Rapport projet CPP :

Yell Ow !

Introduction :

Ce projet nécessite au préalable quelques notions de théorie musicale essentielles à la compréhension de l'intérêt du programme. Nous allons donc essayer de vous présenter, les quelques notions minimum nécessaires à la bonne compréhension des codes.

Dans un premier temps, il est important de comprendre qu'un accord est avant tout un ensemble de notes. C'est l'association de ces notes qui va créer la nature de l'accord. Or, une note se définie par sa hauteur, et l'écart entre deux hauteurs de notes est appelé un interval. C'est donc les intervals qui séparent les différentes notes d'un accord qui permettront de le caractériser, et donc de le nommer.
Pour comprendre la nomenclature d'un accord il faut connaître les rôles des notes qui le composent. Nous distinguerons ici, trois types d'accords : les accords à deux, trois et quatre sons. Chacun de ces accords se compose au moins d'une note de référence appelée fondamentale qui donne son nom à l'accord (un Do Majeur 7 a pour fondamentale le Do). Ensuite dans un accord à deux sons (appelé power-chord) on ajoute une quinte (cinq notes au dessus de la fondamentale : la quinte de Do est le Sol), une quinte et une tierce (trois notes au dessus de la fondamentale : Do - Mi) dans un accord à trois sons, et enfin une quinte, une tierce et une septième dans un accord à quatre sons.

