Rapport final - Projet "Transcription automatique de piano"

Alice Lebrun et Laure Grisez - Groupe 15

22 Mai 2022

Sommaire

1 Introduction

Rappel de la consigne : *"Ce projet porte sur la transcription de la musique de piano. Il consiste à développer un programme permettant d'analyser le son d'un fichier audio pour estimer les notes jouées avec les paramètres associés et en particulier, les instants d'attaque et les durées de ces notes de musique. L'application doit ensuite créer une représentation visuelle intelligible des notes jouées."*

Le projet "Transcription automatique de la musique de piano" a débuté le 22 mars 2022, et nous avions deux mois pour le réaliser. Notre travail a été régulièrement résumé dans des rapports. L'objectif de la matière IN104 est de nous rendre toujours plus autonomes en informatique, et de nous faire prendre en main l'outil git pour le développement collaboratif et la plateforme github pour l'hébergement du développement à travers un projet complexe, celui de la transcription du piano.

Le piano est un instrument très difficile à transcrire, notamment à cause de sa polyphonie. Nous avons alors dû faire des hypothèses simplificatrices et réduire le champ des audios possiblement analysables pour être en mesure de développer un programme performant dans le temps imparti. Nous avons ainsi décidé de nous concentrer sur les fichiers son :

- au format .wav,
- codés en mono ou stéréo 16 bits little endian. Les fichiers stéréo sont convertis en signal mono par le programme en faisant la moyenne des signaux des deux canaux.
- contenant une note, ou plusieurs notes ne se superposant pas.

2 Démarche

Durant le premier mois du projet, nous avons réfléchi à la structure générale de notre programme. Nous avons également effectué des recherches sur internet, pour trouver des codes dont nous pourrions nous inspirer tant pour la partie graphique, la partie son que le traitement du signal. Nous avons lu la documentation de plusieurs bibliothèques, commme SDL ou FFTW3, et avons regardé le code du logiciel de traitement audio Audacity pour le calcul de spectrogramme.

Nous avons décidé de segmenter notre code en cinq modules utilisés par un programme principal. Pour transformer le fichier brut au format .wav en un signal exploitable numériquement et interagir avec le système sonore de l'ordinateur, nous avons créé le module *son*. Nous avons ensuite réfléchi à comment modéliser une note, et nous l'avons défini dans le module *note*. Note gère également la structure de données liée au stockage de la liste de notes à jouer, ainsi que la structure du clavier du piano. Une note est constituée d'un instant de début et d'un instant de fin (en milliseconde depuis le début du morceau) et d'un pointeur vers la touche du clavier qui lui est associée. La touche comporte un nom de note, une fréquence (en Hertz) ainsi qu'un indice et une position permettant de la localiser sur le clavier et au sein de l'image du clavier. Le clavier est un tableau de 88 touches.

À partir des données obtenues grâce à *son* et rendues exploitables grâce à *note*, nous déterminons les notes jouées dans notre audio à l'aide du module *transcription*, qui contient toute l'intelligence relative au traitement du signal, notamment les calculs de transformée de Fourier. Ce module permet de convertir un signal audio en une séquence de notes, et d'en donner le nom. L'algorithme est basé sur un découpage du signal sonore en blocs consécutifs tous de même longueur et en la détermination de la note principale jouée dans ce bloc à l'aide de la maximisation du produit spectral. Deux blocs consécutifs associés à une même note sont fusionnés, et un bloc dont le contenu est trop peu énergétique est associé à un silence.

Le projet de transcription comprend aussi une composante graphique. En effet, après avoir associé le signal sonore à une séquence de notes, il faut animer cette séquence pendant l'audition du signal. Il faut donc afficher le clavier d'un piano et l'animer afin que les notes jouées se colorent tout le temps de leur activation, et également visualiser les notes actives ou prochainement actives sous la forme de rectangles verticaux tombant sur le clavier du piano. Pour gérer cet aspect, nous avons créé les modules *graphique* et *interface*. Basés sur la bibliothèque SDL, ils se chargent respectivement

du moteur graphique pour l'un (activation des capacités de SDL, services de chargement d'image et de copie d'image à l'écran), et de la création et de l'animation de la fenêtre graphique pour l'autre.

Enfin, le module *transcription piano* est le programme principal. Il coordonne tous les modules entre eux afin de réaliser la lecture du fichier à analyser, son analyse, son audition et son animation simultanée.

Après avoir décidé de ce découpage du programme global, nous avons commencé à coder les différentes fonctions utiles, en profitant des vacances d'avril pour beaucoup avancer dans la programmation. Nous avons également commencé à prendre en main git et github, que nous avons continué à utiliser par la suite, pour partager nos avancements respectifs sur le code.

Afin de simplifier la compilation du projet, la détection des dépendances et leur configuration, nous avons décidé d'utiliser l'outil de configuration de code CMake, plutôt que d'écrire directement des fichiers Makefile. Cet outil est bien documenté et se configure à l'aide de fichiers textes dont il est facile de trouver de exemples sur internet.

3 Modules

3.1 Son

Le module *son* contient toutes les fonctions permettant de transformer le signal au format .wav fourni en entrée du programme en une structure informatique exploitable. Le module s'appuie sur la bibliothèque SDL. Il fournit aussi les fonctions faisant le lien avec la carte son et permettant de jouer l'audio.

Le module contient également un test pour déterminer si l'audio de départ est codé en mono ou en stéréo. Dans le cas d'un signal stéréo, on effectue une moyenne pour obtenir un signal en mono à la fin.

Module son.h

```
#ifndef __SON_H__
#define __SON_H__

#include <SDL.h>

/* Initialisation du son */
int initialiser_son();

/* Chargement du fichier son */
int charger_son(char * nom_fichier, SDL_AudioSpec *wav_spec, Uint8 **wav_buffer, Uint32 *wav_length); //crée

/* Conversion du fichier son */
int convertir_son(SDL_AudioSpec *wav_spec, Uint8 *wav_buffer, Uint32 wav_length, double ** donnee_son, doubl

/* Libération du son */
void liberer_son(Uint8 *wav_buffer); //Libère la mémoire allouée pour traiter l'audio.

/* Fermer le moteur audio */
void fermer_son(SDL_AudioDeviceID deviceID);

/* Jouer son */
int jouer_son(SDL_AudioSpec *wav_spec, Uint8 *wav_buffer, Uint32 wav_length, SDL_AudioDeviceID deviceID);
```

```
#endif
```

3.2 Note

Le module *note* gère la définition informatique d'une note, ainsi que les différentes manipulations qui lui sont relatives. Nous avons défini une note comme une structure contenant un instant de début, un instant de fin (les instants sont des nombres entiers de milliseconde) et une touche associée. Nous avons également défini une structure de liste circulaire doublement chaînée pour contenir la liste des notes qui seront jouées dans l'audio. Une touche est essentiellement associée à une fréquence. Il y a une différence fondamentale entre une *note* et une *touche*. Une *touche* représente la touche du piano produisant un certain son, comme un La 3. Une *note* est une portion du signal que l'on annalyse. On en extraira une fréquence fondamentale dans le module *transcription*, qui permettra de déterminer quelle *touche* a été frappée pour produire le son de la *note*.

On a également défini plusieurs variables globales qui permettront de préciser le type de chaque touche. En effet, un piano possède cinq types de touches : les noires, les blanches entre deux noires, les blanches avec une touche noire à gauche et une blanche à noire, les blanches avec une touche noire à droite et une blanche à gauche, et la blanche tout à droite du clavier, qui possède uniquement une touche blanche à sa gauche.

Module note.h

```
#ifndef __NOTE_H__
#define __NOTE_H__

#include <SDL.h>

#define NOMBRE_TOUCHES 88
#define H_BLANCHE 80 //hauteur d'une touche blanche
#define L_BLANCHE 15 //largeur d'une touche blanche
#define H_NOIRE 51
#define L_NOIRE 9
#define CENTRE 0 // Quand la touche blanche est au centre de 2 noires
#define DROITE 1 // Quand la touche blanche est à droite d'une touche noire
#define GAUCHE 2 // Quand la touche blanche est à gauche d'une touche noire
#define NOIRE  3
#define PLEINE 4 // La note blanche la plus à droite est à droite d'une blanche mais n'a pas de touche noire

// Structure note: deux points dans le temps et une touche
struct note_t {
  Uint32 t_debut;
  Uint32 t_fin;
  struct touche_t * touche;
};

// Liste doublement chaînée circulaire des notes présentes dans le signal
struct liste_note_t {
  struct note_t * note;
```

```
  struct liste_note_t * precedent;
  struct liste_note_t * suivant;
};

// Structure touche: un nom, une fréquence (pour l'identification), une image à afficher si la touche est ac
struct touche_t {
  char * nom;
  double frequence;
  int type; //Pour savoir si c'est une blanche à gauche / à droite / au centre par rapport à une touche noir
  SDL_Rect position;
};

struct clavier_t {
  struct touche_t touches[NOMBRE_TOUCHES]; //Un clavier contient 88 touches
};

// Ajoute une note à une liste de note de façon à ce que les notes soient
// dans l'ordre croissant d'instant de début
struct liste_note_t * ajouter_note(struct liste_note_t * liste, Uint32 t_debut, Uint32 t_fin, struct touche_

// Supprime une note de la liste et détruit la note
struct liste_note_t * supprimer_note(struct liste_note_t * liste, struct note_t * note);

// Affiche les caractéristiques d'une note
void afficher_note(struct note_t * note);

// Affiche la liste des notes
void afficher_liste_notes(struct liste_note_t * note);

// Crée un clavier
int creer_clavier(struct clavier_t *clavier);

#endif
```

3.3 Transcription

Le module *transcription* a pour rôle de transformer les données de l'audio en une liste de notes. Pour cela, nous nous basons sur le calcul de la Transformée de Fourier, rendu possible en utilisant la bibliothèque FFTW3.

La stratégie consiste à partitionner le signal en blocs. Si dans ces sous-blocs le signal contient assez d'énergie on identifie la fréquence dominante dans le bloc, ce qui permet d'associer une note (et donc une touche) au bloc en prenant la touche dont la fréquence est la plus proche de celle du bloc. Si la fréquence dominante du bloc correspond à la fréquence dominante du bloc précédent on considère qu'il s'agit de la même note qui se prolonge. Si le bloc ne contient pas assez d'énergie il est associé à un silence.

Pour chaque bloc, nous commençons par calculer le spectrogramme du signal de ce bloc. Ce calcul se fait de la manière suivante :

1. Le bloc est découpé en K sous-blocs de taille N selon le schéma de la figure **??**.

2. Le signal $s^{(k)}(n)$ correspondant à chaque sous-bloc k est filtré par le filtre de Hamming, selon la formule :
   $\forall n \in 0, \ldots, N-1, \quad x^{(k)}(n) = s^{(k)}(n) \times \left(\frac{25}{46} - \frac{21}{46}\cos\left(2\pi\frac{n}{N-1}\right)\right)$
   On remarque que les valeurs du filtre de Hamming ne dépendent pas du sous-bloc k, elles sont donc calculée une seule fois pour le bloc. Nous aurions pu les partager entre les blocs mais nous avions prévu d'avoir des blocs de taille différente et nous n'avons pas eu le temps de modifier le code.
3. Il faut alors calculer la transformée de Fourier discrète $X(p) = FFT(x(n)$ à l'aide de l'algorithme de FFT implémenté dans la librairie FFTW3 pour les signaux réels `fftw_plan_dft_r2c_1d`
4. Le spectrogramme associé au bloc est alors la moyenne des spectrogrammes associés à chaque sous-bloc k. Comme le spectrogramme va être utilisé pour optimiser un produit spectral, il est inutile de le normaliser par K.

