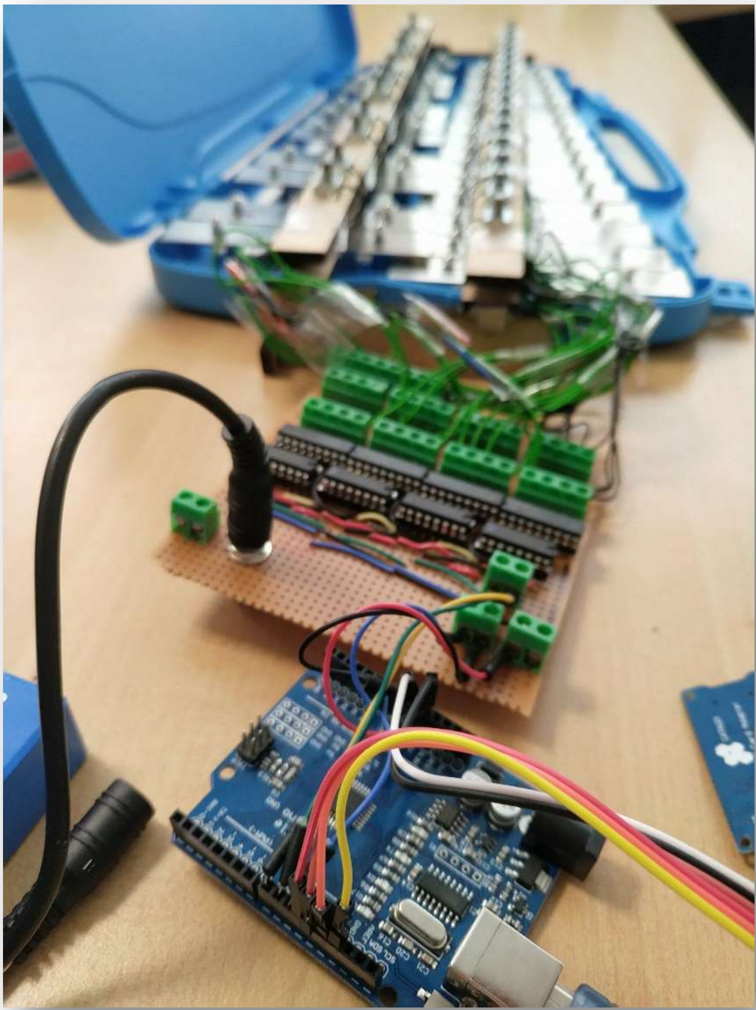
INDUSTRIAL PROJECT REPORT



2017-1018

Automation of a musical instrument

DA SILVA PEREIRA YASMIM
GRANDBOUCHE LUCIE
LI ZUHOER
MALEK GUILLAUME

Table of contents

Acknowledgments ... 3
Introduction ... 4
Part 1: From the sheet to the Arduino ... 5
  Different files formats ... 5
  Music sheet writing rules ... 6
  MusicXML file composition ... 6
  Time conversions used ... 8
  C++ functions ... 9
  Example ... 10
Part 2: From the Arduino to the Glockenspiel ... 11
  Our goal ... 11
  Supplies list ... 11
  How it works ... 12
Conclusion ... 14
Appendix ... 15
  C++ code ... 15
  Arduino code ... 21

Acknowledgments

We would like to express our deep gratitude to Professor Masson our project supervisors, for his patient guidance, his advice and assistance with the Arduino part, enthusiastic encouragement and useful critiques of this project.

I would also like to extend my thanks to Guillaume Malek, a Peip2 student in Polytech Nice Sophia who helped a lot in this project with his musical theory competences, his curiosity and skills.

Introduction

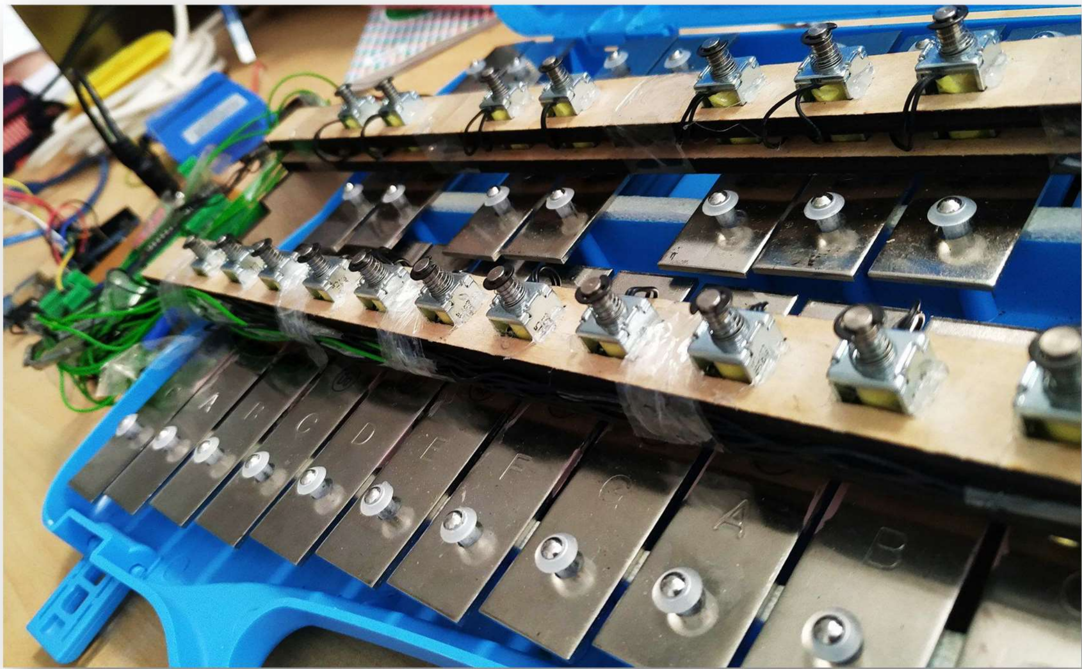
Our industrial project is the automation of a music instrument, this project uses skills in musical theory science computing and electronics.

This project transforms an instrument to an automatic instrument playing music itself, the purpose is to hear music without interface like if someone is playing around you. Our project is divided in two parts:

- Part 1: from the music sheet to the Arduino
- Part 2: from the Arduino to the instrument

The first part will transform a music sheet into a file readable by the Arduino. Anyone can write music sheet on the free software MuseScore2, this software will save the sheet in MusicXML. MusicXML is a file format currently use in the music world, it contains all the information about the sheet its display and its music. We created a code in C++ to transform this MusicXML file into a .txt file. This new file will only contain information about time and note important for playing on the instrument.

