Rapport de projet de 4e année

Orchestre électronique

Élèves ingénieurs : T.ROJ et J.LETELLIER - IMA

Tuteurs école : X.REDON et T.VANTROYS
Tuteur entreprise: L.PRIEUX

Pour l'année scolaire 2015-2016

Remerciements

Nous souhaiterions remercier très chaleureusement L.PRIEUX de nous avoir fait confiance pour ce projet. Nous lui en sommes très reconnaissant. Ce fut un réel plaisir de travailler avec lui ainsi qu'avec tous ses collaborateurs. Nous espérons que notre travail a été à la hauteur de leurs attentes.

Nous souhaiterions également remercier nos tuteurs de projets, qui nous ont toujours aidés et qui ont toujours su nous aiguiller dans nos choix.

Table des matières

1.Introduction

Les objets connectés et les nouvelles technologies font maintenant partie intégrante de nos vies. Bracelets connectés qui surveillent votre santé, applications qui surveillent votre maison, drones autonome, il y a un objet pour n'importe quel application.

Mais tout cela ne serait pas la sans notre « bonne vieille » informatique ! Ces objets, bien que pour l'heure obsolète, ont toujours suscité un intérêt quelconque pour les puristes. Certains en font des objets décoratifs, d'autres les remettent en service, et certains (comme nous et beaucoup d'autres) en font des instruments de musiques.

En effet, ces appareils se présentent comme des instruments, capables parfois de reproduire une mélodie complexe. Alors que l'heure est à la musique électronique, informatique, M.PRIEUX nous a proposé de réaliser un orchestre « peu conventionnel » avec ces vieux appareils.

En commençant par les lecteurs de disquettes 3"5, en passant par les disques durs mécaniques, les imprimantes matricielles jusqu'à l'objet ultime de la « old generation » – le modem-, nous tenterons, en les rassemblant tous, de jouer de la musique.

Nous commencerons par présenter le projet dans lequel s'inscrit cet orchestre, le projet HUMAINS. Puis, nous développerons nos réalisations techniques, en détaillant le principe de commande de chaque instrument. Et enfin, nous terminerons ce rapport en présentant les problèmes rencontrés et les améliorations possibles.

2.Contexte du projet : Le projet HUMAINS

« I miss my pre-internet brain » (Douglas Coupland)

« HUMAINS » est un projet de création artistique prévu pour 2017. Il a été conçu et écrit par L.PRIEUX. C'est un spectacle de théâtre, de marionnettes et de vidéos. Il se définit comme suit :

« *Parcours croisés de trois individus, confrontés à la révolution numérique dans leurs solitudes et dans leurs conversations quotidiennes, et d'une marionnette, métaphore de l'Être Humain dans sa globalité, qui apparaît et disparaît, revient comme un fil rouge, et nous donne à voir, poétiquement, nos propres questionnements. Des projections vidéos, compilations de vidéos glanées sur la toile, nous racontent la multitude humaine connectée. Un spectacle qui questionne avec intelligence, humour et poésie nos comportements face aux technologies numériques.* »

Lors de notre premier entretien, M.PRIEUX nous a développé la principale idée de ce spectacle : montrer aux gens à quel point la technologie change nos vies, et surtout montrer ce que les objets connectés et les nouvelles technologies nous apporte.

C'est donc dans ce contexte que s'inscrit notre projet. Le but étant de créer un orchestre « peu ordinaire » avec de vieux appareils informatiques.

3. Cahier des charges

L'objectif final du projet est d'obtenir un dispositif comprenant une entrée MIDI et comprenant des sorties sur lesquelles on peut venir brancher différents instruments.

Lors d'un premier entretien avec tous nos tuteurs, nous avons pu leurs poser des questions afin de mieux préciser le cahier des charges. Le sujet étant vaste, cet entretien nous a permis de mieux cerner les objectifs, les tenants et les aboutissants.

La première question qui est venue naturellement dans la conversation est le choix du cœur du système. En effet, si nous utilisons comme cœur du dispositif un ARDUINO, il aurait fallu rajouter une Raspberry Pi afin de gérer le logiciel, ce qui ne rentre pas dans le côté « portatif » du système. Nous optons donc pour une Raspberry Pi seule. De plus, une Raspberry Pi offre beaucoup plus de possibilités logicielles et matérielles qu'un Arduino combiné à une Raspberry Pi.

Ainsi, notre dispositif devra contenir :

- Une Raspberry Pi : elle devra gérer les différents serveurs Web et gérer les canaux MIDI.

- Une entrée MIDI afin de pouvoir venir brancher un clavier MIDI

- Une sortie MIDI afin de pouvoir venir brancher un synthétiseur MIDI (une autre proposition a été faite : venir brancher une autre Raspberry Pi muni du logiciel Rosegarden simulant un synthétiseur MIDI).

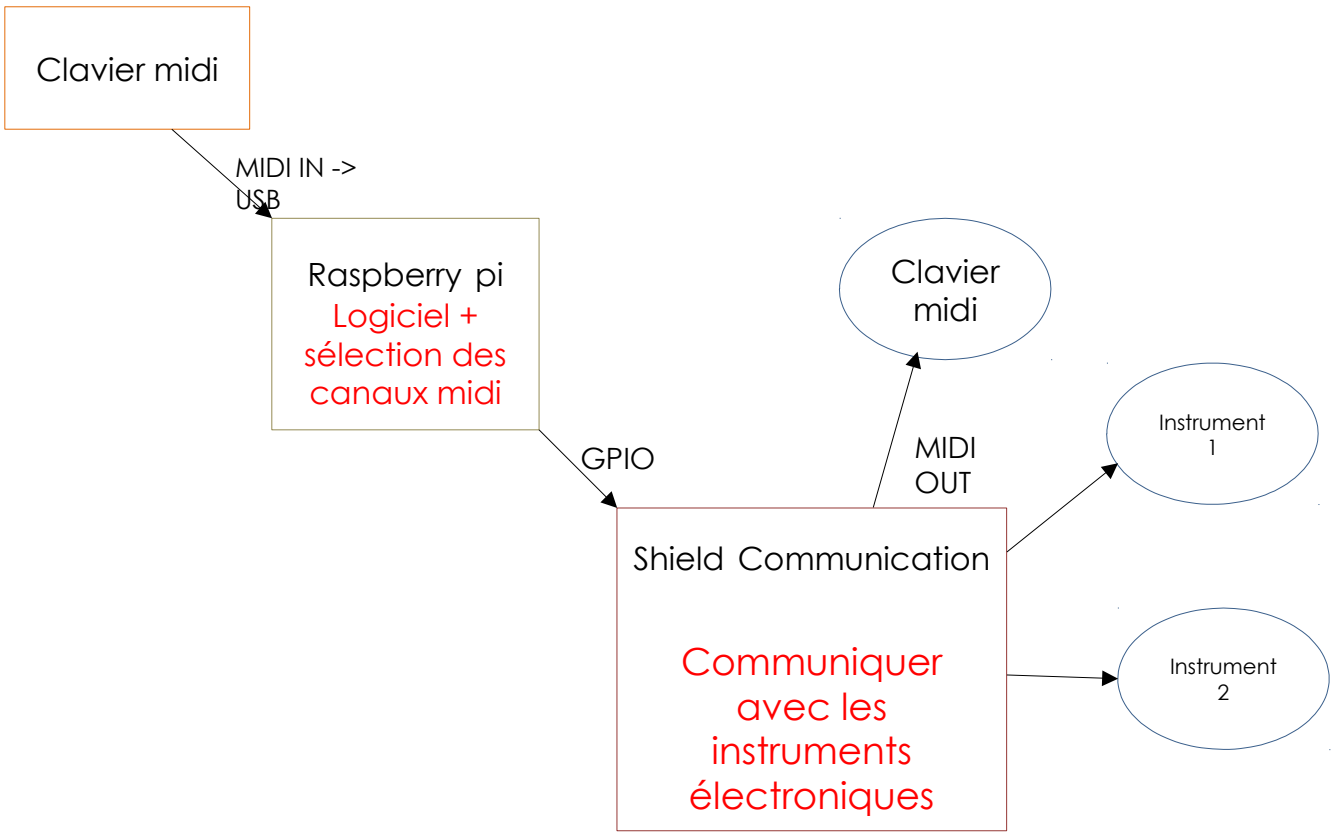
- Une carte permettant de venir brancher les différents instruments.

