Rapport Projet C++
Concert aux Jeux Olympiques

Paul Bouchard
Chaoui Aziz Youssef

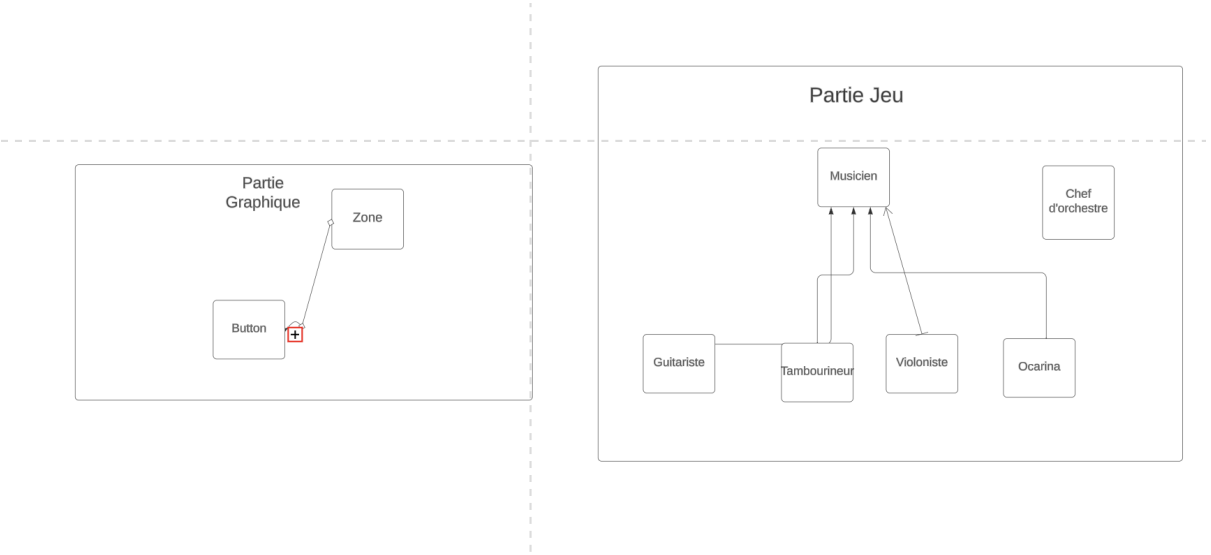
EISE-4

Sommaire

1 - Introduction :

L'objectif de ce projet était de programmer une application liée aux jeux olympiques en C++. Nous avons donc décidé de réaliser la cérémonie d'ouverture des jeux olympiques représentée par une simulation d'un concert dirigé par un chef d'orchestre. Nous avons choisi pour la réalisation de cette application la bibliothèque graphique sfml. Nous nous sommes tournés vers cette dernière car on trouve aisément de nombreux tutos pour s'initier et s'informer sur les différentes fonctionnalités implémentables via sfml.

2 - Diagramme UML :