The second part is the mechanical part, the file (.txt) is put on an SD card, with on Arduino code we will read this file and act in office. The instrument we choose to play on is a glockenspiel (a kind of xylophone in steel), this instrument contains 25 notes. We make an assembly with shift register an Darlington transistor where each output will correspond to a solenoid, the solenoids will act like hammers and hit the note read from the file.



This project is extensible at infinite, as music, as electronics. It can be adapted in many instruments with some adjustment in the mechanical part, and many other styles of music can be played (without constraints). We hope that a next team of passionate will continue this project an improve it.

We create a YouTube Channel where many music can be heard played by our mechanical instrument, here is the link of our channel: https://www.youtube.com/channel/UC4oZE01wRy3X29gztx1tV-g

All the code (C++ and Arduino) can be found in appendix.

Part 1: From the sheet to the Arduino

Different files formats

the aim of this part was to create a file readable by the Arduino which correspond to the music sheet that we wanted to play.

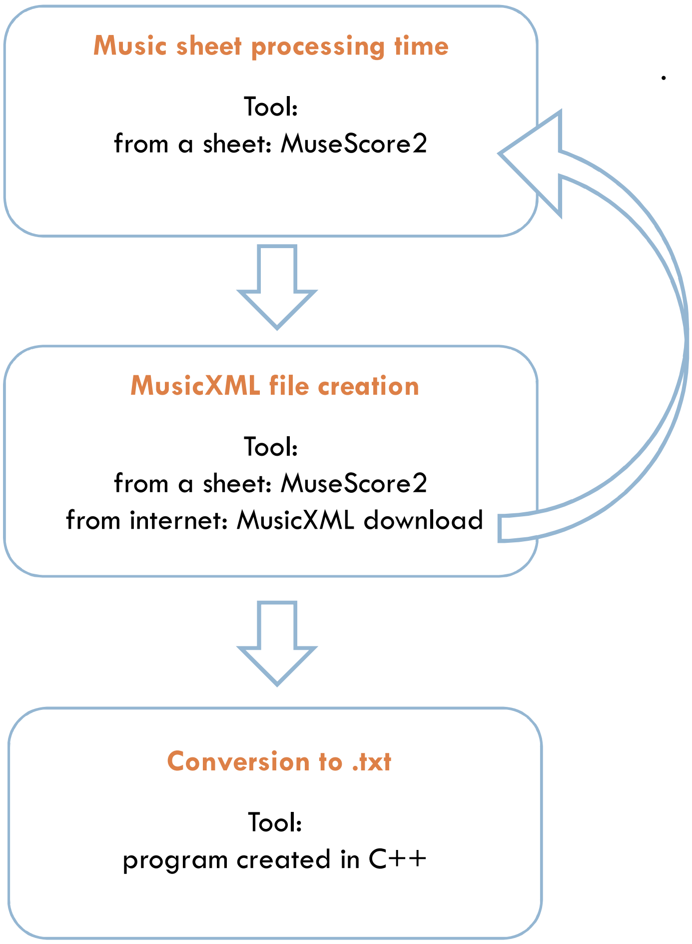
to create this file, we choose some different formats, first we choose to read the music sheet in MusicXML format which contains lots of tags for each note. then we create a .txt file, the file readable by the Arduino.

the difficult part of the project was the knowing of the musical theory and its implementation. without an exact understanding of musical theory this part was impossible to make. (the explanations of the code and how it works will be as clear as possible for those who aren't musicians).

let's talk about MusicXML and how we choose this format. our first choice was to use MIDI files, but the algorithm to transform the MIDI seemed too complicated for the time we had, so we check at others music files. it appears that MusicXML is the best choice, for its readability, plus this format is compatible with other music formats like MIDI, MSV…

During the bibliography time we observe the different tags of the file and how some of them were more important than others (for example some tags are just for the graphism of the music sheet and give none information about the note to be played itself).

Here is a little scheme which explain the relation between files formats.



A processing time must be done to avoid problems during the MusicXML file creation. Therefore, if a sheet is downloaded from the net, you may have a look at it and make some corrections.

If you write the sheet yourself, you must follow the rules established by our team. These rules are strict but every musician can convert a sheet not compatible in a sheet following the rules. You can find these rules in the part "music sheet writing rules".

You can find in appendix 1 the C++ code used to convert a MusicXML file into a .txt

The .txt file will be copy in the SD card mount in the Arduino. Here another code (Arduino code) will read each line of the file extract from the SD cart and do an action in purpose.

Music sheet writing rules

Music is a language. A complicated language with lots of rules and several special features, this make the large variety of music known and the possibility to create even more.

Many things can be changed in a music sheet, the scale, the key, the tempo, the time per measure etc.

There are rules chosen arbitrarily that we decided to follow to write are sheets, the C++ code works only if these rules are followed, if rules aren't followed the code will provide a .txt readable by the Arduino but it will be inaudible (the scale, the time and the note are not going to be right).

The sheet music must:

- Notes written on a staff (only one staff).
- Written in binary times (ternary is forbidden)
- The highest time is the whole
- The lower time is the 16^{th}
- The key signature must be natural (the scale is C Major)
- the key must be C

these rules might seem very restrictive but every musician is able to transform every sheet into a sheet following these rules. Plus, these rules are the most common "style" of music sheet (in classic and modern). ¼ sheets music are already written following these rules.

A possible improvement can be to create a code with no writing rules, the improvement is very difficult and needs lots of time and conversions functions. Plus, a high level in musical theory classes is recommended (at least 8 years to understand every sheets).

MusicXML file composition

As said in the previous chapter, the MusicXML file contain lots of tags and represent a music sheet: the keys, the beats, the duration, the notes, the times, the view, the direction of the stem and many others. Some are not important to our application. We don't care about every page layout tags, there are also some redundancy between times tags (a relative one and an absolute one).

Here are the important tags in the file:

The tags appear in this order (in purple there are unavoidable, in green optional).

```
<measure number="2">
-----
<note>
<rest/>
<step>F</step>
<alter>-1</alter>
<octave>4</octave>
-----
<type>quarter</type>
<dot/>
</measure>
```

Measure as the open tag means the start of a 4 times measures, the close tag marks the end of this measure. A measure always starts with this tag.