Il s'agit de l'algorithme utilisé par le logiciel Audacity pour calculer le spectre d'un signal.

. Puis, nous calculons le produit spectral, et recherchons la fréquence permettant de le maximiser. Nous relions ensuite cette fréquence à la fréquence de la touche la plus proche, ce qui nous permet de déterminer quelle est la note jouée.

Module transcription.h

```
#ifndef __TRANSCRIPTION_H__
#define __TRANSCRIPTION_H__

#include "note.h"

struct liste_note_t* transcrire(double * donnee_son, double * instant_son, int taille, struct clavier_t * cla

#endif
```

3.4 Graphique

Module graphique.h

```
#ifndef __GRAPHIQUE_H__
#define __GRAPHIQUE_H__
#include <SDL.h>

//Cette fonction vérifie qu'on peut lire/écrire des fichiers image et d'afficher des pixels à l'écran.
int initialiser_graphique();

/* Charge le fichier png dans une texture SDL2 */
int charger_texture(char *nom_image, SDL_Renderer * renderer, SDL_Texture **texture);

/* Copie une texture à une position donnée dans le moteur de rendu */
int copier_texture(SDL_Texture *texture, SDL_Rect *position, SDL_Renderer *renderer);

/* Libère une texture SDL2 */
void liberer_texture(SDL_Texture *texture);

/* Libère le moteur audio SDL2 */
```

```
void fermer_graphique();

#endif
```

3.5 Interface

Ce module contient la définition de la fenêtre graphique comme une structure contenant de types issus de la bibliothèque SDL. Plusieurs paramètres essentiels à l'animation de la fenêtre y sont également définis, comme la vitesse de défilement des pixels, ou la largeur et la hauteur des rectangles qui défileront du haut de l'écran vers le bas pour représenter les notes qui seront jouées. Ce module gère aussi toutes les images à afficher, comme les images du clavier ou les touches.

Pour illustrer le clavier du piano, nous avons téléchargé une image de piano, puis nous l'avons rognée afin de ne plus voir que la partie clavier.

Afin d'animer le clavier lorsqu'une touche devient active, nous avons découpé dans l'image du clavier les cinq formes de touches différentes : les touches blanches entre deux noires, les touches blanches à la droite d'une touche noire, les touches blanches à la gauche d'une touche noire, les touches noires et la touche blanche non voisine d'une touche noire (la touche la plus à droite du clavier). Nous avons modifié la couleur de chacune de ces touches selon le code couleur suivant : une touche blanche devient jaune lorsqu'elle est activée (le même jaune que les rectangles signifiant leur activation prochaine ou en cours) et une touche noire devient verte selon le même principe.

Comme il est possible que deux touches voisines soient actives simultanément (même si la version actuelle de l'algorithme de transcription ne le permet pas), il a fallu rendre transparente toute la zone des images rectangulaires des touches qui ne correspondent pas à la touche proprement dite, puis prendre en compte cette transparence (canal alpha) dans SDL à l'aide de l'option `SDL_BLENDMODE_BLEND` de la fonction `charger_texture` du module graphique.

Toutes les manipulations d'image ont été faites à l'aide du logiciel GIMP.

Images du programme

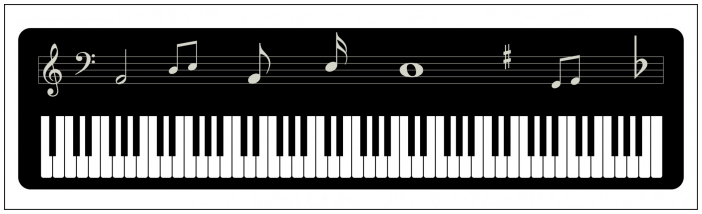

FIGURE 1 – Image initiale pour le clavier

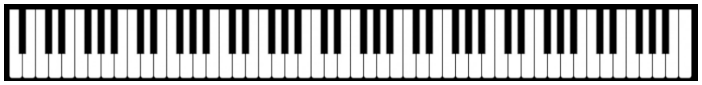

FIGURE 2 – Image finale pour le clavier

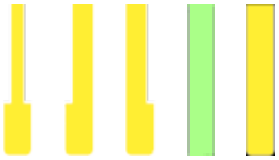

FIGURE 3 – Les différents type de touches pour l'animation du clavier

Module interface.h

```
#ifndef __INTERFACE_H__
#define __INTERFACE_H__

#include <SDL.h>
#include "note.h"

// Vitesse de défilement en pixels par seconde
#define VITESSE_NOTE 100
#define LARGEUR_NOTE 6
#define LARGEUR_INTERFACE 800

struct interface_t {
  SDL_Window *fenetre;
  SDL_Renderer *renderer;
  SDL_Rect position_clavier;
  SDL_Texture * texture_clavier;
  SDL_Texture * textures_touches[5];
};

// Crée l'interface graphique en retournant une structure interface
int creer_interface(struct interface_t *interface);

// Anime l'interface à l'aide de la séquence de notes et du fichier son, pour une durée en milliseconde
void animer_interface(struct interface_t *interface, struct liste_note_t *notes, Uint32 duree, SDL_AudioSpec

// Détruit l'interface et les éléments qui lui sont associés
void liberer_interface(struct interface_t * interface);

#endif
```

4 Complexité, mémoire, gestion des erreurs

4.1 Complexité

La complexité algorithmique du programme tient à deux aspects distincts :

— La nature de l'algorithme de transcription choisi ;
— Les structures de données mises en œuvre.

En ce qui concerne l'algorithme, il se base sur de nombreux calculs de transformée de Fourier discrètes, qui constituent l'essentiel de la charge de calcul.

Toutes ces transformées sont réalisées par l'algorithme de transformée de Fourier rapide qui permet de calculer la transformée de Fourier discrète d'un signal échantillonné en $N = 2^k$ points d'une grille régulière avec une complexité en $\mathcal{O}(N \log(N))$ au lieu de $\mathcal{O}(N^2)$ via l'algorithme naïf de sommation. Afin de réduire au maximum ce coût, nous avons choisi d'utiliser la librairie FFTW3, dont les performances dominent celles des autres implémentations disponibles. Par ailleurs, cette librairie permet d'utiliser des échantillonnages N non forcément puissances de 2 tout en conservant un coût en $\mathcal{O}(N \log(N))$, ce qui constitue un avantage lorsqu'il faut choisir entre $N = 2048$ et $N = 4096$ par exemple. Enfin, cette librairie propose de calculer la FFT en deux temps : une première étape de calibration (qui peut être réutilisée entre plusieurs calculs) qui choisit la meilleure organisation des calculs étant données leur taille et leur localisation en mémoire, et une seconde étape de calcul effectif de la FFT. Cette réutilisation est faite pour l'estimation du spectrogramme d'un segment de signal donné, qui est réalisée en sélectionnant différents sous-segments sur lesquels les calculs de FFT sont effectués avant d'être moyennés.

Bien évidemment nous avons tiré profit de la présence de fonctions de calcul optimisées pour des données réelles ; qui sont deux fois plus rapides que les fonctions associées aux données complexes de même taille. Nous avons également appliqué le fenêtrage de Hamming afin d'améliorer la qualité de l'estimation.

Une autre optimisation a consisté à n'effectuer l'identification d'une note dans un bloc de signal que si ce bloc contenait une énergie suffisante par rapport à l'énergie moyenne par bloc du signal, évitant ainsi d'estimer un spectrogramme sur du bruit de faible niveau.

En ce qui concerne les structures de données, même en se limitant à des pièces de piano jouées une touche à la fois, le nombre de notes générées dépasse plusieurs centaines pour des morceaux de plusieurs minutes. Il est indispensable d'utiliser une structure permettant l'insertion rapide des notes, mais également leur insertion triée par instant de début croissant afin d'arrêter au plus tôt le parcours de cette structure lors de la phase d'animation (les notes trop dans le futur pour donner lieu à une visualisation). Enfin, il est nécessaire de pouvoir supprimer une note rapidement afin d'alléger la structure de stockage dès que la note a été complètement jouée lors de l'animation. Cette structure de données doit être dynamique puisque le nombre d'éléments à y stocker n'est pas connu à l'avance.

Pour ces différentes raisons nous avons avons choisi d'utiliser une liste doublement chaînée circulaire. La circularité permet de parcourir la liste dans n'importe quel ordre, et d'insérer ou de supprimer un élément de manière très efficace. Comme les notes sont identifiées essentiellement dans l'ordre inverse de celui dans lequel elles seront animées, il est très efficace de les insérer dans la liste en partant de la tête et en allant vers la queue dans le sens rétrograde. La queue de la liste est le voisin immédiat de la tête dans ce sens de parcours, et même si plusieurs notes sont identifiées dans un bloc de signal donné, leur insertion se fera en $\mathcal{O}(1)$ comparaisons/modifications de pointeurs. La même complexité est obtenue pour la suppression en parcourant la liste dans l'ordre naturel depuis la tête, puisque ce seront essentiellement les notes de tête qui seront supprimées en premier.

4.2 Mémoire

Afin d'éviter d'utiliser de la mémoire inutilement, nous avons systématiquement testé si les allocations de mémoire avaient été effectuées avec succès.

Dans le module *note*, nous avons décidé de créer la liste des notes en la structurant comme une liste circulaire doublement chaînée pour des raisons d'économie de mémoire. En effet, cette structure permet de supprimer facilement des éléments, ainsi nous pouvons oublier les notes déjà jouées et ne plus les stocker en mémoire.

4.3 Gestion des erreurs

La principale cause d'erreur que nous avons traitée est l'impossibilité d'allouer de la mémoire. Dans ce cas, le programme affiche un court message décrivant brièvement l'erreur rencontrée, puis renvoie la valeur arbitraire -1 pour stopper les calculs.

5 Difficultés rencontrées

Les principales difficultés que nous avons rencontrées dans ce projet ont été d'effectuer un découpage intelligent du programme en sous modules, de prendre l'habitude de github et de nous plonger dans les documentations des différentes librairies.

Pour certains morceaux, le programme se trompe d'un demi-ton, et affiche par exemple un ré dièse au lieu d'un ré. Une autre erreur consiste ausssi à affiché non pas la note jouée, mais la même note un octave au-dessus. Cela est directement lié à la méthode employée pour déterminer la fréquence fondamentale de chaque section de l'audio.

6 Résultats obtenus

Nous avons finalement obtenu un programme qui compile sans erreur, et qui renvoie une fenêtre graphique sur laquelle l'animation des notes apparaît. Notre programme est capable d'analyser des audios de plusieurs minutes, et est d'une précision très satisfaisante pour des audios contenant des notes jouées une par une, liées ou piquées.