Voyons maintenant comment nommer un accord selon les notes qui le composent. Dans les exemples qui suivent, nous prendrons toujours le Do comme fondamentale de notre accord :

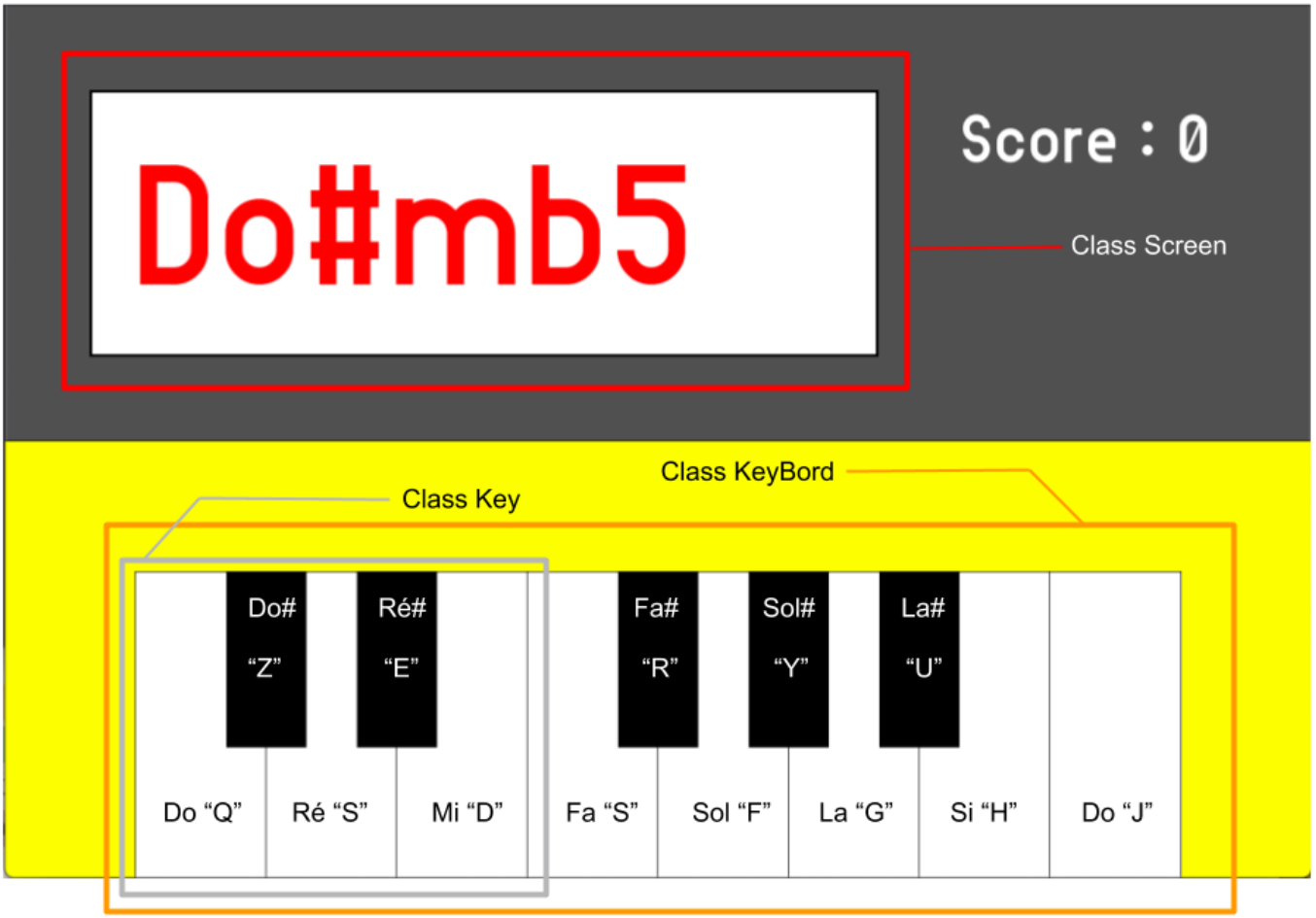
- Si la tierce est mineure on ajoute un "m" minuscule à la suite du nom de la fondamentale (ex : Dom pour Do mineur). Si la tierce est majeure on garde uniquement le nom de la fondamentale (ex : Do pour Do majeur).
- Si la septième est majeure on ajoute "M7" à la fin de l'accord, après les caractérisations de la tierce (ex : DoM7 pour Do majeur avec une septième majeure). Si elle est mineure on ajoute uniquement "7" (ex : Do7 pour Do majeur avec un septième mineure)
- Enfin, la quinte. Celle-ci ne peut être ni majeure, ni mineure. On dit simplement qu'elle est juste ou diminuée. Dans le cas d'un power-chord on ajoute "5" si la quinte est juste à la fin du nom de l'accord, "5dim" si la quinte est diminuée. Dans le cas des autres types d'accord on ajoute "b5" à la fin du nom de l'accord uniquement si la quinte est diminuée.