Ce dispositif devra être capable de :

- Jouer de la musique en direct avec les différents instruments

- Enregistrer un morceau et le jouer en différé.

De ce cahier des charges, nous avons donc pu construire l'architecture de notre système :



Le cahier des charges étant posé, nous pouvons donc détailler l'ensemble de nos réalisations techniques.

4. Réalisations techniques

Dans cette partie, nous détaillerons les instruments un par un, en précisant leur protocole de communication et ce que nous avons fait pour pouvoir les contrôler. Nous détaillerons ensuite ce que nous avons fait afin de pouvoir lire et jouer un fichier MIDI. Enfin, nous détaillerons la commande par l'interface WEB.

4.1. Schéma de base

Voici le schéma de base du montage avec les Raspberry Pi avec les explications :

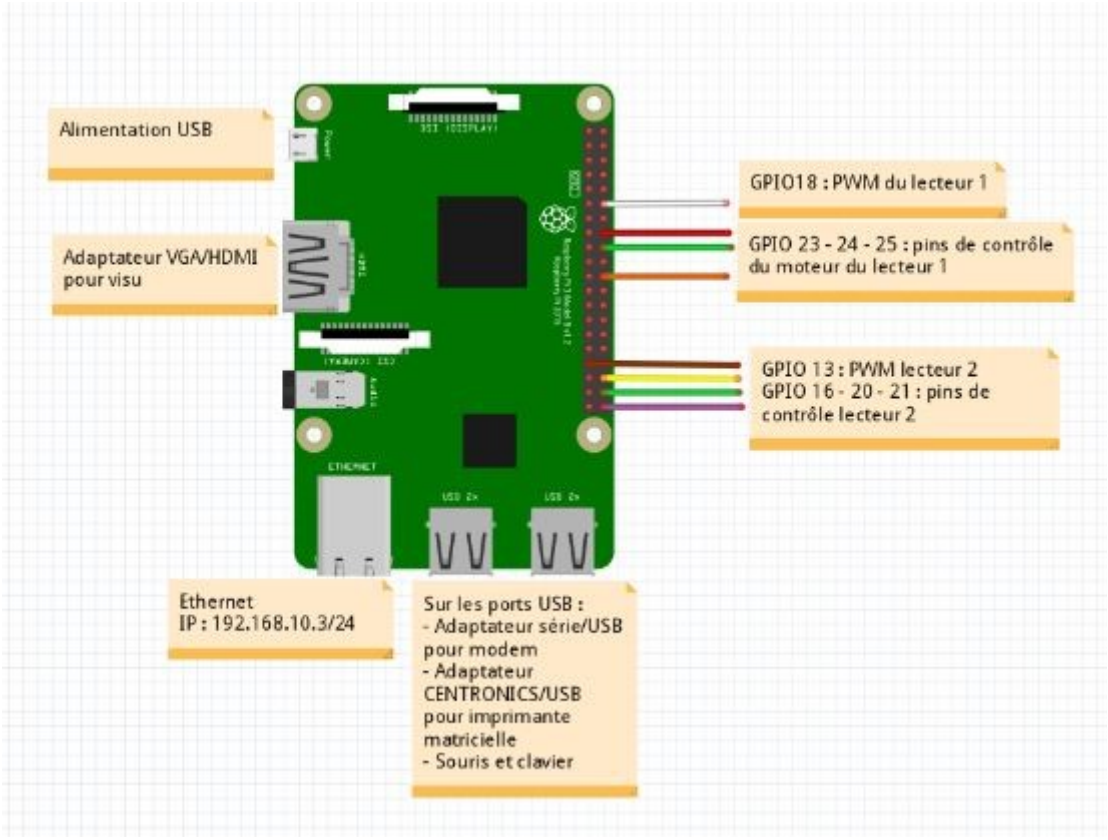


Illustration 1: Montage des instruments avec la Raspberry Pi

Pour coder durant le projet, nous utilisons le langage Python et le langage C. Le langage Python est utilisé parce qu'il est pré-installé sur les Raspberry Pi. Le langage C est utilisé pour les choses plus complexes qu'on ne peut pas effectuer en Python.

4.2. Lecteurs de disquettes

4.2.1. Description de la technique utilisée

Les lecteurs de disquettes sont munis de 2 moteurs :

- Un moteur qui tourne à 300 tr/min afin de permettre un accès plus rapide lors d'une requête.

- Un moteur pas à pas qui permet de déplacer la tête de lecture/écriture précisément sur une piste de la disquette.

Illustration 2: Lecteur de disquette 3"5

Lors de la lecture de la documentation (cf. annexe 2), nous pouvons découvrir qu'un lecteur de disquette fait déjà du bruit durant son fonctionnement. En effet, il a un taux de vibration de 1 octave/déplacement de la tête de lecture. Autant se servir de cette vibration pour faire du bruit.