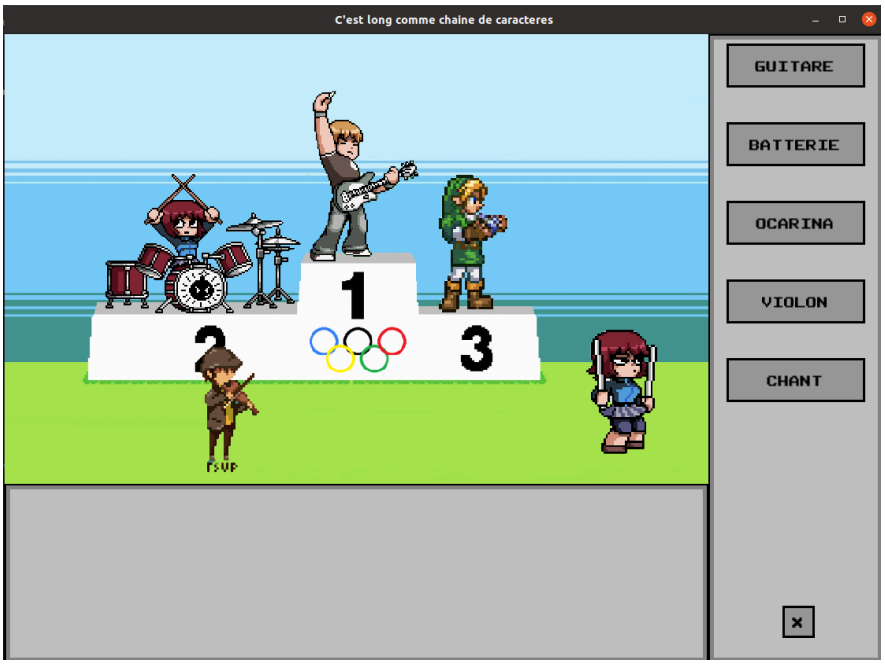


3 - Procédure d'installation :

Pour utiliser notre application il faut d'abord télécharger la librairie sfml via le terminal et la commande : **sudo apt-get install libsfml-dev**
Ensuite, faites un git clone du répertoire du projet dans le répertoire personnel de votre choix. Une fois le clonage réalisé, veuillez extraire le zip "sprites_sons" dans le répertoire courant et sortir les dossiers tambour guitare ocarina violon chef et batterie de sorte à ce qu'ils soient sur le même répertoire que le fichier code. Nous souhaitons avoir l'arborescence suivante:

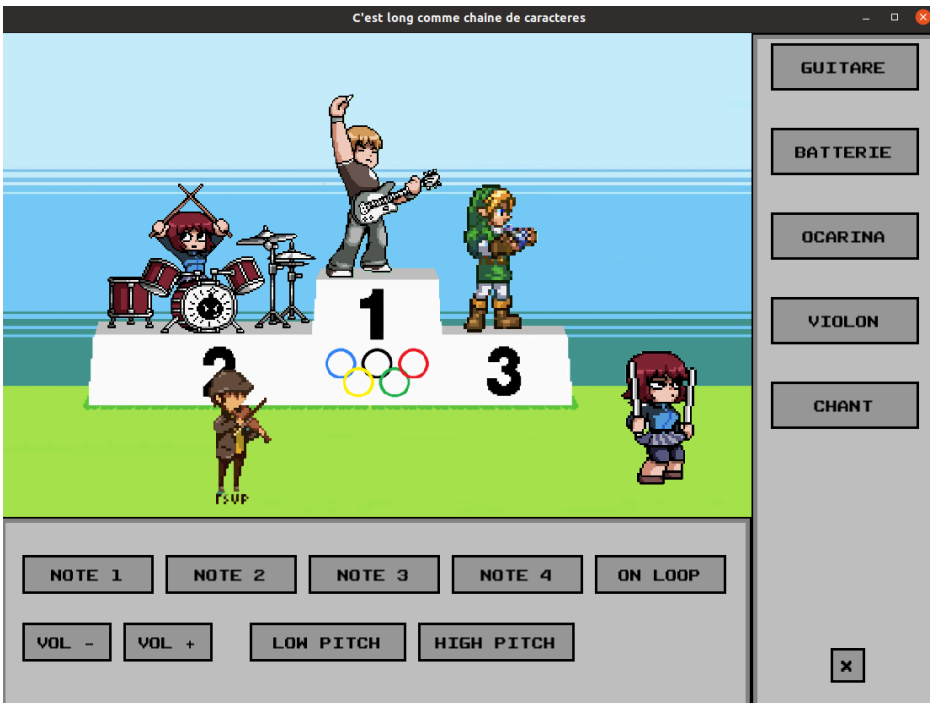
Faire ensuite la commande “**make**” dans le terminal pour compiler le programme puis écrire “**./prog**” pour lancer l’application.

