6ri

Lecteur simple de fichiers audios déclenchés avec des boutons

Nous allons voir deux méthodes:

- en MIDI avec un pad MPX8 et PureData
- en Python avec des boutons physiques de récupération

En MIDI, avec un MPX8 et PureData

L'idée de ce projet est de pouvoir déclencher la lecture de fichiers son (wav) avec un MPX8 de chez AKAI (ou tout autre appareil MIDI pouvant envoyer des notes MIDI) vers un Raspberry Pi 3B, pour ensuite les diffuser sur un système son composé d'une radio à transistors... À noter que l'envoie d'un fichier son doit faire s'arrêter la lecture de celui qui est en cours s'il y en a un.

Schéma de principe

Nous aurions donc:

MPX8 → envoie de note MIDI via USB → **Raspberry Pi 3B** avec PatchboxOS → **PureData** → Patch

Patch PureData

La patch va être développé sur une autre machine pour la bonne raison que je n'ai pas d'écran HDMI à disposition. Le principe: On utilise des fichiers wav. Les objets suivant sont utilisés : Pour charger les fichiers:

- [loadbang] → permet de recharger les morceau au lancement du patch
- [tabplay~] → lit un tableau donné
- [soundfiler] → transforme un fichier en données via le tableau
- tableau → permet de visualiser un tableau

Pour la partie MIDI:

- [notein] → reçoit les notes MIDI et dispense: Note, Vélocité et Channel dans les outlets
- [sel] → exhausteur de goût, écoute ce qui entre et si une valeur correspond à l'un des arguments, envoie un bang sur le outlet correspondant à la place de l'argument dans la liste. Ex: La note MIDI 40 est reçue, celle-ci est en 1ère position dans la liste d'arguments, un bang sera envoyé sur le outlet 1 en partant de la gauche.

La note MIDI 41 est reçue, celle-ci est en 2ème position dans la liste d'arguments, un bang sera envoyé sur le outlet 2, etc.

- [dac~]→ dirige les flux audios vers la sortie son.

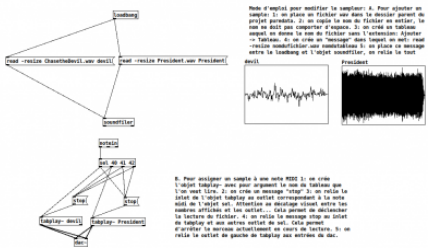
—- Mode d'emploi pour modifier le sampleur: —-

A. Pour ajouter un sample:

1. : on place un fichier wav dans le dossier parent du projet PureData.
2. : on copie le nom du fichier en entier, le nom ne doit pas comporter d'espace.
3. : on créé un tableau auquel on donne le nom du fichier sans l'extension: Ajouter → Tableau.
4. : on crée un “message” dans lequel on met: read -resize nomdufichier.wav nomdutableau
5. : on place ce message entre le loadbang et l'objet soundfiler, on relie le tout

B. Pour assigner un sample à une note MIDI

1. : on crée l'objet tabplay~ avec pour argument le nom du tableau que l'on veut lire.
2. : on relie le inlet de l'objet tabplay au outlet correspondant à la note midi de l'objet sel. Attention au décalage visuel entre les nombres affichés et les outlet... Cela permet de déclencher la lecture du fichier.
3. : on crée un message “stop”
4. : on relie le message stop au inlet du tabplay et aux autres outlet de sel. Cela permet d'arrêter le morceau actuellement en cours de lecture.
5. : on relie le outlet de gauche de tabplay aux entrées du dac.



En Python avec des boutons physiques de récupération

On commence par installer le module “pip” pour Python 3:

```
sudo apt install python3-pip
```

suivi d'un

```
sudo apt install python3-full
```

Puis la bibliothèque Pygame:

```
python3 -m pip install -U pygame --user --break-system-packages
```

La dernière option permet de faire l'installation, sinon, il faut le faire et dans un environnement virtuel de Python....et bof...

