POLYTECH
SORBONNE

Rapport final de projet de programmation CPP

MAIN4 - année 2017/2018

Encadré par :
Cécile BRAUNSTEIN
Jean-Baptiste BREJON

Projet réalisé par :
Fatine BENTIRES ALJ,
Alexia ZOUNIAS-SIRABELLA

Table des matières

Première partie

A savoir sur le jeu

1 Préambule : ce qu'il vous faut télécharger

Afin de lancer le jeu que nous avons crée, voici une liste non exhaustive des bibliothèques à télécharger :

— <SDL2/SDL.h>
— <SDL2/SDL_events.h>
— <iostream>
— <cstring>
— <list>
— <vector>

De plus, pour avoir une visualisation des images crées, il vous faudra placer le dossier **FatineAlexia** sur votre racine.

2 Consignes à respecter

Le projet proposé devait être en lien avec les Paradise Papers. Ces derniers désignent des révélations publiées en Novembre 2017 par le consortium international des journalistes d'investigation sur la base d'une fuite de plus de 13,5 millions de documents confidentiels notamment issus du cabinet d'avocats Appleby.

Cependant le code proposé devait également répondre à diverses contraintes (au minimum) :
— Contenir 8 classes ;
— Avoir 3 niveaux de hiérarchie : **Questions→ Reponses→Score→Detail**
— Utiliser 2 fonctions virtuelles : **virtual initialisation()** dans Questions et Score
— Utiliser 2 surcharges d'opérateurs : **operator()** dans Score et dans Reponses
— Utiliser 2 conteneurs de la *STL* : **Vector et list**
— Commenter le code ;
— Enlever les erreurs *Valgrind* ;
— Dépôt git
— Pas de méthodes/fonctions de plus de 30 lignes
— Utilisation d'un Makefile

En constatant le nombre d'informations révélées par les Paradise Papers, nous avons décidé d'implémenter un quizz afin que chacun puisse évaluer ses connaissances sur le sujet. Ainsi, vous trouverez des questions théoriques concernant les moyens utilisés pour l'évasion fiscale mais aussi d'autres concernant les personnalités et entreprises accusées dans ces révélations.

3 Fonctionnement

Dans le cadre du module de cpp nous avons décidé de créer un quizz sur le sujet du paradise paper. Mais pourquoi un quizz et pas un jeu ? Pour suivre la trace de Madame Braustein bien sûr. En effet, la plupart des examens de cpp écrits correspondent à des QCMs. De plus, les autres groupes implémentent des jeux pour la plupart. Nous voulions donc nous démarquer. Notre vision du jeu est alors la suivante :

Le quizz se décompose en deux parties. Une partie d'entraînement et une partie contre la montre. Une partie d'entraînement se décompose en 10 questions. Les questions sont toujours les mêmes. Il s'agit là de donner uniquement un aperçu de ce que le jeu peut fournir. Une partie contre la montre correspond à 5 questions données aléatoirement. Comme son nom l'indique, l'utilisateur a un temps limité pour l'ensemble des réponses. Les questions changent d'une partie à l'autre. Ainsi, le jeu commence par une page d'accueil. L'utilisateur doit cliquer sur **1** si il choisit le mode d'entraînement et **2** pour le mode contre la montre.

4 UML

Pour avoir une meilleure vision du jeu, voici un UML représentant les différentes classes et leurs agencements :