Les lecteurs de disquettes sont munis d'un moteur pas à pas (cf. annexe 1 pour ses caractéristiques). Ce qui caractérise un moteur pas à pas, c'est son pas de rotation[1]. Si nous mettons un signal suffisamment haut en fréquence afin de sauter des pas, la mécanique fait qu'il va vibrer encore plus, à la fréquence du signal. C'est sur ce principe que nous nous sommes basés afin de réaliser des notes avec un lecteur de disquette.

Ils sont munis de connecteurs 34 pins à angle droit afin de pouvoir communiquer avec le PC. Certaines broches sont utiles pour pouvoir commander le moteur tels que les broches 12, 16, 18 et 20 (cf. annexe 3)

1. Pas de rotation : De combien tourne le moteur lorsqu'il reçoit 1 impulsion du signal.

4.2.2. Réalisation avec la Raspberry Pi

Afin de faire vibrer les lecteurs de disquettes, nous utilisons une MLI[2]. La Raspberry nous offre la possibilité de créer des MLI sur n'importe quel broche. Dans le code que nous utilisons, nous utilisons la bibliothèque de base de contrôle des GPIO[3] afin de les créer.

Pour les GPIO, il faut d'abord préciser quelle numérotation il faut utiliser. Ensuite, il faut leur attribuer leur fonction (entrée ou sortie). Puis, on peut leur affecter des valeurs (*GPIO.LOW* ou *GPIO.HIGH*). Pour les déclarer en PWM (MLI), il faut le préciser.

Jusqu'à présent, nous utilisions qu'un seul lecteur de disquette. Nous avons donc décidé d'en rajouter un. Afin de contrôler 2 lecteurs de disquettes, nous devons passer par des threads. Un thread sera associé au fonctionnement d'un lecteur de disquette. Pour que cela fonctionne, nous utilisons la bibliothèque Python Thread afin de pouvoir effectuer les redéfinitions de méthodes associés aux threads : la méthode *__init__* et la méthode *run.* Les 2 threads sont alors synchronisés pour s'attendre entre eux.

Un exemple en annexe 4 illustre parfaitement l'ensemble de ces propos.

4.3. Disque dur

Illustration 3: Disque dur

Les disques durs sont munis d'une pile de disques (la mémoire), et d'une tête permettant de se placer précisément sur ces disques.

Nous pouvons contrôler la tête du disque dur de la même manière que la tête de lecture d'un lecteur de disquette, en lui envoyant une MLI. Et on reproduit exactement un son à la fréquence demandée.

2. MLI : Modulation à largeur d'impulsion - 3. GPIO : General Purpose Input Output – Entrées/sorties

Cependant, nous avons dû ajouter en amont du disque dur un amplificateur de tension afin d'avoir un volume audible car trop faible.

4.4. Imprimante matricielle

4.4.1 Description de la technique utilisée

Illustration 4: Imprimante matricielle EPSON LQ-570+

Notre imprimante matricielle est une EPSON LQ-570+. Elle est composée de 24 aiguilles. Elle est munie d'une interface parallèle et d'une interface de type Centronics.

Les aiguilles bougent en fonction du caractère à imprimer. Elle utilisera peut-être 8 aiguilles pour une lettre alors qu'elle utilisera 16 aiguilles pour un chiffre ou un caractère spécial par exemple.

Elle peut être commandée par le langage ESC/P2. Le langage ESC/P2 est un langage propriétaire de description de page créé par EPSON. Avec ce langage, on peut contrôler la vitesse de la tête d'impression, mais on peut aussi contrôler la mécanique et la précision de l'impression par exemple. (cf. annexe 5 pour la description du langage).

4.4.2. Réalisation avec la Raspberry Pi

Nous utiliserons donc ce langage afin de communiquer avec l'imprimante. Nous communiquons avec l'imprimante via l'interface Centronics. Au début, nous tentions d'envoyer juste les caractères en mode normal. Mais le mode par défaut de l'imprimante utilise les 24 aiguilles pour écrire tous les caractères. En lisant un peu plus en détail la documentation de l'imprimante et du langage, nous nous sommes aperçus qu'il existait un mode graphique, où l'imprimante utilisait juste le nombre d'aiguilles en fonction du caractère reçu.

Afin de dire à l'imprimante le nombre d'aiguilles à utiliser, nous passons en mode graphique. Il faut ensuite préciser la densité du point (en pixels), et préciser la position de départ. Ensuite, en fonction du caractère reçu, elle utilisera le bon nombre d'aiguilles.

Et en fonction des données que l'on envoie, on remarque que la vibration n'est pas la même. Le code qui est à l'origine de ce fonctionnement est présent dans les annexes (cf. annexe 6).

4.5. Le Modem

4.5.1. Description technique de la méthode utilisée

Illustration 5: Modem 56k Message de U.S Robotics

Le Modem[4] 56k est l'illustration parfaite de l'appareil « old school ». Il était utilisé lors de l'apparition des connexions Internet. Il était en charge d'établir la connexion avec le Central de communication pour établir la liaison entre les 2 postes, puis cryptait les données qui passaient. Il est muni d'une interface série RS-232.

Le bruit qu'il émettait était caractéristique des transmissions. A l'époque, tout était en analogique. Les sons dépendaient des commandes qu'il utilisait.

Les commandes pour le contrôler sont des commandes Hayes (cf. annexe 7). Il suffit donc d'utiliser ces commandes pour contrôler le Modem.

4.5.2. Réalisation avec la Raspberry Pi

Nous utilisons donc une communication série avec le Modem pour le contrôler. Il faut d'abord entrer en mode Commande, puis envoyer les commandes que l'on désire. Le programme présent en annexe 8 nous illustre ce principe.

4.Modem : Modulateur - Démodulateur

4.6. Clavier d'ordinateur

Lors de la présentation du cahier des charges, nous avons mentionné le fait que notre dispositif devait contenir une entrée MIDI afin de pouvoir y venir brancher un clavier MIDI. Pour coller totalement au thème du sujet et du projet, nous avons remplacé ce clavier MIDI par un clavier d'ordinateur.

Lorsque l'on appuie sur une touche, cela déclenche un événement matériel que l'ordinateur traite en conséquence. Ce sont ces événements qui nous intéressent. En utilisant le module PyGame de Python, nous pouvons récupérer la touche sur laquelle on appuie. Cette touche est convertie en note via une table d'association touche/fréquence, et elle est envoyée par un signal PWM aux lecteurs de disquettes. Le programme traitant cette partie est présent en annexe 9.