Le script python 3.11 qui fonctionne est le suivant (à adapter à vos GPIO, noms de fichiers à déclencher, etc...):

```
#Always comment your code like a violent psychopath will be maintaining it and they know where you live  ;)

from pygame import mixer                                    #importe la classe Mixer depuis le module "pygame" pour gérer les
fichiers son
from gpiozero import Button                                 #importe que l'élément "Button"depuis la bibliothèque GPIOzero

ChasetheDevil = Button(13)                                  #désigne la broche GPIO 13, l'active et lui donne un nom
President = Button(21)                                      #désigne la broche GPIO 21, l'active et lui donne un nom
mixer.pre_init(44100,-16,2, 2048)                           #configure les différents paramètres de alsa pour lire le fichier son. En
deça de 2048, il y a des                                                                    décrochages et la lecture
des fichiers n'est pas fluide.
mixer.init()                                                #créé une instance de la classe "mixer" de la bibliothèque "Pygame"

def chase():                                                #déclare une fonction pour gérer l'arrêt du morceau en cours, son volume,
son chargement e>
    mixer.music.stop()                                      #stoppe le fichier actuellement en lecture
    mixer.music.set_volume(1.0)                             #règle le volume
    mixer.music.load('/home/waxmonde/samples/ChasetheDevil.wav')    #charge le fichier son en indiquant le chemin entier pour pouvoir
démarrer le script depui>
    mixer.music.play()                                      #trouve les bons numéros du loto de hier

def pres():                                                 #déclare une fonction pour gérer l'arrêt du morceau en cours, son volume,
son chargement e>
    mixer.music.stop()                                      #stoppe le fichier actuellement en lecture
    mixer.music.set_volume(1.0)                             #règle le volume
    mixer.music.load('/home/waxmonde/samples/President.wav')        #charge le fichier son en indiquant le chemin entier pour pouvoir
démarrer le script depui>
    mixer.music.play()                                      #répare ta gazinière


while True:                                                 #initialise une boucle "while" (conditionnelle)
    ChasetheDevil.when_pressed = chase                      #lorsque le bouton auquel est assigné ChasetheDevil est pressé, cela
lance la fonction "c>
    President.when_pressed = pres                           #lorsque le bouton auquel est assigné President est pressé, cela lance la
fonction "pres"

pause()                                                     #mets le script en pause, il attend donc une nouvelle instruction en
permanence
```

Puisque notre sortie sera en mono, nous pouvons avoir cette ligne configuration dans le script python:

```
mixer.pre_init(44100,-16, 1, 127)
mixer.init()
```

Où les paramètres sont (fréquence d'échantillonnage, la taille, le nombre de canaux de sortie, la taille du buffer). Dans notre cas, on place la fréquence d'échantillonnage à 44100 Hz, ce qui est de qualité CD, on demande un seul canal de sortie, ce qui fera moins de calculs, un a une taille de buffer assez petite, le tout fait que la latence est assez réduite et permette de pouvoir éventuellement jouer avec les samples.

Notre nouveau script Python est donc le suivant:

```
#Always comment your code like a violent psychopath will be maintaining it and they know where you live  ;)

from pygame import mixer                                                    #importe la classe Mixer depuis le module "pygame" pour gérer
les fichiers son
from gpiozero import Button                                                 #importe que l'élément "Button"depuis la bibliothèque GPIOzero

RadioVaxmonedeu = Button(5)
RadioWaxmonde01 = Button(7)
Waxmonde10 = Button(13)                                         #désigne la broche GPIO 13, l'active et lui donne un nom
WaxmondeRalenti = Button(15)                                                #désigne la broche GPIO 21, l'active et lui donne un
nom

mixer.pre_init(44100,-16, 1, 127)
mixer.init()                                                                #créé une instance de la classe "mixer" de la bibliothèque
"Pygame"

def radiovaxmonedeu():                                                              #déclare une fonction pour gérer l'arrêt du morceau
en cours, son volu>
    mixer.music.stop()                                                      #stoppe le fichier actuellement en lecture
    mixer.music.set_volume(1.0)                                             #règle le volume
    mixer.music.load('/home/waxmonde/samples/radio_vaxmonedeu_01.wav')              #charge le fichier son en indiquant le chemin entier
pour pouvoir démar>
    mixer.music.play()

def radiowaxmonde01():                                                              #déclare une fonction pour gérer l'arrêt du morceau
en cours, son volu>
    mixer.music.stop()                                                      #stoppe le fichier actuellement en lecture
    mixer.music.set_volume(1.0)                                             #règle le volume
    mixer.music.load('/home/waxmonde/samples/radio_waxmonde_01.wav')                #charge le fichier son en indiquant le chemin entier
pour pouvoir démarre>
    mixer.music.play()                                                      #répare ta gazinière


def waxmonde10():                                                                   #déclare une fonction pour gérer l'arrêt du morceau en
cours, son volume, so>
    mixer.music.stop()                                                      #stoppe le fichier actuellement en lecture
    mixer.music.set_volume(1.0)                                             #règle le volume
    mixer.music.load('/home/waxmonde/samples/waxmonde_10.wav')   #charge le fichier son en indiquant le chemin entier pour pouvoir démarrer
le script depuis >
    mixer.music.play()                                                      #trouve les bons numéros du loto de hier

def waxmonderalenti():                                                              #déclare une fonction pour gérer l'arrêt du morceau
en cours, son volu>
    mixer.music.stop()                                                      #stoppe le fichier actuellement en lecture
    mixer.music.set_volume(1.0)                                             #règle le volume
    mixer.music.load('/home/waxmonde/samples/waxmonde_ralenti.wav')                 #charge le fichier son en indiquant le chemin entier pour
pouvoir démarrer>
    mixer.music.play()                                                      #répare ta gazinière


while True:                                                                 #initialise une boucle "while" (conditionnelle)
    RadioVaxmonedeu.when_pressed = radiovaxmonedeu                                  #lorsque le boutton auquel est assigné ChasetheDevil
est pressé, cela>
    RadioWaxmonde01.when_pressed = radiowaxmonde01                                      #lorsque le boutton auquel est assigné President
est pressé, cela>
    Waxmonde10.when_pressed = waxmonde10
    WaxmondeRalenti.when_pressed = waxmonderalenti

pause()
```