7 Conclusion

Ce projet nous a permis de développer de nouvelles compétences liées à la gestion d'un projet informatique en autonomie : recherches de bibliothèques adaptées à nos besoins, lecture de documentation, segmentation du travail, prise en main de l'outil github... Nous avons également dû choisir entre avancer dans la programmation et prendre le temps de d'abord rendre nos fonctions les moins coûteuses possibles en terme de complexité. Nous avons pris le parti de nous concentrer en premier lieu sur l'avancement du code et la résolution des erreurs, avant de réfléchir à des moyens toujours plus astucieux d'alléger la complexité de nos fonctions. Avec plus de temps, nous aurions sûrement pu réduire encore le coût de certains modules.

Annexe : code C

Programme principal : *transcription_piano.c*

```
#include <SDL.h>
#include "son.h"
#include "transcription.h"
#include "interface.h"

void affiche_avec_separateur(char* message, char *sep) {
  for (int i = 0; i < strlen(message); ++i)
    printf("%s", sep);
  printf("\n");
  printf("%s\n", message);
```

```
  for (int i = 0; i < strlen(message); ++i)
    printf("%s", sep);
  printf("\n");
}

int main(int argc, char* argv[]) {
  if (argc != 2) {
    printf("Usage: transcription_piano fichier.wav\n");
    return 1;
  }
  char* nom_fichier = argv[1]; //nom de l'audio

  //On initialise le son en vérifiant qu'il n'y a pas eu d'erreur.
  if (initialiser_son() != 0) {
    return 1;
  }

  SDL_AudioSpec wav_spec;
  Uint8 *wav_buffer;
  Uint32 wav_length;

  //On charge le son en vérifiant qu'il n'y a pas eu d'erreur.
  if (charger_son(nom_fichier, &wav_spec, &wav_buffer, &wav_length) != 0) {
    return 1;
  }

  double * donnees_son; //amplitude du signal à chaque instant
  double * instants_son; //instants reliés aux amplitudes
  int taille;

  //On convertit le son en vérifiant qu'il n'y a pas eu d'erreur.
  if(convertir_son(&wav_spec, wav_buffer, wav_length, &donnees_son, &instants_son, &taille) !=0) {
    return 1;
  }
  Uint32 duree = 1000 * instants_son[taille - 1];
  //On cree le clavier
  struct clavier_t clavier;

  creer_clavier(&clavier);

  // Crée la transcription du signal en liste de notes
  char message[256] = "Transcription de ";
  strcat(message, nom_fichier);
  affiche_avec_separateur(message, "=");
  struct liste_note_t * notes = transcrire(donnees_son, instants_son, taille, &clavier);
  // On libère les structures du signal
```

```
  free(donnees_son);
  free(instants_son);
  affiche_avec_separateur("Resultat de la transcription", "-");
  afficher_liste_notes(notes);
  // Crée l'interface graphique
  struct interface_t interface;
  if (creer_interface(&interface) != 0) {
    printf("Erreur: impossible de créer l'interface graphique.\n");
    return 1;
  };
  // Lance l'animation des notes transcrites
  affiche_avec_separateur("Animation de la lecture", "-");
  animer_interface(&interface, notes, duree, &wav_spec, wav_buffer, wav_length);
  // Libère toutes les ressources
  liberer_son(wav_buffer);
  liberer_interface(&interface);
  SDL_Quit();
  return 0;
}
```

Module *son* :

```
#include "son.h"

/* Initialisation du son */
int initialiser_son() {
  if (SDL_Init(SDL_INIT_AUDIO) != 0) {
    SDL_Log("Erreur dans l'initialisation du son %s", SDL_GetError());
    return 1;
  }
  return 0;
}


/* Chargement du fichier son */
int charger_son(char * nom_fichier, SDL_AudioSpec *wav_spec, Uint8 **wav_buffer, Uint32 *wav_length) {
  if (SDL_LoadWAV(nom_fichier, wav_spec, wav_buffer, wav_length) == NULL) {
    SDL_Log("Le fichier %s n'a pas pu être ouvert: %s\n", nom_fichier, SDL_GetError());
    return 1;
  }
  if (wav_spec -> format != AUDIO_S16LSB) {
    SDL_Log("Le format n'est pas supporté.\n"); //SDL_Log est équivalent à un printf
    return 1;
  }
  return 0;
}
```

```
/* Conversion du fichier son */
int convertir_son(SDL_AudioSpec *wav_spec, Uint8 *wav_buffer, Uint32 wav_length, double ** donnee_son, doubl
  *taille = wav_length / (2 * wav_spec -> channels);
  //on divise par 2 pour passer de 16 bits à 8 bits, et par wav_spec -> channels car c'est le nombre de canna
  *donnee_son = malloc(*taille * sizeof(double));
  if (*donnee_son == NULL) {
    SDL_Log("Erreur: la mémoire n'a pas pu être allouée");
    return 1;
  }
  *instant_son = malloc(*taille * sizeof(double));
  if (*instant_son == NULL) {
    SDL_Log("Erreur: la mémoire n'a pas pu être allouée");
    free(*donnee_son);
    return 1;
  }
  double duree = (double)(*taille) / wav_spec->freq; //La durée n'est pas forcément un entier, on convertit

  //On teste si le fichier est en mono ou stéréo

  switch (wav_spec->channels) {
  case 1: // On est en mono.
    for (int i = 0; i < *taille; ++i) {
      (*instant_son)[i] = (i * duree) / (*taille - 1); //On parenthèse (i*duree) pour être sûr de faire une
      (*donnee_son[i]) = (double)(*(short *)&wav_buffer[2 * i]) / (1 << 15); //conversion des données audio
    } // mono
    break;
  case 2: // On est en stéréo.
    for(int i = 0; i< *taille; ++i) {
      (*instant_son)[i] = (i * duree) / (*taille - 1); //On parenthèse (i*duree) pour être sûr de faire une
      short gauche = *(short *)&wav_buffer[4 * i];
      short droite = *(short *)&wav_buffer[4 * i + 2];
      //Cette fois, on lit toujours les octets par 2 mais tous les 2 paquets, donc 4i->4i+2 puis 4i+4->4i+6.
      (*donnee_son)[i] = (double)((int)gauche + droite) / (1<<16); //conversion des données audio en valeurs
    }
    break;
  default:
    SDL_Log("Le format n'est pas supporté. \n");
    return 1;
  } // switch
  return 0;
}


/*Libération du son*/
void liberer_son(Uint8 *wav_buffer){
  SDL_FreeWAV(wav_buffer);
```

```
}

/*Libération du moteur audio*/
void fermer_son(SDL_AudioDeviceID deviceID) {
  SDL_CloseAudioDevice(deviceID);
}

/*Jouer son*/
int jouer_son(SDL_AudioSpec *wav_spec, Uint8 *wav_buffer, Uint32 wav_length, SDL_AudioDeviceID deviceID) {
  if (SDL_QueueAudio(deviceID, wav_buffer, wav_length) != 0) {
    SDL_Log("Erreur: impossible de mettre le morceau de musique dans la file audio: %s\n", SDL_GetError());
    return 1;
  }
  SDL_PauseAudioDevice(deviceID, 0);
  return 0;
}

//test git
```

