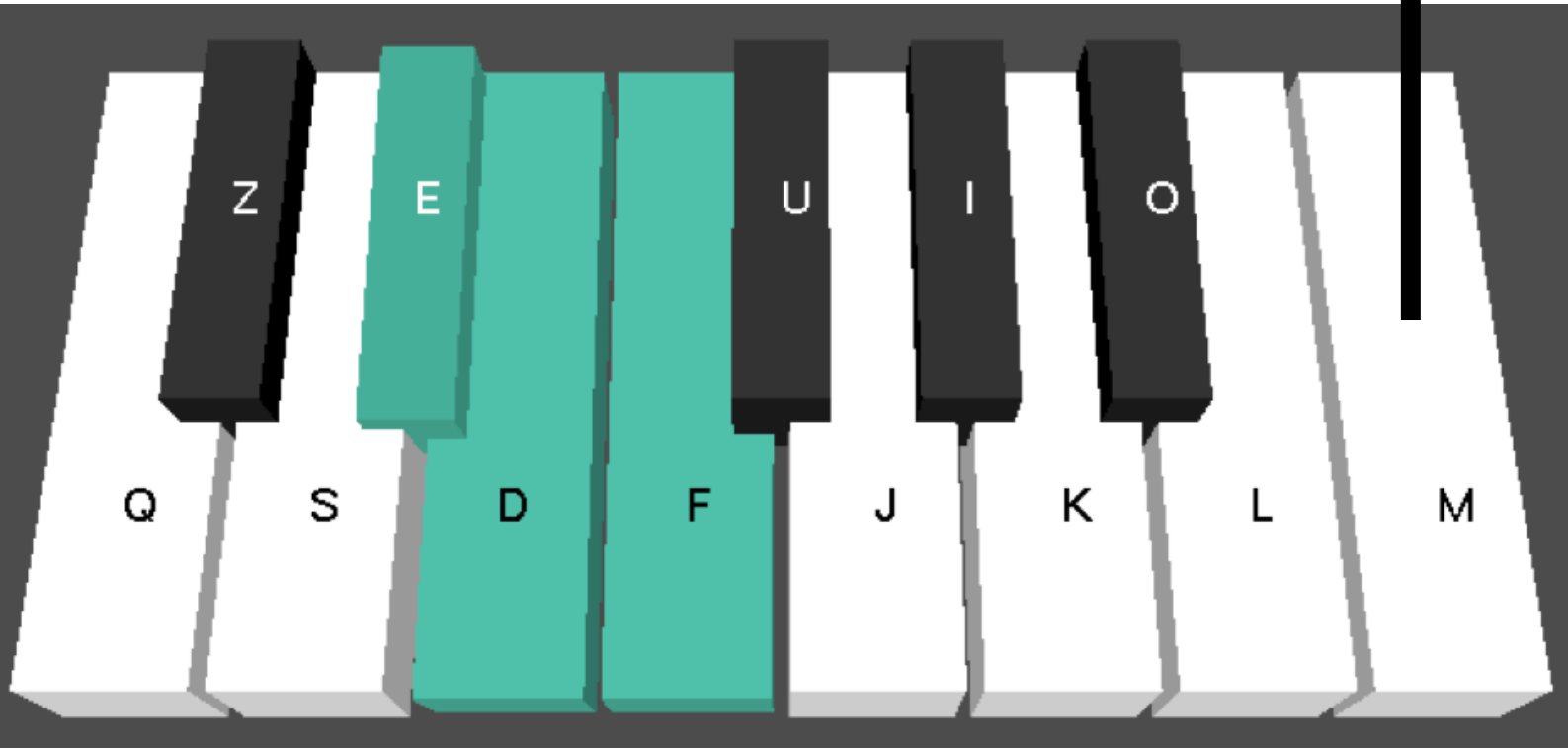
RAPPORT P1RV
PIANO VIRTUEL OPENGL

Antoine DEPERCIN
Maxence LAUVERNIER

2022/2023



Piano Virtuel - OpenGL

SOMMAIRE

ANTOINE DE PERCIN / MAXENCE LAUVERNIER

Piano Virtuel - OpenGL

I. INTRODUCTION

Dans le cadre de notre P1RV nous devions modéliser un « piano virtuel » en OpenGL. Le programme devait afficher les touches du piano en 3D avec lesquelles l'utilisateur.ice pouvait interagir (le choix de l'interaction était libre), et bien entendu générer des sons en fonction de la ou les touches appuyées.