Voilà, il semble que vous ayez, toutes les cartes en main pour entrevoir le principe de ce programme et en comprendre les enjeux.

Lucas Marillesse
Julien Roze

Présentation du projet :

Ce programme est un logiciel utilitaire dont le but est d'apprendre et de réviser ses accords musicaux. Le programme affiche un clavier de piano dont les touches sont reliés au clavier de l'ordinateur. La touche "q" correspond à la 1ere touche blanche du piano, le "s" la seconde etc. Les touches noires sont les touches de la ligne azerty placé comme sur un piano : intercalés entre les touches blanches (le z est la première touche noire car elle est entre le q et le s). Un nom d'accord aléatoire composé de 2, 3 ou 4 notes apparaît également à l'écran. Le but de ce programme est de recomposer l'accord demandé en appuyant sur les différentes touches du clavier de l'ordinateur correspondant au notes qui composent l'accord proposé. A chaque bonne réponse un compteur de point s'incrémente. Une partie est composée d'une série de 10 accords. une fois la partie terminée, le piano reste affiché pour jouer librement. Cependant si l'on veut recommencer une partie, il faut relancer le programme. l'implémentation d'un bouton rejouer n'a pas encore été mise en place.



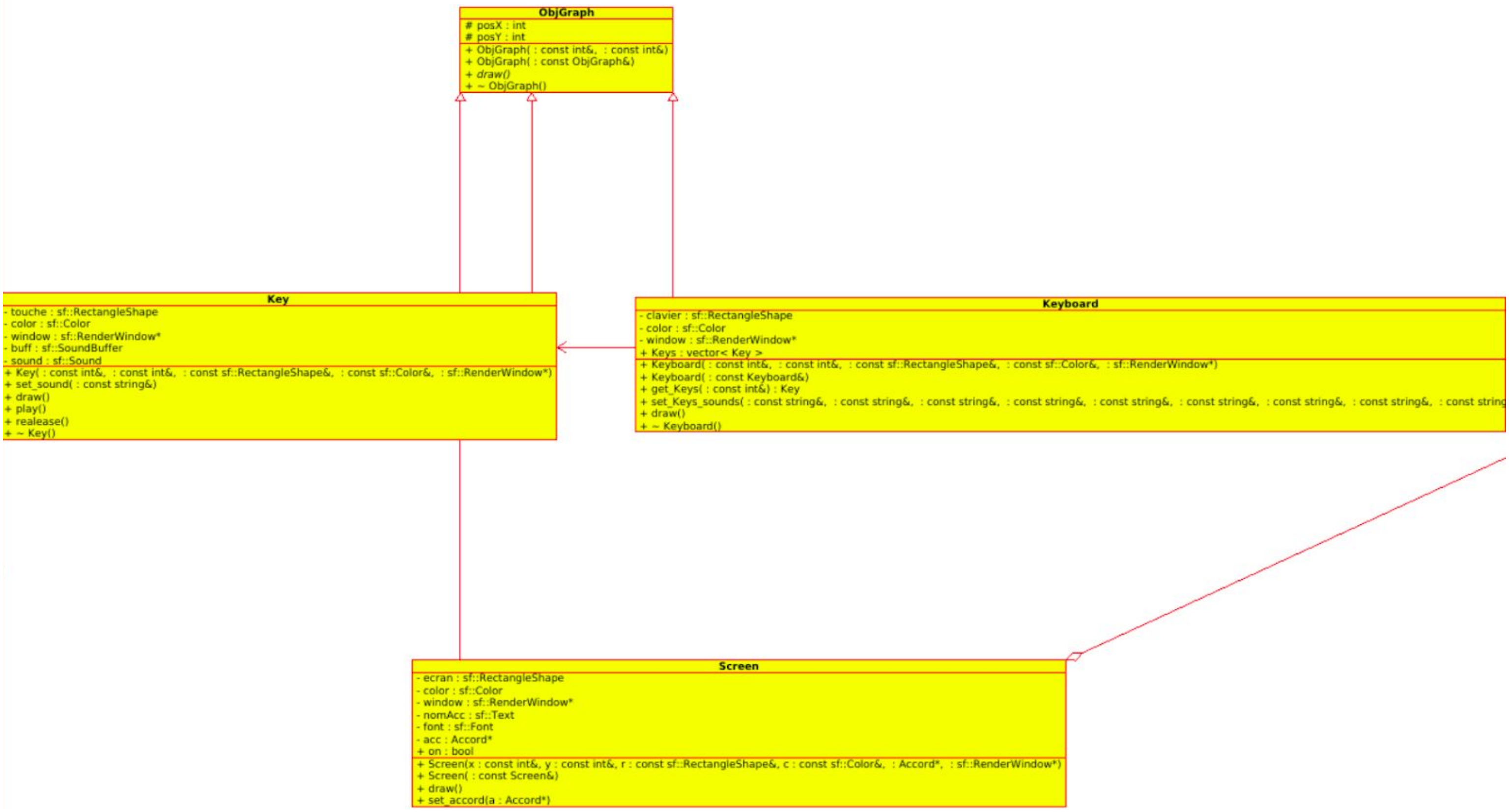
Lucas Marillesse
Julien Roze

Réalisation du programme :