Then we can have many notes in each measure so an open note tag marks the beginning of a note and all this relative information.

The rest tag is a facultative one, if it's present it means that the note is a silence.

The step tag is the “name” of the note.

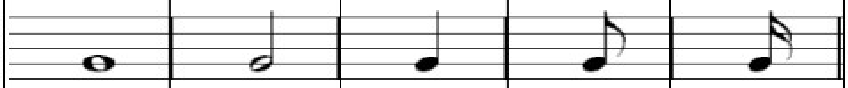
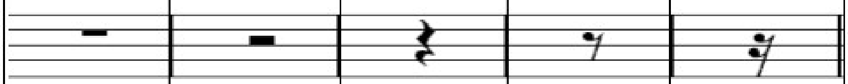
The alter tag is optional, if it's present it means that the note is altered.

The octave tag indicates the octave of the note.

The type tag indicates the duration of the note, the duration is relative to the beat of the piece. (because in our rule we decide to choose a 4-time piece, the duration “quarter” will for example be a crotchet).

The dot tag is optional it indicates if the note is “dot”, if a note is dot its duration will be: the duration without the dot + half of the duration without the note.

For a better understanding here is a table with the conversion between types and the representation with the name we use in France.

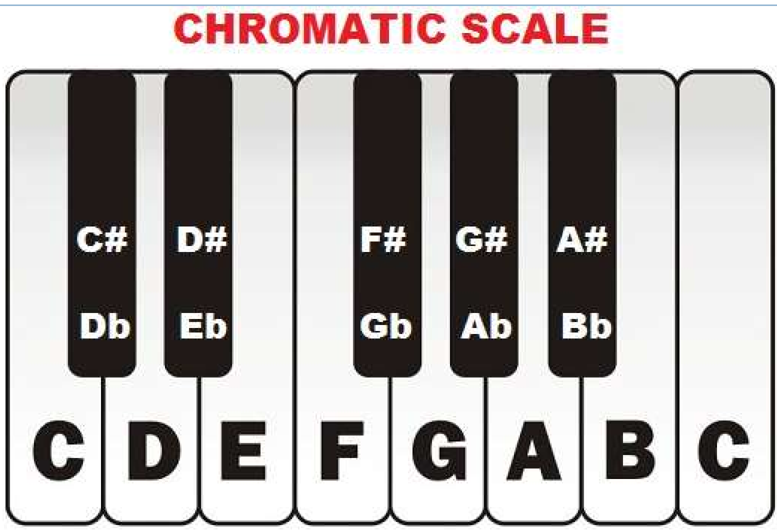
Type					
					
	Ronde	Blanche	Noire	Croche	Double croche
					
	Pause	Demi-pause	Soupir	Demi-soupir	Quart de soupir
	Whole	Half	Quarter	Eighth	16th

In the Alter tag there is two different value possible: +1 for a sharp (#) and -1 for bemol (♭). The value 0 is also possible for a natural note (but we won't use it in our code).

Here is the conversions table between all our notes. The step tag gives the information about the step of the note in the US format, so the first conversion is from the US format to the French format.

The other conversion is from bemol to sharp. One thing hard to get in musical theory is the pitches in the scale. In the C major scale between C and D there is 1 pitch, but between E and F there is only half-one.

For someone who doesn't want to get embarrassed with the different scales there is an easy way to figure if there is a pitch or half a pitch between two notes: if there is a black key between two withes it means there is a tone, if not there is only half one.



If a note is sharp you must add half a tone to the note. If the note is bemol you must subtract half a tone. This brings us to the fact that a F bemol is no longer a F but becomes an E…

All the correspondence is written in these tree tables, we decided to use only sharps to avoid conflicts with the Arduino, so if a note is bemol it will automatically be convert into its sharp equivalent.

	sol3	la3	si3	do4	re4	mi4	fa4	sol4	la4	si4	do5	re5	mi5	fa5
step	G	A	B	C	D	E	F	G	A	B	C	D	E	F
octave	3	3	3	4	4	4	4	4	4	4	5	5	5	5

	sold3	lad3	do4	dod4	red4	fa4	fad4	sold4	lad4	do5	dod5	red5	fa5	fad5
step	G	A	C	C	D	E	F	G	A	B	C	D	E	F
octave	3	3	4	4	4	4	4	4	4	4	5	5	5	5
alter	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1

	sold3	lad3	si3	dod4	red4	mi4	fad4	sold4	lad4	si4	dod5	red5	mi5	fad5
step	A	B	C	D	E	F	G	A	B	C	D	E	F	G
octave	3	3	4	4	4	4	4	4	4	5	5	5	5	5
alter	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

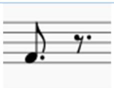
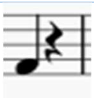
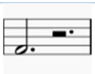
Time conversions used

The US and the French musical theory systems aren't the same, we discover this during the bibliography phase.

Music sheets writing rules have a huge impact in the time. Because we decided to write sheet only in 4 times per measure in binary (which means that the crotchet note will last 1 time).

In the file readable by the Arduino, each line will correspond to a time. The tempo will be defining and each line will be read in ratio to this tempo. The shortest time is a quarter time so every time must be multiplicated by four to get the number of line associated.

Here is a conversion table in detail for each time available

Classic time	**representation**	**Relative time for our program**
¼		1
½		2
½ + ¼		3
1		4
1 + ½		5
2		6
2 + 1		7
4		8

C++ functions

To extracts information from the XML file we created many functions:

```
int rechercher(string s, string rech) : this function permit to search the string
s in the string rech. If the string isn't found the function will return -1 if
it's found the function will return the position of the string.

String conversion(string enQuoi, string balise): this function will permit to
change the alter and the time from the conversions tables.
```

String extraction(string ligne, string balise) : this function will return the information contain in a ligne between an open tag and a close tag.

String note(string alter, string note_base, string octave, bool rest) : the function will return the name of the note from the note_base, the octave and the alter. Plus if the note is a silence the bool rest will be "true" and the return will be "0".

Int temps(string ty, bool dot) : the function will return the time where the note must be played and also the time to wait between another note played.

String reecrit(string fich) : this function will cross the XML file in parameter and write another file (.txt) with only the tags needed per each measure.