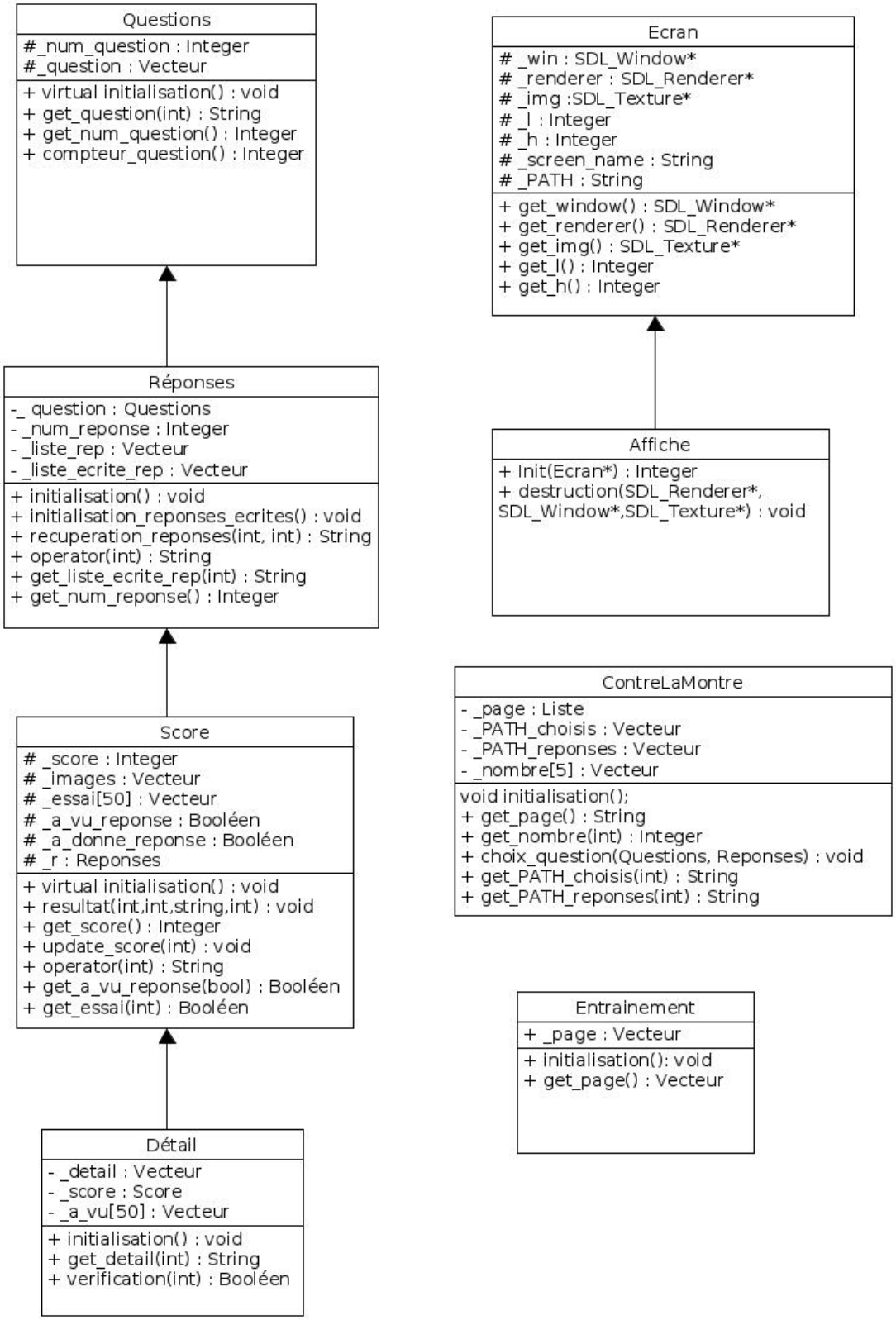

FIGURE 1 – Diagramme UML

Deuxième partie

Partie technique

5 Les questions, réponses et détails du jeu

5.1 L'implémentation des questions

La classe **Questions** nous permet de réunir les images contenant les différentes questions du quizz en leur associant divers attributs :

- Un identifiant unique correspondant au numéro de la question stocké dans *_num_question*
- Un vecteur *_question* regroupant tous les chemins des images avec les questions

Ce tableau est rempli grâce à la méthode virtuelle*initialisation()*. De plus, le chemin de l'image d'une question *i* est facilement récupérable grâce à la méthode *get_question(int i)*. Pour finir, la fonction *compteur_question()* permet de passer à la question suivante. Cette dernière sera notamment utile pour le mode **Entraînement** détaillé plus bas.
Une amélioration possible aurait été de réunir les chemins des images dans un fichier texte au lieu d'utiliser un tableau mais, par manque de temps, nous avons décider de l'implémenter avec un vecteur pour y accéder très facilement.

5.2 L'obtention des réponses du jeu

La classe **Réponses**, héritant de **Questions**, nous permet de réunir les images contenant les réponses aux questions en leur ajoutant différents attributs :

- Le numéro de la réponse stocké dans *_num_reponse*, associé à une question
- Un vecteur *_liste_rep* regroupant tous les chemins des images avec les réponses, la fonction récupérant cette liste étant un opérateur de surcharge *operator()*
- Un vecteur *_liste_ecrite_rep* regroupant toutes les réponses (A, B ou C)

Deux méthodes d'initialisation permettent de remplir le tableau des chemins des images et celui des réponses, *initialisation()* et *initialisation_reponse_ecrites()*. Comme pour la classe **Questions**, il est possible de récupérer le chemin de la réponse ou la réponse elle-même pour une question *i*. Là encore, il aurait été possible d'améliorer le code en utilisant un fichier contenant l'intégralité des chemins.

