Rapport BE C++

Badier Isaure, De Freitas Guillaume, Laidet Matthieu

11 mai 2022

1 Présentation du projet

Notre projet est une boîte à énigmes. Il s'agit d'un jeu qui se compose d'un enchaînement d'énigmes basées sur l'interaction du joueur avec la boîte, au travers de différents capteurs. Le tout est narré sur un écran OLED. **Lire la suite de ce rapport sans avoir fait le jeu ruine l'expérience car nous allons détailler la résolution des énigmes**.

2 Déroulement du jeu

Votre ami a passé une soirée mouvementée. Vous vous inquiétez pour lui et décidez de le retrouver.

Lors de la première énigme, vous entendez une mélodie, peut être est-ce la sonnerie du téléphone de votre ami ? Pour vous en assurer il faudra la reproduire (à la manière d'un *Simon*).
Chaque bouton correspond à une note de musique et l'appui sur le mauvais bouton vous fait recommencer l'énigme.

Une fois assuré qu'il s'agit bien de la sonnerie de votre ami vous devez rejoindre la pièce d'où provient le son. Cette énigme est un labyrinthe en caractères ASCII. Il vous faut rejoindre la sortie en déplaçant le "o" avec les 4 boutons (utilisés comme croix directionnelle).

Vous l'avez trouvé mais il dort profondément. Pour le réveiller il faudra appuyer sur le bouton cerclé de noir. Cela ne suffira pas alors vous vous résolvez à l'éblouir. C'est à dire projeter la lumière de votre téléphone portable dans le trou à droite de la boîte (utilisation d'un capteur de luminosité).

Votre ami est réveillé, mais il ne sait plus où il est. Pour lui montrer les environs il faut retourner la boite (utilisation d'un accéléromètre).

Ça y est, il reprend ses esprits et constate un message de son ex sur son téléphone ! Malheureusement son code a été changé lors de cette soirée. L'énigme consiste à trouver le code caché sur la boîte. Dans chaque mot écrit sur la boîte, vous trouverez un chiffre. Pour les remettre dans le bon ordre, il suffit de reconstituer la phrase *Désolé pour hier soir* en hommage au groupe Tryo. Pour saisir le code, on utilise les boutons gauche et droit pour naviguer entre les digits et les boutons hauts et bas pour incrémenter ou décrémenter la valeur du digit.

Enfin il peut voir ce que son ex lui a répondu. Il vous faudra juste reconstituer le message en déplaçant la bonne lettre au bon endroit en utilisant les boutons comme croix directionnelle. Vous connaissez la cause de la rupture entre votre ami et son ex. A vous de le convaincre de vous raconter lorsqu'il se réveillera (Fin du jeu).

3 Conception et explication du code

La progression dans le jeu étant assez linéaire nous avions au départ comme volonté de créer un enchaînement simple consistant à poser une énigme et à passer à la suivante. De ce principe là, a découlé un effort de standardiser au maximum tout ce qui pouvait l'être. L'ensemble des explications concernant la conception s'appuieront sur le diagramme de classe présenté en figure 1.

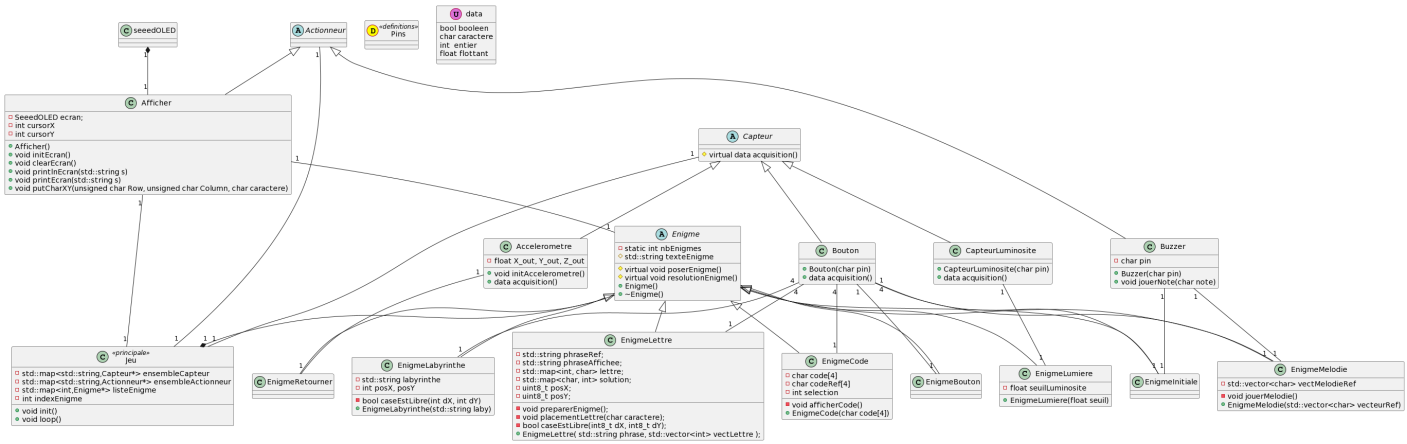

Figure 1: Diagramme de classe

Notre jeu est composé de trois type d'objets principaux : des objets liés aux interactions avec le hardware que sont les actionneurs et les capteurs ainsi que des objets logiciels que sont les énigmes. Ces trois types d'objets avec le jeu forment les quatre classes centrales de notre projet. Le jeu possède ainsi trois ensembles respectivement d'énigmes, de capteurs et d'actionneurs qui peuvent être mis en relation.
Chaque Capteur renvoie une union de type `data` via la méthode `acquisition()`. Cette donnée est la grandeur mesurée qui sera ensuite traitée pour la résolution d'une énigme.
Chaque énigme est exécutée par deux méthodes : `poserEnigme()` et `resolutionEnigme()`.