Module *note* :

```
#include "note.h"

// Crée une note et l'ajoute dans la liste par ordre croissant de date de début
struct liste_note_t * ajouter_note(struct liste_note_t * liste, Uint32 t_debut, Uint32 t_fin, struct touche_
  struct note_t * note = malloc(sizeof(struct note_t));
  if (note == NULL) {
    return NULL;
  }
  struct liste_note_t *nouvelle_liste = malloc(sizeof(struct liste_note_t));
  if (nouvelle_liste == NULL) {
    free(note);
    return NULL;
  }
  // Remplir la note
  note->t_debut = t_debut;
  note->t_fin = t_fin;
  note->touche = touche;
  nouvelle_liste->note = note;
  // Si la liste est vide elle devient égale à la liste constituée de la nouvelle note
  if (liste == NULL) {
    // On boucle sur le maillon unique
    nouvelle_liste->suivant = nouvelle_liste;
    nouvelle_liste->precedent = nouvelle_liste;
    return nouvelle_liste;
  }
  /* Sinon
```

```
     Insérer la note à la bonne place. On parcourt la liste en remontant par les antécédents car les notes
     sont produites essentiellement dans l'ordre chronologique, les nouvelles notes ont donc plus de chance
     de se ranger en fin de liste
     On part du précédent de la tête, c'est-à dire la queue (qui peut être égale à la tête si la liste ne
     contient qu'un élément) */
  struct liste_note_t * courant = liste;
  // Tant qu'on n'a pas bouclé et que la note démarre avant la note courante on remonte la liste
  while (courant->precedent != liste && t_debut < courant->note->t_debut) {
    courant = courant->precedent;
  }
  // Arrivé ici on a bouclé ou on a trouvé une note de la liste actuelle postérieure à la note courante
  // Si on a bouclé c'est qu'il faut mettre la note à la fin
  if (courant == liste) {
    liste->precedent->suivant = nouvelle_liste;
    nouvelle_liste->precedent = liste->precedent;
    nouvelle_liste->suivant = liste;
    liste->precedent = nouvelle_liste;
  }
  else {
    nouvelle_liste->suivant = courant;
    nouvelle_liste->precedent = courant->precedent;
    courant->precedent->suivant = nouvelle_liste;
    courant->precedent = nouvelle_liste;
  }
  return liste;
}

// Supprime une note de la liste et détruit la note
struct liste_note_t * supprimer_note(struct liste_note_t * liste, struct note_t * note) {
  if (note == NULL || liste == NULL) {
    return liste;
  }
  // Si la liste ne contient qu'un élément...
  if (liste->suivant == liste) {
    // ...et qu'il correspond à la note
    if (liste->note == note) {
      free(liste->note);
      free(liste);
      return NULL;
    }
    // Sinon rien à faire
    else return liste;
  } // Un seul élément
  // Sinon on parcourt les éléments
  struct liste_note_t * courant = liste;
  while (courant->note != note) {
```

```
    courant = courant->suivant;
    // Si on est revenu en tete de liste on quitte la boucle
    if (courant == liste) {
      break;
    }
  } // while
  // A ce point on est sorti soit parce qu'on a parcouru toute la liste en vain soit parce que la note corres
  if (courant->note == note) {
    courant->suivant->precedent = courant->precedent;
    courant->precedent->suivant = courant->suivant;
    // Si on supprime la tête, la nouvelle tête est le suivant
    if (courant == liste) {
      liste = courant->suivant;
    }
    free(courant->note);
    free(courant);
  }
  return liste;
}

// Affiche les caractéristiques d'une note
void afficher_note(struct note_t * note) {
  if (note == NULL) {
    printf("note=NULL\n");
  } else {
    printf("note=(%s t_debut=%dms t_fin=%dms)\n", note->touche->nom, note->t_debut, note->t_fin);
  }
}

// Affiche la liste des notes
void afficher_liste_notes(struct liste_note_t * liste) {
  int nombre_notes = 0;
  if (liste == NULL) {
    printf("liste=NULL\n");
  } else {
    struct liste_note_t * courant = liste;
    do {
      ++nombre_notes;
      afficher_note(courant->note);
      courant = courant->suivant;
    } while(courant != liste);
  } // else
  printf("Nombre de notes=%d\n", nombre_notes);
}

// Crée un clavier
```

```
int creer_clavier(struct clavier_t *clavier) {
  // On construit maintenant les 88 touches du piano...
  clavier->touches[ 0].nom = "La-1";
  clavier->touches[ 0].frequence = 27.5;
  clavier->touches[ 0].type = GAUCHE;
  clavier->touches[ 0].position.x = 14;
  clavier->touches[ 0].position.y = 0;
  clavier->touches[ 0].position.w = L_BLANCHE;
  clavier->touches[ 0].position.h = H_BLANCHE;
  //
  clavier->touches[ 1].nom = "La-1dième";
  clavier->touches[ 1].frequence = 29.135235094880564;
  clavier->touches[ 1].type = NOIRE;
  clavier->touches[ 1].position.x = 800.0 / 1750.0 * 37;
  clavier->touches[ 1].position.y = 0;
  clavier->touches[ 1].position.w = L_NOIRE;
  clavier->touches[ 1].position.h = H_NOIRE;
  //
  clavier->touches[ 2].nom = "Si-1";
  clavier->touches[ 2].frequence = 30.867706328507698;
  clavier->touches[ 2].type = DROITE;
  clavier->touches[ 2].position.x = 800.0 / 1750.0 * 46;
  clavier->touches[ 2].position.y = 0;
  clavier->touches[ 2].position.w = L_BLANCHE;
  clavier->touches[ 2].position.h = H_BLANCHE;
  //
  clavier->touches[ 3].nom = "Do0";
  clavier->touches[ 3].frequence = 32.70319566257477;
  clavier->touches[ 3].type = GAUCHE;
  clavier->touches[ 3].position.x = 800.0 / 1750.0 * 79;
  clavier->touches[ 3].position.y = 0;
  clavier->touches[ 3].position.w = L_BLANCHE;
  clavier->touches[ 3].position.h = H_BLANCHE;
  //
  clavier->touches[ 4].nom = "Do0dième";
  clavier->touches[ 4].frequence = 34.64782887210895;
  clavier->touches[ 4].type = NOIRE;
  clavier->touches[ 4].position.x = 800.0 / 1750.0 * 103;
  clavier->touches[ 4].position.y = 0;
  clavier->touches[ 4].position.w = L_NOIRE;
  clavier->touches[ 4].position.h = H_NOIRE;
  //
  clavier->touches[ 5].nom = "Ré0";
  clavier->touches[ 5].frequence = 36.70809598967588;
  clavier->touches[ 5].type = CENTRE;
  clavier->touches[ 5].position.x = 800.0 / 1750.0 * 113;
```

```
clavier->touches[ 5].position.y = 0;
clavier->touches[ 5].position.w = L_BLANCHE;
clavier->touches[ 5].position.h = H_BLANCHE;
//
clavier->touches[ 6].nom = "Ré0dièse";
clavier->touches[ 6].frequence = 38.89087296526005;
clavier->touches[ 6].type = NOIRE;
clavier->touches[ 6].position.x = 800.0 / 1750.0 * 136;
clavier->touches[ 6].position.y = 0;
clavier->touches[ 6].position.w = L_NOIRE;
clavier->touches[ 6].position.h = H_NOIRE;
//
clavier->touches[ 7].nom = "Mi0";
clavier->touches[ 7].frequence = 41.203444614108676;
clavier->touches[ 7].type = DROITE;
clavier->touches[ 7].position.x = 800.0 / 1750.0 * 145;
clavier->touches[ 7].position.y = 0;
clavier->touches[ 7].position.w = L_BLANCHE;
clavier->touches[ 7].position.h = H_BLANCHE;
//
clavier->touches[ 8].nom = "Fa0";
clavier->touches[ 8].frequence = 43.65352892912542;
clavier->touches[ 8].type = GAUCHE;
clavier->touches[ 8].position.x = 800.0 / 1750.0 * 179;
clavier->touches[ 8].position.y = 0;
clavier->touches[ 8].position.w = L_BLANCHE;
clavier->touches[ 8].position.h = H_BLANCHE;
//
clavier->touches[ 9].nom = "Fa0dièse";
clavier->touches[ 9].frequence = 46.249302838954236;
clavier->touches[ 9].type = NOIRE;
clavier->touches[ 9].position.x = 800.0 / 1750.0 * 202;
clavier->touches[ 9].position.y = 0;
clavier->touches[ 9].position.w = L_NOIRE;
clavier->touches[ 9].position.h = H_NOIRE;
//
clavier->touches[10].nom = "Sol0";
clavier->touches[10].frequence = 48.999429497718594;
clavier->touches[10].type = CENTRE;
clavier->touches[10].position.x = 800.0 / 1750.0 * 211;
clavier->touches[10].position.y = 0;
clavier->touches[10].position.w = L_BLANCHE;
clavier->touches[10].position.h = H_BLANCHE;
//
clavier->touches[11].nom = "Sol0dièse";
clavier->touches[11].frequence = 51.91308719749307;
```

```
clavier->touches[11].type = NOIRE;
clavier->touches[11].position.x = 800.0 / 1750.0 * 235;
clavier->touches[11].position.y = 0;
clavier->touches[11].position.w = L_NOIRE;
clavier->touches[11].position.h = H_NOIRE;
//
clavier->touches[12].nom = "La0";
clavier->touches[12].frequence = 55.0;
clavier->touches[12].type = CENTRE;
clavier->touches[12].position.x = 800.0 / 1750.0 * 246;
clavier->touches[12].position.y = 0;
clavier->touches[12].position.w = L_BLANCHE;
clavier->touches[12].position.h = H_BLANCHE;
//
clavier->touches[13].nom = "La0dièse";
clavier->touches[13].frequence = 58.27047018976117;
clavier->touches[13].type = NOIRE;
clavier->touches[13].position.x = 800.0 / 1750.0 * 269;
clavier->touches[13].position.y = 0;
clavier->touches[13].position.w = L_NOIRE;
clavier->touches[13].position.h = H_NOIRE;
//
clavier->touches[14].nom = "Si0";
clavier->touches[14].frequence = 61.735412657015445;
clavier->touches[14].type = DROITE;
clavier->touches[14].position.x = 800.0 / 1750.0 * 278;
clavier->touches[14].position.y = 0;
clavier->touches[14].position.w = L_BLANCHE;
clavier->touches[14].position.h = H_BLANCHE;
//
clavier->touches[15].nom = "Do1";
clavier->touches[15].frequence = 65.40639132514958;
clavier->touches[15].type = GAUCHE;
clavier->touches[15].position.x = 800.0 / 1750.0 * 311;
clavier->touches[15].position.y = 0;
clavier->touches[15].position.w = L_BLANCHE;
clavier->touches[15].position.h = H_BLANCHE;
//
clavier->touches[16].nom = "Do1dièse";
clavier->touches[16].frequence = 69.29565774421795;
clavier->touches[16].type = NOIRE;
clavier->touches[16].position.x = 800.0 / 1750.0 * 335;
clavier->touches[16].position.y = 0;
clavier->touches[16].position.w = L_NOIRE;
clavier->touches[16].position.h = H_NOIRE;
//
```

```
clavier->touches[17].nom = "Ré1";
clavier->touches[17].frequence = 73.41619197935181;
clavier->touches[17].type = CENTRE;
clavier->touches[17].position.x = 800.0 / 1750.0 * 345;
clavier->touches[17].position.y = 0;
clavier->touches[17].position.w = L_BLANCHE;
clavier->touches[17].position.h = H_BLANCHE;
//
clavier->touches[18].nom = "Ré1dièse";
clavier->touches[18].frequence = 77.78174593052015;
clavier->touches[18].type = NOIRE;
clavier->touches[18].position.x = 800.0 / 1750.0 * 368;
clavier->touches[18].position.y = 0;
clavier->touches[18].position.w = L_NOIRE;
clavier->touches[18].position.h = H_NOIRE;
//
clavier->touches[19].nom = "Mi1";
clavier->touches[19].frequence = 82.4068892282174;
clavier->touches[19].type = DROITE;
clavier->touches[19].position.x = 800.0 / 1750.0 * 377;
clavier->touches[19].position.y = 0;
clavier->touches[19].position.w = L_BLANCHE;
clavier->touches[19].position.h = H_BLANCHE;
//
clavier->touches[20].nom = "Fa1";
clavier->touches[20].frequence = 87.30705785825089;
clavier->touches[20].type = GAUCHE;
clavier->touches[20].position.x = 800.0 / 1750.0 * 411;
clavier->touches[20].position.y = 0;
clavier->touches[20].position.w = L_BLANCHE;
clavier->touches[20].position.h = H_BLANCHE;
//
clavier->touches[21].nom = "Fa1dièse";
clavier->touches[21].frequence = 92.49860567790851;
clavier->touches[21].type = NOIRE;
clavier->touches[21].position.x = 800.0 / 1750.0 * 434;
clavier->touches[21].position.y = 0;
clavier->touches[21].position.w = L_NOIRE;
clavier->touches[21].position.h = H_NOIRE;
//
clavier->touches[22].nom = "Sol1";
clavier->touches[22].frequence = 97.99885899543723;
clavier->touches[22].type = CENTRE;
clavier->touches[22].position.x = 800.0 / 1750.0 * 443;
clavier->touches[22].position.y = 0;
clavier->touches[22].position.w = L_BLANCHE;
```

```
clavier->touches[22].position.h = H_BLANCHE;
//
clavier->touches[23].nom = "Sol1dièse";
clavier->touches[23].frequence = 103.8261743949862;
clavier->touches[23].type = NOIRE;
clavier->touches[23].position.x = 800.0 / 1750.0 * 467;
clavier->touches[23].position.y = 0;
clavier->touches[23].position.w = L_NOIRE;
clavier->touches[23].position.h = H_NOIRE;
//
clavier->touches[24].nom = "La1";
clavier->touches[24].frequence = 110.0;
clavier->touches[24].type = CENTRE;
clavier->touches[24].position.x = 800.0 / 1750.0 * 478;
clavier->touches[24].position.y = 0;
clavier->touches[24].position.w = L_BLANCHE;
clavier->touches[24].position.h = H_BLANCHE;
//
clavier->touches[25].nom = "La1dièse";
clavier->touches[25].frequence = 116.5409403795224;
clavier->touches[25].type = NOIRE;
clavier->touches[25].position.x = 800.0 / 1750.0 * 501;
clavier->touches[25].position.y = 0;
clavier->touches[25].position.w = L_NOIRE;
clavier->touches[25].position.h = H_NOIRE;
//
clavier->touches[26].nom = "Si1";
clavier->touches[26].frequence = 123.47082531403095;
clavier->touches[26].type = DROITE;
clavier->touches[26].position.x = 800.0 / 1750.0 * 510;
clavier->touches[26].position.y = 0;
clavier->touches[26].position.w = L_BLANCHE;
clavier->touches[26].position.h = H_BLANCHE;
//
clavier->touches[27].nom = "Do2";
clavier->touches[27].frequence = 130.81278265029923;
clavier->touches[27].type = GAUCHE;
clavier->touches[27].position.x = 800.0 / 1750.0 * 543;
clavier->touches[27].position.y = 0;
clavier->touches[27].position.w = L_BLANCHE;
clavier->touches[27].position.h = H_BLANCHE;
//
clavier->touches[28].nom = "Do2dièse";
clavier->touches[28].frequence = 138.59131548843595;
clavier->touches[28].type = NOIRE;
clavier->touches[28].position.x = 800.0 / 1750.0 * 567;
```

```
clavier->touches[28].position.y = 0;
clavier->touches[28].position.w = L_NOIRE;
clavier->touches[28].position.h = H_NOIRE;
//
clavier->touches[29].nom = "Ré2";
clavier->touches[29].frequence = 146.83238395870367;
clavier->touches[29].type = CENTRE;
clavier->touches[29].position.x = 800.0 / 1750.0 * 577;
clavier->touches[29].position.y = 0;
clavier->touches[29].position.w = L_BLANCHE;
clavier->touches[29].position.h = H_BLANCHE;
//
clavier->touches[30].nom = "Ré2dièse";
clavier->touches[30].frequence = 155.56349186104035;
clavier->touches[30].type = NOIRE;
clavier->touches[30].position.x = 800.0 / 1750.0 * 600;
clavier->touches[30].position.y = 0;
clavier->touches[30].position.w = L_NOIRE;
clavier->touches[30].position.h = H_NOIRE;
//
clavier->touches[31].nom = "Mi2";
clavier->touches[31].frequence = 164.81377845643485;
clavier->touches[31].type = DROITE;
clavier->touches[31].position.x = 800.0 / 1750.0 * 609;
clavier->touches[31].position.y = 0;
clavier->touches[31].position.w = L_BLANCHE;
clavier->touches[31].position.h = H_BLANCHE;
//
clavier->touches[32].nom = "Fa2";
clavier->touches[32].frequence = 174.61411571650183;
clavier->touches[32].type = GAUCHE;
clavier->touches[32].position.x = 800.0 / 1750.0 * 643;
clavier->touches[32].position.y = 0;
clavier->touches[32].position.w = L_BLANCHE;
clavier->touches[32].position.h = H_BLANCHE;
//
clavier->touches[33].nom = "Fa2dièse";
clavier->touches[33].frequence = 184.9972113558171;
clavier->touches[33].type = NOIRE;
clavier->touches[33].position.x = 800.0 / 1750.0 * 666;
clavier->touches[33].position.y = 0;
clavier->touches[33].position.w = L_NOIRE;
clavier->touches[33].position.h = H_NOIRE;
//
clavier->touches[34].nom = "Sol2";
clavier->touches[34].frequence = 195.99771799087455;
```

```
clavier->touches[34].type = CENTRE;
clavier->touches[34].position.x = 800.0 / 1750.0 * 675;
clavier->touches[34].position.y = 0;
clavier->touches[34].position.w = L_BLANCHE;
clavier->touches[34].position.h = H_BLANCHE;
//
clavier->touches[35].nom = "Sol2dièse";
clavier->touches[35].frequence = 207.65234878997248;
clavier->touches[35].type = NOIRE;
clavier->touches[35].position.x = 800.0 / 1750.0 * 699;
clavier->touches[35].position.y = 0;
clavier->touches[35].position.w = L_NOIRE;
clavier->touches[35].position.h = H_NOIRE;
//
clavier->touches[36].nom = "La2";
clavier->touches[36].frequence = 220.0;
clavier->touches[36].type = CENTRE;
clavier->touches[36].position.x = 800.0 / 1750.0 * 710;
clavier->touches[36].position.y = 0;
clavier->touches[36].position.w = L_BLANCHE;
clavier->touches[36].position.h = H_BLANCHE;
//
clavier->touches[37].nom = "La2dièse";
clavier->touches[37].frequence = 233.08188075904488;
clavier->touches[37].type = NOIRE;
