                mfcc = mfcc[:, :max_len]

            return mfcc

        # Predicts one sample
        def predict(filepath, model):
            sample = preproc(filepath)
            sample_reshaped = sample.reshape(1, feature_dim_1, feature_dim_2, channel)
            return ['bas', 'droite', 'aurevoir', 'bonjour', 'non', 'haut', 'gauche', 'oui'][
                    np.argmax(model.predict(sample_reshaped))
            ]
```

```
In [5]: droites = files("droite")
        gauches = files("gauche")
        hauts = files("haut")
        bass = files("bas")
```

```
In [6]: np.random.randint(0,len(droites))
```

```
Out[6]: 863
```

```
In [7]: def getPred(arr, lbl):
            fname = "mixes/"+lbl+"/"+arr[np.random.randint(0,len(arr))]
            pred = predict(
                    fname,
                    model=model
                )
            #print(pred)
            snd = pygame.sndarray.make_sound((load_audio_file(fname) * 32768).astype(np.int16))
            return (
                pred,
                snd
            )
```

```
In [8]: """
        Sample Python/Pygame Programs
        Simpson College Computer Science
        http://programarcadegames.com/
        http://simpson.edu/computer-science/
        """
        import pygame
```

```
import pygame.mixer as mx


BLACK = (0, 0, 0)
WHITE = (255, 255, 255)

sprites = {
    0: "data/game_img/bg.png",
    1: "data/game_img/wall.png",
    2: "data/game_img/coin.png",
    3: "data/game_img/fin.png",
    4: "data/game_img/player.png"
}

size = 13
field = [[0 for x in range(size)] for y in range(size)]
field = [
    [1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,0,1,0,2,0,2,0,1,1,0,2,1],
    [1,2,1,0,0,2,0,2,0,2,0,0,1],
    [1,0,1,2,1,1,1,1,0,1,2,0,1],
    [1,2,1,0,1,0,2,1,0,1,0,2,1],
    [1,0,2,0,1,2,0,0,2,0,2,0,1],
    [1,1,1,1,1,0,0,1,1,1,1,1,1],
    [1,2,0,2,0,2,0,2,0,2,0,2,1],
    [1,0,0,0,2,1,1,1,0,0,1,0,1],
    [1,2,1,1,0,1,3,1,1,0,1,2,1],
    [1,0,0,0,2,1,2,0,1,0,1,0,1],
    [1,2,0,2,0,1,0,2,0,2,0,2,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1]
]

mx.quit()
mx.init(channels=1)

#sndhaut = mx.Sound("data/audio/haut/haut_adel_01.wav")
#snddroite = mx.Sound("data/audio/droite/droite_adel_01.wav")
#sndgauche = mx.Sound("data/audio/gauche/gauche_adel_01.wav")
#sndbas = mx.Sound("data/audio/bas/bas_adel_01.wav")


class Player(pygame.sprite.Sprite):
    """ The class is the player-controlled sprite. """

    def __init__(self, x, y):
        """Constructor function"""
        # Call the parent's constructor
        super().__init__()
```

```
            # Set height, width
            #self.image = pygame.Surface([15, 15])
            self.image = pygame.image.load(sprites[4])
            #self.image.fill(BLACK)

            # Make our top-left corner the passed-in location.
            self.rect = self.image.get_rect()
            self.rect.x = x
            self.rect.y = y

    class SimpleSprite(pygame.sprite.Sprite):
        """ The class is the simple sprite. """

        def __init__(self, x, y, code):
            """Constructor function"""
            # Call the parent's constructor
            super().__init__()

            # Set height, width
            #self.image = pygame.Surface([15, 15])
            self.image = pygame.image.load(sprites[code])
            #self.image.fill(BLACK)

            # Make our top-left corner the passed-in location.
            self.rect = self.image.get_rect()
            self.rect.x = x
            self.rect.y = y
            self.code = code

        def setCode(self, code):
            self.code = code
            self.image = pygame.image.load(sprites[code])
```

In [9]:
```
    # Call this function so the Pygame library can initialize itself
    pygame.init()
    mx.quit()
    mx.init(channels=1)

