Rapport de projet : C++

EVRARD Jade – ROYER Jérémy – EISE4

Thème du projet : There is no Planet B !

Nom du projet : Escape Green

Description de l'application développée

Le thème qui nous a été imposé est en lien avec l'environnement et l'écologie. En effet, il peut être intéressant de développer différentes interfaces permettant de sensibiliser la population aux problématiques liées à l'environnement.

Nous avons fait le choix de programmer un jeu qui prend la forme d'un espace game virtuel. Le but du jeu est donc de sauver la planète en résolvant différentes énigmes au cours du temps. Comme dans un vrai espace game, le joueur passe d'une salle à une autre en répondant correctement aux questions et/ou défis qui lui seront posés. Une fois toutes les énigmes résolues, le joueur a réussi à s'échapper et a donc réussi à SAUVER LA PLANETE. Pour rester dans le thème de l'environnement, tous nos mini-jeux internes aux différentes salles sont à visée didactique. Notre jeu est composé de 2 salles : la première est sur le thème des déchets tandis que la seconde est sur le thème des fruits de saison. Dans chaque salle, le joueur devra à chaque fois résoudre des énigmes de 3 types différents : un mini-jeu, une question et une charade. Aucune erreur n'est permise ! Tant que le joueur n'a pas 100% de bonnes réponses, il ne peut passer à la suite du jeu.

Nous avons également préféré ne pas mettre de chronomètre (à la différence des vrais escape game). Ce choix a été fait car nous préférons que tout le monde puisse finir le jeu et apprendre sur les bonnes attitudes à avoir quant au tri des déchets et à la sélection des fruits de saison.

Respect des contraintes

Tout d'abord, il était exigé de réaliser 8 classes minimum ainsi que 3 niveaux de hiérarchie. Dans notre jeu, nous avons utilisé 10 classes au total (en comprenant les classes abstraites) et 3 niveaux de hiérarchie. Nous avons donc respecté ces 2 contraintes.

Par ailleurs, nous avons créé 2 fonctions virtuelles différentes dans la classe abstraite *Game*. La première est la fonction *game_show* qui prend en paramètre un booléen tandis que la seconde est la fonction *verif_click* qui prend en paramètre un entier. Ces 2 fonctions sont des fonctions virtuelles pures. Elles ont été créées car toutes les classes filles utilisent des fonctions *game_show* et *verif_click* qui doivent être redéfinies pour chaque classe fille car l'interface graphique est différente pour chacune de ces classes.

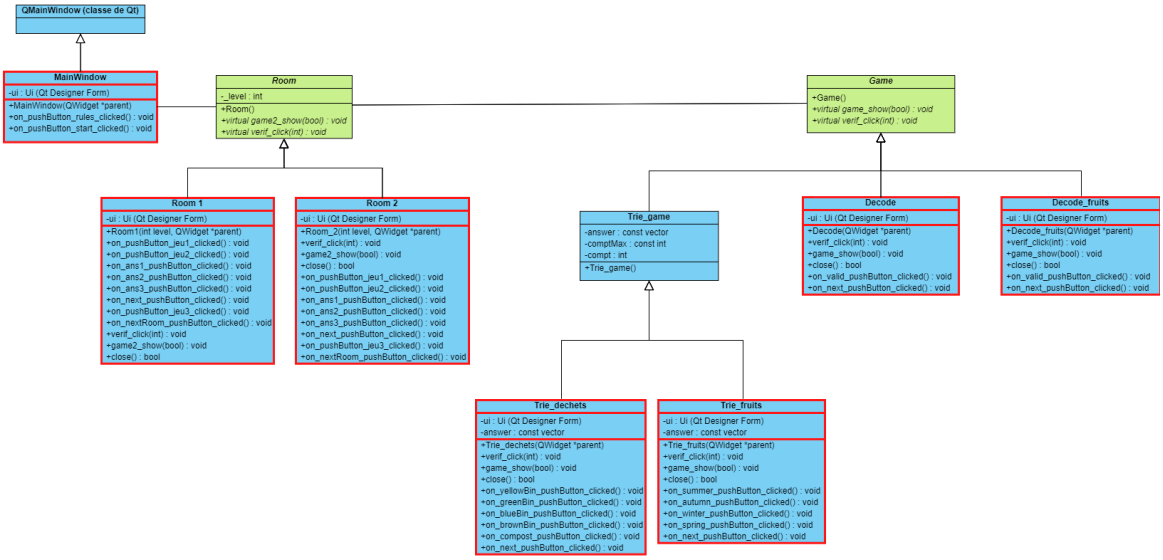
Nous avons utilisé un conteneur de la STL, qui est un *vector*. Il nous permet de stocker la réponse à chaque image lors des jeux de tri. On compare donc le bouton cliqué à la réponse contenu dans le conteneur.

Il nous était également demandé de réaliser un diagramme UML complet de l'application. Celui-ci a été fait et est à retrouver dans la partie suivante ci-dessous.

De plus, de nombreux commentaires ont été mis tout au long du code pour expliquer au mieux chaque partie du code comme souhaité. Notre code peut également être compilé sans erreurs et sans warnings. Enfin, toutes nos fonctions/méthodes respectent la limite de 30 lignes maximum (hors commentaires, lignes vides et assert).

Finalement, nous utilisons un Makefile pour compiler notre projet. Nous l'utilisons sous Windows avec le logiciel Qt Creator.