clavier->touches[37].position.x = 800.0 / 1750.0 * 733;
clavier->touches[37].position.y = 0;
clavier->touches[37].position.w = L_NOIRE;
clavier->touches[37].position.h = H_NOIRE;
//
clavier->touches[38].nom = "Si2";
clavier->touches[38].frequence = 246.94165062806198;
clavier->touches[38].type = DROITE;
clavier->touches[38].position.x = 800.0 / 1750.0 * 742;
clavier->touches[38].position.y = 0;
clavier->touches[38].position.w = L_BLANCHE;
clavier->touches[38].position.h = H_BLANCHE;
//
clavier->touches[39].nom = "Do3";
clavier->touches[39].frequence = 261.62556530059857;
clavier->touches[39].type = GAUCHE;
clavier->touches[39].position.x = 800.0 / 1750.0 * 775;
clavier->touches[39].position.y = 0;
clavier->touches[39].position.w = L_BLANCHE;
clavier->touches[39].position.h = H_BLANCHE;
//
```

```
clavier->touches[40].nom = "Do3dièse";
clavier->touches[40].frequence = 277.182630976872;
clavier->touches[40].type = NOIRE;
clavier->touches[40].position.x = 800.0 / 1750.0 * 799;
clavier->touches[40].position.y = 0;
clavier->touches[40].position.w = L_NOIRE;
clavier->touches[40].position.h = H_NOIRE;
//
clavier->touches[41].nom = "Ré3";
clavier->touches[41].frequence = 293.6647679174075;
clavier->touches[41].type = CENTRE;
clavier->touches[41].position.x = 800.0 / 1750.0 * 809;
clavier->touches[41].position.y = 0;
clavier->touches[41].position.w = L_BLANCHE;
clavier->touches[41].position.h = H_BLANCHE;
//
clavier->touches[42].nom = "Ré3dièse";
clavier->touches[42].frequence = 311.12698372208087;
clavier->touches[42].type = NOIRE;
clavier->touches[42].position.x = 800.0 / 1750.0 * 832;
clavier->touches[42].position.y = 0;
clavier->touches[42].position.w = L_NOIRE;
clavier->touches[42].position.h = H_NOIRE;
//
clavier->touches[43].nom = "Mi3";
clavier->touches[43].frequence = 329.6275569128699;
clavier->touches[43].type = DROITE;
clavier->touches[43].position.x = 800.0 / 1750.0 * 841;
clavier->touches[43].position.y = 0;
clavier->touches[43].position.w = L_BLANCHE;
clavier->touches[43].position.h = H_BLANCHE;
//
clavier->touches[44].nom = "Fa3";
clavier->touches[44].frequence = 349.2282314330039;
clavier->touches[44].type = GAUCHE;
clavier->touches[44].position.x = 800.0 / 1750.0 * 875;
clavier->touches[44].position.y = 0;
clavier->touches[44].position.w = L_BLANCHE;
clavier->touches[44].position.h = H_BLANCHE;
//
clavier->touches[45].nom = "Fa3dièse";
clavier->touches[45].frequence = 369.9944227116344;
clavier->touches[45].type = NOIRE;
clavier->touches[45].position.x = 800.0 / 1750.0 * 898;
clavier->touches[45].position.y = 0;
clavier->touches[45].position.w = L_NOIRE;
```

```
clavier->touches[45].position.h = H_NOIRE;
//
clavier->touches[46].nom = "Sol3";
clavier->touches[46].frequence = 391.9954359817493;
clavier->touches[46].type = CENTRE;
clavier->touches[46].position.x = 800.0 / 1750.0 * 907;
clavier->touches[46].position.y = 0;
clavier->touches[46].position.w = L_BLANCHE;
clavier->touches[46].position.h = H_BLANCHE;
//
clavier->touches[47].nom = "Sol3dièse";
clavier->touches[47].frequence = 415.3046975799452;
clavier->touches[47].type = NOIRE;
clavier->touches[47].position.x = 800.0 / 1750.0 * 931;
clavier->touches[47].position.y = 0;
clavier->touches[47].position.w = L_NOIRE;
clavier->touches[47].position.h = H_NOIRE;
//
clavier->touches[48].nom = "La3";
clavier->touches[48].frequence = 440.0;
clavier->touches[48].type = CENTRE;
clavier->touches[48].position.x = 800.0 / 1750.0 * 942;
clavier->touches[48].position.y = 0;
clavier->touches[48].position.w = L_BLANCHE;
clavier->touches[48].position.h = H_BLANCHE;
//
clavier->touches[49].nom = "La3dièse";
clavier->touches[49].frequence = 466.16376151809;
clavier->touches[49].type = NOIRE;
clavier->touches[49].position.x = 800.0 / 1750.0 * 965;
clavier->touches[49].position.y = 0;
clavier->touches[49].position.w = L_NOIRE;
clavier->touches[49].position.h = H_NOIRE;
//
clavier->touches[50].nom = "Si3";
clavier->touches[50].frequence = 493.8833012561242;
clavier->touches[50].type = DROITE;
clavier->touches[50].position.x = 800.0 / 1750.0 * 974;
clavier->touches[50].position.y = 0;
clavier->touches[50].position.w = L_BLANCHE;
clavier->touches[50].position.h = H_BLANCHE;
//
clavier->touches[51].nom = "Do4";
clavier->touches[51].frequence = 523.2511306011974;
clavier->touches[51].type = GAUCHE;
clavier->touches[51].position.x = 800.0 / 1750.0 * 1007;
```

```
clavier->touches[51].position.y = 0;
clavier->touches[51].position.w = L_BLANCHE;
clavier->touches[51].position.h = H_BLANCHE;
//
clavier->touches[52].nom = "Do4dièse";
clavier->touches[52].frequence = 554.3652619537443;
clavier->touches[52].type = NOIRE;
clavier->touches[52].position.x = 800.0 / 1750.0 * 1031;
clavier->touches[52].position.y = 0;
clavier->touches[52].position.w = L_NOIRE;
clavier->touches[52].position.h = H_NOIRE;
//
clavier->touches[53].nom = "Ré4";
clavier->touches[53].frequence = 587.3295358348153;
clavier->touches[53].type = CENTRE;
clavier->touches[53].position.x = 800.0 / 1750.0 * 1041;
clavier->touches[53].position.y = 0;
clavier->touches[53].position.w = L_BLANCHE;
clavier->touches[53].position.h = H_BLANCHE;
//
clavier->touches[54].nom = "Ré4dièse";
clavier->touches[54].frequence = 622.253967444162;
clavier->touches[54].type = NOIRE;
clavier->touches[54].position.x = 800.0 / 1750.0 * 1064;
clavier->touches[54].position.y = 0;
clavier->touches[54].position.w = L_NOIRE;
clavier->touches[54].position.h = H_NOIRE;
//
clavier->touches[55].nom = "Mi4";
clavier->touches[55].frequence = 659.2551138257401;
clavier->touches[55].type = DROITE;
clavier->touches[55].position.x = 800.0 / 1750.0 * 1073;
clavier->touches[55].position.y = 0;
clavier->touches[55].position.w = L_BLANCHE;
clavier->touches[55].position.h = H_BLANCHE;
//
clavier->touches[56].nom = "Fa4";
clavier->touches[56].frequence = 698.456462866008;
clavier->touches[56].type = GAUCHE;
clavier->touches[56].position.x = 800.0 / 1750.0 * 1107;
clavier->touches[56].position.y = 0;
clavier->touches[56].position.w = L_BLANCHE;
clavier->touches[56].position.h = H_BLANCHE;
//
clavier->touches[57].nom = "Fa4dièse";
clavier->touches[57].frequence = 739.988845423269;
```

```
clavier->touches[57].type = NOIRE;
clavier->touches[57].position.x = 800.0 / 1750.0 * 1130;
clavier->touches[57].position.y = 0;
clavier->touches[57].position.w = L_NOIRE;
clavier->touches[57].position.h = H_NOIRE;
//
clavier->touches[58].nom = "Sol4";
clavier->touches[58].frequence = 783.9908719634989;
clavier->touches[58].type = CENTRE;
clavier->touches[58].position.x = 800.0 / 1750.0 * 1139;
clavier->touches[58].position.y = 0;
clavier->touches[58].position.w = L_BLANCHE;
clavier->touches[58].position.h = H_BLANCHE;
//
clavier->touches[59].nom = "Sol4dièse";
clavier->touches[59].frequence = 830.6093951598906;
clavier->touches[59].type = NOIRE;
clavier->touches[59].position.x = 800.0 / 1750.0 * 1163;
clavier->touches[59].position.y = 0;
clavier->touches[59].position.w = L_NOIRE;
clavier->touches[59].position.h = H_NOIRE;
//
clavier->touches[60].nom = "La4";
clavier->touches[60].frequence = 880.0;
clavier->touches[60].type = CENTRE;
clavier->touches[60].position.x = 800.0 / 1750.0 * 1174;
clavier->touches[60].position.y = 0;
clavier->touches[60].position.w = L_BLANCHE;
clavier->touches[60].position.h = H_BLANCHE;
//
clavier->touches[61].nom = "La4dièse";
clavier->touches[61].frequence = 932.3275230361802;
clavier->touches[61].type = NOIRE;
clavier->touches[61].position.x = 800.0 / 1750.0 * 1197;
clavier->touches[61].position.y = 0;
clavier->touches[61].position.w = L_NOIRE;
clavier->touches[61].position.h = H_NOIRE;
//
clavier->touches[62].nom = "Si4";
clavier->touches[62].frequence = 987.7666025122486;
clavier->touches[62].type = DROITE;
clavier->touches[62].position.x = 800.0 / 1750.0 * 1206;
clavier->touches[62].position.y = 0;
clavier->touches[62].position.w = L_BLANCHE;
clavier->touches[62].position.h = H_BLANCHE;
//
```

```
clavier->touches[63].nom = "Do5";
clavier->touches[63].frequence = 1046.502261202395;
clavier->touches[63].type = GAUCHE;
clavier->touches[63].position.x = 800.0 / 1750.0 * 1239;
clavier->touches[63].position.y = 0;
clavier->touches[63].position.w = L_BLANCHE;
clavier->touches[63].position.h = H_BLANCHE;
//
clavier->touches[64].nom = "Do5dièse";
clavier->touches[64].frequence = 1108.7305239074888;
clavier->touches[64].type = NOIRE;
clavier->touches[64].position.x = 800.0 / 1750.0 * 1263;
clavier->touches[64].position.y = 0;
clavier->touches[64].position.w = L_NOIRE;
clavier->touches[64].position.h = H_NOIRE;
//
clavier->touches[65].nom = "Ré5";
clavier->touches[65].frequence = 1174.6590716696307;
clavier->touches[65].type = CENTRE;
clavier->touches[65].position.x = 800.0 / 1750.0 * 1273;
clavier->touches[65].position.y = 0;
clavier->touches[65].position.w = L_BLANCHE;
clavier->touches[65].position.h = H_BLANCHE;
//
clavier->touches[66].nom = "Ré5dièse";
clavier->touches[66].frequence = 1244.5079348883241;
clavier->touches[66].type = NOIRE;
clavier->touches[66].position.x = 800.0 / 1750.0 * 1296;
clavier->touches[66].position.y = 0;
clavier->touches[66].position.w = L_NOIRE;
clavier->touches[66].position.h = H_NOIRE;
//
clavier->touches[67].nom = "Mi5";
clavier->touches[67].frequence = 1318.5102276514804;
clavier->touches[67].type = DROITE;
clavier->touches[67].position.x = 800.0 / 1750.0 * 1305;
clavier->touches[67].position.y = 0;
clavier->touches[67].position.w = L_BLANCHE;
clavier->touches[67].position.h = H_BLANCHE;
//
clavier->touches[68].nom = "Fa5";
clavier->touches[68].frequence = 1396.9129257320162;
clavier->touches[68].type = GAUCHE;
clavier->touches[68].position.x = 800.0 / 1750.0 * 1339;
clavier->touches[68].position.y = 0;
clavier->touches[68].position.w = L_BLANCHE;
```

```
clavier->touches[68].position.h = H_BLANCHE;
//
clavier->touches[69].nom = "Fa5dièse";
clavier->touches[69].frequence = 1479.9776908465383;
clavier->touches[69].type = NOIRE;
clavier->touches[69].position.x = 800.0 / 1750.0 * 1362;
clavier->touches[69].position.y = 0;
clavier->touches[69].position.w = L_NOIRE;
clavier->touches[69].position.h = H_NOIRE;
//
clavier->touches[70].nom = "Sol5";
clavier->touches[70].frequence = 1567.981743926998;
clavier->touches[70].type = CENTRE;
clavier->touches[70].position.x = 800.0 / 1750.0 * 1371;
clavier->touches[70].position.y = 0;
clavier->touches[70].position.w = L_BLANCHE;
clavier->touches[70].position.h = H_BLANCHE;
//
clavier->touches[71].nom = "Sol5dièse";
clavier->touches[71].frequence = 1661.2187903197814;
clavier->touches[71].type = NOIRE;
clavier->touches[71].position.x = 800.0 / 1750.0 * 1395;
clavier->touches[71].position.y = 0;
clavier->touches[71].position.w = L_NOIRE;
clavier->touches[71].position.h = H_NOIRE;
//
clavier->touches[72].nom = "La5";
clavier->touches[72].frequence = 1760.0;
clavier->touches[72].type = CENTRE;
clavier->touches[72].position.x = 800.0 / 1750.0 * 1406;
clavier->touches[72].position.y = 0;
clavier->touches[72].position.w = L_BLANCHE;
clavier->touches[72].position.h = H_BLANCHE;
//
clavier->touches[73].nom = "La5dièse";
clavier->touches[73].frequence = 1864.6550460723606;
clavier->touches[73].type = NOIRE;
clavier->touches[73].position.x = 800.0 / 1750.0 * 1429;
clavier->touches[73].position.y = 0;
clavier->touches[73].position.w = L_NOIRE;
clavier->touches[73].position.h = H_NOIRE;
//
clavier->touches[74].nom = "Si5";
clavier->touches[74].frequence = 1975.5332050244976;
clavier->touches[74].type = DROITE;
clavier->touches[74].position.x = 800.0 / 1750.0 * 1438;
```

```
clavier->touches[74].position.y = 0;
clavier->touches[74].position.w = L_BLANCHE;
clavier->touches[74].position.h = H_BLANCHE;
//
clavier->touches[75].nom = "Do6";
clavier->touches[75].frequence = 2093.0045224047904;
clavier->touches[75].type = GAUCHE;
clavier->touches[75].position.x = 800.0 / 1750.0 * 1471;
clavier->touches[75].position.y = 0;
clavier->touches[75].position.w = L_BLANCHE;
clavier->touches[75].position.h = H_BLANCHE;
//
clavier->touches[76].nom = "Do6dièse";
clavier->touches[76].frequence = 2217.4610478149784;
clavier->touches[76].type = NOIRE;
clavier->touches[76].position.x = 800.0 / 1750.0 * 1495;
clavier->touches[76].position.y = 0;
clavier->touches[76].position.w = L_NOIRE;
clavier->touches[76].position.h = H_NOIRE;
//
clavier->touches[77].nom = "Ré6";
clavier->touches[77].frequence = 2349.3181433392624;
clavier->touches[77].type = CENTRE;
clavier->touches[77].position.x = 800.0 / 1750.0 * 1505;
clavier->touches[77].position.y = 0;
clavier->touches[77].position.w = L_BLANCHE;
clavier->touches[77].position.h = H_BLANCHE;
//
clavier->touches[78].nom = "Ré6dièse";
clavier->touches[78].frequence = 2489.0158697766497;
clavier->touches[78].type = NOIRE;
clavier->touches[78].position.x = 800.0 / 1750.0 * 1528;
clavier->touches[78].position.y = 0;
clavier->touches[78].position.w = L_NOIRE;
clavier->touches[78].position.h = H_NOIRE;
//
clavier->touches[79].nom = "Mi6";
clavier->touches[79].frequence = 2637.020455302962;
clavier->touches[79].type = DROITE;
clavier->touches[79].position.x = 800.0 / 1750.0 * 1537;
clavier->touches[79].position.y = 0;
clavier->touches[79].position.w = L_BLANCHE;
clavier->touches[79].position.h = H_BLANCHE;
//
clavier->touches[80].nom = "Fa6";
clavier->touches[80].frequence = 2793.825851464034;
```

```
clavier->touches[80].type = GAUCHE;
clavier->touches[80].position.x = 800.0 / 1750.0 * 1571;
clavier->touches[80].position.y = 0;
clavier->touches[80].position.w = L_BLANCHE;
clavier->touches[80].position.h = H_BLANCHE;
//
clavier->touches[81].nom = "Fa6dièse";
clavier->touches[81].frequence = 2959.9553816930784;
clavier->touches[81].type = NOIRE;
clavier->touches[81].position.x = 800.0 / 1750.0 * 1594;
clavier->touches[81].position.y = 0;
clavier->touches[81].position.w = L_NOIRE;
clavier->touches[81].position.h = H_NOIRE;
//
clavier->touches[82].nom = "Sol6";
clavier->touches[82].frequence = 3135.963487853998;
clavier->touches[82].type = CENTRE;
clavier->touches[82].position.x = 800.0 / 1750.0 * 1603;
clavier->touches[82].position.y = 0;
clavier->touches[82].position.w = L_BLANCHE;
clavier->touches[82].position.h = H_BLANCHE;
//
clavier->touches[83].nom = "Sol6dièse";
clavier->touches[83].frequence = 3322.4375806395647;
clavier->touches[83].type = NOIRE;
clavier->touches[83].position.x = 800.0 / 1750.0 * 1627;
clavier->touches[83].position.y = 0;
clavier->touches[83].position.w = L_NOIRE;
clavier->touches[83].position.h = H_NOIRE;
//
clavier->touches[84].nom = "La6";
clavier->touches[84].frequence = 3520.0;
clavier->touches[84].type = CENTRE;
clavier->touches[84].position.x = 800.0 / 1750.0 * 1638;
clavier->touches[84].position.y = 0;
clavier->touches[84].position.w = L_BLANCHE;
clavier->touches[84].position.h = H_BLANCHE;
//
clavier->touches[85].nom = "La6dièse";
clavier->touches[85].frequence = 3729.310092144724;
clavier->touches[85].type = NOIRE;
clavier->touches[85].position.x = 800.0 / 1750.0 * 1661;
clavier->touches[85].position.y = 0;
clavier->touches[85].position.w = L_NOIRE;
clavier->touches[85].position.h = H_NOIRE;
//
```

```
  clavier->touches[86].nom = "Si6";
  clavier->touches[86].frequence = 3951.066410048998;
  clavier->touches[86].type = DROITE;
  clavier->touches[86].position.x = 800.0 / 1750.0 * 1670;
  clavier->touches[86].position.y = 0;
  clavier->touches[86].position.w = L_BLANCHE;
  clavier->touches[86].position.h = H_BLANCHE;
  //
  clavier->touches[87].nom = "Do7";
  clavier->touches[87].frequence = 4186.009044809583;
  clavier->touches[87].type = PLEINE;
  clavier->touches[87].position.x = 800.0 / 1750.0 * 1703;
  clavier->touches[87].position.y = 0;
  clavier->touches[87].position.w = L_BLANCHE;
  clavier->touches[87].position.h = H_BLANCHE;
  return 0;
}
```