4.7. Le protocole MIDI

4.7.1. Description générale

Illustration 6: Logo de l'interface MIDI

Le protocole MIDI[5] est un protocole de communication dédié à la musique. C'est également un format de fichier utilisé entre les logiciels et les instruments MIDI ou les séquenceurs. Apparu dans les années 1980, c'est devenu un standard très répandu dans le matériel électronique de musique.

Informatiquement, il s'agit d'une liaison série symétrique à 31,25 kb/s. Le standard de connectique utilisé est le DIN-5. Il est possible de jouer sur 16 instruments différents avec 1 liaison.

5.MIDI : Musical Instrument Digital Interface

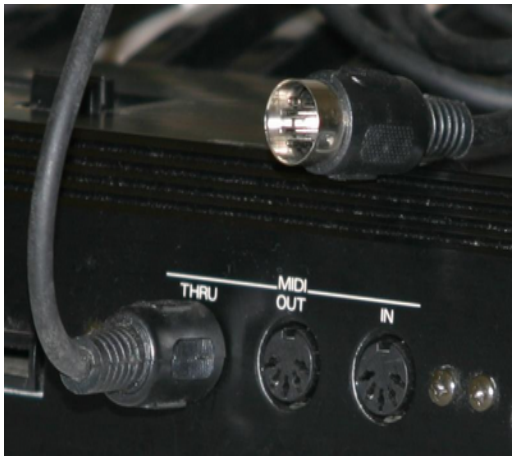


Illustration 7: Illustration des ports MIDI

4.7.2 Format des messages MIDI

Nous avons précisé que le format MIDI était aussi un format de fichier. Nous allons utiliser ces fichiers afin de détecter les notes jouées et les jouer sur les instruments.

Il existe 3 types de fichiers MIDI : le type 0 est pour les fichiers ne contenant qu'une seule piste sur les 16 canaux. Le type 1 est pour les fichiers où les canaux sont séparés, et les pistes sont jouées simultanément.

Le type 2 est pour les fichiers où les canaux sont séparés, mais les pistes sont jouées séquentiellement. Le type est le plus utilisé.

Un fichier MIDI contient :

- une plage d'entête « MThd » (contient le tempo, la résolution de temps, …)
- une plage de données « MTrk »

Dans la plage de données, les notes sont codées sous la forme d'événements (cf. annexe 10). Il est possible d'avoir au début des fichiers MIDI une plage de Métadonnées contenant le titre, l'auteur, etc.

4.7.3 Réalisation avec la Raspberry Pi

L'algorithme Python pour décrypter le fichier MIDI est assez long et complexe. En résumé, il s'agit d'un algorithme basé sur le résultat d'une commande Shell qui analyse le fichier MIDI et retranscrit le fichier sous la forme d'un fichier texte contenant tous les événements du fichier. Ce sont des lignes de la forme suivante :

midi.noteOnEvent(time= < time>, channel = < channel>, data = [< numéro de la note > , < hauteur >]

Ainsi, l'algorithme fait de l'analyse de chaînes afin de récupérer toutes les valeurs importantes du fichier telles que toutes les notes jouées durant le morceau, les temps de déclenchement des notes, etc…

Notre fonction retourne 3 tableaux et 2 entiers qui sont représentatifs du fichier MIDI (notes, temps, hauteur de notes, résolution temporelle et clé utilisée). Le tout est envoyé par les threads aux lecteurs de disquettes afin de jouer la partition. Afin de faire la liaison entre les notes MIDI et les notes classiques, nous utilisons un dictionnaire.

Le programme est présent dans notre archive GIT.

4.8. Interface Web pour le contrôle des instruments

Maintenant que nous arrivons à contrôler tous les instruments et que nous arrivons à faire de la musique avec à l'aide d'un fichier MIDI, nous devons créer une interface afin de commander ces instruments.

Nous avons donc décidés d'utiliser un serveur web sur la Raspberry Pi (Lighttpd) afin d'y placer la page de contrôle des instruments. Elle est assez minimaliste pour l'instant et contient juste les boutons afin d'activer les différentes fonctionnalités de notre système. Nous utilisons les langages HTML, PHP et CSS pour coder cette page.

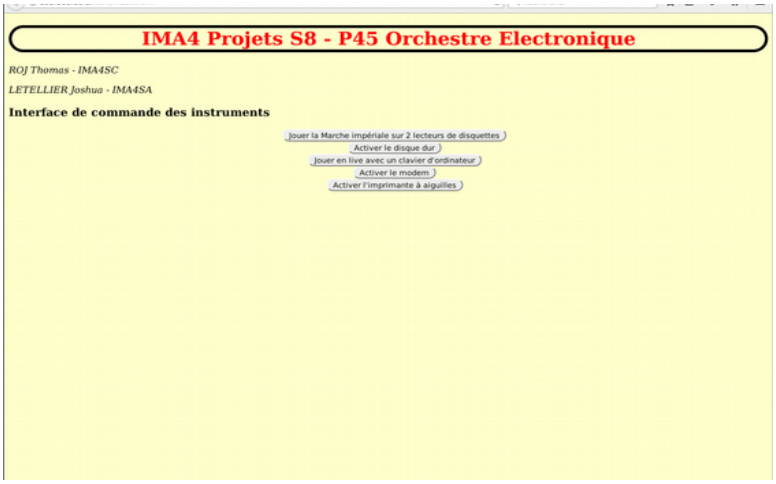


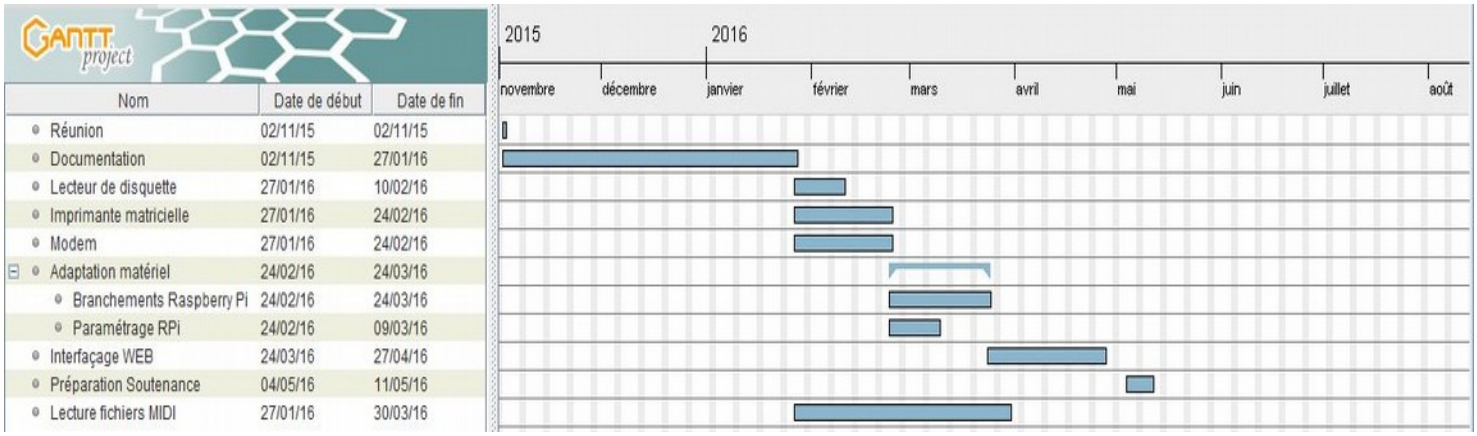
Illustration 8: Page Web de contrôle des instruments