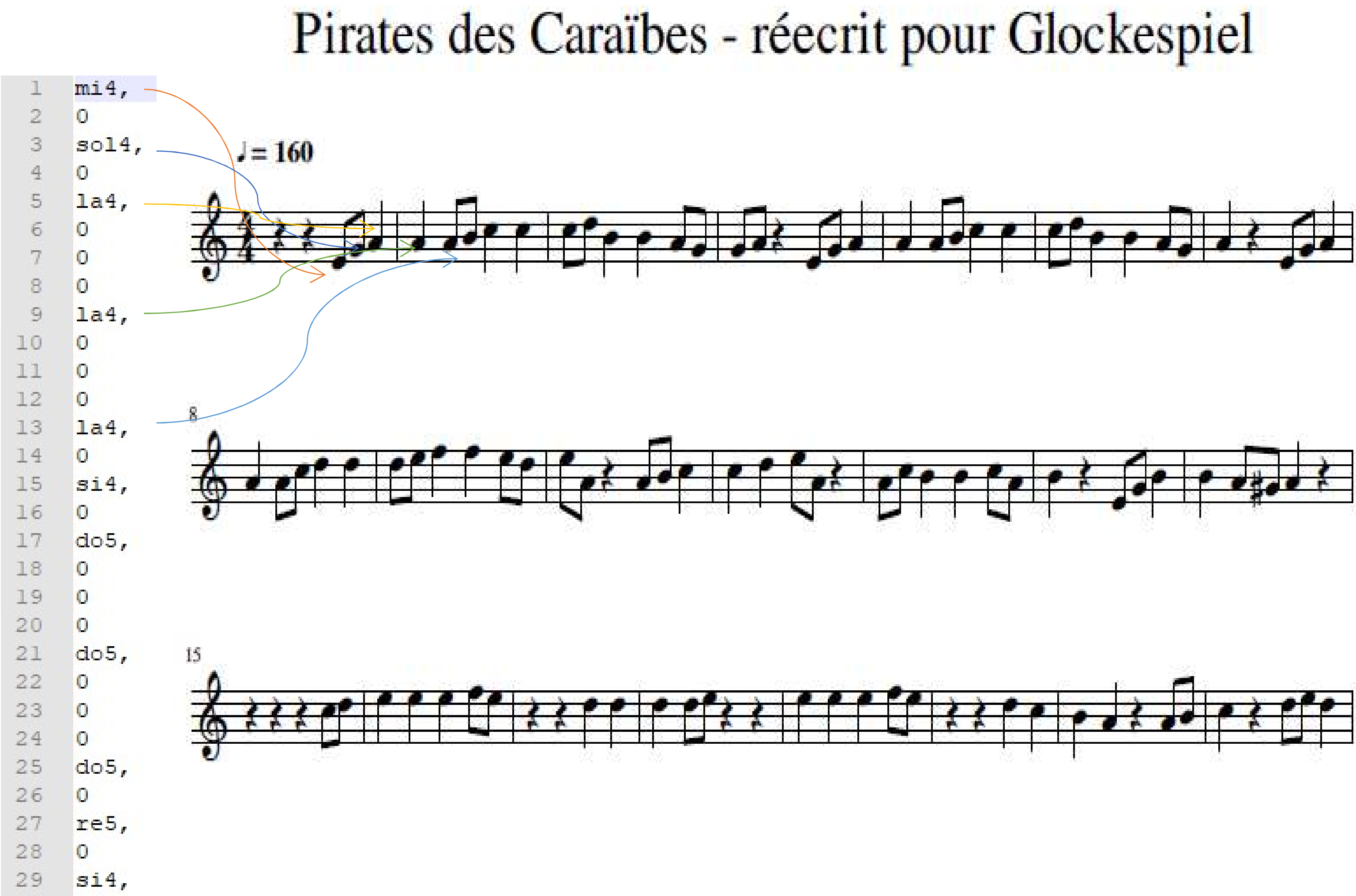
Void fichier_sortie(string fichentree, string fichsort) : this function is the reunification of every else. You put an XML file and a .txt file will be created in the same folder.

Int main(void) : this is our main program it only call the fichier_sortie function.

All the code is available in the appendix section

Example

Here an example of a music sheet out of our C++ code.



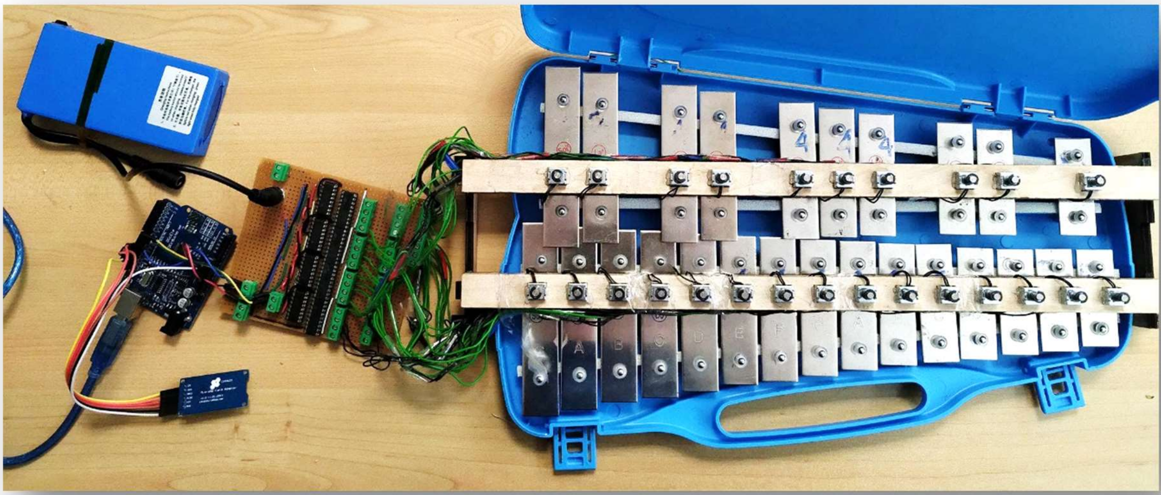
Part 2: From the Arduino to the Glockenspiel

Our goal

With the first part complete any music sheet can be transformed into a file readable line after line where each line contains several notes to be played. Each line will be read in order to follow the tempo.

The glockenspiel we worked on contains 25 notes from G3 to G5, two octaves. We are going to use one solenoid for each note, every solenoid will be activated during a short time to push the note and make a resonance.

Here is a picture of the complete set up. In the next part you can find a complete supplies list.



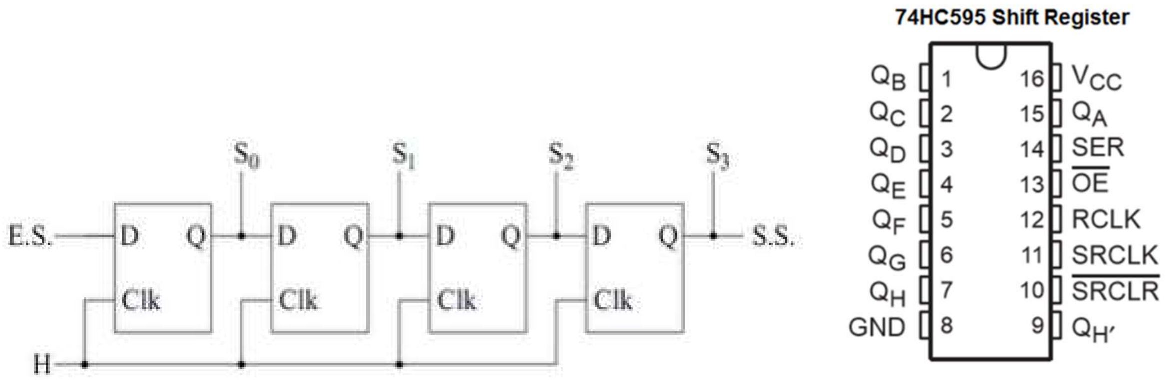
Supplies list

Here is a list of all the furniture we use for this mechanical part:

- 1 Arduino Uno
- 1 SD card lector
- 1 micro SD card – 32 Gbits
- 4 shift registers – 8 bits (74HC595)
- 4 Darlington transistors ships (ULN2803)
- 25 solenoids
- 1 external battery
- A glockenspiel
- A wood shelf
- Multiples cables

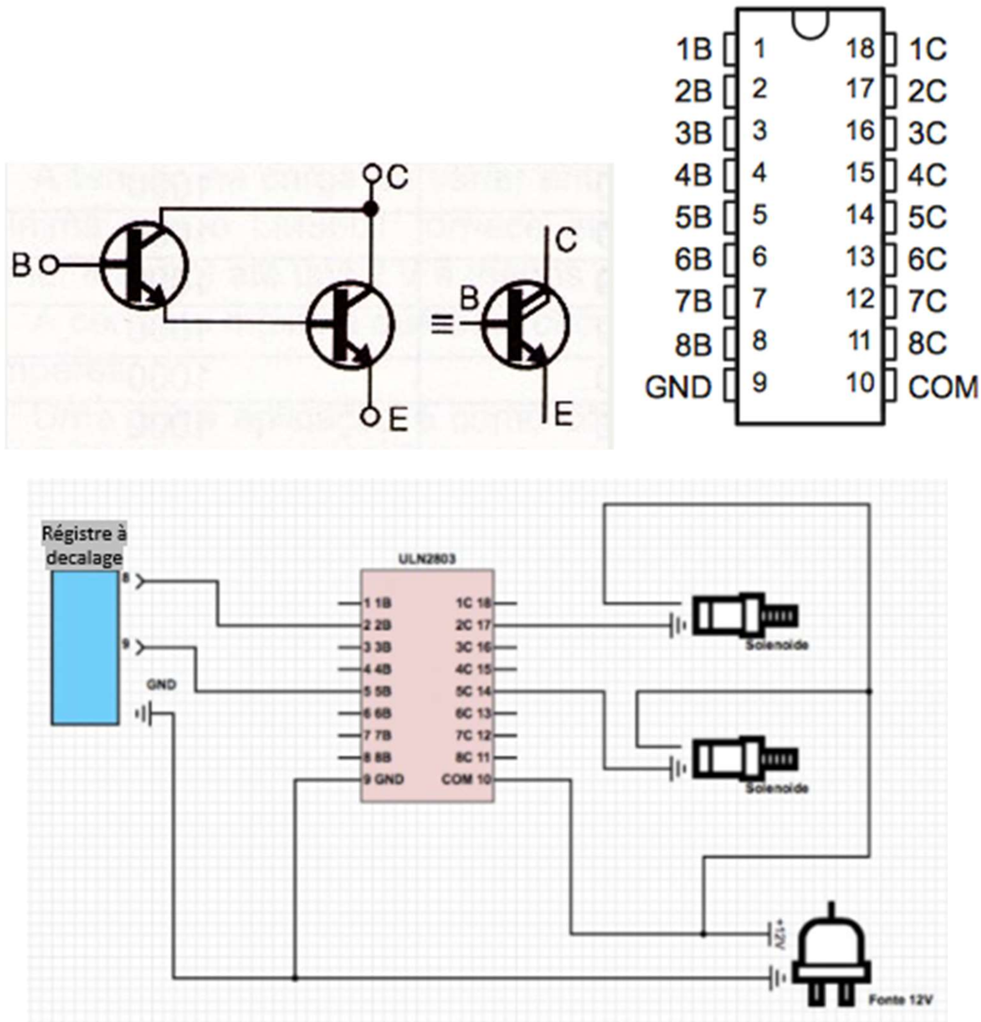
How it works

We need for shifts registers 8 bits to place one output for each note/solenoid. We decides to use shift register to minimize the number of cable and use a smaller Arduino card, our project doesn't need a big powerful card, the Arduino Uno is sufficient.



Our four shift registers

The solenoid needs lots of power to be activated, and the 5V supplied by the Arduino is not sufficient. We use a Darlington transistor ship to bring more power to the solenoids, we also use an external battery.



Example of Darlington transistor assembly

Each shift register will contain 8 notes, the last will contain only one (if we want to change the instrument and use a 32 notes instrument, there's no need to change the shift registers assembly).

```
#define ARRAYSIZE 8
String notes1[ARRAYSIZE] = {"sol3", "sold3", "la3", "lad3", "si3", "do4", "dod4","re4"};
String notes2[ARRAYSIZE] = {"red4", "mi4", "fa4", "fad4", "sol4" , "sold4","la4", "lad4"};
String notes3[ARRAYSIZE] = {"si4", "do5", "dod5","re5","red5","mi5","fa5", "fad5"};
String notes4[ARRAYSIZE] = {"sol5"};
```

The most significant bit (MSB) will the first note contained in each register, for example the MSB of the register one will be "sol3".

In our program we defined that 10 notes can be played at the same time (as a chord), this means that it's possible to have 10 notes in the same line.