Nous avons donc décidé d'établir la conception du projet lors de la première séance. En définissant en premier lieu les attendus minimums du projet et les fonctionnalités supplémentaires que nous pouvions ajouter si le projet était fini avant la fin du P1RV.

Nous avons ensuite décidé de la manière dont nous allions travailler. Nous avons donc créé un répositoire Git pour le projet et scindé le projet en deux parties, la gestion du son et la gestion de l'affichage et des mouvements. Nous avons donc veillé à travailler sur des parties différentes afin d'éviter les problèmes.

Voici le lien vers le répositoire Git : https://github.com/Antoine-dP/PianoVirtuel

II. AFFICHAGE

a. Choix de la bibliothèque d'affichage

Tout d'abord, nous avons dû choisir la bibliothèque d'affichage 3D pour faire l'interface de notre piano 3D. Comme nous avions déjà travaillé avec OpenGL pour de telles applications, il était logique de choisir cette bibliothèque.

b. Premier affichage de touches

Nous avons ensuite commencé avec un affichage simple des touches avec des rectangles 2D vus du dessus (fig. 1). Nous avons placé 8 touches blanches, soit une octave, puisque cela semblait le plus réaliste pour une utilisation au clavier. Deux octaves auraient rendu cette utilisation plus compliquée à cause du nombre de touches insuffisant sur une rangée du clavier.

Ces placeholders de touches nous ont permis de valider le fonctionnement du système d'affichage de base. Ceci nous a aussi permis de construire le projet autour de cet affichage simplifié, pour pouvoir construire en parallèle la détection de touches et la gestion du son. Cet affichage d'objets 2D a aussi facilité la tâche de trouver des propriétés de la perspective convenables pour la suite.

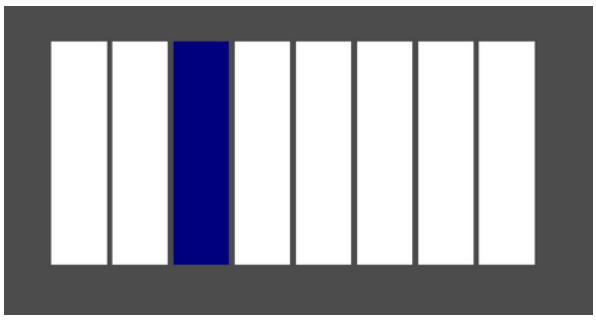


Figure 1, affichage 2D du piano

Une fois l'affichage de base validé, nous avons ajouté les touches noires pour avoir une octave complète (fig. 2).

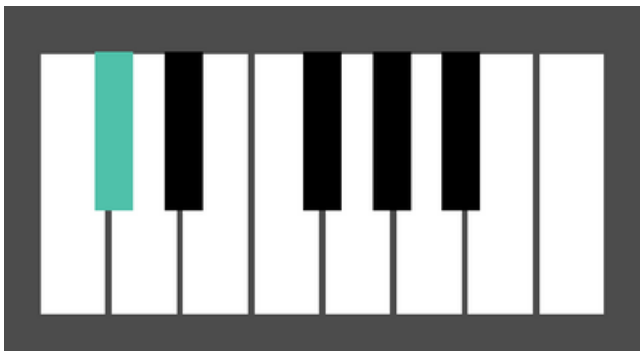


Figure 2, affichage 2D avec touches noires

C'est à ce stade que nous avons cherché à compartimenter le code en différentes classes, et au lieu de centraliser l'affichage de toutes les touches dans une fonction, chaque instance de touche contient sa fonction d'affichage. Nous avons crée deux classes, WhiteKey et BlackKey, qui héritent d'une classe abstraite Key, et on appelle la fonction show() sur chaque touche du piano pour obtenir l'affichage final. Ceci permet aussi la modification des fonctions d'affichage sans modifier le cœur du code.

c. Changement de couleur

A ce stade, la détection de touches au clavier fonctionnait suffisamment pour être utilisée pour être affichée en direct. Nous devions donc montrer à l'écran quelle touche de clavier correspond à quelle touche de piano, puis afficher quand une touche est appuyée.

La première étape d'affichage de lettres du clavier s'est faite avec la fonction glutBitmapCharacter de la bibliothèque GLUT. On inclut l'affichage de la lettre dans la fonction show() de chaque touche, qui affiche la lettre associée à cette touche (l'attribut keyboardLetter de la classe Key).