Il y a certainement possibilité de simplifier le script avec un système de dictionnaire ou autre.

Prochaine étape: faire en sorte que ce script se lance automatiquement au démarrage du Raspberry Pi!

Lancement automatique du script python au démarrage du Raspberry Pi

Si vous avez testé votre script mais qu'il ne se lance pas et que vous avez ce genre de retour dans la console:

```
 Can't set permissions (436) for /home/patch/waxmonde/.lgd-nfy0, No such file or directory
```

Il est possible qu'il faille donner les droits en lecture/écriture/utilisation au dossier dans lequel se trouve votre script python.

En effet, il y en a besoin pour la gestion des GPIO et créer le fichier .lgd-nfy0. Il faut donc faire un

```
sudo chmod 777 "chemin de votre dossier où dans lequel se situe votre script"
```

On va utiliser la méthode “systemd” (si j'ose dire…) qui gère le lancement de “services” au démarrage du Raspberry Pi (en gros…).

Tout est décrit ici: https://www.youtube.com/watch?v=kQjnesqBMzQ et tout se passera en ligne de commande.

- On crée un nouveau fichier pour le nouveau service:

```
sudo nano /etc/systemd/system/waxmonde.service
```

→ vous pouvez donner le nom du service que vous voulez bien entendu…

- Dans nano, on édite le fichier avec ce qui suit et qui fonctionne au 16/10/2025:

```
[Unit]
Description=Lecteur de samples Waxmonde v1.0
After=alsa-restore.service

[Service]
ExecStart=/usr/bin/python3 /home/patch/waxmonde/gpio_music_box08.py
WorkingDirectory=/home/patch/waxmonde/
User=patch

[Install]
WantedBy=multi-user.target
```

Vous pouvez retrouver les explications sur les différentes sections du code sur les internets.

Il y a par exemple la documentation officielle ici: https://systemd.io/

Et la documentation pour systemd sur debian ici (PatchboxOs étant basée sur Debian) : https://wiki.debian.org/systemd Vous pouvez sauvegarder votre fichier, sachant qu'il est possible qu'il y ait des réglages à faire ultérieurement.

- À la suite de quoi, on donne les droits qu'il faut à ce fichier:

```
sudo chmod 644 /etc/systemd/system/waxmonde.service
```

Vous pouvez vous référer au man de chmod pour plus de détails sur cette commande.

- On démarre le service:

```
sudo systemctl start waxmonde.service
```

systemctl est la commande qui permet de gérer les lancements, arrêts, etc. des différents services à chaud.