    # Create an 800x600 sized screen
    screen = pygame.display.set_mode([455, 455])

    # Set the title of the window
    pygame.display.set_caption("Jeu de Deep Learning")

    # Create the player paddle object
    #player = Player(0, 0)
    all_sprites_list = pygame.sprite.Group()
```

```
my_sprites_list = []

fld = field.copy()
for i in range(size):
    for j in range(size):
        spr = SimpleSprite(i*35,j*35,fld[j][i])
        all_sprites_list.add(spr)
        my_sprites_list.append(spr)
#all_sprites_list.add(player)
done = False

def setPlayerPos(x,y,px,py):
    if my_sprites_list[y*size+x].code == 1:
        return (px,py)
    nextpx, nextpy = (max(min(x,size-2),1), max(min(y,size-2),1))
    my_sprites_list[nextpy*size+nextpx].setCode(4)
    if fld[nextpx][nextpy] == 2:
        fld[nextpx][nextpy] = 0
    elif fld[nextpx][nextpy] == 3:
        pygame.display.quit()
        pygame.quit()
        done=True
    if(nextpx != px or nextpy != py):
        my_sprites_list[py*size+px].setCode(fld[px][py])
    return (nextpx, nextpy)

px,py = (1,1)
my_sprites_list[py*size+px].setCode(4)

clock = pygame.time.Clock()


while not done:

    for event in pygame.event.get():
        try:
            if event.type == pygame.QUIT:
                done = True

            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT:
                    pred, snd = getPred(gauches,"gauche")
                    snd.play()

                elif event.key == pygame.K_RIGHT:
                    pred, snd = getPred(droites,"droite")
                    snd.play()
```

```
                elif event.key == pygame.K_UP:
                    pred, snd = getPred(hauts,"haut")
                    snd.play()

                elif event.key == pygame.K_DOWN:
                    pred, snd = getPred(bass,"bas")
                    snd.play()

                #print(np.array(fld)[py-1:py+2,px-1:py+2])

                if pred=="gauche":
                    while my_sprites_list[(py-1)*size+px].code != 1:
                        px, py = setPlayerPos(px,py-1,px,py)
                        pygame.time.delay(100)
                        screen.fill(WHITE)
                        # Draw sprites
                        all_sprites_list.draw(screen)
                        # Flip screen
                        pygame.display.flip()
                elif pred=="droite":
                    while my_sprites_list[(py+1)*size+px].code != 1:
                        px, py = setPlayerPos(px,py+1,px,py)
                        pygame.time.delay(100)
                        screen.fill(WHITE)
                        # Draw sprites
                        all_sprites_list.draw(screen)
                        # Flip screen
                        pygame.display.flip()
                elif pred=="haut":
                    while my_sprites_list[py*size+px-1].code != 1:
                        px, py = setPlayerPos(px-1,py,px,py)
                        pygame.time.delay(100)
                        screen.fill(WHITE)
                        # Draw sprites
                        all_sprites_list.draw(screen)
                        # Flip screen
                        pygame.display.flip()
                elif pred=="bas":
                    while my_sprites_list[py*size+px+1].code != 1:
                        px, py = setPlayerPos(px+1,py,px,py)
                        pygame.time.delay(100)
                        screen.fill(WHITE)
                        # Draw sprites
                        all_sprites_list.draw(screen)
                        # Flip screen
                        pygame.display.flip()
            # -- Draw everything
```

```
                    # Clear screen
                    screen.fill(WHITE)
                    # Draw sprites
                    all_sprites_list.draw(screen)
                    # Flip screen
                    pygame.display.flip()
                except:
                    print("fin")
                    break

            # Pause
            clock.tick(40)


        pygame.quit()
```

In [10]:
```
pygame.quit() #emergency exit
```

In [ ]: