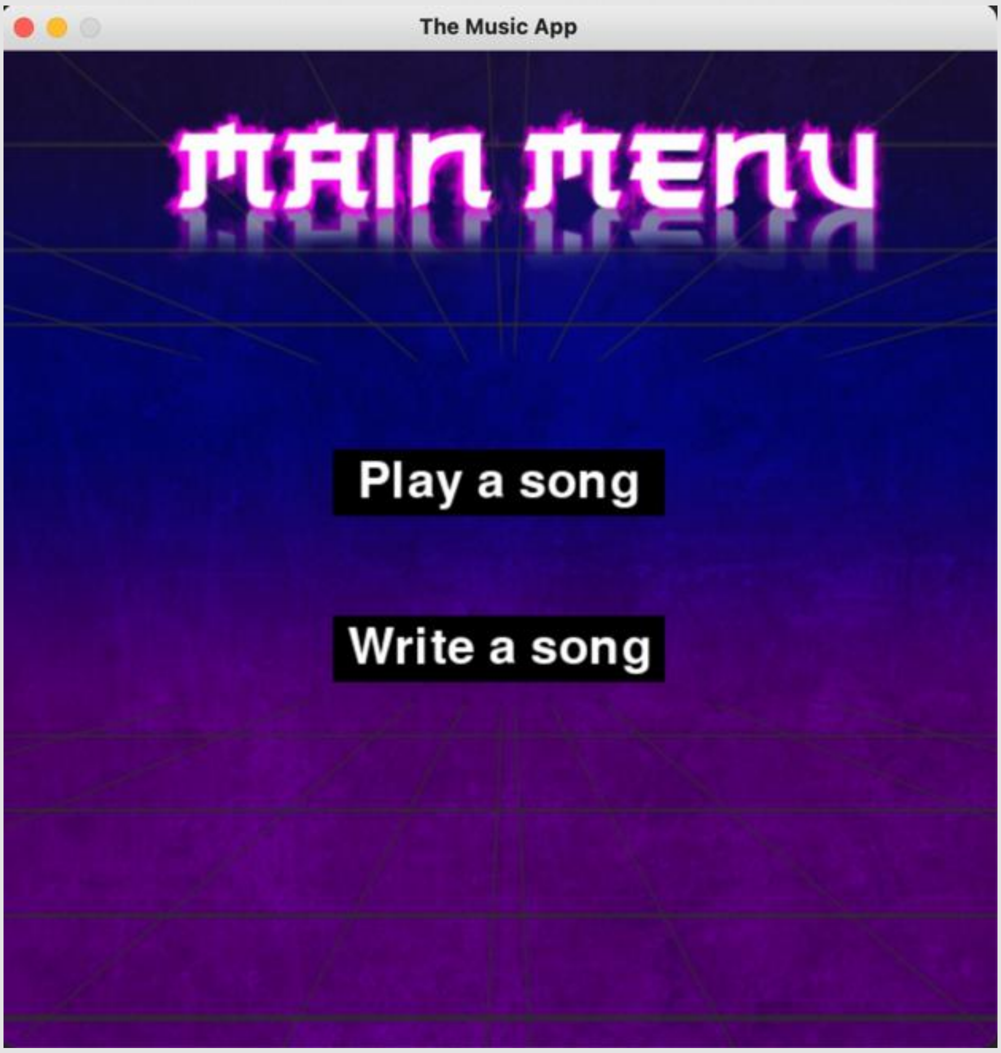
January 2021

# THE MUSIC APP



Cleophas Fournier
Paul Zamanian
L1- groupe int5

## Summary:

I) Introduction

II) Application presentation

- Interface presentation
  - song player
  - song writer

- Technical presentation
  - Organization
  - Settings
  - Main
  - Menu
  - Menufunctions

III) Algorithms of functions from file “MAIN”

IV) Conclusion

# I) Introduction

During our first year at EFREI Paris, we were able to make a small music application, that can read music of any partition written under this form:

"SOLc p Zc SOLn LAn SOLn DOn Zc SIb SOLc p Zc SOLn LAn SOLn REn Zc DOb SOLc p Zc"

At first, we did not have any particular ideas of how we could realize this project, but both of us being fans of music, we knew we would take great pleasure realizing this project.

We jumped headfirst into the conception of the basic functions, the ones able to read and write new music partitions, not really knowing how much time this was going to take us and how many functions we were going to create.

We knew from the start that we wanted to create a GUI (graphical user interface) to manipulate these functions and make the application as user friendly as possible. To achieve this goal, we used the module "pygame", after a lot of research and testing, we found this module to be the most appropriate for our use.

We had a lot of fun creating this application and are constantly adding new functionalities. Here are some of them we added to our program.

## Added functionalities:

One of the main changes we did was to the function "sound", we changed the sound produced by the function by adding into the project wav files to produce a piano sound. We used the application "garageband" to create the piano sounds. This means that we no longer had to associate a note with a frequency, but we still had to associate each note to a wav file.

We also tried to prevent the application from crashing as much as possible by adding securities. For example, when writing a new song, the program won't let you start writing it without entering a title. Or when entering an amount for transposition when modifying a song, the program checks if the entered amount is and integer and displays and error message on window if it is not.

Another functionality we added was to the Markov chains. At first, we did not fully understand how we were supposed to choose which durations to add to the new notes, so we chose to create another successors table for duration. Surprisingly, this method works very well to create a similar partition to the selected one.

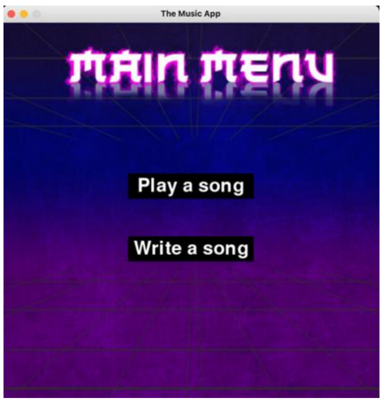
We also added file management. Indeed, in our project we have a folder called "TXT" that contains all the partitions files. This makes it easy to add new files and extend the data base.

The goal was to make the application look as professional as we could, we obviously still have a lot to learn but we had fun adding small details that make the manipulation and testing feel more alive. For example, when writing a new song, when typing in the title of our new song, we created a small box to fit in the text, that gets bigger depending on the size of the title. We also added a small white bar that flashes (like in word), and responsive buttons that change color when the mouse is hovering over them.

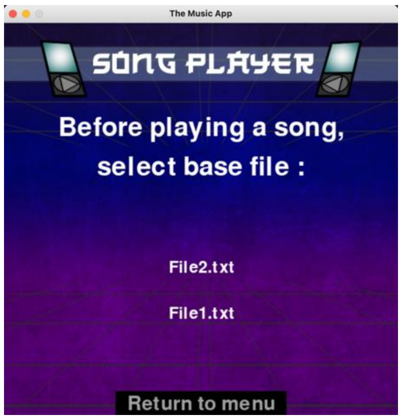
## II) Application presentation:

After hours of coding, we are delighted to present to you our music application: "The music App". Before getting into the technical aspect of how and why we chose to code this application as so, let us present briefly our application and its capabilities. Let us start by showing you the graphical interface of our program. We will then show you the technical aspect of our project.

### 1) Interface presentation:

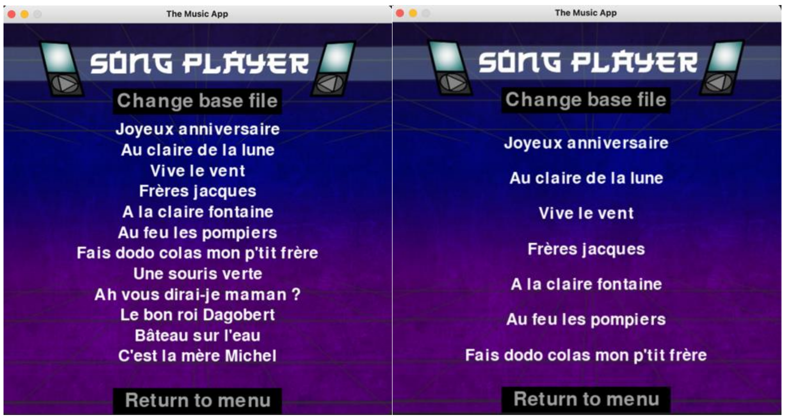


Welcome to the main screen, from here the user can choose to play a song from the ones already in the file provided with the application or to write a new one. Let's start by playing a song.



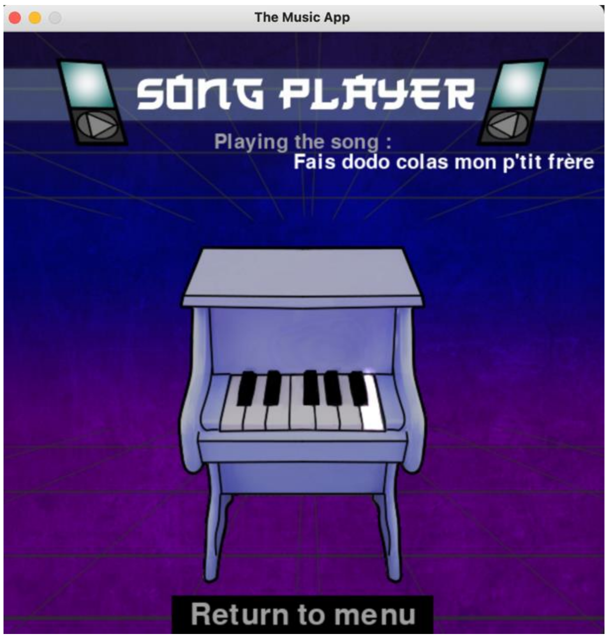
We wanted to be able to add new text files to enrich the database. Therefore, we added an option when clicking on the button: "Play a song", to choose from which file we would like to read a song. This option is available only if there is more than 1 file in the folder "TXT files".

After selecting a file, you enter the song player. This screen allows the user to choose a song from the selected file. The program takes all titles from the file and draws them on screen. The display of these titles depends on the number of titles contained in the file. The more songs there are on the file, the more the titles will be close to each other, to be able to display all of them on screen, as you can see with this example:



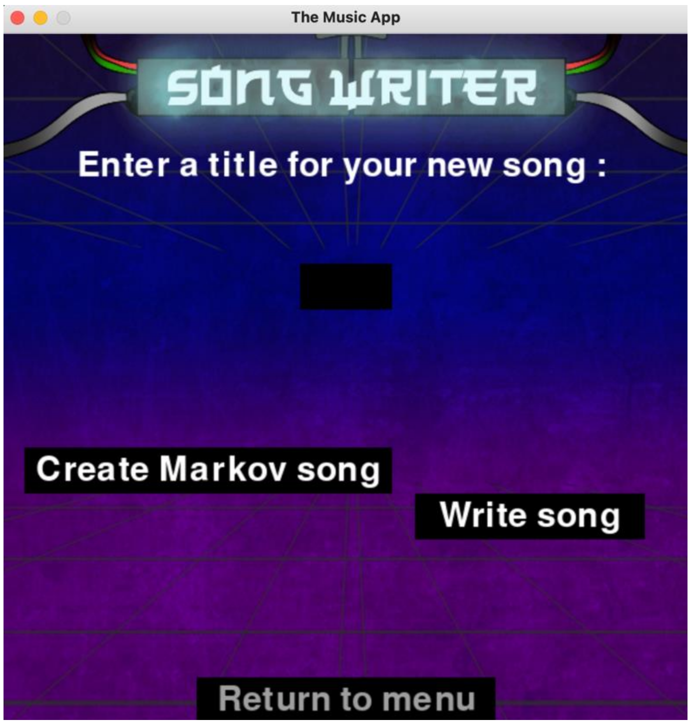
We now have the choice to select a song, when selecting one, we enter the sound tuner, that allows to add modifications to the song we have selected. Such as transposition and/or inversion.

Once pressing the play button of the sound tuner, the song starts playing, and the program displays a piano animation with the keys getting pressed lighting up:



Now let’s press the button “Return to menu” and go back to the main screen. Let’s go to the write a new song.

When entering the song writer, the first action to do is to select a title for our new song:

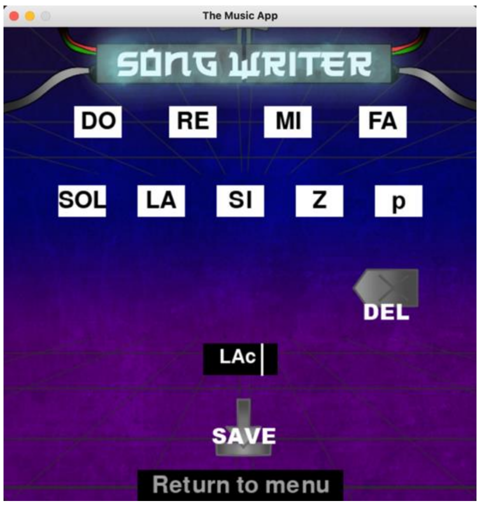


When we enter a title, we have 2 options:

- Write a song note by note
- Write a song using Markov chains

<u>Note-by-note writer:</u>

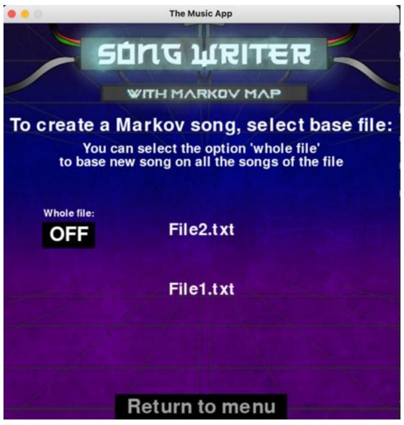
For the note-by-note writer, to make the app as user friendly as possible, we made buttons for each note, and each duration. The user selects a note, and then selects the duration associated with each note.



The new song gets written on the screen for the user to see what he is writing. He has the option to delete the last note that he has entered if he chooses to. When finished composing his new song, by pressing the button save, the user can choose the file he wants to save the song to.

The song is then automatically saved to the database on the selected file and the user is redirected to the song player where he can test out his new song. When a file contains a song written by the user, in the song player, next to the song title, there is a “delete” button that allows the user to delete the song he has written.

## Markov chains writer:



To create a Markov song, select base file:

You can select the option 'whole file'
to base new song on all the songs of the file

Whole file: OFF

File2.txt

File1.txt

When writing a song using the Markov chains writer, the user has to first select a file to base his new song on. He can choose to select the whole file or just a song in the file by using the switch "whole file".

If the user selects a file and the whole file switch is off, he is able to choose a song from the file he selected. If he chooses to create the new song based on the whole file, this next screen shows up.
The program then displays on screen the successors tables for the durations and the notes of the song:

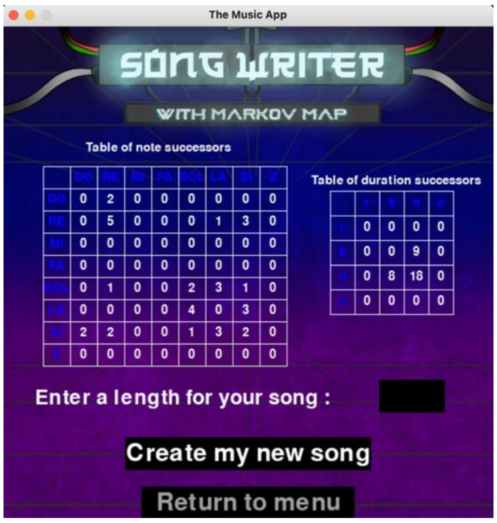


Table of note successors

| | DO | RE | MI | FA | SOL | LA | SI | Z |
|---|---|---|---|---|---|---|---|---|
| DO | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| RE | 0 | 5 | 0 | 0 | 0 | 1 | 3 | 0 |
| MI | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| FA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SOL | 0 | 1 | 0 | 0 | 2 | 3 | 1 | 0 |
| LA | 0 | 0 | 0 | 0 | 4 | 0 | 3 | 0 |
| SI | 2 | 2 | 0 | 0 | 1 | 3 | 2 | 0 |
| Z | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table of duration successors

| | r | b | n | c |
|---|---|---|---|---|
| r | 0 | 0 | 0 | 0 |
| b | 0 | 0 | 9 | 0 |
| n | 0 | 8 | 18 | 0 |
| c | 0 | 0 | 0 | 0 |

Enter a length for your song :

The last step to create a song is to enter a length for his new song. When clicking "Create my new song", the user again has the choice of the file he would like to add his song to and is redirected to the "play song" menu to test out his new song.

## 2) Technical presentation:

### 1) Organization

For both of us, this was our first-time coding such an application, so organization was one of our top priorities. This is how we organized our code.

We separated all our functions in four files:

- Settings
- Main
- Menu
- Menufunctions

We created different dependencies in our project to manipulate images, wav and text files efficiently. One is for images (background images, animations, and button images), one is for our sound files, and the last one is for the text files containing partitions.

Our program runs with the file "Menu" that contains the main loop and manipulates all of the other functions of the program. In this next part, we will go through in detail the contents of the different files and explain as best we can how our program works.

### 2) Settings:

The file "settings" contains all of the settings of our application. We chose to do this to facilitate the development phase. When we wanted to change a setting, instead of going through our code and finding where we initialized it, we can just change it in the settings file, and it would change it everywhere in our code.

We created a dictionary to associate note duration with duration in seconds and lists of duration and notes to be able to manipulate them efficiently.

This file contains all of the colors we use in our GUI in RBG values, and our window settings.
We chose to implement a grid, to facilitate the development part. This consists of simply dividing the window into groups of pixels. When drawing an object (button, text, switch) on the screen we can just write as parameter the number of "group of pixels" instead of typing in the exact number of pixels. This method, used with a function that we called "draw_grid" (that draws on screen a grid of the groups of pixels), saved us a consequent amount of time while coding.

However, this method has its limits because it can sometimes lack precision when placing objects on screen. So, some of our functions from the file "menufunctions" have an additional parameter: "nogrid", set by default to False to be able to be more precise if there is the need to.

The file also contains for loops to create dictionaries for file manipulation. One dictionary for the text files (with file number, file title, and filelength), and another one for the wav files (with note, associated with filename). Indeed, we chose to change the “sound” function provided with the subject of the project, to make the sound of the app feel more realistic.

The last part of the file “Settings” loads all the images we use in the program, such as background images, piano animations, or button images.

## 3) Main

The file “main” contains all of the basic functions for sound manipulation of our application. The functions of this file are:

- Sound(note, duration)
- get_key(value, dictionary)
- get_song(file, number)
- make _music_sheet(list_notes, invert, transpos, amount_of_transposition=0)
- get_all_songs_titles(file)
- transpose(list_of_notes, transposition):
- transposenote(note, transposition):
- invert (list):
- inverse_note(note):
- addsong(strin, title, file):
- delsong(numb, file):
- create_markov_map(numb, file):
- sum_of_maps(m1, m2):
- create_markov_map_all_songs(file):
- markov_new_song(markovmapnotes, total_notes, markovmapdurations, total_durations, songlength):

How to play a song:

In order to play a song, there are different steps:

- First with the function “get_song”, the program finds the correct line and returns a list of all the different notes.

- Then, with the function “make_music_sheet”, it creates a list of tuples containing the note and duration (in seconds), applying the corresponding modifications if there are any. This function makes the partition readable by the function “sound”. This is done with the dictionary “duration_of_notes” from the file settings.

- Then the last function to execute to play a song is “sound”. At first, we used the function provided to us, but then modified it to make the sounds more realistic. So, no need to associate the note with a frequency anymore, but we need to associate each note with a file. This is done with the dictionary “notes_dict” from the file settings.

How to create a Markov song:

These are the different steps to create a markov song:

- The first function to execute is create_markov_map or create_markov_map_all_songs. These functions create 2 matrixes and 2 arrays and returns them. The two matrixes are successors tables for durations and notes, and the arrays are the total number of notes played in the song or file.

- Once the previous function creates the matrixes, we can send all of these elements to the "markov_new_song" function which return a new song (as a list) using the markov algorithm.

If you would like to test these functions without running the GUI, we have prepared for you a range of 6 tests at the bottom of the file "Main".
All of the functions of this file are explained in detail in the part Algorithms of functions.

## 4) Menu

The file "Menu" contains all of the screens from the menu. These are the different menu screens we used:

- start_screen():
- choose_file():
- choose_song():
- song_tuner(song_number):
- play_the_song(next_action):
- choose_title_new_song():
- choose_file_to_add_song_to(bg, new_song, user_title):
- write_song(user_title):
- Markov_create_song(TITLE):

This file also contains the main loop of the program, at the very bottom. This loop rotates between different conditions and looks for the next action to execute. All of our menu screens return the next action to execute, so when executing a screen, it sets the next action to be what the screen returns.
There are some exceptions though, for example the screen "Choose title" returns the title of selected song and the next action.

These screen functions use all of the functions of the file "Main" to manipulate sound depending on the user's selection.