- On rend le servie actif au démarrage:

```
sudo systemctl enable waxmonde.service
```

- On vérifie le statu du service, à savoir s'il a été lancé, si cela a fonctionné et si il est actuellement en route ou à l'arrêt:

```
sudo systemctl status waxmonde.service
```

Dans notre cas, on avait ceci en réponse:

```
× waxmonde.service - Lecteur de samples Waxmonde v1.0
     Loaded: loaded (/etc/systemd/system/waxmonde.service; enabled; preset: enabled)
     Active: failed (Result: exit-code) since Sun 2025-05-25 13:18:30 CEST; 8h ago
   Duration: 19.682s
    Process: 585 ExecStart=/usr/bin/python3 /home/waxmonde/samples/gpio_music_box08.py (code=exited, status=1/FAILURE)
   Main PID: 585 (code=exited, status=1/FAILURE)
        CPU: 1.999s

May 25 13:18:29 patchbox python3[585]: pygame 2.1.2 (SDL 2.26.5, Python 3.11.2)
May 25 13:18:29 patchbox python3[585]: Hello from the pygame community. https://www.pygame.org/contribute.html
May 25 13:18:29 patchbox python3[585]: Traceback (most recent call last):
May 25 13:18:29 patchbox python3[585]:   File "/home/waxmonde/samples/gpio_music_box08.py", line 9, in <module>
May 25 13:18:29 patchbox python3[585]:     mixer.init()                                                    #créé une instance de la
classe "mixer" de la b>
May 25 13:18:29 patchbox python3[585]:     ^^^^^^^^^^^
May 25 13:18:29 patchbox python3[585]: pygame.error: ALSA: Couldn't open audio device: Unknown error 524
May 25 13:18:30 patchbox systemd[1]: waxmonde.service: Main process exited, code=exited, status=1/FAILURE
May 25 13:18:30 patchbox systemd[1]: waxmonde.service: Failed with result 'exit-code'.
```

```
May 25 13:18:30 patchbox systemd[1]: waxmonde.service: Consumed 1.999s CPU time.
lines 1-18/18 (END)
```

Ce qui signifie que le service a été lancé, que le script s'est lancé mais qu'il a rencontré une erreur.

L'erreur ici est que alsa ne sait pas sur quel carte son se lancer, alors que le script lorsqu'il est lancé à la main, en ssh, s'exécute sans soucis.

La question a été posée sur le forum de la distribution PatchboxOs, utilisée ici sur le raspberry pi:

https://community.blokas.io/t/how-to-start-a-python-script-at-the-boot-of-patchboxos/5852

Et par ce biais des explications ont été trouvées ici:

https://forums.raspberrypi.com/viewtopic.php?t=360496

Il faut donc créer un fichier de conf dans /etc/, le tout en sudo donc:

Dans un premier temps on va chercher la bonne dénomination de carte son à utiliser par alsa. La commande est la suivante (pas besoin d'être sudo):

```
aplay -l

**** List of PLAYBACK Hardware Devices ****
card 0: vc4hdmi [vc4-hdmi], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
Subdevices: 1/1
Subdevice #0: subdevice #0
card 1: Headphones [bcm2835 Headphones], device 0: bcm2835 Headphones [bcm2835 Headphones]
```

Dans le fichier de conf que l'on va créer, il faudra donc spécifier la carte son ainsi que la connectique (device) qui nous intéressent avec "hw:1,0". Ici, le premier chiffre est pour la carte utilisée, le second, après la virgule, est pour le 'device' utilisé. Il pourrait y avoir plusieurs devices par carte dans le cas d'une carte son externe avec entrées et sorties multiples.

```
sudo touch /etc/asound.conf
```

← la commande sudo permet d'avoir les droits d'écriture, la commande touch créé le fichier dont le nom est précisé après avec son emplacement, ici /etc/asound.conf. C'est à cet endroit que alsa viendra chercher la configuration qui va bien pour lui permettre de diffuser la lecture de fichiers son.

```
sudo nano /etc/asound.conf
```

← la commande nano ouvre un éditeur de texte basique mais puissant en ligne de commande et permet de mettre ce qui suit dans le fichier asound.conf:

```
 pcm.!default {
        type asym
        playback.pcm {
        type plug
        slave.pcm "hw:1,0"
        }
        }
```

On sauvegarde le fichier, dans nano il faut faire : ctrl+O, puis entrée, puis ctrl+X On peut rebooter le raspberry pi (sudo reboot)

On teste sans se connecter en ssh (au cas où et puisque c'est comme cela que cela fonctionnera en production) en appuyant sur un des boutons pour déclencher la lecture d'un fichier son.

On est content.e et on va fêter ça dans un endroit sympa comme son jardin puisqu'on a du linge à étendre!!!