5.3 La possibilité d'avoir plus de détails

La classe détails découle de la classe Score donnant ainsi 4 niveaux de hiérarchie. Une fois les réponses données, et pour certaines uniquement, il est possible d'avoir plus de détails. Ainsi, on retrouve :

- La variable *_detail* qui va contenir les chemins des images donnant le détail
- La fonction *initialisation()*, même principe que précédemment. La seule distinction est qu'elle initialise uniquement la variable _detail pour les réponses qui possèdent du détail.
- l'appel à la classe Score
- La fonction *verification* qui vérifie si la réponse a été vue ou pas. Ainsi, si ce n'est pas le cas on ne peut pas voir le détail de la réponse (pour éviter toute tentative de triche)

6 Le Score

La classe Score est une classe très importante. Elle permet de compter les points de l'utilisateur et fait ainsi particulièrement attention aux exclusions. Elle découle de la classe *Reponses*. Elle est composée de plusieurs fonctions. On retrouve entre autre :

- Un constructeur par défaut et un constructeur initialisant Reponse
- Une fonction virtuelle initialisation() pour l'initialisation des chemins des images qui vont afficher le score
- Un *get_score()* permettant de récupérer le score
- Une fonction *update_score()* qui remet le score à jour
- Un opérateur de surcharge *operator()* renvoyant les chemins des images
- Une fonction *get_a_vu_reponse()* qui vérifie si l'utilisateur a vu la réponse ou non

- Une fonction *Resultat* qui récupère la réponse de l'utilisateur, la compare à la réponse stockée, vérifie qu'il s'agit du premier essai d'appui sur la touche et qu'il n'a pas vu la réponse ni le détail avant d'itérer le score

```
void Score::resultat(int numero_question, int mode, std::string touche, int i)
{
std::string reponse_faite=_r.recuperation_reponses(numero_question,mode);
bool verdict= get_a_vu_reponse(_a_vu_reponse);

if(reponse_faite==touche && verdict==false && _essai[i]==false)
{
  _score= _score +1;
  std::cout << "_score_boucle=" << _score << std::endl;
 _essai[i]=true; //on passe essai de la boucle correspondante a true apres
}
else
    _essai[i]=true;
}
```

7 L'affichage des fenêtres

7.1 La classe Écran

Passons à l'explication de l'affichage des fenêtres. La classe Écran permet l'initialisation des variables SDL nécessaires à l'affichage d'un Écran et des différents composants de cet Écran. Ainsi, elle se compose de :

- Variables de type SDL_Window, SDL_Renderer et SDL_Texture.
- 2 constructeurs, un par défaut et un prenant en entrée les informations nécessaires (largeur, hauteur, chemin...)
- un destructeur permettant la libération de la mémoire

Le but de la classe écran est de mettre en place les éléments utilisés par la classe Affiche.

7.2 La classe Affiche

La classe Affiche hérite de la classe Écran. Son but est d'afficher l'écran de la page d'accueil. Elle comprend :

- Une fonction *Init* qui initialise l'écran, le renderer et l'image en récupérant les données d'Ecran et en utilisant les fonctions *IMG_LoadTexture* et *SDL_QueryTexture*

- Une fonction *destruction* qui libère la mémoire

Ces deux fonctions sont celles qui vont permettre une interaction avec l'utilisateur via une plate forme d'affichage.

8 Les modes du jeu

8.1 La partie Entraînement

Comme évoqué plus haut, le jeu présente deux modes de jeu. Nous allons nous intéresser ici au mode Entraînement qui permet au joueur de découvrir le fonctionnement du jeu avec 10 questions sélectionnées. Ces dernières seront toujours les mêmes et dans le même ordre ce qui lui permettra également d'avoir le temps de lire tous les renseignements donnés. La classe **Entrainement** ne contient qu'un attribut nommé *_page* qui correspond à un **conteneur STL** de type *vector* et va contenir le chemin de la page propre à ce mode de jeu (la page d'accueil). On retrouve également deux méthodes différentes des constructeurs :

— *initialisation()* qui permet d'initialiser *_page* au chemin correct
— *get_page* qui renvoie le chemin si besoin est

En utilisant les autres classes, il sera donc aisé de faire défiler les 10 questions sélectionnées, leur réponse ainsi que les détails qui peuvent être associés.