Module *transcription.c*

```
#include <string.h>
#include <math.h>
#include <fftw3.h>
#include <stdbool.h>

#include "transcription.h"

#define HARMONIQUES 2
#define EPS_AMPLITUDE 0.01
#define LARGEUR_SOUS_BLOC 2000
#define NOMBRE_SOUS_BLOCS 9
#define LARGEUR_BLOC (LARGEUR_SOUS_BLOC * (NOMBRE_SOUS_BLOCS + 1) / 2)

//#define SAUVE_DANS_FICHIER

int calculer_spectrogramme(double * donnees_son, double * instant_son, int debut, double ** spectrogramme, do
  double * signal_bloc = malloc(LARGEUR_SOUS_BLOC * sizeof(double)); // On alloue la mémoire pour stocker le
  if (signal_bloc == NULL) {
    printf("Erreur: impossible d'allouer le bloc de signal.");
    return 1;
  }
  double * filtre = malloc(LARGEUR_SOUS_BLOC * sizeof(double)); // On alloue la mémoire pour stocker les coe
  if (signal_bloc == NULL) {
    printf("Erreur: impossible d'allouer le bloc de signal.");
    free(signal_bloc);
    return 1;
  }
```

```
fftw_complex * fft = malloc(LARGEUR_SOUS_BLOC * sizeof(fftw_complex)); //On alloue la mémoire pour la trans
if(fft == NULL) {
  printf("Erreur: impossible d'allouer le bloc de signal.");
  free(signal_bloc);
  free(filtre);
  return 1;
}

//On construit le tableau des fréquences.
double tMax = instant_son[LARGEUR_SOUS_BLOC - 1]; // tMax est la durée du segment de signal considéré: la d
//les instants sont répartis sur une grille régulière, donc on a translaté fin - début à bloc-0 pour déterr
double df = 1.0 / tMax; //pas fréquentiel

//Comme le signal est réel, sa fft sera symétrique (des infos utiles et leur conjugué). On peut donc ne ga
int demiBloc = LARGEUR_SOUS_BLOC / 2;

for(int i = 0; i < demiBloc; ++i) {
  (*frequences)[i] = (i+0.5) * df; //demiBloc car la transformée de Fourier d'un signal réel est paire donc
} // for i

//Initialisation du spectrogramme
for (int i = 0; i < demiBloc; ++i) {
  (*spectrogramme)[i] = 0.0;
} // for i

//Initialisation du filtre
for (int i = 0; i < LARGEUR_SOUS_BLOC; ++i) {
  filtre[i] = 25.0 / 46.0 - 21.0 / 46.0 * cos(2 * M_PI * i / (LARGEUR_SOUS_BLOC - 1.0));
} // for i

//Boucle sur les sous-blocs
for (int n = 0; n < NOMBRE_SOUS_BLOCS; ++n) {
  //Application de la fenêtre de Hamming
  for(int i = 0; i < LARGEUR_SOUS_BLOC; ++i) {
    signal_bloc[i] = donnees_son[debut + n * demiBloc + i] * filtre[i]; // fenêtre de Hamming
  } // for i -> fenêtrage


  //On réalise la transformée de Fourier du signal.
  fftw_plan plan;
  //Le plan peut être réutilisé tant que signal_bloc et fft sont aux mêmes adresses mémoire->gain en perfo
  if (n == 0) plan = fftw_plan_dft_r2c_1d(LARGEUR_SOUS_BLOC, signal_bloc, fft, FFTW_ESTIMATE); //bloc = ta
  fftw_execute(plan);//Calcul effectif de la FFT

  for(int i = 0; i < demiBloc; ++i) {//on part de 0 jusu'à demiBloc-1, on a donc bien demiBloc indices à b
    (*spectrogramme)[i] += fft[i][0]*fft[i][0] + fft[i][1]*fft[i][1]; //partie réelle ^2 + partie im. ^2
```

```
    } // for i -> spectrogramme
  } // for n -> moyenne sur les blocs
  return 0;
}

//Produit spectral = log(produit de i = 1 à H de mod(X(if))^2
// On ne prend que les harmoniques accessibles donc on calcule une
// moyenne pour équilibrer
double calculer_produit_spectral(int indice_f, double * spectrogramme, double * frequences, int taille) { //
  double produit_spectral = 0.0;
  int num = 0;
  for (int i = 1; i <= HARMONIQUES; ++i) {
    int index = i * indice_f;
    if (index < taille) {
      produit_spectral += log(spectrogramme[index]);
      ++num;
    }
  }
  return produit_spectral / num;
}

//On maximise le produit spectral pour trouver la fréquence fondamentale. On renvoie l'indice correspondant
int maximiser_produit_spectral(double * spectrogramme, double* frequences, int taille, struct clavier_t* cla
  int indice = 1;
  double produit_max = calculer_produit_spectral(indice, spectrogramme, frequences, taille);
  for (int i = 2; i < taille; ++i) {
    double produit = calculer_produit_spectral(i, spectrogramme, frequences, taille);
    if (produit > produit_max) {
      produit_max = produit;
      indice = i;
    }
  }//for
  // La fréquence qui maximise le produit
  double f = frequences[indice];
  // Recherche de la note de piano correspondante
  indice = 0;
  double ecart_min = fabs(f - clavier->touches[indice].frequence);
  for (int i = 1; i < NOMBRE_TOUCHES; ++i) {
    double ecart = fabs(f - clavier->touches[i].frequence);
    if (ecart < ecart_min) {
      ecart_min = ecart;
      indice = i;
    }
  }
  printf("[maximiser_produit_spectral] Fréquence=%gHz touche=%s (%d) fréquence touche=%gHz\n", f, clavier->t
  return indice;
```

```
}

// Convertit un signal audio en une séquence de notes. La stratégie consiste à
// partitionner le signal en blocs de largeur LARGEUR_BLOC. Si le signal contient assez
// d'énergie on identifie la fréquence dominante dans le bloc. Si elle correspond
// à la fréquence dominante du bloc précédent on considère qu'il s'agit de la même
// note qui se prolonge
struct liste_note_t* transcrire(double * donnees_son, double * instant_son, int taille, struct clavier_t* cla
  //
  // Initialisation des structures pour le calcul de note
  //
  // On ajuste la taille pour qu'elle soit un multiple de LARGEUR_BLOC
  // La division est entière (pas de partie fractionnaire)
  int nombre_blocs = taille / LARGEUR_BLOC;
  taille = nombre_blocs * LARGEUR_BLOC;
  double * carre_signal = malloc(taille * sizeof(double));
  if (carre_signal == NULL) {
    printf("Erreur: impossible d'allouer le signal au carré.\n");
    return NULL;
  }
  double *spectrogramme = malloc((LARGEUR_SOUS_BLOC / 2) * sizeof(double));
  if (spectrogramme == NULL) {
    printf("Erreur: impossible d'allouer le spectrogramme.\n");
    free(carre_signal);
    return NULL;
  }
  double *frequences = malloc((LARGEUR_SOUS_BLOC / 2) * sizeof(double));
  if (frequences == NULL) {
    printf("Erreur: impossible d'allouer les fréquences.\n");
    free(carre_signal);
    free(spectrogramme);
    return NULL;
  }
  //
  // Calcul d'énergie du signal
  //
  double energie = 0.0;
  for(int i = 0; i < taille ; ++i) {
    double carre = donnees_son[i] * donnees_son[i];
    carre_signal[i] = carre;
    energie += carre;
  }//for i
  //
  // Détection des notes
  //
  // On va parcourir le signal par blocs de DUREE_BLOC pour chercher une note dans chaque bloc
```

```
  int debut = 0;
  struct liste_note_t *liste = NULL;
  // Ces variables permettent de savoir si la note détectée sur le bloc courant prolonge la note précédente
  int indice_note_en_cours = -1;
  double t_debut_en_cours = 0.0;
  double t_fin_en_cours = 0.0;
  for (int index_bloc = 0; index_bloc < nombre_blocs; ++index_bloc) {
    int fin = debut + LARGEUR_BLOC;
    double energie_moyenne_bloc = energie * LARGEUR_BLOC / taille;
    // On vérifie que le bloc contient assez d'énergie pour qu'on y cherche une note
    double energie_bloc = 0.0;
    for (int i = debut; i < fin; ++i) {
      energie_bloc += carre_signal[i];
    }
    // S'il n'y a pas assez d'énergie pour chercher une note dans le bloc c'est un silence
    if (energie_bloc < EPS_AMPLITUDE * energie_moyenne_bloc) {
      // S'il y a une note en attente on l'ajoute à la liste
      if (indice_note_en_cours != -1) {
        struct touche_t *touche_freq_max = &clavier->touches[indice_note_en_cours];
        liste = ajouter_note(liste, (Uint32)(1000 * t_debut_en_cours), (Uint32)(1000 * t_fin_en_cours), touch
      }
      // On réinitialise la note en cours
      t_debut_en_cours = 0.0;
      t_fin_en_cours = 0.0;
      indice_note_en_cours = -1;
    } // Pas assez d'énergie dans le bloc
    else { // Assez d'énergie dans le bloc
      if (calculer_spectrogramme(donnees_son, instant_son, debut, &spectrogramme, &frequences) == 0) { // sp
#ifdef SAUVE_DANS_FICHIER
        char filename[256] = "spectrogramme";
        sprintf(&filename[13], "%d", index_bloc);
        strcat(filename, ".csv");
        FILE * file = fopen(filename, "w");
        for (int i = 0; i < LARGEUR_SOUS_BLOC / 2; ++i) {
          fprintf(file, "%.20e;%.20e\n", frequences[i], spectrogramme[i]);
        }
        fclose(file);
        strcpy(filename, "signal");
        sprintf(&filename[6], "%d", index_bloc);
        strcat(filename, ".csv");
        file = fopen(filename, "w");
        for (int i = 0; i < LARGEUR_BLOC; ++i) {
          fprintf(file, "%.20e;%.20e\n", instant_son[debut + i], donnees_son[debut + i]);
        }
        fclose(file);
#endif
```

```
        int indice_note_bloc = maximiser_produit_spectral(spectrogramme, frequences, LARGEUR_SOUS_BLOC / 2, c
        // Vérifier si la note trouvée complète la note courante
        if (indice_note_bloc == indice_note_en_cours) {
          t_fin_en_cours = instant_son[fin];
        } // On continue la note en cours
        // Sinon on stocke la note en cours et la note en cours passe à la note courante
        else {
          if (indice_note_en_cours != -1) {
            struct touche_t * touche_freq_max = &clavier->touches[indice_note_en_cours];
            printf("[transcrire] On stocke la note en cours, touche=%s\n", touche_freq_max->nom);
            liste = ajouter_note(liste, (Uint32)(1000 * t_debut_en_cours), (Uint32)(1000 * t_fin_en_cours), t
          }
          t_debut_en_cours = instant_son[debut];
          t_fin_en_cours = instant_son[fin];
          indice_note_en_cours = indice_note_bloc;
        } // On passe à une nouvelle note en cours
      } // Si le calcul du spectrogramme s'est bien passé
    } // Assez d'énergie dans le bloc
    debut = fin;
  } // while
  // S'il reste une note en cours on l'ajoute à la liste
  if (indice_note_en_cours != -1) {
    struct touche_t *touche_freq_max = &clavier->touches[indice_note_en_cours];
    printf("[transcrire] On stocke la note en cours, touche=%s\n", touche_freq_max->nom);
    liste = ajouter_note(liste, (Uint32)(1000 * t_debut_en_cours), (Uint32)(1000 * t_fin_en_cours), touche_fi
  }
  return liste;
}
```