Ajout d'une carte Audio Amp Shim

Le projet va maintenant s'installer dans une radio dont le haut-parleur est sous 4 Ohms pour une puissance de 2W. Il y a une carte qui correspond à ces paramètres at qui s'appelle la Audio Amp Shim, de chez Pimoroni. Elle prend peu de place, communique en i2s et sert d'amplificateur de bonne qualité. Il est à noter que le i2s a besoin de 3 broches GPIO pour fonctionner. Voici la documentation à ce sujet: Pin Out Audio Amp Shim

Ce sont donc des ports que nous ne pourrons pas utiliser pour d'autres fonctionnalités, comme des boutons par exemple. Il faut donc aussi désactiver le SPI, sans quoi, les GPIO normalement utilisés par ce protocole seront vu comme utilisés par

lgpio et le script se lancera puis s'arrêtera.

Il faudra aussi brider le volume de sortie, puisque la puissance admissible par le haut-parleur est inférieure à la puissance maximale délivrable par l'amplificateur de la carte.

Nouvelle configuration de Alsa

Il faut commencer par modifier le fichier /boot/firmware/config.txt pour dèsactiver les autres cartes son.

```
sudo nano /boot/firmware/config.txt

# For more options and information see
# http://rptl.io/configtxt
# Some settings may impact device functionality. See link above for details

# Uncomment some or all of these to enable the optional hardware interfaces
dtparam=i2c_arm=on
dtparam=i2s=on
dtparam=spi=off

# Enable audio (loads snd_bcm2835)


# Additional overlays and parameters are documented
# /boot/firmware/overlays/README

# Automatically load overlays for detected cameras
camera_auto_detect=1

# Automatically load overlays for detected DSI displays
display_auto_detect=1

# Automatically load initramfs files, if found
auto_initramfs=1

# Enable DRM VC4 V3D driver
#dtoverlay=vc4-kms-v3d
#max_framebuffers=2
dtoverlay=hifiberry-dac
dtoverlay=i2s-mmap
force_eeprom_read=0
# Don't have the firmware create an initial video= setting in cmdline.txt.
# Use the kernel's default instead.
disable_fw_kms_setup=1

# Run in 64-bit mode
arm_64bit=1

# Disable compensation for displays with overscan
disable_overscan=1

# Run as fast as firmware / board allows
arm_boost=1

[cm4]
# Enable host mode on the 2711 built-in XHCI USB controller.
# This line should be removed if the legacy DWC2 controller is required
# (e.g. for USB device mode) or if USB support is not required.
otg_mode=1

[all]
```

Il y a des explications par ici: HiFiBerry

Puisque nous n'utilisons plus la sortie casque pour alimenter le haut-parleur, il faut retourner dans le fichier de configuration de alsa. Avant cela, il faut se souvenir que cela n'est pas seulement une marque de levure chimique mais bien un gestionnaire de cartes son. Il y a pas mal de documentation à ce sujet ici: Site du porjet Alsa et ici: Alsasoundrc qui explique comment fonctionne le fichier de configuration de Alsa. Très intéressant!

Nous allons donc voir dans un premier temps comment s'appelle pour Alsa notre carte son.

```
aplay -l
```

Nous retourne ceci:

```
**** List of PLAYBACK Hardware Devices ****
card 0: sndrpihifiberry [snd_rpi_hifiberry_dac], device 0: HifiBerry DAC HiFi pcm5102a-hifi-0 [HifiBerry DAC HiFi pcm5102a-hifi-0]
  Subdevices: 0/1
  Subdevice #0: subdevice #0
```

Il n'y a que cela, donc notre carte son va s'appeler 0:0. Nous modifions donc le fichier /etc/asound.conf

```
sudo nano /etc/asound.conf

pcm.!default {
type asym
playback.pcm {
type plug
slave.pcm "hw:0,0"
}
}
```

Il n'y aura eu qu'à changer “hw:1,0” par: “hw:0,0” et c'est tout! Et on sauvegarde le fichier, comme de bien entendu.

On reboot le raspberry pi pour que la modification du fichier soit prise en compte et cela fonctionne. On s'habitue doucement au succès...

Article extrait de : http://www.lesporteslogiques.net/wiki/ - **WIKI Les Portes Logiques**
Adresse :
http://www.lesporteslogiques.net/wiki/openatelier/projet/sampler_puredata_pour_raspberry_pi_sans_ecran
Article mis à jour: **2025/10/16 09:18**