4 - Application Graphique :

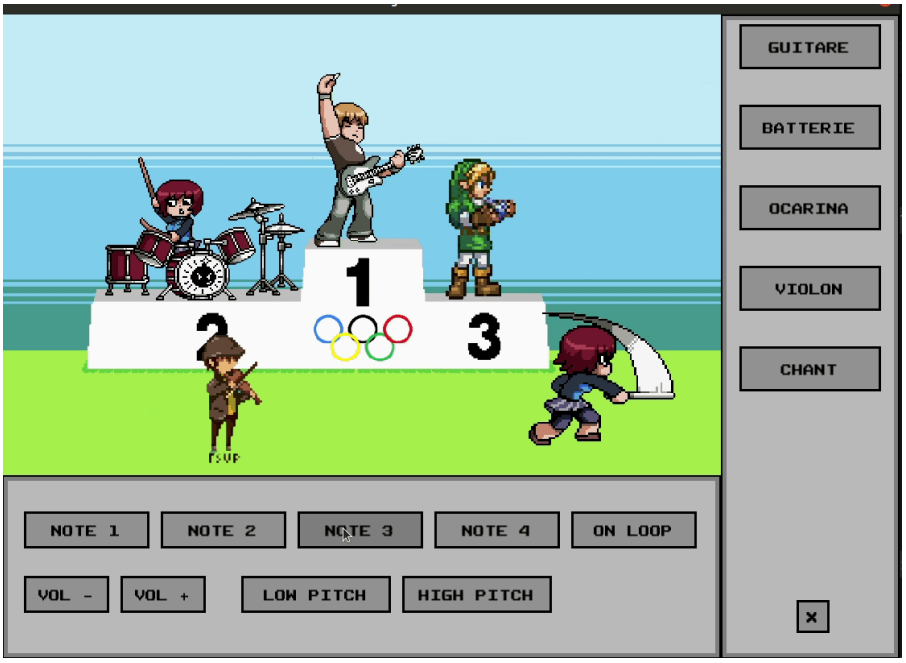


Voici ci-dessus l’affichage initial de l’application.
On observe tout d’abord nos sprites qui sont initialisés dans leur stance c'est-à-dire leur posture de base. On voit au premier plan la chef d’orchestre, et nos musiciens en second plan (ne pas faire attention à la ressemblance entre la batteuse et la chef d'orchestre, elles sont jumelles).

Lors de l’appui sur un des boutons appelant nos musiciens, il y aura la génération d’un affichage de boutons comme suit :



L’appui sur un des boutons note génère simultanément un son ainsi qu’une animation commune qui anime le chef d’orchestre et le musicien souhaité et ce le long du morceau joué. Voici un aperçu de l’animation (continuez à regarder, c’est un peu lent) :



Il est également possible de combiner les notes et de les jouer en boucle pour pouvoir générer une multitude de morceaux différents . Le joueur peut également accorder (en réalité changer la hauteur du son joué) ses instruments et jouer plus ou moins fort.

Enfin, si le joueur souhaite terminer le concert, il peut tout simplement quitter la fenêtre en appuyant sur la croix en bas à droite de sa fenêtre.

5 - Mise en valeur des contraintes :

Nous avons réalisé au total les 8 classes requises.

Nous nous sommes également servis de conteneurs pour réaliser la méthode virtuelle setParameters, nous avons choisi d'utiliser des vecteurs car ceux-ci permettent d'utiliser un système de coordonnées simples et un système de liste pour initialiser les paramètres de chacun des sprites sans nous soucier du nombre de postures que possèdent chacun des musiciens.

De plus, nous avons également nos deux méthodes virtumooelles que sont modif pitch et modif volume ces méthodes sont déclarées et bien définies plus tard dans les classes filles suivant le tableau de fichier sons propres à chacun des musiciens.

Comme vous pouvez le constater nous avons fait le choix fort de ne pas réellement faire 3 niveaux de hiérarchie car cela nous semblait peu justifié, nous réfléchissions sur la fin à la réalisation d'une classe supérieure orchestre qui serait un conteneur de nos musiciens et de notre chef d'orchestre mais cela semblait léger pour justifier efficacement cette hiérarchie.

Nous avons également veillé à conserver des méthodes courtes et à commenter abondamment le code.