Un serveur CGI[6] est aussi installé afin de faire la liaison entre le site Web et les programmes a exécuter. Il contient tous les interpréteurs dont nous avons besoin afin d'exécuter nos programmes (Python et C principalement).

6. CGI : Common Gateway Interface

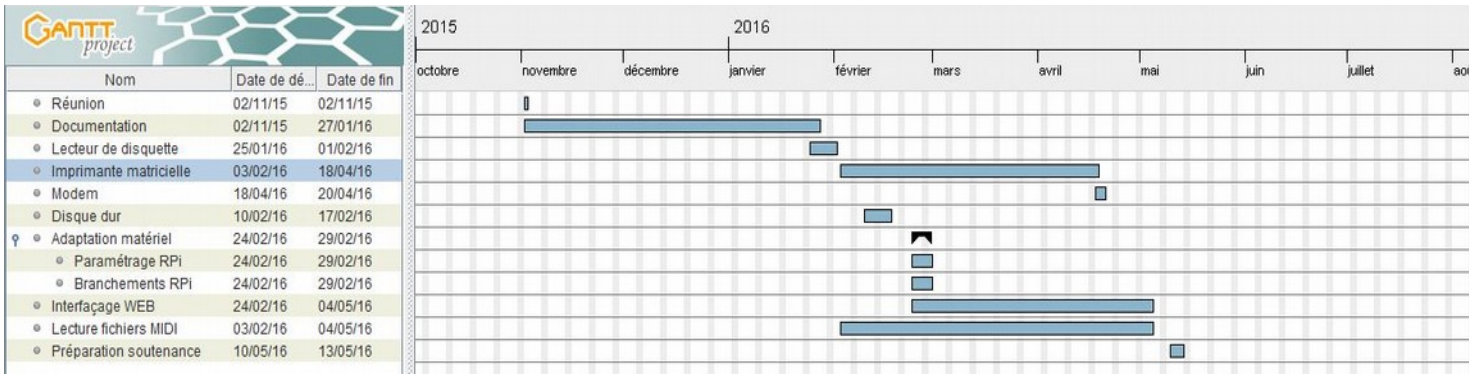
5. Gestion de projet

Voici le calendrier prévisionnel du début de projet :



Nom	Date de début	Date de fin
Réunion	02/11/15	02/11/15
Documentation	02/11/15	27/01/16
Lecteur de disquette	27/01/16	10/02/16
Imprimante matricielle	27/01/16	24/02/16
Modem	27/01/16	24/02/16
Adaptation matériel	24/02/16	24/03/16
Branchements Raspberry Pi	24/02/16	24/03/16
Paramétrage RPi	24/02/16	09/03/16
Interfaçage WEB	24/03/16	27/04/16
Préparation Soutenance	04/05/16	11/05/16
Lecture fichiers MIDI	27/01/16	30/03/16

Et voici le calendrier final de notre projet :



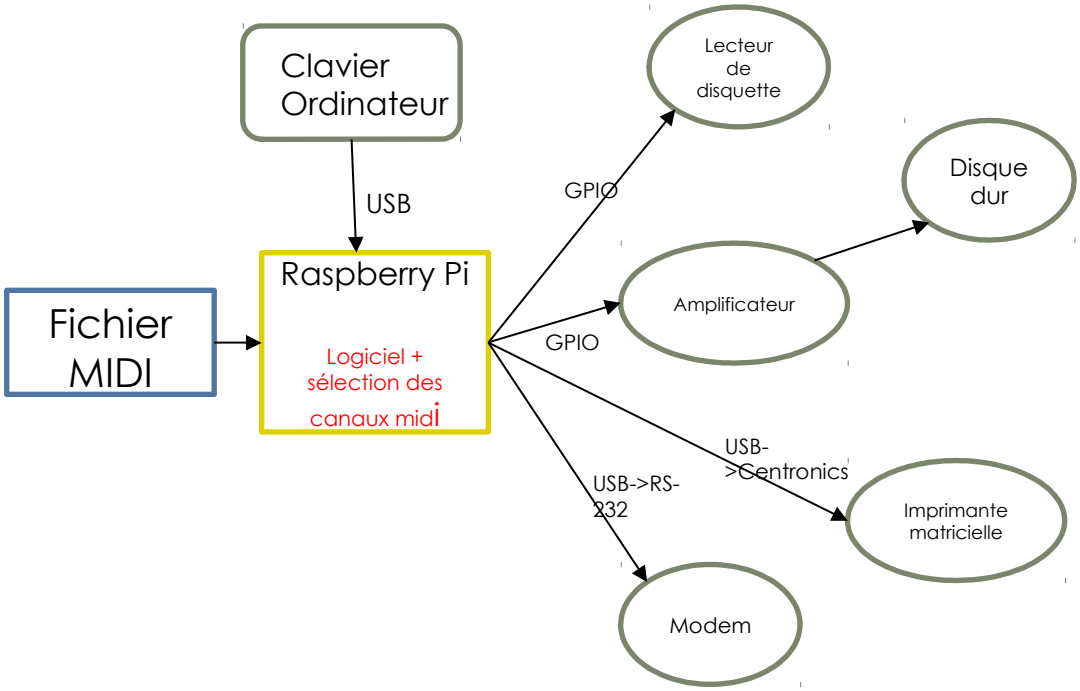
Nom	Date de dé...	Date de fin
Réunion	02/11/15	02/11/15
Documentation	02/11/15	27/01/16
Lecteur de disquette	25/01/16	01/02/16
Imprimante matricielle	03/02/16	18/04/16
Modem	18/04/16	20/04/16
Disque dur	10/02/16	17/02/16
Adaptation matériel	24/02/16	29/02/16
Paramétrage RPi	24/02/16	29/02/16
Branchements RPi	24/02/16	29/02/16
Interfaçage WEB	24/02/16	04/05/16
Lecture fichiers MIDI	03/02/16	04/05/16
Préparation soutenance	10/05/16	13/05/16

On peut remarquer la prise importante de notre temps pour l'imprimante matricielle, qui nous a causé beaucoup plus de difficultés que prévu (mauvaise communication, pas le bon matériel …). On peut aussi remarquer l'arrivée tardive du modem (retard du matériel) puis la courte durée du travail effectué sur ce dernier du fait qu'il était pas possible d'engager une communication complète. La lecture des fichiers MIDI a pris le temps que nous avions prévus.

