Projet Long : Création d'un logiciel d'autotune

PENELOUX Nicolas

SCHEINER Ymri

A - Introduction

- Qui sommes-nous ? Quel est l'objectif du projet ?

- → Etudiant de Master Informatique parcours DATA.

- → Création d'un logiciel d'autotune accessible à tous.

- Dans quel scénarios pouvons-nous l'utiliser ?

- → Faire vos propres chansons.

- → Ajuster votre voix déraillante sur des fichiers audios.

B – Architecture, conception et gestion du projet

- Décomposition du projet en sous problèmes

- → Analyse du projet : ce qu'il nous faut, ce que nous avons besoin de faire. Dans l'ordre : lire un fichier, l'analyser et le comprendre, et le corriger.

- Compétences techniques nécessaires

- → Nouveau langage de programmation

- → Savoir manipuler des fichiers audios et comprendre le fonctionnement des signaux, en adaptant des algorithmes pour les utiliser.

- Répartition et durée du développement
- → Par sous problème / partie, d'une durée équivalente (~ 1 mois) entre les parties.

- Modules principale et relations entre eux
- → Lecteur de fichier .wav, transformation du .wav en signaux lisibles, puis insertion dans le pitch tracking.

-

- Choix de l'organisation particulière :

- → Progression pas à pas : il faut les résultats précédents pour pouvoir tester et vérifier, pour ensuite progresser et passer à la phase suivante.

- Test du projet :

- → Création d'algorithme de tests.

- → Vérification fait main par nous même.

- Difficulté rencontrées :

- → Nouveau langage de programmation, compréhension des fonctionnements des signaux en informatique ainsi que les algorithmes à utilisé.

C – Programmation

```rust
pub fn new_reader(file_path : &str) -> Result<WavReader, std::io::Error>{
    let file = File::open(file_path)?;
    let mut reader = BufReader::new(file);

    // On lis l'en-tête du fichier WAV
    let mut header = [0u8; 44]; //44 correspond à la taille standard d'un entête Wav
    reader.read_exact(&mut header)?;

    // Récupérer les paramètres du fichier WAV à partir de l'en-tête
    let channels = u16::from_le_bytes([header[22], header[23]]);
    let sample_width = u16::from_le_bytes([header[34], header[35]]);
    let frame_rate = u32::from_le_bytes([
        header[24], header[25], header[26], header[27]
    ]);

    // Lire les données audio brutes
    let mut raw_data = Vec::new();
    reader.read_to_end(&mut raw_data)?;

    // Convertir les données audio brutes en échantillons
    let mut samples = Vec::new();
    let sample_size = sample_width as usize / 8;
    for i in (0..raw_data.len()).step_by(sample_size) {
        let mut sample_bytes = [0u8; 4];
        sample_bytes[..sample_size].copy_from_slice(&raw_data[i..i + sample_size]);
        let sample_value = match sample_width {
            8 => sample_bytes[0] as i16,
            16 => i16::from_le_bytes([sample_bytes[0], sample_bytes[1]]),
            24 => i32::from_le_bytes([0, sample_bytes[0], sample_bytes[1], sample_bytes[2]]) as i16,
            32 => i32::from_le_bytes([sample_bytes[0], sample_bytes[1], sample_bytes[2], sample_bytes[3]]) as i16,
            _ => return Err(std::io::Error::new(std::io::ErrorKind::InvalidData, "Invalid sample width")),
        };
        samples.push(sample_value);
    }

    Ok(WavReader { channels, sample_width, frame_rate, samples })
}
```

D – Conclusion

- Version 2 du projet

- → Un projet complet, en ajoutant des fonctionnalités intéressantes pour les utilisateurs tout comme modifier le son, la tonalité, les variations … faire des « filtres » que les utilisateurs pourraient utilisé à leur guise.

- Quel changement ?

- → Y contribuer d'avantage, un langage de programmation plus simple (même s'il devrait être un peu moins adapté).