Module *graphique* :

```
#include <SDL.h>
#include <SDL_image.h>

#include "graphique.h"


/* Initialise le moteur graphique SDL2 */
int initialiser_graphique()
{
  if (SDL_Init(SDL_INIT_VIDEO) < 0) {
    SDL_Log("Erreur lors de l'initialisation du graphique: %s\n", SDL_GetError());
    return 1;
  }
  if (IMG_Init(IMG_INIT_PNG) != IMG_INIT_PNG) {
    SDL_Log("Error lors de l'initialisation de la gestion des images.\n");
    return 1;
```

```
  }
  return 0;
}

/* Charge le fichier png dans une texture SDL2 */
int charger_texture(char *nom_image, SDL_Renderer * renderer, SDL_Texture **texture)
{
  SDL_Surface *image = IMG_Load(nom_image);
  if (image == NULL) {
    SDL_Log("Erreur lors du chargement de l'image %s: %s\n", nom_image, IMG_GetError());
    return 1;
  }
  *texture = SDL_CreateTextureFromSurface(renderer, image);
  if (*texture == NULL) {
    SDL_Log("Erreur: impossible de créer la texture associée à la surface.\n");
    SDL_FreeSurface(image);
    return 1;
  }
  SDL_FreeSurface(image);
  if (SDL_SetTextureBlendMode(*texture, SDL_BLENDMODE_BLEND) != 0) {
    SDL_Log("Erreur: impossible de prendre en compte le cannal alpha de la texture: %s\n", SDL_GetError());
    liberer_texture(*texture);
    return 1;
  }
  return 0;
}

/* Copie une texture à une position donnée dans le moteur de rendu */
int copier_texture(SDL_Texture *texture, SDL_Rect *position, SDL_Renderer *renderer) {
  if (SDL_RenderCopy(renderer, texture, NULL, position) != 0) {
    SDL_Log("Erreur: impossible de copier la texture: %s\n", SDL_GetError());
    return 1;
  }
  return 0;
}

/* Libère une texture SDL2 */
void liberer_texture(SDL_Texture *texture) {
  SDL_DestroyTexture(texture);
}

/* Libère le moteur audio SDL2 */
void fermer_graphique() {
  IMG_Quit();
}
```