Malgré les difficultés matérielles et logicielles que nous avons pu rencontrer, nous avons (presque) tenus notre calendrier prévisionnel.

6. Conclusion

Voici l'architecture de notre système à la fin du projet.



Nous pouvons remarquer l'absence du Shield Communication. Celui-ci a été supprimé du projet car nous pouvons brancher les instruments directement sur la Raspberry Pi sans dégrader celle-ci (aucun risque de surtension ou sur-courant dû à des dégradations internes des instruments).

Bien sûr, ce projet n'est pas abouti. L'étude et la mise en musique des fichiers MIDI est à perfectionner, du fait que l'algorithme n'est pas universel (les fichiers MIDI ne sont pas tous les mêmes, problème de formatage des fichiers). De plus, le système n'est pas transportable dans l'état. La création d'un support est donc à prévoir. L'interface Web est aussi à perfectionner, tant au niveau graphique qu'au niveau fonctionnel. L'affichage des notes serait une fonctionnalité à ajouter, ainsi qu'un menu montrant les instruments connectés.

Ce projet fût très intéressant. L'idée de rapprocher ancienne et nouvelle technologie était un peu surprenante au début mais c'était une très bonne idée. Nous avons pu sortir ces instruments de leur usage quotidien afin qu'ils deviennent, peut-être un jour, des « stars » lors d'un concerto.

Annexes

Annexe 1 : Caractéristique du moteur pas à pas du lecteur de disquette TEACFD235HF-A291

Head position mechanism	Stepping motor and lead screw
Stepping motor	4-phase, 20 steps per revolution
Stepping motor drive	2 steps per track
Track 00 detection method	Photo-interrupter
Track to track time	3ms (excludes settling time, refer to item 8.3.4)
Settling time	15ms or less (excludes track to track time)
Average track seek time	94ms (includes settling time)

Annexe 2 : Caractéristiques de vibration du moteur pas à pas

Vibration	14.7m/s² (1.5G)l or less (10 ~ 100Hz, 1 octave/m sweep rate)	—	19.6m/s² (2G) or less (10 ~ 100Hz, 1/4 octave/ m sweep rate)
	9.8m/s² (1.0G) or less (100 ~ 200Hz, 1 octave/ m sweep rate)		
	4.9m/s² (0.5G) or less (200 ~ 600Hz, 1 octave/ m sweep rate)		

Annexe 3 : Correspondance entre les broches du FDD et leur fonction

Pin Nos.	Signals	Pin Nos.	Signals	Direc
1	NC	2	NC	–
3	P. key	4	NC	–
5	NC	6	NC	–
7	0V	8	INDEX	Output
9	0V	10	NC	–
11	0V	12	DRIVE SELECT 1	Input
13	0V	14	NC	–
15	0V	16	MOTOR ON	Input
17	0V	18	DIRECTION SELECT	Input
19	0V	20	STEP	Input
21	0V	22	WRITE DATA	Input
23	0V	24	WRITE GATE	Input
25	0V	26	TRACK 00	Output
27	0V	28	WRITE PROTECT	Output
29	0V	30	READ DATA	Output
31	0V	32	SIDE ONE SELECT	Input
33	0V	34	DISK CHANGE	Output

Annexe 4 : Code d'exemple d'utilisation de 2 lecteur de disquette

```
from threading import Thread
import time
from RPi import GPIO
class DD(Thread):
        #Constructeur
        def __init__(self, pin, t_notes, t_tps):
                Thread.__init__(self);
                self.pin = pin
                self.t_notes = t_notes
                self.t_tps = t_tps
         #Méthode appelée par le thread
        def run(self):
                p = GPIO.PWM(self.pin,220)
                p.start(15)
                i = 0
                while(i<len(self.t_notes)-1):
                        p.ChangeFrequency(self.t_notes[i])
                        time.sleep(self.t_tps[i])
                        i+=1
                GPIO.cleanup()
notes=[220,1,220,1,220,1,174.5,1,261.5,1,220,1,174.5,1,261.5,1,220,1]
tps=[1,0.02,1,0.02,1,0.02,0.4,0.05,0.6,0.05,0.4,0.05,0.6,0.05,0.4,0.05,0.
6,0.05]
GPIO.setmode(GPIO.BCM)
#DD1
GPIO.setup(23,GPIO.OUT)
GPIO.setup(24,GPIO.OUT)
GPIO.setup(25,GPIO.OUT)
#DD2
GPIO.setup(16,GPIO.OUT)
GPIO.setup(20,GPIO.OUT)
GPIO.setup(21,GPIO.OUT)
#PWM
GPIO.setup(18,GPIO.OUT)
GPIO.setup(13,GPIO.OUT)
#Act DD1
GPIO.output(23,GPIO.LOW)
GPIO.output(24,GPIO.LOW)
GPIO.output(25,GPIO.LOW)
#Act DD2
GPIO.output(16,GPIO.LOW)
GPIO.output(20,GPIO.LOW)
GPIO.output(21,GPIO.LOW)
#Creation des threads
thread1 = DD(18,notes,tps)
thread2 = DD(13,notes,tps)
#Lancement des threads
thread1.start()
thread2.start()
#Synchro
thread1.join()
thread2.join()
```

Annexe 5 : Description des commandes du langage ESC/P2 utilisées

ESC (G Select graphics mode

ESC/P 2

Format

ASCII	ESC	(	G	n_L	n_H	m
Hex	1B	28	47	n_L	n_H	m
Decimal	27	40	71	n_L	n_H	m

Parameter range

$n_L = 1$
$n_H = 0$
m = 1, 49

Function

Selects graphics mode (allowing you to print raster graphics)

ESC (V Set absolute vertical print position

ESC/P 2

Format

ASCII	ESC	(	V	n_L	n_H	m_L	m_H
Hex	1B	28	56	n_L	n_H	m_L	m_H
Decimal	27	40	86	n_L	n_H	m_L	m_H

Parameter range

$n_L = 2$, $n_H = 0$
$0 \leq m_L \leq 255$, $0 \leq m_H \leq 127$

Function

Moves the vertical print position to the position specified by the following formula:

(vertical position) = (($m_H \times 256$) + m_L) × (defined unit) + (top-margin position)

ESC (U Set unit

ESC/P 2

Format

ASCII	ESC	(	U	n_L	n_H	m
Hex	1B	28	55	n_L	n_H	m
Decimal	27	40	85	n_L	n_H	m

Parameter range

n_L = 1, n_H = 0

m = **5**, 10, 20, 30, 40, 50, 60

Function

Sets the unit to m / 3600 inch. The printer uses this unit when moving the print position, setting the page length, and setting the top and bottom margins with the following commands: ESC (V, ESC (v, ESC \, ESC $, ESC (C, ESC (c, **<MOVX>**, and **<MOVY>**.