On observe également l'absence de fuites mémoire sur valgrind:

```
youssef@ubuntu:~/Downloads/Paul$ valgrind ./prog --leak-check=full
==5971== Memcheck, a memory error detector
==5971== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5971== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==5971== Command: ./prog --leak-check=full
==5971==
==5971== Conditional jump or move depends on uninitialised value(s)
==5971==    at 0x5D86565: pa_shm_cleanup (in /usr/lib/x86_64-linux-gnu/pulseaudio/libpulsecommon-13.99.so)
==5971==    by 0x5D867A1: pa_shm_create_rw (in /usr/lib/x86_64-linux-gnu/pulseaudio/libpulsecommon-13.99.so)
==5971==    by 0x5D764B6: pa_mempool_new (in /usr/lib/x86_64-linux-gnu/pulseaudio/libpulsecommon-13.99.so)
==5971==    by 0x5CFB9B1: pa_context_new_with_proplist (in /usr/lib/x86_64-linux-gnu/libpulse.so.0.21.2)
==5971==    by 0x4FE1955: ??? (in /usr/lib/x86_64-linux-gnu/libopenal.so.1.19.1)
==5971==    by 0x4FE3EB2: ??? (in /usr/lib/x86_64-linux-gnu/libopenal.so.1.19.1)
==5971==    by 0x4FAF9FE: ??? (in /usr/lib/x86_64-linux-gnu/libopenal.so.1.19.1)
==5971==    by 0x53334DE: __pthread_once_slow (pthread_once.c:116)
==5971==    by 0x4FAECA3: alcOpenDevice (in /usr/lib/x86_64-linux-gnu/libopenal.so.1.19.1)
==5971==    by 0x48BF011: ??? (in /usr/lib/x86_64-linux-gnu/libsfml-audio.so.2.5.1)
==5971==    by 0x48BEF0C: sf::AlResource::AlResource() (in /usr/lib/x86_64-linux-gnu/libsfml-audio.so.2.5.1)
==5971==    by 0x48C3552: sf::SoundSource::SoundSource() (in /usr/lib/x86_64-linux-gnu/libsfml-audio.so.2.5.1)
==5971==
Button details:
Width: 35| Height: 35
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
Les boutons sont de mêmes dimensions
sh: 1: leaks: not found
==5971==
==5971== HEAP SUMMARY:
==5971==     in use at exit: 297,089 bytes in 3,062 blocks
==5971==   total heap usage: 230,271 allocs, 227,209 frees, 377,967,301 bytes allocated
==5971==
==5971== LEAK SUMMARY:
==5971==    definitely lost: 0 bytes in 0 blocks
==5971==    indirectly lost: 0 bytes in 0 blocks
==5971==      possibly lost: 304 bytes in 2 blocks
==5971==    still reachable: 296,785 bytes in 3,060 blocks
==5971==         suppressed: 0 bytes in 0 blocks
==5971== Rerun with --leak-check=full to see details of leaked memory
==5971==
==5971== Use --track-origins=yes to see where uninitialised values come from
==5971== For lists of detected and suppressed errors, rerun with: -s
==5971== ERROR SUMMARY: 2 errors from 1 contexts (suppressed: 2 from 2)
```

6 - Nos fiertés :

Nos fiertés principales résident dans:
- la fonction défilement sprite:
Cette fonction consiste en un parcours d'un tableau de sprite en changeant l'index au bout d'une certaine valeur ce qui donne l'impression que le sprite est animé puisque la succession de postures à un rythme élevé donne cet effet.

- sprites et placement des sprites:
Après de longues recherches pour obtenir les sprites que nous avons utilisés (c'est compliqué de trouver plusieurs sprites d'un personnage qui joue d'un instrument), nous sommes satisfait de la disposition actuelle dans laquelle nous avons réussi à utiliser différentes perspectives et incorporer avec satisfaction les différents personnages dans le décor.

7 - Bibliographie :

https://youtu.be/T31MoLJws4U

Tutoriels 2.1 (SFML / Apprendre) (sfml-dev.org)

https://www.spriters-resource.com/xbox_360/scottpilgrimvstheworldthegame/sheet/70802/

https://en.cppreference.com/w/

https://www.deviantart.com/gregarlink10/art/Link-Ocarina-of-Time-UPDATED-868055542