Diagramme UML de l'application

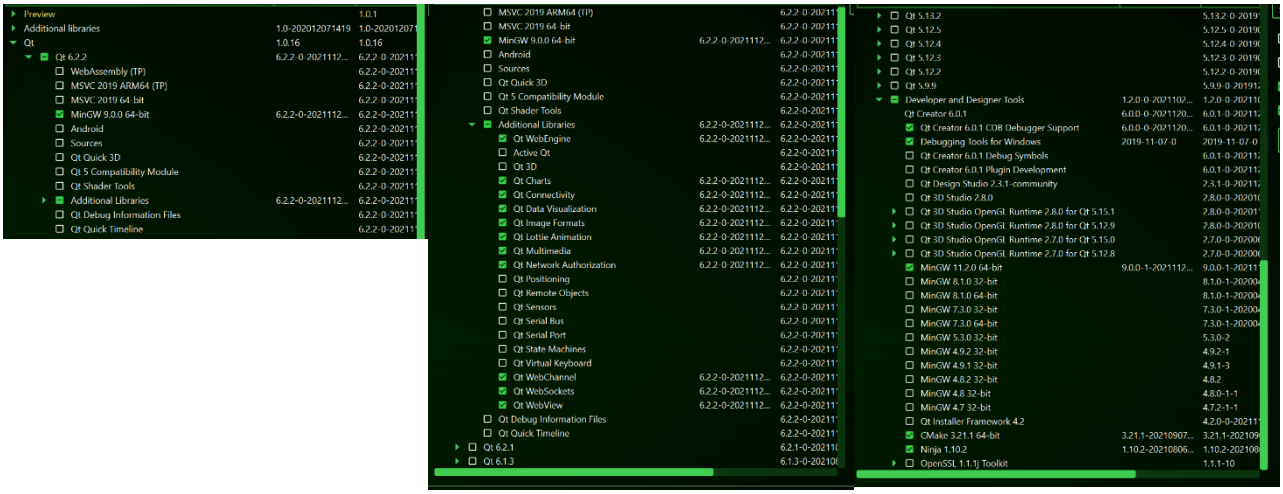


Procédure d'installation et d'exécution du code

Afin d'exécuter notre jeu, il vous faudra tout d'abord télécharger le logiciel Qt Creator. Ensuite, vous pouvez télécharger les dossiers *Escape_green* et *pictures* présent sur le GitHub. Puis, il faut ouvrir Qt Creator et dans le menu *Fichier* choisir *Ouvrir un fichier ou un projet…* (Ctrl+o). Choisissez le fichier *Escape_green.pro* dans le dossier *Escape_green*. Vous n'avez plus qu'à exécuter le code en mode debug (Ctrl+r).

Liens utiles :

- Qt Creator : https://www.qt.io/download-qt-installer



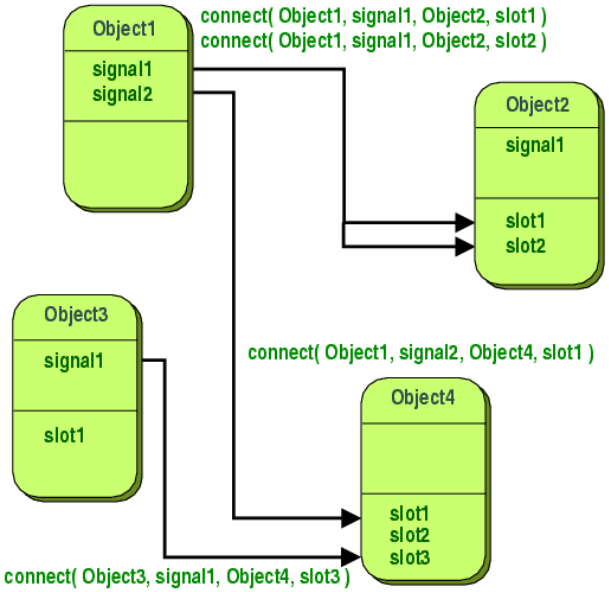
- GitHub : http://github.com/Jeremy-ROYER/EscapeGreen

Parties d'implémentation dont nous sommes le plus fier

Durant la conception de notre projet, nous avons rencontré plusieurs problèmes. Entre la prise en main de la programmation graphique et la programmation évènementielle, nous avons fait face à beaucoup de nouveautés.

Un des plus gros obstacles que l'on a dû surmonter était de devoir changer de fenêtre puis revenir sur la précédente une fois la partie de jeu terminée. C'est-à-dire qu'on devait créer une fenêtre (n°2 pour l'exemple), cacher la fenêtre active (n°1 pour l'exemple) et afficher la fenêtre qui venait d'être créée. Le problème est qu'il fallait ré-afficher la fenêtre n°1, une fois que la fenêtre n°2 a été fermée, sachant qu'il n'y a pas de temps fixe pour la « vie » de la fenêtre n°2 .

Nous avons donc utilisé, pour résoudre la difficulté, le mécanisme des signaux et des « slots ». Ce mécanisme permet, sous Qt, de connecter deux objets (QObject). Lorsque l'objet 1 émet



un signal alors il est émis jusqu'à l'objet 2 et lorsque ce dernier le reçoit alors un slot de l'objet 2 est exécuté. Nous avons donc exploité ce mécanisme en créant un signal *closing()* et en surchargeant la fonction *close()* de la classe qui est envoyée à la fenêtre qui doit se rouvrir lorsque la fenêtre actuelle se ferme. Lorsque la fenêtre reçoit le signal alors celle-ci active son slot *show()* qui permet de l'afficher aux yeux de l'utilisateur.

Pour éviter les fuites de mémoire, nous avons utilisé l'attribut *Qt ::WA_DeleteOnClose* sur les fenêtres qui permet de supprimer la fenêtre lorsque celle-ci est fermée.