ESC . c v h m n_L n_H d_1 d_2 . . . d_k

c = 0	Selects full graphics mode; all data bytes are treated as print data
1	Selects run length encoded compressed mode; data treated as follows: counter byte, data, counter byte, data
v	Specifies vertical dot density (independent of number of pinsin head) $(\text{vertical dot density}) = \frac{3600}{v}$ dpi
h	Specifies horizontal dot density (independent of number of pins in head) $(\text{horizontal dot density}) = \frac{3600}{h}$ dpi
m	Specifies vertical dot count (1, 8, or 24)
n_L, n_H	Specifies horizontal dot count $(\text{horizontal dot count}) = ((n_H \times 256) + n_L)$
d_1 . . . d_k	Data or counter / data combination

Annexe 6 : Programme de contrôle de l'imprimante matricielle

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <termios.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <strings.h>

struct termios saveterm;

int init_serial(char *device,int speed) {
// Fonction pour initialiser le port Serie
        struct termios new;
        int fd=open(device,O_RDWR|O_NOCTTY);// On
ouvre un descripteur de fichier en mode lecture
et ecriture, ici ça sera notre port série
        if(fd<0){perror(device); exit(-1);}
        tcgetattr(fd,&saveterm); // save current
port settings
        bzero(&new,sizeof(new));
        new.c_cflag=CLOCAL|CREAD|speed|CS8;
        new.c_iflag=0;
        new.c_oflag=0;
        new.c_lflag=0;  // set input mode (non-
canonical, no echo,...)
        new.c_cc[VTIME]=0; // inter-character
timer unused
        new.c_cc[VMIN]=1; // blocking read until
1 char received
        tcflush(fd, TCIFLUSH);
        tcsetattr(fd,TCSANOW,&new);
        return fd;
}