When coding the GUI, we rapidly realized we were going to need another file to put in our interface functions. Thus, the file "Menufunctions"

## 5) Menufunctions

This file contains functions that we use to display elements on the different menu screens. Most of these functions are used in every single menu screen, so it seemed essential to call them instead of rewriting them in every single screen on the file "Menu". These are the functions of this file:

- return_to_menu(screen):
- draw_text(screen, text, size, color, x, y, nogrid=False, midtop=True):
- draw_grid(screen):
- button(screen, text, button_color, text_color, x, y, w, h):
- text_button(screen, text, text_color, x, y, w, h):
- switch(screen, x, y, w, h, state, color):
- text_box(screen, text, x, y, w, h, time):
- img_button(screen, name, imgunpressed, imgpressed, x, y, w=0, h=0, decalage=43):
- check_for_error(amount):
- delete_last_note(string):
- piano_animation(screen, note):

- are_you_sure(screen, songtitle, file):
- draw_map(screen, matrix, x, y, w, h, list_for_axis):

## Piano animation:

For the animation of the piano, we created a function called piano_animation. We send to this function the note that the song player is playing, and it finds the correct image using the dictionary of "Settings" "notesdict". In order to make it more realistic, when finished playing a note we call the function again and send it the note "Z" to show the image of the piano with no key pressed.

## Buttons:

For all of the button functions, we managed to make the buttons change color when being hovered over by the mouse to make the App feel intuitive. The way this works is by checking the position of the mouse and changing the variable associated with color when the mouse is above the button.

The button functions return None when the button is not pressed or return the text of the button if it is pressed. This makes it easy to check if the button is pressed or not.
These functions only work if placed in a while loop.

## III) Algorithms of the functions of file “Main”

We chose to translate to algorithmic language all the functions of the file “Main” because they are the ones the most in link with sound manipulation. For each function, we include a part “information” to explain the objective of the function.

### SOUND:

**Information:**
The purpose of this function is to play a sound, it plays a sound only if the note is note a “Z”.
**Return:**
This function has no return.

Function sound (note: string; duration: integer)
**Copied parameters:** note, duration
**Output folder:** notesdict[]: dictionary of notes associated with soundfile
**Specific functions :**

```
        Fadeout() : function to decrease the volume gradually for a more pleasant rendering
        Play(): plays the sound of a file
        Sleep(): wait for an amount of seconds
```

```
BEGIN
    If (note = “Z”) Then
        Sleep(duration)
    Else then
        soundfile ← notesdict[note]
        fadeout (0)
    End if
END
```

### GET_SONG

**Information:**
This function allows to retrieve the list of notes of a song by specifying its number and the file
**Return:**
This function returns a list.

Function get_song(file : name of a text file , number : integer)
**copied parameters** :file, number
**local parameters :** list_of_notes : 1D array of notes
**Specific functions:**

```
    split() :  convert a string into a list according to a certain character
    next() : retrieves next item from the iterator
    strip() : removes characters from a list ( in this case : \n)
    size() : take the size of a list, dictionary, file
BEGIN
    For line ← 1 to size(file) Do #for every line in the file
        If ("#" + number + " ") = line[:3] or ("#" + number + " ") = line[:4] Then
            list_of_notes ← split(str(next(file)), " ")
            list_of_notes[-1] ← strip(list_of_notes[-1])
            Return list_of_notes
        End If
    End For
```

```
END
```

## MAKE_MUSIC_SHEET

**Information:**
This function creates a list of tuples of the form (note, duration) with note as a string and duration as an integer.
To do so it uses a list of notes.
This is also the function that apply transposition and/or inversion as needed
**Return**:
This function returns a list of tuples.

```
Function make_music_sheet(list_notes : 1D array; invert,transpos : Boolean ; amount_of_transposition : integer)
copied parameters :list_notes, invert, transpos, amount_of_transposition :(set to 0 if no value is assigned)
local parameters :
            new_list : 1D array
            note_and_duration : 1D array
            index, duration : integer
specific functions:
            invert (list) : function to reverse a list
            transpose(list, amount): function to transpose a list
            size() :take the size of a list, dictionary, file
            duration_of_notes: dictionary that associates durations in seconds with letter

BEGIN
    If invert = True then
        list_notes ← invert(list_notes)
    End If
    If transpos = True then
        list_notes ← transpose(list_notes, int(amount_of_transposition))
    End If
    For index ← 1 To size(list_notes) Do
        note_and_duration[index] <- list_notes[index]
        duration ← 0
        If note_and_duration{index] != "p" Then
            duration ← duration + duration_of_notes[note_and_duration[-1]]
            note ← note_and_duration[:-1]

            If index != size(list_notes) - 1 and list_notes[index+1] = "p" Then
                duration ← duration + duration/2
            End If
            new_list[index]←(note,  duration)
        End If
    End For
    Return new_list
END
```

## GET_ALL_SONG_TITLES

**Information**:
The goal of this function is to create a dictionary associating a song number with the title of the song. It is mainly for drawing the titles on the screen with the GUI
**Return**:
This Function returns a dictionary.

```
Function get_all_song_titles(file : name of a text file)
Copied parameters: file
Local variables: all_titles : dictionary
Specific functions:
        dict() :creates a dictionary
        strip() : removes special characters in this case : “\n”
        size() :take the size of a list, dictionary, file
        readlines(): reads all lines of a file

BEGIN
        dict(all_titles)
        readlines(file)
        For line ← 1 to size(lines) Do
                If line[0] = "#" Then
                        If line[2] = " " Then           # if the number of the song is only one digit
                                all_titles[line[1]] ← strip(line[3:])
                        Else Then                       # if the number of the song is 2 digits
                                all_titles[line[1:3]] ← strip(line[3:])
                        End If
                End If
        End For
        Return all_titles
END
```