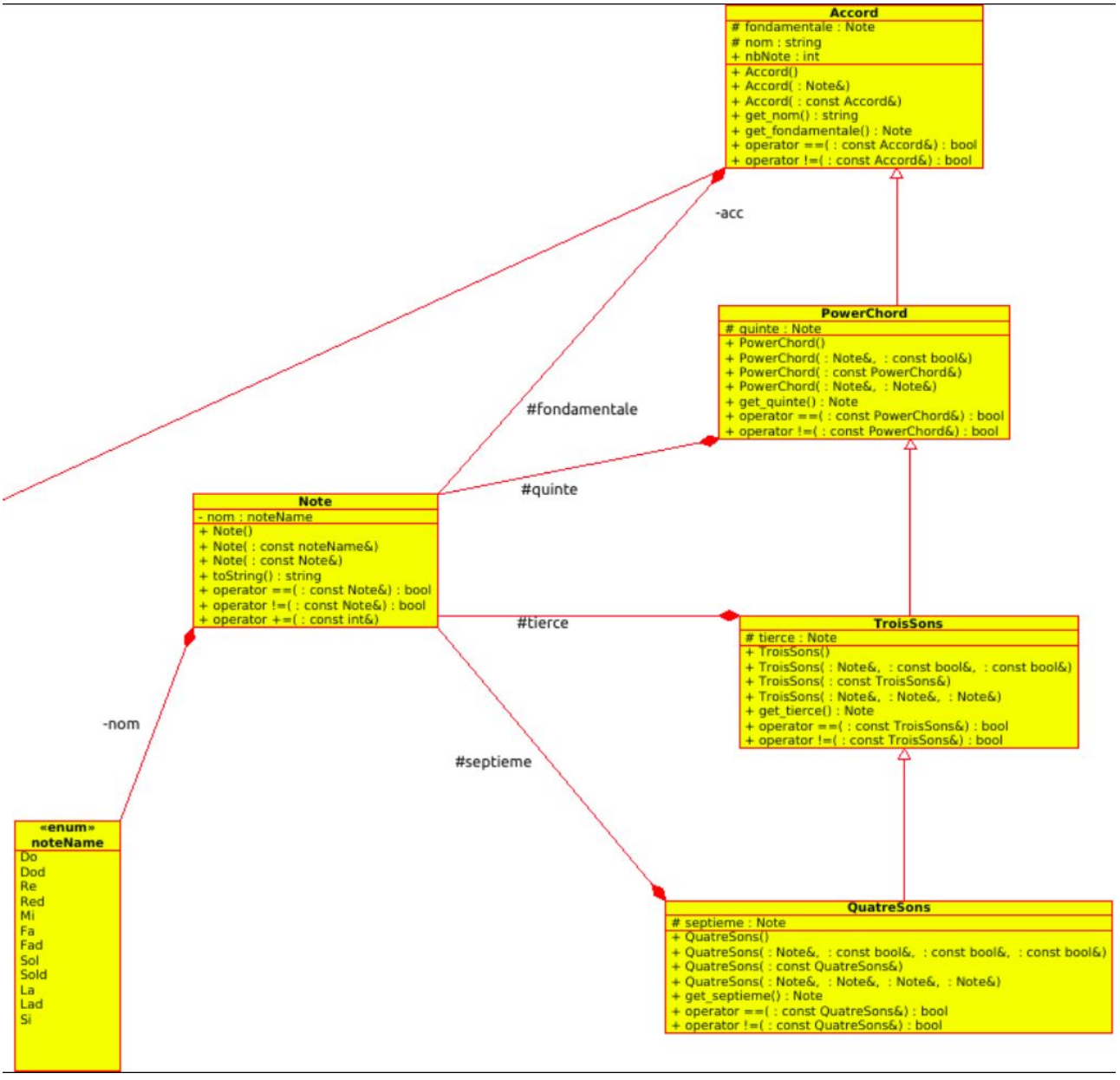
Ce programme est composé de deux parties majeures : Une interface graphique gérant l'affichage du clavier et des éléments de gameplay, et une partie implémentant la hiérarchie des accords et des mécanismes de jeux. vous trouverez en annexe le diagramme de classe complet afin de mieux comprendre les différentes relations qui existent au sein du projet.

La Partie Graphique utilise la bibliothèque SFML et gère

- l'affichage de la fenêtre
- l'affichage du clavier passant par l'affichage des différents rectangles correspondant aux notes.
- l'affichage des accords à jouer.
- la gestion des fichiers sonores.
- la programmation événementielle de la réaction du piano lors de la pression des touches sur le clavier de l'ordinateur avec le changement de couleur et l'émission du sons correspondant à la note.