8.2 La partie Contre la montre

Le deuxième mode correspond à celui contre la montre. Ce dernier est un peu plus innovant puisqu'une partie correspond à 5 questions données aléatoirement parmi les 30 questions crées et n'autorise la réponse qu'en un temps limité (ici 20 secondes). Il aurait été possible de créer une nouvelle classe comprenant différents niveaux de difficultés (difficultés des réponses et temps plus ou moins limité), mais par manque de temps nous ne l'avons pas fait. Ainsi, la classe est composée :

— D'une fonction *initialisation()* qui charge le chemin et reposant donc sur le même principe que les fonctions initialisation des autres classes
— D'un *get_page()* permettant de récupérer le chemin de la page dédiée à cette partie (page de consigne) stocké dans la variable *_page* qui correspond à un **conteneur STL** de type *list*
— D'une fonction *get_nombre(int i)* qui va stocker les nombres aléatoires générés
— De deux vecteurs de string *_PATH_choisis[50]* et *_PATH_reponses[50]* qui vont contenir respectivement les chemins des questions et réponses sélectionnées aléatoirement
— D'une fonction *choix_question(Questions question, Reponses reponse)* qui va commencer par générer 5 nombres aléatoires puisqu'une partie est composée de 5 questions puis va récupérer les questions et réponses de ces nombres aléatoires et les stocker dans les deux variables énoncées précédemment

La contrainte de temps se trouve directement dans le fichier main.cpp. On commence par utiliser une variable de type *time_t* que l'on appelle start. On crée également deux doubles correspondant au temps qui passe (*minuteur*) et à la limite de temps (*temps_ecoule*). Si le minuteur est supérieur au temps écoulé, on affiche la photo avec le score marquant la fin de la partie.

```
void ContreLaMontre::choix_question(Questions question, Reponses reponse)
{
    int tmp;
    srand(time(NULL));
    for(int i=0; i<5; i++)
    {
        tmp=rand()%31;
        _nombre[i] = tmp;
        while(tmp==10)
            _nombre[i]=rand()%31;
    }

    for(int i=0; i<5; i++)
    {
        _PATH_choisis[i]=question.get_question(_nombre[i]);
        _PATH_reponses[i]=reponse.operator()(_nombre[i]);
    }
    _PATH_choisis[5]=question.get_question(10);
}
```

9 Le fichier principal : main.cpp

9.1 L'agencement des fichiers

Le fichier main.cpp correspond au cœur du programme. Pour rédiger cette partie nous nous sommes basés sur le tutorat SDL d'Alexandre Laurent disponible en ligne. Ainsi, le fichier main.cpp est composé de plusieurs parties. Dans un premier temps, il fait appel à toutes les classes créées et les bibliothèques nécessaires. Puis il se décompose d'une fonction *quizz()* qui regroupe le squelette du programme et d'une fonction *main()* qui correspond à la fonction principale.

La fonction quizz() est composée d'une première partie permettant d'afficher la fenêtre ainsi que l'image de départ. Elle fait ainsi appel aux classes *Ecran* et *Affiche* définies précédemment. La seconde partie correspond à l'appel des fonctions initialisation() de toutes les classes permettant le chargement des chemins des images. Elle met également en place les itérateurs utilisés dans les boucles. Enfin, la dernière partie correspond au traitement des événements avec l'utilisation de *SDL_Event* et la mise en place d'un switch.

9.2 La gestion des touches du clavier

Au sein du switch, plusieurs fonctionnalités sont disponibles. Globalement, il s'agit du même principe en fonction de si on joue une partie d'entraînement ou une partie contre la montre (modélisé par la variable etat_courant==1 ou ==2) :

- *SDLK_ESCAPE* pour quitter le jeu
- *SDLK_1* si l'utilisateur souhaite jouer une partie d'entraînement
- *SDLK_2* si l'utilisateur souhaite jouer une partie contre la montre
- *SDLK_a* si l'utilisateur clique sur A, on fait appel à la fonction *resultat()* de score. Si la réponse est bien A, on incrémente le score
- *SDLK_b* et *SDLK_c* basés sur le même principe que précédemment
- *SDLK_d* si l'utilisateur veut voir plus de détails à la réponse donnée
- *SDLK_RIGHT* si l'utilisateur appuie sur la flèche de droite pour passer à la question suivante. Dans ce cas là, on passe la fonction *get_a_vu_reponse* de score à false, indiquant qu'il vient de changer de question et qu'il n'a pas encore vu la réponse.
- *SDLK_UP* si l'utilisateur veut regarder la réponse. Dans ce cas là après action on passe la fonction précédente à true.
- *SDLK_DOWN* si l'utilisateur veut voir son score. Ceci n'est possible que lorsque le jeu est fini donc si l'itérateur i de la partie d'entraînement vaut 11 (puisqu'il est itéré une fois de plus que le nombre de questions) ou si l'itérateur de la partie contre la montre j=6 (toujours sur le même principe).