## Transpose

**Information**:
this function allows you to transpose an entire list of notes by calling the "transposenote" function which allows you to transpose a note.
It sends the note to the transposenote function only if the note is different to “Z” and “p”
**Return**:
list
<u>Function transpose(list_of_notes : 1D array ; transposition : integers) return list</u>
**Copied parameters:** list_of_notes, transposition
**Local variables:**

```
        newline : 1D array
        note: string
```

**Specific function:**

```
        transposenote() detail right after this function
        size(): take the size of a list, dictionary, file
BEGIN
        For index ←1 to size(list_of_notes) Do
                If list_of_notes[index][:1] =/ "Z" and list_of_notes[index][:1] =/ "p" Then
                        newline[index] ← transposenote(list_of_notes[index], transposition)
                Else Then
                        newline[index] ← list_of_notes[index]
                End If
        End For
        Return newline
End
```

## Transpose_note:

**Information**:
This function transposes a note sent by the function transpose
**Returns**:
string
<u>Function transpose_note(note:string, amount:integer) return list</u>
**Copied parameters:** note, amount
Notes: list of notes from settings:( ["DO", "RE", "MI", "FA", "SOL", "LA", "SI", "Z"])

**Local parameter**: list: list of strings

```
                    Index:integer
Specific functions:
                    Index(): gets the index of a value in a list
BEGIN
          list <- notes[:-1]   # removes the Z of the list "notes"
          index ← list.index(note[:-1]) + amount
          while index >= 7 do
             index ← index – 7
          end while
          note ← list[index] + note[-1:]
          return note
END
```

## Invert

**Information :**
This function allows you to invert a list of notes, As for the transpose function, it calls the function invert_note only if the note is not a p
**Return :**
This function returns a list

Function invert(list : 1D array):      returns a list
**Copied parameters** : list
**Local variables:** inversed_list: newlist of strings
**Specific functions :**
size() :take the size of a list, dictionary, file

```
BEGIN
   For index <- 1 To size(list) Do
          If list[index] != "p" Then
                    inversed_list[index] ←inverse(element)
          Else Then
                    inversed_list[index] ← element
          End If
   End For
   Return inversed_list
END
```

## Invert_note

**Information :**
This functions inverts a note
**Return :**
This function returns a string

Function inverse_note(note : string) return string
**Copied parameters** : note
**Local variables :**
newduration :  string
durations : 1D array
notes : 1D array of notes (from settings file)
duration_of_notes : 1D array of durations(from settings file)

**Specific Function:**
size() :take the size of a list, dictionary, file
index() :returns the index of the specified element in the list.

```
BEGIN
          newnote ← notes[(size(notes) - notes.index(notes[:-1])) % size(notes)]
```

```
        newduration 🡐 durations[(size[durations] - durations.index(notes[-1:])) % size(durations)]
        Return newnote + newduration
END
```

## Addsong

**Information** :
This function allows you to add a song to a text file.

<u>Function addsong(notes : string or list ; title : string, file : string) no return</u>
**Copied parameters** : notes, title, file
**Local variables** : Fd, n : integers
**Specific function :**

```
        write() :  write in a file
        close() : close the file
        type() :  find the type of the variable
        join() : Join all items in a list into a string
        size() : takes the size of a list, dictionary, file

BEGIN
        If type(notes) = list Then
                notes ← " ".join(notes)          #if notes is a list, it creates a string with these notes
                                                 separating them by spaces
        Else Then
                notes ← notes[:-1]               #if it is a string, it takes of the space at the last position.
                                                 This is done because when writing the song in write player,
                                                 the program automatically adds a space after each note
        End If
        filesong ← open the txt folder with file as name in adding mode
        fd ← open the txt folder with file as name in reading mode
        n ← 0
        For line ← 1 to size(fd) Do
                n ← n+1
        End For
        If n >= 2 Then
                n ← n//2
        End If
        n ← n+1
        filesong ← write("#" + str(n) + " " + title + "\n" + notes[:-1] + "\n")
        filesong ← close
END
```

## Delsong

**Information:**
This function allows you to delete a song in the chosen text file.
For that we use the function "del" which allows us to delete a line. you have to delete twice the line with the same index because once the first delete (the title) the next line will take its place.

The second part of this function is to restore order in the number of songs. In fact, if one does not delete the last song, the song numbers get disordered. So in this function we correct this.

<u>Function delsong(numb : integer, file : string)</u>
**Copied parameters** : numb, file
**Local variables :**

```
                    number_of_before : integer
                    editfile : take a txt file
                    line : line of the txt file
```

**Specific function :**

```
                    readlines(): read lines in a textfile
                    del() : used to delete objects.
                    size() :take the size of a list, dictionary, file
```

```
Begin
    filesong ← open the txt folder with file as name in reading mode
    lines ← readlines(filesong)
    numb ← (numb - 1)*2
    del(lines[numb])
    del(lines[numb])
    filesong ← close
    editfile ← open the txt folder with file as name in writting mode
    number_of_before ← 0
    For line ← 1 To size(line) Do
        If line[0] = "#" Then
            If number_of_before + 1 /= int(line[1:3]) Then
                editfile←write("#" + str(int(number_of_before)+1 )+ line[3:])
            Else Then
                editfile ← write(line)
            End If
            number_of_before ← number_of_before + 1
        Else Then
            editfile ← write(line)
        End If
    End For
    editfile ← close
End
```