Lorsque la propriété isPressed d'une touche est vraie (touche appuyée, voir Partie III.), la fonction show() affichera la touche d'une couleur différente. Comme la sélection de couleur en OpenGL est une machine d'état, il fallait simplement override la couleur noire par défaut avec la couleur "appuyée" pour la durée de l'affichage des faces de la touche. On peut donc voir en direct lorsqu'une touche est appuyée (fig. 2).

d. Passage en 3D

Puis, nous sommes passés à un affichage 3D. Un premier essai à été effectué (fig. 3) avec des pavés à 6 faces de couleur identique. Chaque touche est instanciée à une position, son centre, et ajoutée à un vecteur de pointeurs d'objets Key. A chaque rafraichissement de l'image, on parcourt le vecteur de pointeurs pour appeler la fonction show() sur chacun des objets. Ainsi, on a pu ajuster facilement l'apparence des touches et leur position.

Nous avons rapidement compris qu'une couleur par type de touche (blanc / noir) ne suffirait pas pour un affichage clair. Sans nuance les touches se confondent, leurs limites ne sont pas visibles, et tout a l'air très plat.

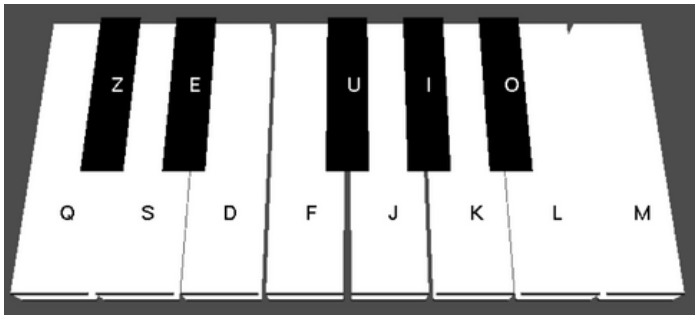


Figure 3, affichage 3D bicolore avec délimitations de bords

Notre première idée de solution était de démarquer les arêtes avec des segments noirs pour mieux voir les coins des touches. Ceci n'a pas fonctionné (fig. 3 ci-dessus) car les lignes n'apparaissaient pas partout. Les lignes verticales et horizontales s'affichaient bien, mais celles en diagonale non. Nous avons pensé que cela pourrait être dû à une épaisseur trop faible de la ligne, mais élargir les arêtes n'a pas fonctionné pour bien les voir.

Une autre idée était d'utiliser des shaders. Ils permetteraient de pouvoir distinguer les touches entre elles et donneraient de la profondeur à l'image. Après plusieurs heures d'essais, nous n'avons pas réussi à faire fonctionner une version plus récente de OpenGL (2.0), à cause de conflits de bibliothèques. Nous avons donc conclu que puisque nous avions uniquement besoin des shaders pour améliorer la qualité esthétique du piano, nous pourrions trouver une autre solution plus rapide pour utiliser le temps restant à implémenter des fonctionnalités plus utiles.

La solution adoptée était simplement d'utiliser trois couleurs différentes, une pour chaque orientation des faces (fig. 4). Ainsi, on distingue les touches entre elles, et les faces entre elles.

Figure 4, affichage 3D avec trois couleurs par touche

III. DETECTION DE TOUCHES

a. Détection d'appui et de release au clavier

Pour la détection de touches, nous avons commencé par un "Input Manager" qui enregistrait la touche appuyée à un instant t. Nous avons donc utilisé le callback glutKeyboardFunc. Quand une touche est appuyée, le callback enregistre le caractère associé à la touche dans un objet "currentPressedKey". Si une touche est déjà appuyée, le callback overwrite le caractère précédent.

Pour le release, nous avons utilisé le callback glutKeyboardUpFunc pour savoir si la touche qui est enregistrée dans le currentPressedKey a été relevée. Ainsi, nous avons vu vérifier le bon fonctionnement des méthodes d'affichage au début du projet (fig. 1, fig. 2).

b. Support multi-touches

Un problème inhérent de ce système est qu'il n'y a pas la possibilité d'appuyer sur plusieurs touches simultanément, un défaut conséquent pour un piano. Il fallait donc modifier le système de détection de touches pour pouvoir gérer des appuis multi-touches.

Nous sommes donc passés d'un système centralisé (le currentPressedKey), à un système décentralisé où chaque touche enregistre son état actuel (bool isPressed). Le callback d'appui appelle alors une méthode pressKey(char, true), avec en char la touche appuyée et le callback de release appelle pressKey(char, false).