Pour chaque énigme `poserEnigme()` affiche le texte de l'énigme et effectue les étapes d'initialisation. Par exemple pour l'énigme de la mélodie, on joue la mélodie une première fois.

`resolutionEnigme()` contient une structure de contrôle de type `while(!enigmeValidee)` dans laquelle on vérifie les actions du joueur jusqu'à validation de l'énigme.

Le jeu se déroule en deux phases `init()` et `loop()`. Dans la phase d'initialisation, on crée et alloue nos capteurs, énigmes et actionneurs que l'on stocke dans les map suivantes. De cette façon on maîtrise la vie de ces objets et on peut les désallouer dans le destructeur de Jeu.

```
std::map<std::string, Capteur *> ensembleCapteur;
std::map<std::string, Actionneur *> ensembleActionneur;
std::map<int, Enigme *> listeEnigme;
```

La phase de loop quant à elle ne fait qu'itérer dans la liste d'énigme. Une fois la liste passée le programme propose de jouer une nouvelle partie.

4 Déroulement du projet

Pour pouvoir travailler en parallèle en sachant ce que les autres faisaient, nous avons passé beaucoup de temps au début du projet à réfléchir ensemble à la conception. Nous n'avons commencé à coder qu'à la troisième séance du BE. En revanche, nous avions préparé les headers de chaque classe afin d'harmoniser les notations d'attributs et de méthodes.

Sur la suite du projet, nous nous sommes séparés le travail en travaillant chacun sur une énigme basée sur un capteur différent. De cette façon chacun avait à implémenter la librairie de son capteur ainsi que de son énigme. Les tests ont été effectués via des affichages console le temps de prendre en main la librairie `SeedOLED`. Bien que la librairie `Afficher` existait, les fonctions n'étaient pas implémentées. Nous aurions dû implémenter plus rapidement Afficher avec des `cout` de façon à ne pas avoir à reprendre le code des énigmes.

La classe abstraite énigme avait des méthodes dont chaque énigme hérite, nous avons ensuite rajouté pour chacune les attributs et méthodes qui leur sont propres (par exemple le seuil de luminosité pour l'énigme Lumière).

Une fois tous les capteurs implémentés pour leur énigmes respectives, on a pu rajouter de nouvelles énigmes basées sur des capteurs déjà utilisés. En effet, l'énigme du bouton qui consiste juste à appuyer sur un bouton a permis l'implémentation de la classe `Bouton` que les énigmes Code, Lettre et Labyrinthe réutilisent.

Le seul point flou dans notre conception au départ était la durée de vie des instances de classes. Afin de la maîtriser pleinement et de pouvoir avoir autant d'instances que nécessaire, nous avons décidé d'instancier énigmes et capteurs dans le Jeu plutôt que dans leurs headers respectif (comme c'est le cas pour le Serial d'Arduino). De fait, pour ne pas avoir de variables globales, nous sommes passés par des pointeurs de classe abstraites dans une structure sur laquelle on peut itérer. Au départ nous voulions stocker les capteurs dans un `vector<Capteur*>` et non une `map<"string",Capteur*>`. Cependant, un vecteur de `capteur*` ne nous a pas permis d'accéder aux membre des objets de classes héritées dont la référence était contenue dans le vecteur. Par exemple, un `Accelerometre` est un `Capteur`. Sa référence peut être stockée dans un `vector<Capteur*>`. Cependant, nous ne pourrons pas accéder à la méthode `Accelerometre::initAccelerometre()`, cela ne compile pas.

Nous avons également été bloqués par le faible nombre de ports input/output de la carte Espressif8266. Nous utilisons dans notre projet quatre boutons qui sont des entrées ainsi que un buzzer qui est une sortie. Nous avions donc besoin de cinq GPIO différents car l'accéléromètre et l'écran OLED utilisent une communication I2C et le capteur de luminosité une entrée analogue.
Sur l'Espressif utilisé, certains ports comme D3, D4 ou D10 sont utilisés pour flasher le code sur la carte. Il était donc impossible de brancher un bouton dessus, car le téléversement ne fonctionne pas si l'on force un de ces pins à une valeur. Nous avons donc dû brancher le buzzer sur le pin D3. Néanmoins, ce pin a une valeur différente de 0 lors du flash : nous avons donc rajouté un interrupteur afin de pouvoir éteindre le buzzer pendant le flash et préserver nos oreilles. L'énigme initiale (qui propose de commencer la partie) écrit la valeur 0 sur le pin D3, permettant son utilisation par la suite.