In the SD card there is many files already transformed with the C++ code, the users will choose the music they want to play by writing the name of the file in the Arduino launch code. If the file exists and can be read, the code will be executed.

First, the code will reset the registers by putting then to zero. Then parse the notes contains in the same line (supposing that it's possible to have from 0 to 10 notes in the same line). The C++ code puts a coma after each note, the parser will be the coma. If the note found correspond to a note allocate in a register, then the register will receive a "1" at the place correspondent.

Example: the note mi4 is detected in a line, the register 2 will change from B00000000 to B01111111.

While the register has a "1", an impulsion will activate the solenoid correspondent at the note. After a 20 ms delay we put the register back at 0. We experiment that 20 ms delay is the good time to have a great resonance on the glockenspiel (if we change the instrument, the time must be recalculated).

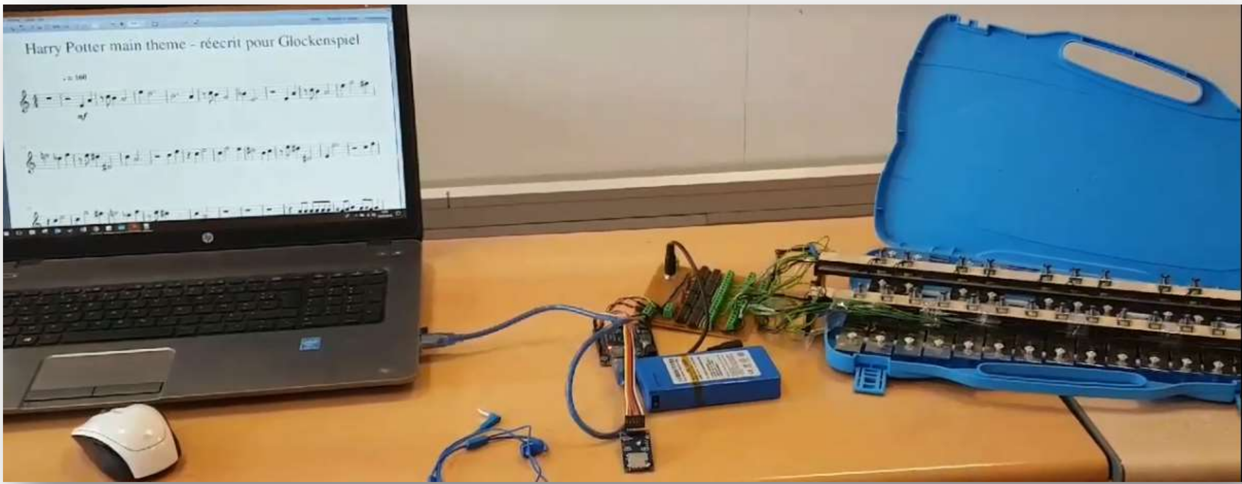
Then we wait a time correspondent at the tempo before reading the next line. If a 0 is found (means that this time is a silence, the program will just wait at the tempo before reading the next line).

All the Arduino code is in appendix.

Conclusion and improvements

This project consisted of automating a mechanical instrument. To make an instrument play a song by itself. We choose to automate a Glockenspiel. We also took some liberties from the original project. Indeed, instead of using the MIDI format, we chose to use the MusicXML format because it is easier to handle it.

To do so, the project has been divided in two parts. The first part was to convert a XML music sheet in a text file, understandable by the Arduino. The second one was to activate solenoids which will play notes on the Glockenspiel from the Arduino. These two parts works properly, and the Glockenspiel is now able to play any given song (with some constraints from the music sheet).



What we can do next to improve the project is to add a LED display and buttons, which will allow us to choose songs directly from the SD card, and not from a computer. We can also improve the aesthetic of the support, by removing the duct tape and by fixing everything together with glue.
A great improvement will be to delete all the constraints from the music sheet. There is also the possibility to play on any instrument.