void close_serial(int fd) { // fermeture du Port
Série
        tcsetattr(fd,TCSANOW,&saveterm);
        close(fd);
}
int main(void) {
        int fd = init_serial("/dev/usb/lp0",
9600);
        int cpt_data = 0;
        printf("Debut de contrôle imprimante\n");
        unsigned char c=0x1B; //ESC
        unsigned char e = 0x0D; //CR
        unsigned char f = 0x0A; //LF
        unsigned char g = 0x28; // (
        unsigned char h = 0x47; // G
        unsigned char i = 0x01; //1
        unsigned char j = 0x00; //0
        unsigned char de = 0x02; //2
        unsigned char s = 0x7F; //127
        unsigned char v = 0x40; //64
        unsigned char ae = 0x56; //V
        unsigned char af = 0x55; //U
        unsigned char p = 0x2E; //.
        unsigned char hu = 0x08;
        unsigned char full_data = 0xFF;
        unsigned char empty_data = 0x00;
        unsigned char semi_data = 0x55;
        unsigned char data = 0x0F;
        unsigned char data_t = 0xF0;
        unsigned char data_o = 0xA1;
        unsigned char data_a = 0x22;
        unsigned char data_p = 0xB7;
        unsigned char fin = 0x0C; //FF
        unsigned char nh = 0xC8;
        write(fd, &c, 1); //ESC
        write(fd, &g, 1); // (
        write(fd, &h, 1);
        write(fd, &i, 1);
        write(fd, &j, 1);
        write(fd, &i, 1); //ESC ( G 1 0 1 pour
passer en mode graphique
        write (fd, &c,1);
        write(fd, &g, 1);
        write(fd, &af, 1);
        write(fd, &i, 1);
        write(fd, &j, 1);
        write(fd, &i, 1);
        write(fd, &j, 1); //ESC ( U 10 10 dot
density
        write(fd, &c, 1);
        write(fd, &g,1);
        write(fd, &ae, 1);
        write(fd, &de, 1);
        write(fd, &i, 1);
        write(fd, &s,1);
        write(fd, &v, 1); //ESC ( V 2 0 127 64
position de depart
        write(fd, &c, 1);
        write(fd, &p, 1);
        write(fd, &j, 1);
        write(fd, &i, 1);
        write(fd, &j, 1);
        write(fd, &i, 1);
        write(fd, &j, 1);
        write(fd, &hu, 1);
        write(fd, &i, 1);
        write(fd, &j, 1);
        write(fd, &c, 1);
        write(fd, &p, 1);
        write(fd, &j, 1);
        write(fd, &i, 1);
        write(fd, &j, 1);
        write(fd, &i, 1);
        write(fd, &j, 1);
        write(fd, &hu, 1);
        write(fd, &i, 1);
        write(fd, &nh, 1);
        while (cpt_data < 200) {
                write(fd, &(full_data), 1);
                write(fd, &(data_o), 1);
                write(fd, &(data), 1);
                cpt_data++;
        }
        cpt_data = 0;
        while(cpt_data < 200) {
                write(fd, &(semi_data), 1);
                write(fd, &(data_o), 1);
                write(fd, &(full_data), 1);
                cpt_data++;
        }
        cpt_data = 0;
        while (cpt_data < 200) {
                write(fd, &(data), 1);
                write(fd, &(data_p), 1);
                write(fd, &(data_t), 1);
                cpt_data++;
        }
        cpt_data = 0;
        while(cpt_data < 200) {
                write(fd, &(data_p), 1);
                write(fd, &(full_data), 1);
                write(fd, &(semi_data), 1);
                cpt_data++;
        }
        close_serial(fd);
        return 0;}
```

Annexe 7 : Commandes Hayes pour la commande des MoDems

Basic Commands

<control key>S Stop or restart help screens.

<control key>C *or*
<control key>K Stop help screens.

S Use in conjunction with *D*, *S*, or *&* commands (or just AT) to display a basic command list; online help.

A Manual Answer: goes off hook in answer mode. Pressing any key aborts the operations.

A/ Re-executes the last issued command. Used mainly to redial. Does not require the AT prefix or a Carriage Return.

Any key Aborts off-hook dial/answer operation and hangs up.

AT Required command prefix, except with A/, +++, and A>. Use alone to test for OK result code.

B*n* **U.S./ITU-T answer sequence**

*	*B0*	*ITU-T answer sequence*
	B1	U.S. answer tone

D*n* **Dials the specified phone number, includes the following:**

	0-9	Numeric digits
	#, *	Extended touch-tone pad tones
	L	Dials the last dialed number
*	P	Pulse (rotary) dial
	R	Originates call using answer (reverse) frequencies
	Sn	Dials the phone number string stored in NVRAM at position n (n = 0–3); phone numbers are stored with the &Zn=s command
	T	Tone dial
	,	(Comma) Pause, See S8 definition; which it is linked to
	;	(Semicolon) Return to Command mode after dialling
	/	Delays for 125 ms. before proceeding with dial string
	W	Wait for second dial tone (X2 or X4); linked to S6 register
	@	Dials, waits for quiet answer, and continues (X3 or higher)
	$	Displays a list of Dial commands

E*n* **Sets local echo**

	E0	Echo OFF
*	*E1*	*Modem displays keyboard commands*

Annexe 8 : Programme C de contrôle du modem

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <termios.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

/** Constantes **/

#define SERIALDEV       "/dev/ttyUSB0"
#define BAUDRATE        B9600
#define TAILLE_TAMPON   3

void Reponse(int ds)
{
  register int i;
  char d[TAILLE_TAMPON];
  sync();
  for(i=0;i<TAILLE_TAMPON;i++){
    if(read(ds,d+i,1)!=1)
{ perror("Reponse.read"); exit(-1); }
    if(d[i]==0x0d) break;
  }
  int size=i;
  for(i=0;i<size;i++)
    if(d[i]!=0x0d)
fprintf(stdout,"%c",d[i]);
  if(size>0) fprintf(stdout," (");
  for(i=0;i<size;i++){
    fprintf(stdout,"%.2x",d[i]);
    if(i<size-1) fprintf(stdout," ");
  }
  if(size>0) fprintf(stdout,")\n");
}

void ModeCommande(int ds)
{
  #ifdef DEBUG
  printf("{ModeCommande}\n");
  #endif
  char *cmd="++++++++++++++++++";
//Obligation de saturer le buffer pour send
commande
  sleep(1);
  sync();
  write(ds,cmd,strlen(cmd));
  Reponse(ds);
}

void Answer(int ds)
{
  #ifdef DEBUG
  printf("{Answer}\n");
  #endif
  char *cmd="TAATAATAATAATAATA\r";
//Saturation buffer
  write(ds,cmd,strlen(cmd));
  Reponse(ds);}
```

```c
static struct termios sauvegarde;

/** Ouverture d'un port serie **/

int ouvertureSerie(char *periph,int
vitesse)
{
  struct termios nouveau;
  int df=open(periph,O_RDWR|O_NOCTTY);
  if(df<0) return -1;

  tcgetattr(df,&sauvegarde); /* save
current port settings */
  bzero(&nouveau,sizeof(nouveau));
  nouveau.c_cflag=CLOCAL|CREAD|vitesse|CS8;
  nouveau.c_iflag=0;
  nouveau.c_oflag=0;
  nouveau.c_lflag=0;
  nouveau.c_cc[VTIME]=0;
  nouveau.c_cc[VMIN]=1;
  tcflush(df, TCIFLUSH);
  tcsetattr(df,TCSANOW,&nouveau);

  return df;
}

/** Fermeture d'un port serie **/

void fermetureSerie(int df)
{
  tcsetattr(df,TCSANOW,&sauvegarde);
  close(df);
}

/** Programme principal **/

int main(void) {
  int ds;
  ds=ouvertureSerie(SERIALDEV,BAUDRATE);
  if(ds<0){
    fprintf(stderr,"Erreur sur la connexion
série.\n");
    exit(-1);
  }
  char enter = '\r';
  ModeCommande(ds);
  write(ds, &enter, 1);
  write(ds, &enter, 1);
  write(ds, &enter,1);
  write(ds,&enter,1);
  write(ds,&enter,1);
  Answer(ds);
  sleep(15);
  write(ds,&enter,1);
  fermetureSerie(ds);
  return 0;
}
```

Annexe 9 : Programme de gestion du clavier

```
import dico_clavier
import sys
import time
import pygame
from pygame.locals import *
from RPi import GPIO

def main(args):
  GPIO.setmode(GPIO.BCM)
  GPIO.setup(23,GPIO.OUT)
  GPIO.setup(24, GPIO.OUT)
  GPIO.setup(25,GPIO.OUT)
  GPIO.setup(18,GPIO.OUT)
  p = GPIO.PWM(18,1)
  p.start(35)
  pygame.init()
  pygame.display.set_mode((640,480))
  continuer = 1
  while continuer:
               for event in pygame.event.get():
                        if (event.type == KEYDOWN):
p.ChangeFrequency(dico_clavier.midi_note[event.key])
                                 time.sleep(0.2)

main(sys.argv)
```

Annexe 10 : Evènements MIDI

Les événements sont des données qui servent à coder un son à produire, dont l'ouverture ou la fermeture de note et quelques paramètres... Les événements sont codés sur trois octets, le bit de poids fort du premier octet est positionné à 1.

- 1 octet: 4 bits pour le type d'événement - 4 bits (x0 à xF) pour préciser un canal de 1 à 16, noté ci-après **c**
- 1 octet - première donnée: souvent la note, de 0 (C_0/Do_0) à 127 (x7F) (G_{10}/Sol_{10})
- 1 octet - seconde donnée de 0 à 127 (x7F) quand il y a lieu

x9 Note On Event

Cet événement déclenche la production de la note précisée.

Le troisième octet «vélocité» permet de coder une rapidité, une force d'attaque: **1** est le plus lent/faible, **127** le plus rapide/forte. Les claviers non sensitifs codent normalement cette valeur à **x40** (64). La valeur **0** est une façon d'éteindre une note.

x9**c** - note - vélocité

Pour produire la note A3 (440Hz) sur le premier canal, le premier octet sera composé de **x9** pour les 4 bits de poids fort et **x0** pour les bits de poids faible: %10010000= x90; le second est l'ordre du la_3: 45, le **0** étant le C_0.

x8 Note Off Event

Cet événement met fin à la note précisée.

La fin de note est un événement qui doit être précisé. Le troisième octet «vélocité» détermine la rapidité d'extinction (0 le plus lent, 127 le plus rapide).

x8**c** - note - vélocité