La méthode pressKey cherche alors une touche dans le vecteur de touches qui a en attribut le char correspondant. Si une est trouvée, son bool isPressed est modifié à true ou false, en fonction de quel callback l'appelle. Chaque touche peut alors indépendamment gérer son affichage, et modifier sa couleur ou sa position si elle est appuyée.

c. Détection d’objet au clic souris

Une fois que l’utilisation du piano au clavier était achevée, nous avons cherché à faire fonctionner l’appui de touches de piano avec la souris.

Comme le callback de souris renvoie la position (x, y) du clic, la première solution tentée était de calculer quelle touche était sélectionnée en fonction de la position du clic sur l’écran. Ceci semblait fonctionner avec les touches 2D et une perspective verticale comme la figure 2, mais la 3D avec une perspect

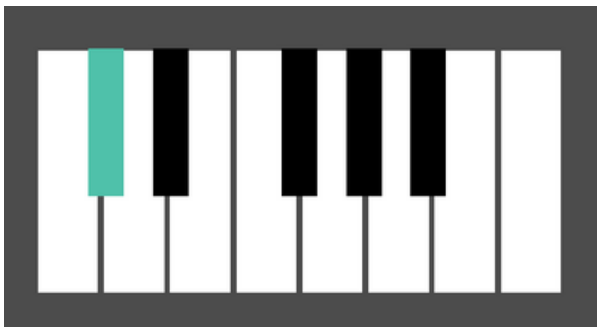


Figure 2

Figure 4

Comme un calcul direct de la position du clic dans l’espace 3D ne semblait pas réalisable, nous avons cherché à détecter quel objet était sélectionné lors du clic. Nous avons cherché avec une bibliothèque, mais les méthodes suggérées n’ont jamais fonctionné comme voulu avec notre code, nous avons donc dû abandonner cette piste.

d. Détection de couleur au clic

Par contre, lors de ces tests, nous avons trouvé la méthode glReadPixels() qui peut renvoyer la couleur du pixel sélectionné. Nous avons alors eu l’idée d’avoir des couleurs différentes pour chaque touche. Ainsi, lorsqu'une touche est sélectionnée, la couleur du pixel du clic nous indiquerait quelle touche a été sélectionnée. La couleur de la touche est alors de la forme (R’, G, B), avec R’ = (R - keyID / 255). On peut alors déduire le keyID, donc quelle touche est appuyée, en fonction de la valeur R’ lue par glReadPixels().

C’est alors cette méthode qui a été utilisée pour pouvoir sélectionner les touches à la souris. Cette méthode suffit dans notre cas car il y a peu de touches, mais elle peut poser problème s'il faut ajouter des touches. Avec 13 touches la couleur est peu modifiée, mais un avec un plus grand nombre de touches on pourrait remarquer la différence de couleur.

IV. MOUVEMENT

Pour le mouvement des touches de piano lors de l'appui d'une touche de clavier, nous avons cherché à effectuer un mouvement le plus réaliste possible. Pour cela, une rotation de 5°, en prenant comme point fixe le haut de la touche, nous a semblé approprié.

Pour effectuer cette rotation d'une seule touche, nous avons utilisé les fonctions glTranslatef() et glRotatef() (fig. 5). Il faut d'abord effectuer une translation pour placer le point de rotation de la touche en (0, 0, 0), puis faire la rotation et effectuer la translation inverse. On peut ensuite placer les GL_QUADS, et enfin faire translation / rotation inverse / translation inverse. Notre touche est alors dans la bonne position avec la rotation voulue.

```
if (isPressed)
{
    for (int i = 0; i < 6; i++)
    {
        // Draw face
        // Rotate key if it's pressed
        glTranslatef(-position.x, -position.y, -position.z - L / 2);
        glRotatef(rotateAngle, 1, 0, 0);
        glTranslatef(position.x, position.y, position.z + L / 2);

        // Draw faces
        glBegin(GL_QUADS);
        // Make blue
        glColor3f(couleurBlanche[i][0] * 79.0f / 255.0, couleurBlanche[i][1] * 194.0f / 255.0, couleurBlanche[i][2] * 170.0f / 255.0);
        for (int j = 0; j < 4; j++)
        {
            glVertex3f(toucheBlanche[face[i][j]][0] * W / 2 + position.x,
                toucheBlanche[face[i][j]][1] * H + position.y,
                toucheBlanche[face[i][j]][2] * L / 2 + position.z);
        }
        glEnd();

        // Undo rotation for other key
        glTranslatef(-position.x, -position.y, -position.z - L / 2);
        glRotatef(-rotateAngle, 1, 0, 0);
        glTranslatef(position.x, position.y, position.z + L / 2);
    }
}
```