Appendix

C++ code

```cpp
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <limits>
#include <string>
#include <stdio.h>
#include <ctype.h>

using namespace std;


/////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////VARIABLES GENERALES////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////


int tempo = 60; // le tempo par défaut sera la seconde
//std::string note_base_tab[7]= ["C","D","E","F","G","A","B","C"];
//string note_conv_tab[12] = ["do1","dod1","re1","red1","mi1","fa1","fad1","sol1","sold1","la1","lad1","si1","do2"];



/////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////FONCTIONS UTILES///////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////


//fonction de recherche du string rech dans le string s
//renvoie la position du début de la recherche dans la string
//renvoie -1 si la string n'est pas présente
int rechercher(string s, string rech){
    for ( size_t pos = 0 ; pos != string::npos && pos < s.size() ; ++pos){
        size_t indice = s.find(rech,pos);
        if( pos != string::npos){
            return indice;
        }
        pos = indice + 1;
    }
    return -1; // on revnoie -1 si on ne trouve pas la chaîne de caractère
}
```

```
//fonction permettant d'extraire une valeur contenue dans une balise
//string ligne : la ligne a parser
//string balise : la balise à reperer
//On ne recupère qu'un caractère
string extraction(string ligne , string balise){
    string info="";
    if(rechercher(ligne,balise)!=1){
        for (unsigned i=balise.size()+1 ; i<balise.size()+2 ; i++){
                info += ligne[rechercher(ligne,balise)+i];
            }
    }
    return info;
}


string conversion(string enQuoi, string balise){
    string conv=""; //la conversion
    //conversion alter
    if(enQuoi == "1" && balise == "alter"){
        return conv = "diese";
    }
    if(enQuoi == "-" && balise == "alter"){
        return conv = "bemol";
    }
    //conversion tps
    if(enQuoi == "w" && balise == "type"){
        return conv = "whole";
    }
    if(enQuoi == "h" && balise == "type"){
        return conv = "half";
    }
    if(enQuoi == "q" && balise == "type"){
        return conv = "quarter";
    }
    if(enQuoi == "e" && balise == "type"){
        return conv = "eighth";
    }
    if(enQuoi == "1" && balise == "type"){
        return conv = "16th";
    }
    return conv;
}
```

```
//alter : "diese" "bemol" ou "", note_base : "A","B","C","D","E","F","G" octave : "4", "5"
//note par défaut renvoyée : do1
//si la note n'est pas trouvé au bon octave il la placera automatiquement dans l'octave 4
//si c'est un silence rest = true
std::string note(string alter, string note_base, string octave , bool rest){
    std::string n="do1";
    if(rest) {
        return "0";//il s'agit d'un silence
    }
    if(alter == "diese"){
        if(note_base == "A"){
            n="lad";
        }
        if(note_base == "B"){//le si dièse n'existe pas -> do
            n="do";
        }
        if(note_base == "C"){
            n="dod";
        }
        if(note_base == "D"){
            n="red";
        }
        if(note_base == "E"){//le mi dièse n'existe pas -> fa
            n="fa";
        }
        if(note_base == "F"){
            n="fad";
        }
        if (note_base == "G"){
            n="sold";
        }
    }
    if(alter == "bemol"){
        if(note_base == "A"){
            n="sold";
        }
        if(note_base == "B"){
            n="lad";
        }
        if(note_base == "C"){//le do bemol n'existe pas -> si
            n="si";
        }
        if(note_base == "D"){
            n="dod";
        }
        if(note_base == "E"){
            n="red";
        }
        if(note_base == "F"){//le fa bemol n'existe pas -> mi
```

```
//Fonction qui extrait toutes les balises interressantes du fichier XML de base et les met dans un autre fichier "zik.txt"
//on part du repère de la première meusure.
//string fich est le fichier
string reecrit(string fich){
    //int comptempo = 0;
    string tempo = "6"; // par défaut le tempo sera la seconde
    string tab;
    ifstream ios(fich); // ouverture du fichier
    ofstream fichier1("zik.txt");
    if(ios){
        while(getline(ios,tab)){
            while(getline(ios,tab)){
                if (rechercher(tab,"<measure number=\"")!=-1){
                    fichier1 << tab << endl;
                }
                if(rechercher(tab,"<rest/>")!=-1){
                    fichier1 << tab << endl;
                }
                if(rechercher(tab,"<note>")!=-1){
                    fichier1 << tab << endl;
                }
                if(rechercher(tab,"<per-minute>")!=-1){
                    tempo = extraction(tab,"per-minute");
                    //comptempo = 1;
                    fichier1 << tab << endl;
                }
                if(rechercher(tab,"<step>")!=-1){
                    fichier1 << tab << endl;
                }
                if(rechercher(tab,"<alter>")!=-1){
                    fichier1 << tab << endl;
                }
                if(rechercher(tab,"<octave>")!=-1){
                    fichier1 << tab << endl;
                }
                if(rechercher(tab,"<type>")!=-1){
                    fichier1 << tab << endl;
                }
                if(rechercher(tab,"<dot/>")!=-1){
                    fichier1 << tab << endl;
                }
                if(rechercher(tab,"</note>")!=-1){
                    fichier1 << tab << endl;
                }
                if(rechercher(tab,"</measure>")!=-1){
                    break;//fin de la meusure
                }
            }
        }
    }
```

```
//cette fonction ecrit a partir d'un fichier xml en entrée,
//un fichier au format souhaité en sortie
void fichier_sortie(string fichentree, string fichsort){
            string temmp =  reecrit(fichentree); // le tempo et l'ecriture du fichier intermediaire
            string tab;
            string alt;
            string stp;
            string oct;
            string typ;
            bool dot;
            bool rest;
            int compteur = 0;
            int temps_it = 0;
            ifstream ios("zik.txt");
            ofstream fichierend(fichsort);
            if(ios){
                fichierend <<temmp<< "0" << endl; // on ecrit le tempo en haut du fichier
                while(getline(ios,tab)){
                    if(rechercher(tab,"rest")!=-1){
                        rest = true;
                    }
                    if(rechercher(tab,"dot")!=-1){
                              dot = true;
                    }
                    if(rechercher(tab,"type")!=-1){
                        typ = conversion(extraction(tab,"type"),"type");
                    }

                    if(rechercher(tab,"alter")!=-1){
                        alt = conversion(extraction(tab,"alter"),"alter");
                    }
                    if(rechercher(tab,"step")!=-1){
                        stp = extraction(tab,"step");
                        compteur++;
                    }
                    if(rechercher(tab,"octave")!=-1){
                        oct =extraction(tab,"octave");
                        compteur++;
                    }
                    if(rechercher(tab,"</note>")!=-1){
                        temps_it = temps(typ,dot);
                        compteur++;
                    }
                    if(stp !="" && oct !="" && compteur==3){
                        fichierend <<note(alt,stp,oct,rest)<<"," <<endl;
                        for(int i=1 ; i<temps_it ; i++){
                            fichierend << "0" << endl;
                        }
```

```
                        alt ="";
                        rest = false;
                        dot = false;
                        compteur=0;
                    }
                }
            }
            ios.close(); //on ferme le fichier
}

/////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////MAIN////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////////////




int main (void) {
            fichier_sortie("test8.xml","sortie8.txt");
            //fichier_sortie("test2.xml","sortie2.txt");
            //fichier_sortie("test3.xml","sortie3.txt");
            //fichier_sortie("test1.xml","sortie1.txt");
            //fichier_sortie("test4.xml","sortie4.txt");





    return EXIT_SUCCESS;
}



/* doc utile pour les fichiers
 * getline(flux, char) : lit une ligne complète
 * flux.get(char) : lit un caractère
 * flux >> variable : pour recuperer a partir du fichier jusqu'a un délimiteur
 *
 *
 *
 * flux.eof() : savoir si on a atteint la fin du fichier -> booléen
 * flux.ignore(nbCaractere, caractèredeFin) : ignorer un certain nb de caractère ou ignorer tout jusqu'a ce que caracteredeFin soit rencontré
 * flux.ignore(numeric_limits<int>::max(),'\n') : compter les lignes dans un fichier
 *
 *flux.clear() :  permet ed revenir au debut du fichier
 *
 *
 *
 * str.find(str2) renvoie la position du premier element trouvé ou npos si il n'est pas trouvé.
 */
```

```
        if(note_base == "F"){//le fa bemol n'existe pas -> mi
            n="mi";
        }
        if (note_base == "G"){
            n="fad";
        }
    }
    if(alter==""){
        if(note_base == "A"){
            n="la";
        }
        if(note_base == "B"){
            n="si";
        }
        if(note_base == "C"){
            n="do";
        }
        if(note_base == "D"){
            n="re";
        }
        if(note_base == "E"){
            n="mi";
        }
        if(note_base == "F"){
            n="fa";
        }
        if (note_base == "G"){
            n="sol";
        }
    }
    return n+octave;

}
```

```
int temps(string ty, bool dot){
    int tps = 1;
    if(ty=="16th" && dot){
        tps = 1;
    }
    if(ty=="eighth"){
        tps = 2;
        if(dot){
            tps = 3;
        }
    }
    if(ty=="quarter"){
        tps = 4;
        if(dot){
            tps = 6;
        }
    }
    if(ty=="half"){
        tps = 8;
        if(dot){
            tps = 12;
        }
    }
    if(ty=="whole"){
        tps = 16;
        if(dot){
            tps = 24;
        }
    }
    return tps;
}
```