Module *interface*

```
#include <stdbool.h>
#include <SDL.h>
#include "son.h"
#include "graphique.h"
#include "interface.h"

// Crée l'interface graphique en retournant une structure interface
int creer_interface(struct interface_t *interface) {
  // Initialisation du moteur graphique
  if (initialiser_graphique() != 0) {
    return 1;
  }
  // Initialisation de la gestion des événements
  if (SDL_Init(SDL_INIT_EVENTS) != 0) {
    printf("Erreur: impossible d'initialiser la gestion des événements : %s\n", SDL_GetError());
    return 1;
  }
  // Création de la fenêtre de l'application
  SDL_DisplayMode DM;
  SDL_GetCurrentDisplayMode(0, &DM);
  Uint32 hauteur_ecran = DM.h-120;
  interface->fenetre = SDL_CreateWindow("Transcription piano", SDL_WINDOWPOS_CENTERED, 0, LARGEUR_INTERFACE,
  if (interface->fenetre == NULL) {
    SDL_Log("Erreur: impossible de créer la fenêtre.\n");
    return 1;
  }
  // Création du moteur de rendu
  interface->renderer = SDL_CreateRenderer(interface->fenetre, -1, SDL_RENDERER_ACCELERATED);
  if (interface->renderer == NULL) {
    SDL_Log("Erreur: impossible de créer le moteur de rendu.\n");
    SDL_DestroyWindow(interface->fenetre);
    return 1;
  }
  // La surface associée au clavier n'est plus utile
  if (charger_texture("Images/clavier.png", interface->renderer, &interface->texture_clavier) != 0) {
    SDL_Log("Erreur: impossible de charger la texture du clavier.\n");
    SDL_DestroyWindow(interface->fenetre);
  }
  interface->position_clavier.x = 0;
  interface->position_clavier.y = hauteur_ecran - 100;
  interface->position_clavier.w = 800;
  interface->position_clavier.h = 88;
  // On lit les textures associées aux différents types de touches
  // Pour chacune des textures de touche active
  // Charge l'image correspondante
  char *noms_textures_touches[5] = {"Images/touche_centre.png",
```

```
    "Images/touche_droite.png",
    "Images/touche_gauche.png",
    "Images/touche_noire.png",
    "Images/touche_pleine.png"};
  for (int i = 0; i < 5; ++i) {
    if (charger_texture(noms_textures_touches[i], interface->renderer, &interface->textures_touches[i]) != 0)
      for (int j = 0; j < i; ++j)
        liberer_texture(interface->textures_touches[j]);
      SDL_Log("Erreur: impossible de charger la texture des touches.\n");
      liberer_texture(interface->texture_clavier);
      SDL_DestroyWindow(interface->fenetre);
      return 1;
    } // charger_texture
  } // for i
  return 0;
}

// Anime l'interface à l'aide de la séquence de notes et du fichier son
void animer_interface(struct interface_t *interface, struct liste_note_t *liste, Uint32 duree, SDL_AudioSpec
  Uint32 delai_interface = (1000 * interface->position_clavier.y) / VITESSE_NOTE;
  Uint32 t_demarrage = SDL_GetTicks();
  bool son_demarre = false;
  SDL_AudioDeviceID deviceId = SDL_OpenAudioDevice(NULL, 0, wav_spec, NULL, 0);
  Uint32 t_actuel = 0;
  bool animer_interface = true;
  while (animer_interface) {
    printf("                                                \r");
    printf("t=%dms", t_actuel);
    fflush(stdout);
    SDL_Event event;
    if (SDL_PollEvent(&event)) {
      if (event.type == SDL_QUIT) {
        animer_interface = true;
        break;
      }
    }
    struct liste_note_t * tete = liste;
    t_actuel = SDL_GetTicks() - t_demarrage;
    if (!son_demarre && t_actuel >= delai_interface) {
      if (jouer_son(wav_spec, wav_buffer, wav_length, deviceId) != 0) {
        animer_interface = true;
        break;
      }
      son_demarre = true;
    }
    Uint32 t_horizon = t_actuel + delai_interface;
```

```
  SDL_SetRenderDrawColor(interface->renderer, 0, 0, 0, 255);
  SDL_RenderClear(interface->renderer);
  SDL_RenderCopy(interface->renderer, interface->texture_clavier, NULL, &interface->position_clavier);
  // On arrête le parcours de la liste à la fin de la liste
  // ou dès qu'on rencontre une note trop dans le futur
  // car les notes sont triées par date de début croissantes
  bool trop_tot = false;
  bool trop_tard = false;
  bool dessiner_notes = true;
  while (dessiner_notes) {
    t_actuel = SDL_GetTicks() - t_demarrage;
    t_horizon = t_actuel + delai_interface;
    struct note_t * note = tete->note;
    Uint32 t_debut = note->t_debut + delai_interface;
    Uint32 t_fin   = note->t_fin + delai_interface;
    trop_tot  = t_debut > t_actuel + delai_interface;
    trop_tard = t_fin <= t_actuel;
    // Quelque chose à afficher
    if (!trop_tot && !trop_tard) {
      // La touche est active
      if (t_actuel >= t_debut) {
SDL_Rect position = note->touche->position;
position.y = interface->position_clavier.y + 6;
        copier_texture(interface->textures_touches[note->touche->type], &position, interface->renderer);
        printf(" note=%s", note->touche->nom);
      }
      SDL_Rect rect;
      // X rectangle
      rect.x = note->touche->position.x + (note->touche->position.w - LARGEUR_NOTE) / 2;
      // w rectangle
      rect.w = LARGEUR_NOTE;
      // y et h rectangle
      Uint32 t_min = (t_actuel  > t_debut ?  t_actuel : t_debut);
      Uint32 t_max = (t_horizon < t_fin   ? t_horizon :   t_fin);
      rect.h = ((t_max - t_min) * VITESSE_NOTE) / 1000;
      if (rect.h > 0) {
        rect.y = ((t_horizon - t_max) * VITESSE_NOTE) / 1000;
        if (rect.y + rect.h >= interface->position_clavier.y) {
          rect.h = interface->position_clavier.y - rect.y;
        }
        if (note->touche->type == NOIRE) {
          SDL_SetRenderDrawColor(interface->renderer, 170, 255, 136, 255);
        }
        else {
          SDL_SetRenderDrawColor(interface->renderer, 255, 238, 51, 255);
        }
```

```
          SDL_RenderFillRect(interface->renderer, &rect);
        } // rect.h > 0
      }
      // On supprime la note de la liste si sa date de fin est antérieure à l'instant présent
      if (trop_tard) {
        liste = supprimer_note(liste, note);
      }
      tete = tete->suivant;
      dessiner_notes = (liste != NULL) && (tete != liste) && !trop_tot;
    } // while tete != NULL
    SDL_RenderPresent(interface->renderer);
    printf("\r");
    fflush(stdout);
    Uint32 t_fin_anim = SDL_GetTicks() - t_demarrage;
    if (t_fin_anim < t_actuel + 33) {
      SDL_Delay(t_actuel + 33 - t_fin_anim);
    }
    animer_interface = liste != NULL;
  } // while (!fini)
  if (t_actuel > duree + delai_interface) {
    SDL_Delay(t_actuel - (duree + delai_interface));
  }
  fermer_son(deviceId);
}

// Détruit l'interface et les éléments qui lui sont associés
void liberer_interface(struct interface_t * interface) {
  SDL_DestroyRenderer(interface->renderer);
  SDL_DestroyWindow(interface->fenetre);
}
```