## Create_markov_map

**Information:**
This function creates Markov maps for the inputted song of a file. It takes a file and song number. We chose to use the number of repetitions of each note to create the new song. We also chose to create a Markov map for note durations as well.
**Return:**
So this function has 4 returns:

- 2D array: succession table for notes
- 2D array: succession table for durations
- 2 * 1D array: total repetitions for notes and for durations
-

<u>Function create_markov_map(numb : string, file : string): return 2 * 2D arrays, 2 * 1D arrays,</u>

**Copied parameters :** numb, file, durations(list of durations from settings file)
**Local variables :**

```
                    Songlist: list of strings
                    total_durations : 1D array
                    total_notes : 1D array
                    Markovmap_durations : 2D array
                    Markovmap_notes : 2D array
                    before_duration : 1D array
                    before_note : 1D array
                    I : integer
```

Specific function:

Get_song(file, numb): retrieves the notes and duration in the form of a list
filter() :filters the given iterable with the help of a function that tests each element in the iterable to be true or not.
size() :take the size of a list, dictionary, file
list() : returns a list
index() :returns the index of the specified element in the list.

```
BEGIN
    songlist ← get_song(file, numb)
    songlist ← list(filter(lambda :  note /= "p", songlist))   #removes all the “p” from list

    # calculate the total number of durations
    total_durations ← [0, 0, 0, 0]
    for index← to size(songlist) do
      total_durations[durations.index(songlist[index][-1:])] <- 1+
    total_durations[durations.index(songlist[index] [-1:])]

    # fill lines of matrix for duration with 0
    for i← to 5 do
      markovmap_durations[i]← ([0, 0, 0, 0])

    # count sucessors of each duration
    for i← to size(songlist)  do
      if i > 0 then
        before_duration <- songlist[i - 1][-1:]
        markovmap_durations[durations.index(before_duration)][durations.index(songlist[i][-1:])] <-
    markovmap_durations[durations.index(before_duration)][durations.index(songlist[i][-1:])] + 1
      end if
    end for

    #remove durations
    for i← size(songlist) do
      songlist[i] ← songlist[i][:-1]

    #calculate total number of notes notes:
    total_notes ← [0 ,0 ,0, 0, 0, 0, 0, 0]
    for index←1 to size(songlist) do
      total_notes[notes.index(note)] <- total_notes[notes.index(note)]+1
    end for

    #create matrix of successors for each note
    markovmap_notes ← []
    for i← to 8 do
      markovmap_notes.append([0, 0, 0, 0, 0, 0, 0, 0])
    end for

    #count sucessors of each note
    for i← to size(song list)-1 do
      if i>0 then
        before_note ← songlist[i-1]
        markovmap_notes[notes.index(before_note)][notes.index(songlist[i])] ←
    markovmap_notes[notes.index(before_note)][notes.index(songlist[i])] + 1
      End if
    End for
    return markovmap_notes, total_notes, markovmap_durations, total_durations
END
```

## Create_markov_map_all_songs

**Information:**

## The goal of this function si to create markov maps of all the songs of a file

Function create_markov_map_all_songs(file : string)

```
Copied parameters : file
Local variables :
        markov_map_notes_all_songs : 2D array
        markov_map_durations_all_songs : 2D array
        total_notes : 1D array
        total_durations : 1D array
        map : array of arrays
        markov_map_notes: 2D array
        all_notes: 1D array
        markov_map_durations: 2D array
        all_durations: 1D array
Specific functions :
        size() :take the size of a list, dictionary, file
        sum_of_maps() :
        create_markov_map() :
        get_a ll_songs_titles() :
        Sum_of_maps() :
BEGIN
        # this function uses the function "create_markov_map" to create a markov map of a whole file,
        # it returns 4 elements, the 2 markov maps(notes and duration)
        # and the 2 lists containing the number of occurrences of each note and each duration

        song_titles ← get_a ll_songs_titles(file)

        #create all 4 maps and lists, and fill them with zeros
        For i ← 1 to 8 Do
                markov_map_notes_all_songs[i] ← [0, 0, 0, 0, 0, 0, 0, 0]
        End For
        For i ← 1 to 4 Do
                markov_map_durations_all_songs[i] ← [0, 0, 0, 0]
        End For
        total_notes ← [0, 0, 0, 0, 0, 0, 0, 0]
        total_durations ← [0, 0, 0, 0]
        For song ← 1 to size(song_titles) Do
        # adds all the maps and lists to obtain the sum of all 4 lists and maps

                map ← create_markov_map(song, file)
                markov_map_notes ← map[0]
                all_notes ← map[1]
                markov_map_durations ← map[2]
                all_durations ← map[3]

                # sums markov map for the notes
```

## Markov_new_song

Function markov_new_song(markovmapnotes : 2D array, total_notes : 1D array , markovmapdurations : 2D array, total_durations : 1D array, songlength : integer)

```
Copied parameters : markovmapnotes, total_notes, markovmapdurations, total_durations, songlength
Local variables :
k, i, j : integers
succesors_matrix_line : 1D array
firstnote : string
notes : 1D array
new_song : 1D array
```