Arduino code

```
//*****************Definition de la carte SD*****************
#include <SD.h> //Load SD library
#include <SPI.h>
int chipSelect = 4; //chip select pin for the MicroSD Card Adapter
File file; // file object that is used to read and write data

//*****************Broches connectées 74HC595*****************
const int DS = 8; // Données
const int ST_CP = 9; // Verrou
const int SH_CP = 10; // Horloge

//*****************définition des 13 notes du métalophone*****************
#define ARRAYSIZE 8
String notes1[ARRAYSIZE] = {"sol3", "sold3", "la3", "lad3", "si3", "do4", "dod4","re4"};
String notes2[ARRAYSIZE] = {"red4", "mi4", "fa4", "fad4", "sol4" , "sold4","la4", "lad4"};
String notes3[ARRAYSIZE] = {"si4", "do5", "dod5","re5","red5","mi5","fa5", "fad5"};
String notes4[ARRAYSIZE] = {"sol5"};

// définition des 4 registres HC595 soit 32 bits
byte registre1 = B11111111; // le MSB correspond à la note sol3
byte registre2 = B11111111; // le MSB correspond à la note red4
byte registre3 = B11111111; // le MSB correspond à la note si4
byte registre4 = B11111111; // le MSB correspond à la note sol5

byte RAZ = B00000000;


// nombre maximum de notes par ligne
String StringSplits[10];
```

```
//*****************variables diverses*****************
unsigned long temps, temps2;
int var=0;
int tempo = 70;
int i,j;

void setup() {
Serial.begin(9600); // start serial connection to print out debug messages and data

pinMode(DS, OUTPUT);
pinMode(ST_CP, OUTPUT);
pinMode(SH_CP, OUTPUT);

pinMode(chipSelect, OUTPUT); // chip select pin must be set to OUTPUT mode
if (!SD.begin(chipSelect)) { // Initialize SD card
  Serial.println("Could not initialize SD card."); // if return value is false, something went wrong.
}
if (SD.exists("sortie9.txt")) {
  Serial.println("File exists.");
}
temps=millis();
}

void loop() {

if (var==0) { // on ne va faire ce "if" qu'une seule fois
  file = SD.open("mousse.txt", FILE_READ); // Ouverture du fichier en mode lecture

  if (file) { // Si le fichier "file" existe
    Serial.println("--- Reading start ---");
    while (file.available()) {
      String list = file.readStringUntil('\r\n'); // Lecture d'une ligne
      Serial.println(list);
```

```
//******************Mise à 0 des registres HC595*****************
        for (i=0; i<8; i++){
          bitClear(registre1, i);
          bitClear(registre2, i);
          bitClear(registre3, i);
          bitClear(registre4, i);
        }
//******************Affectation des notes dans les registres*****************
        if (list.equals("0\r")) { // Si il n'y a pas de note à jouer
          Serial.println("pas de note");
        }

        else { // Si il y des notes à jouer
          for (j = 0; j < 10; j++) { // séparation des notes en supposant qu'il y en a au maximum 10 (il faudra cherch
            StringSplits[j] = GetStringPartAtSpecificIndex(list, ',', j);
            if (StringSplits[j]!= "") { // si il y a rellement une notes (sur les 10 il peux y en avoir des vides)
              for (i=0; i<8; i++){
                  if (StringSplits[j] == notes1[i]){
                    Serial.println(StringSplits[j]);
                     bitSet(registre1, (7-i));
                  }
                  if (StringSplits[j] == notes2[i]){
                    Serial.println(StringSplits[j]);
                     bitSet(registre2, (7-i));
                  }
                  if (StringSplits[j] == notes3[i]){
                    Serial.println(StringSplits[j]);
                     bitSet(registre3, (7-i));
                  }
                  if (StringSplits[j] == notes4[i]){
                    Serial.println(StringSplits[j]);
                     bitSet(registre4, (7-i));
                  }
              }
            }
          }
```

```
            Serial.print("registre1 = ");
            Serial.println(registre1,BIN);
            Serial.print("registre2 = ");
            Serial.println(registre2,BIN);
            Serial.print("registre3 = ");
            Serial.println(registre3,BIN);
            Serial.print("registre4 = ");
            Serial.println(registre4,BIN);
          }

           digitalWrite(ST_CP, LOW);
           shiftOut(DS, SH_CP, LSBFIRST, registre4);
           shiftOut(DS, SH_CP, LSBFIRST, registre3);
           shiftOut(DS, SH_CP, LSBFIRST, registre2);
           shiftOut(DS, SH_CP, LSBFIRST, registre1);
           digitalWrite(ST_CP, HIGH);
           delay(20);
           digitalWrite(ST_CP, LOW);
           shiftOut(DS, SH_CP, LSBFIRST, RAZ);
           shiftOut(DS, SH_CP, LSBFIRST, RAZ);
           shiftOut(DS, SH_CP, LSBFIRST, RAZ);
           shiftOut(DS, SH_CP, LSBFIRST, RAZ);
           digitalWrite(ST_CP, HIGH);



        delay(tempo);
        }
//*****************Fin d'affectation des notes dans les registres*****************

      file.close(); // fermeture du fichier
      Serial.println("--- Reading end ---");

   }
   else {
      Serial.println("Could not open file (reading).");
```

```
        delay(500); // wait for 5000ms
        var=1;
 }
           for (i=0; i<8; i++){
              bitClear(registre1, i);
              bitClear(registre2, i);
              bitClear(registre3, i);
              bitClear(registre4, i);
           }
         for (i=0; i<=15; i++){
             digitalWrite(ST_CP, LOW);
             shiftOut(DS, SH_CP, LSBFIRST, registre4);   //on envoi
             shiftOut(DS, SH_CP, LSBFIRST, registre3);
             shiftOut(DS, SH_CP, LSBFIRST, registre2);
             shiftOut(DS, SH_CP, LSBFIRST, registre1);
             digitalWrite(ST_CP, HIGH);
         }
 }
```