Figure 4, extrait de code pour l'affichage d'une touche blanche appuyée

Nous avons réfléchi à ajouter une animation au moment de l'appui d'une touche, mais comme les touches de clavier ont seulement deux états possibles il est impossible de savoir avec quelle force et quelle vitesse la touche a été appuyée. Une animation pourrait être plus perturbante que bénéfique pour l'utilisateur, surtout si elle est trop lente. Comme le mouvement des touches n'était pas visuellement désagréable à l'état actuel, nous avons choisi de conserver le mouvement immédiat

V. GESTION DU SON

a. Trouver la librairie adaptée pour la gestion du son

La première étape à réaliser pour avoir une gestion du son était celle de la détermination de la librairie C++ appropriée. Le but étant que lorsqu'une touche du piano est appuyée, le programme renvoie la note appropriée. Nous avions en plus quelques contraintes concernant la gestion du son :

- Possibilité de jouer plusieurs notes en simultané (comme un piano réel)
- Possibilité d'appuyer plusieurs fois sur la même note : dans cette situation, si une touche est appuyée alors que la note associée est déjà en train d'être jouée, le son en cours est arrêté et c'est le nouveau son qui prend le dessus. Comme lors d'un appui répétitif sur une touche de piano.
- Enfin lors d'un appuie bref de la touche, le son ne doit pas à l'instant du relâchement de la touche.

Pour cela nous avons effectué des recherches pour trouver la librairie adaptée, cela n'a pas été facile, car si il existe beaucoup de librairies son pour C++, leur documentation est souvent très sommaire et il n'y a pas d'indicateurs de difficultés. Nous avons d'abord opté pour la librairie Bass24 puis nous avons décidé de garder la librairie IrrKlang.

- **Bass24**: sa prise en main était très difficile du fait du manque de documentation en ligne, la seule aide que nous avons trouvé était un fichier texte relatant toutes les fonctions sans trop expliquer leur but. De plus Bass24 ne permettait pas de l'ajout d'effet comme le pitching.

- **IrrKlang**: sa prise en main était beaucoup plus facile que Bass24. La documentation était beaucoup plus présente et cette librairie présentait des effets pouvant être ajoutés aux sons.

b. Se familiariser avec la librairie choisie

La seconde étape à été de comprendre comment fonctionnait la librairie. Pour cela nous avons analysé le site de la librairie qui fournissait plusieurs tutoriels pour apprendre à utiliser les fonctionnalités de base de la librairie. Il y avait également de nombreux tutoriels sur YouTube afin d'approfondir les possibilités liées à la librairie.

Nous avons donc pu tester la librairie dans un programme de test, afin de parvenir à générer des sons depuis Visual Studio à partir de fichiers. Nous avons alors rencontré plusieurs problèmes, le premier étant que IrrKlang n'acceptait pas les fichiers mp3 alors que sa documentation indiquait le contraire. Le second en revanche fut un peu plus long à résoudre, la fonction appelée par IrrKlang pour jouer le son ne reconnaissait pas le chemin vers le son. Après des recherches et l'aide de nombreux forums, nous avons réussi à jouer un son. Il a donc fallu ensuite trouver les sons adéquats pour le piano.

c. Trouver les fichiers sons pour les notes

La troisième étape fut celle de trouver les fichiers sons nécessaires pour jouer les notes de notre piano. Cette étape semblait au début triviale, mais s'est avérée au final plus ardue que prévu du fait de la difficulté à trouver une octave complète de notes de piano. En effet trouver les notes individuellement était facile, mais trouver les 12 notes correspondantes à la même octave et provenant du même piano était plus difficile que prévu. Nous avons réussi à les trouver sur le site freesound.org.

d. Première tentative d’algorithme de gestion du son

La première tentative d’algorithme avait pour but de gérer le son d'une seule octave du piano, l’algorithme mis en œuvre reposait sur deux variables globales. La première appelée vb était un vecteur de booléen composé de 13 booléens représentant l’état des sons joués par les 13 touches du piano. Ainsi si le son correspondant à la note d’une touche était en cours d’émission le booléen associé à la note était passé à vrai et à faux lorsque aucun son n’était émis par la touche. Cela permettait d’éviter la superposition de sons lors de l’appuie répétitif d’une même touche. En effet, sans cette condition, si l’utilisateur.ice maintenait l’appuie sur une touche, un son aurait été joué à chaque frame, ce que nous ne voulions pas. La deuxième variable locale était appelée piano, il s’agissait d’un vecteur de string constitué des 13 chemins vers les fichiers .wav correspondant aux 13 notes associées aux touches du piano.