La partie musicale implémente la hiérarchie suivante mettant en place le programme précédemment expliqué. La classe accord est la classe principale de cette hiérarchie. Elle est composé d'une note fondamentale de type note qui définit l'accord créé (si la fondamentale est un do l'accord joué sera alors un accord de do) Les classes PowerChord, TroisSons, et QuatresSons sont des enfants de la classe Accord et possèdent dans leur attributs une note (quinte tierce et 7eme). le PowerChord sera donc composé de deux notes le TroisSons de trois et le QuatreSons de quatre. Ces notes seront les notes à jouer pour faire sonner l'accord. Chaque classe possédant une note en plus de la précédente hérite de cette dernière et récupère donc ces attributs. Les accords sont donc enrichis d'une note à chaque niveau de hiérarchie.



Cette hiérarchie utilise des surcharges d'opérateurs de comparaison (== et !=) dans la classe accord et ses filles, pour reconnaître si les accords joués sont les bons en comparant un à un les attributs des différentes classes, contenant les notes qui composent l'accord. des surcharges d'opérateurs += pour changer la hauteur d'une note en lui ajoutant n demis tons selon le type d'accord choisis et - pour connaître l'interval présent entre deux notes.

À l'intérieur du main, un tableau de type Accord est utilisé pour stocker les dix accords qui seront utilisés lors d'une partie. Or ces accords ne seront jamais du type accord en tant que tels, nous utilisons le polymorphisme pour y stocker des classes filles de Accord. Pour cette raison, nous devions utiliser des méthodes virtuelles. Ici, c'est donc les surcharges d'opérateurs qui sont rendues virtuelles puisque c'est d'elle que nous nous servons dans le main.

Le programme va donc utiliser dans un main les deux parties précédentes pour faire fonctionner le jeu. il crée un tableau d'accord aléatoire en utilisant les différentes classes de la hiérarchie et les compare aux entrées reçues par les événements claviers gérés par la SFML. Une machine à état gère les différents cas selon leur nombre de note et attends le nombre exacte de note que contient l'accord. il compare ensuite l'accord joué et l'accord attendu grâce aux surcharges d'opérateur et augmente le nombre de point si la solution proposée est juste. Le score est affiché à l'écran grâce à la classe screen.

Procédure d'utilisation :

Ce Logiciel utilise la bibliothèque graphique SFML pour afficher les différents éléments visuels. Il faut donc, pour pouvoir utiliser le programme, installer cette librairie.
Pour installer SFML il suffit de suivre les étapes suivantes :

- ouvrir un terminal
- entrez la commande suivante : " sudo apt-get install libsfml-dev "
- suivez les instructions écrites sur le terminal. (entrez le mot de passe, accepter l'installation …

- pour compiler le programme entrez la ligne de commande suivante "make clean" puis la commande "make"
- lancez l'exécutable avec la commande ./Yellow

lorsque le programme est lancée, la partie commence directement. il vous suffit de jouer selon les explications données précédemment. Une partie se joue en dix accords.

Nos fiertés :

Parmis tout ce que nous avons implémenté dans ce programme, nous sommes assez content du système de classe que nous avons mis en place pour la création du piano. Notre premier essai de piano fonctionnel représentait un main de 260 lignes de code à lui tout seul. Désormais, la création de ce piano se fait en l'appel d'un seul constructeur. Puis en utilisant une seule méthode de l'objet créé, nous avons configuré les sons du piano et nous disposons alors d'un objet fonctionnel.
Ensuite la mise en place de la machine à état qui nous permet de gérer les différentes phases de jeu est une idée ingénieuse pour la gestions des différents cas. Bien que celle-ci soit simple, elle est une solution essentielle au fonctionnement de notre programme.
Pour conclure nous sommes relativement fière de la façon dont nous avons réussi à utiliser le langage objet dans le but de simplifier la programmation à l'intérieur du main. La hiérarchie de classe mise en place nous a permis de rendre le programme plus concis et donc plus claire dans son implémentation.