beforenote : string
succesors_matrix_notes : 2D array
succesors_matrix_durations : 2D array
durations :
firstduration :
beforeduration : 1D array
Specific function :
size() :take the size of a list, dictionary, file
index() :returns the index of the specified element in the list.
random.choice() : randomly choose an item from a list and other sequence types.

```
BEGIN
    # This function creates a new song using the markov map technique
    # it returns the partition created
    # create a list of lists of all successors for each note

    For k ← 1 to size(markovmapnotes) Do
        For i ← 1 to size(line) Do
            For j ← 1 to line[i] Do
                succesors_matrix_line[j] ←notes[i]
            End For
        End For
        succesors_matrix_notes[k] ← succesors_matrix_line[i]
    End For
    # take the most played note in song and set it as first note

    firstnote ← notes[total_notes.index(max(total_notes))
    new_song[1] ← firstnote
    # create new partition only composed of notes by randomly choosing in lists of all successors

    For i ← 1 to int(songlength) Do
        beforenote ← new_song[i-1]
        new_song[i] ← random.choice (succesors_matrix_notes [beforenote.notes.index (beforenote)])
    End For
    # create a list of lists of all successors for each duration

    For k ← 1 to size(markovmapdurations) Do
    succesors_matrix_durations
        For i ← 1 to size(line) Do
            For j ← 1 to size(line[i]) Do
                succesors_matrix_line[j] ← durations[i]
            End For
        End For
        succesors_matrix_durations[k] ← succesors_matrix_line[i]
    End For
    # take the most played duration in song and add it to the first note
    firstduration ← durations[total_durations.index(max(total_durations))]
    new_song[0] ← new_song[0] + firstduration
    # add to new partition durations by randomly choosing in lists of all duration successors

    For i ← 1 to int(songlength)):
        beforeduration ← new_song[i-1][-1:]
        new_song[i] ← new_song[i] + (random.choice
(succesors_matrix_durations[beforeduration.index(beforeduration)]))
    End For

    Return new_song
END
```

# IV) Conclusion

## I) Difficulties encountered

It was our first-time coding such a large application. We have realized very small games before but nothing as consequent. It was therefore a challenge for us to carry out this project. We came across a few difficulties:

At first, we directly jumped head-first into coding without thinking about structuring our code, or how we were going to use it. This led to many modifications being made during the coding process, which made us lose many hours coding some parts we never used.

We also had a hard time thinking about the logic of our GUI. It was hard to think about how to navigate between multiple screens even though in the end we figured out a method that works very well. We thought about using classes (thanks to YouTube tutorials) but we did not go that way because we weren't 100% sure we understood all the concepts. It was extremely difficult to code the GUI, the module we chose: "Pygame" does not have any useful functions for this use, we therefore had to code everything such as buttons and text boxes…ext.

Another difficulty we've encountered was understanding some of what was asked. The part about the Markov maps was very hard to understand a first, and even harder to code.

We also had hard times sharing the work. We quickly realized that we had different ways of thinking, and that all functions had to work together. It was a real challenge for us to coordinate every piece of work we did separately especially to not be able to see each other.

Overall, even though we had difficulties to carry out this project, we took great pleasure in realizing it and learnt many important aspects of programming.

## II) What we learned from this project

Being new to python we have learned a great amount of knowledge thanks to this project. Whether it be in relation with the coding aspect or the teamwork aspect.
Thanks to this project we have improved our coding competence. We learned a lot about organization, and about the mentality needed to code such a project.

One of the main elements we learned was that the project never ends. We quickly realized that the more you add to a project, the more you want to add to it. In addition to that, the more sophisticated the program gets, the more complicated data structures we have to manipulate. To today, we are still adding elements to the program or modifying certain algorithms to make it as best we can.

We also learned from our experience (as users) with application and games of all sorts. It was very fun to compare our application with the knowledge we had about applications in general. We thought about what makes a certain application better than another one. Even though it is obvious we still have so much to learn, we did our best to make it as dynamic as possible (buttons that change color, text box ect...)

We quickly realized how time consuming it was to code and how the most basic element of an application sometimes requires tens of lines of code. Another element we learnt was that the choice of modules (at least in python) is extremely important, before using a module we have to do a consequent amount of research to make sure not only that the module is applicable for what we want to do, but also that using them is simple enough.

We noted that competences completely external to programming like musical knowledge or digital drawing abilities was really helpful for a project like this.
This taught us that it was very useful to know how to use these types of skills in order to reuse them in a project or to learn about the skills from our project partners.

We also learned to ask for help and the importance of not getting stuck in the face of a problem by seeking advice from friends, teachers or even from the internet.

It was also important to get in the habit of checking if the programs of our functions worked well and were well secured to have a stable and reliable project.

## III) What can be improved

We constantly have new ideas of what to add to this project. One of the elements that we tried to add with no success was an animation of a music note to accompany the piano. It was strange but “pygame” did not want to recognize the format of the images even though it was the same format as the other ones. To this day this mystery remains unsolved.

Another element we tried to add with no success was a pause button when playing a song. It seems that the while loop while playing a song was maybe too long to execute to detect other key presses.

It might also have been interesting to add a functionality allowing to modify a song already written in the database by the user. It would have been great to incorporate a “edit” button to modify it directly through the app. This would have allowed to improve the different songs added without having to rre-write it from scratch.

We could have also added to the functionality "markov chain mode" to take into account the "p" as well. This would made it possible to have even more precise statistics on the songs and to make them even more similar to the various databases such as the partitions files or directly partition song.

Overall, we took this project as an opportunity to learn python coding in detail. We realized how addictive coding is and building an application from scratch can be. It is obvious we still have many aspects to learn, and this project comforted us in our choice of studies. We are looking forward to the next project in C.

Don’t hesitate to send us an email or message by teams if you have any questions regarding the project.

Cleophas Fournier, cleophas.fournier@efrei.net
Paul Zamanian, paul.c.zamanian@gmail.com