La gestion du son était gérée dans la fonction pressKey, présente dans l’algorithme de gestion de clavier et de souris, cette fonction est appelée lorsqu’une touche est appuyée (par touche du clavier ou appuie souris). Elle était constituée d’une série de switch jouant le son et changeant le signe du booléen associé à la touche.

e. Problème d’appuie simultané

Le premier algorithme fonctionnait relativement bien, mais avait un problème majeur, lors de l’appuie en simultané de plusieurs touches, leurs sons ne se jouaient pas en même temps, il y avait un léger décalage. Ce problème nous posa problème un long moment, car la résolution à travers le code ne semblait pas fonctionner. Cela s’explique du fait que le problème n’avait rien à voir avec le code mais avec les fichiers sons. En effet, les fichiers sons, bien que venant du même enregistrement, n'avait pas le même délai de silence avant de lancer la note. Nous avons donc changé les fichiers sons en en prenant des nouveaux. Le problème fut alors réglé.

f. Création d’octave

Une fois la gestion du son fonctionnelle, on pouvait désormais s’attaquer aux fonctionnalités supplémentaires désirées. La première étant l’ajout d’octave. Il était impossible de trouver plusieurs octaves du même piano enregistré, nous avons donc décidé d’utiliser dans un premier temps les effets fournis par la librairie IrrKlang, cette idée a vite été abandonnée du fait du manque d’effet dans la librairie. La librairie ne fournissait pas une fonction de pitching, mais seulement une fonction d’accélération du temps et de modification de la hauteur.

Ainsi pour faire passer une de nos notes d'une octave à la suivante, nous devions modifier la hauteur(modifiant de fait la vitesse), puis la vitesse (modifiant de fait la hauteur), le compromis a trouvé était très dur, et n'ayant pas l'oreille musicale, cela était très dur.

Nous avons donc utilisé une méthode plus lourde mais fonctionnelle en utilisant Audacity. Audacity permet de modifier les notes pour leur faire gagner ou perdre une ou deux octaves. Cela a permis d'avoir des fichiers sons correspondants à une octave au-dessus et une octave en dessous. Nous n'avons pas fait plus d'octaves car la méthode utilisée par Audacity avait tendance à détériorer le son à chaque application de cette méthode, le son aurait été trop mauvais si nous avions construit plus d'octaves.

g. Changement d'octave

Afin de changer d'octave nous avons dû mettre en place un algorithme permettant le switch d'octaves à l'appuie des touches « 0 », « 1 » et « 2 », afin de passer à la première, deuxième et troisième octave. Cet algorithme était plutôt simple, nous avons crée des vecteurs de vecteurs d'octaves pour chaque instrument : ainsi un instrument est représenté par un vecteur comprenant 3 vecteurs de string (chemin vers les notes) associés aux octaves. Nous avions ensuite un vecteur global comprenant les instruments. À l'aide de variables globales, nous pouvions alors passer d'instrument à instrument et d'octaves à octaves. Cet algorithme fonctionne parfaitement.

h. Changement d'instrument

Comme indiqué dans le paragraphe ci-dessus, nous avons aussi rajouté un instrument (comme il est possible de le faire sur certains synthétiseurs). Ici il s'agit d'une guitare, dont on peut jouer les notes au piano. Nous avons donc rajouté tout un vecteur de guitare au vecteur global des instruments.

VI. CONCLUSION

Ce projet nous a permis, à travers la création d'un piano virtuel, de nous familiariser avec de nombreuses nouvelles notions. Nous avons pu appliquer nos connaissances d'OpenGL vues en cours, mais aussi expérimenter avec des librairies nouvelles, et même anticiper le cours de MEDEV et commencer à utiliser l'outil Git. Ce projet a également été l'occasion de réaliser un premier projet "long" en informatique, qui nous a incité à apprendre à gérer notre temps.

Finalement, nous espérons que vous prendrez autant de plaisir à jouer à notre piano virtuel que nous en avons eu à le concevoir.