POLYTECH SORBONNE

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Rapport de projet C++

Lisa Hennebelle et Meriem Lagha

C++- Cécile Braunstein

Introduction

Le but de ce mini-projet est la réalisation d'un projet entièrement codé en c++ et sur le thème "Smoke". Comme le sujet était très large, nous avons eu des idées à foison: jeu de carte à coder en c++, point and click etc. L'idée principale étant de coder un jeu impliquant des objets en rapport avec la fumée s'est dégagée et nous nous sommes mises d'accord pour un jeu simple et éducatif.

Instruction de lancement de l'application:

Le jeu a été codé avec l'outil de développement graphique QTcreator. Pour lancer l'application, il faut télécharger notre archive qui contient les librairies Qt nécessaires à son bon fonctionnement ainsi que le makefile permettant de le compiler. Ensuite il suffit de se placer dans le bon répertoire, de lancer la commande make et de lancer l'executable.

Description du jeu

Le principe est relativement simple: Lorsque l'utilisateur lance le jeu, une fenêtre est affichée avec un petit rappel des règles. Ensuite, il doit choisir entre deux versions du jeu: Lite ou Dark. Ce choix ne détermine que du changement de la vision du jeu, d'un point de vue graphique.
Une fois cette sélection effectuée, il obtient une vue sur une fenêtre contenant plusieurs images. Ce sont des objets parsemés dans le décor. L'utilisateur possède aussi un sac, en bas à droite de l'écran, dans lequel il pourra ranger les objets qu'il sélectionne à l'aide des flèches du clavier.
Certains d'entre eux sont spéciaux car ils sont reliés à la fumée. A ce moment, il peut appuyer sur la touche "entrée" de son clavier pour faire apparaître une fenêtre donnant des informations spécifiques à cet objet. Cela permet aussi de colorer l'objet d'une teinte rouge pour permettre de le placer plus facilement dans le sac. Une fois que tous les objets en rapport avec la fumée sont placés dans le sac, le jeu est terminé et l'utilisateur a gagné! Mais attention le jeu est limité en temps: il ne dispose que de 3 minutes pour repérer les objets en rapport avec la fumée et les mettre dans le sac.

1) Mise en place en C++

Pour pouvoir coder un tel jeu, il fallait réfléchir aux classes et fonctionnalités desdites classes qui nous permettraient une fois assemblées de former notre application.
Nous avons donc découpé notre jeu comme suivant :

- Une classe Objet qui permet d'implémenter un objet graphique, à une certaine position de l'écran, et qui peut être déplacé grâce aux touches du clavier. Cet objet ne peut pas être placé dans le sac et dès que l'utilisateur essaye il reçoit un message lui indiquant qu'il s'est trompé d'objet.

- Une classe Smoke qui est une classe fille d'Objet et qui possède des fonctionnalités supplémentaires telles que l'affichage d'informations en rapport, ou le changement de couleur une fois la touche entrée sélectionnée. Un objet Smoke est aussi capable de détecter s'il a été mis dans le sac ou non. Si c'est le cas, il met son attribut flag à 1. Cette fonctionnalité permettra de tester dans la classe Game si le jeu est terminé.

- -Une classe Sac, spécialement dédiée à l'objet sac, qui n'est pas déplaçable, mais dont les positions sont importantes pour déterminer si on a terminé le jeu. Pour tester ces positions dans les fichiers les nécessitant, nous avons rajouté les macros SACX et SACY pour nous permettre de le placer à l'endroit que nous voulions dans l'environnement au fur et à mesure du développement sans devoir changer 6 lignes de code à chaque modification.

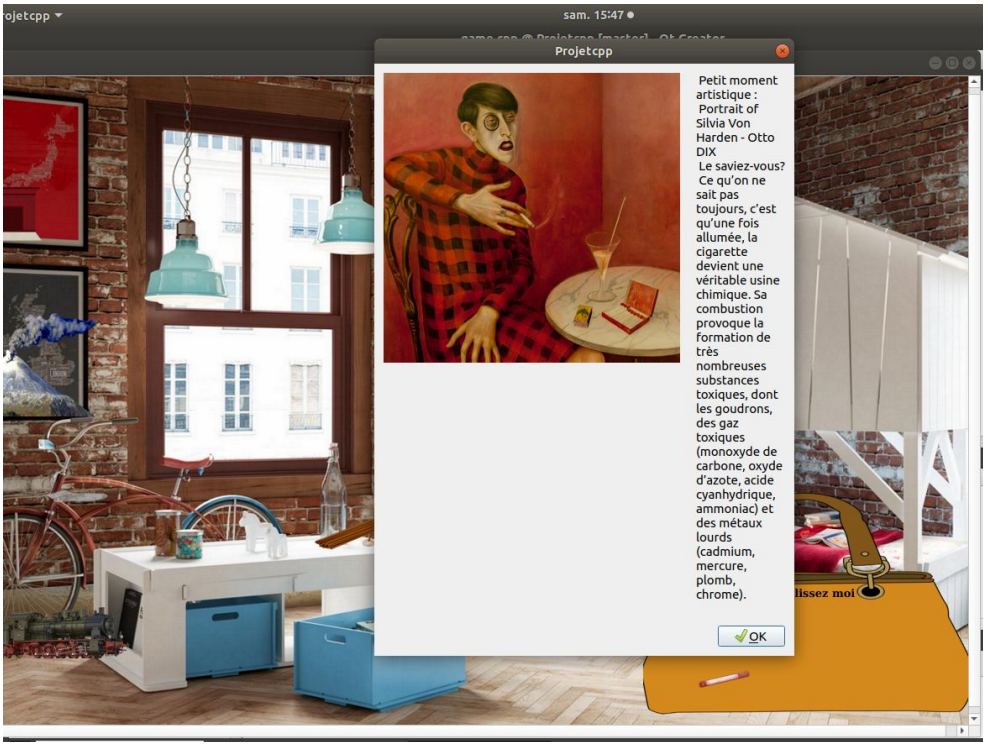
- Une classe Background qui permet de créer le fond d'écran du jeu, qui lui non plus n'est pas déplaçable mais doit apparaître derrière tous les autres objets. Son constructeur prend en argument un entier pour pouvoir choisir le fond d'écran en fonction de la game sélectionnée (Dark ou Lite).


- Une classe Game qui permet d'avoir des listes d'objets et de smoke pour contenir l'intégralité des éléments apparaissant sur l'écran et de pouvoir les créer, puis les tester pour déterminer si le jeu est terminé. Son constructeur prend en argument un entier : si jamais c'est l'entier 0, on construit une game de type Lite sinon on construit une game de type Dark.
  Elle comporte une fonction qui est appelée périodiquement grâce à la fonctionnalité de QT de créer des connections avec des Timers: à la fin du timer la fonction passée en argument de la fonction connect() est appelée et le timer se remet en route.
  cette fonction est isGameOver(). Elle permet de tester la valeur des positions des objets de type Smoke pour s'assurer que le jeu est ou n'est pas terminé. C'est elle qui crée des boites de dialogue pour informer l'utilisateur de la fin du jeu, qu'il ai gagné avant la fin du timer ou non.

  Dans cette classe est aussi initialisé un attribut de type QTimer qui nous permet d'instancier un timer de 3 minutes à partir du moment où on a créé le jeu. Une fois ce temps écoulé, si l'utilisateur n'a pas réussi à regrouper tous les objets, une boîte de dialogue s'ouvre pour lui indiquer que c'est la fin du temps imparti.

game Lite

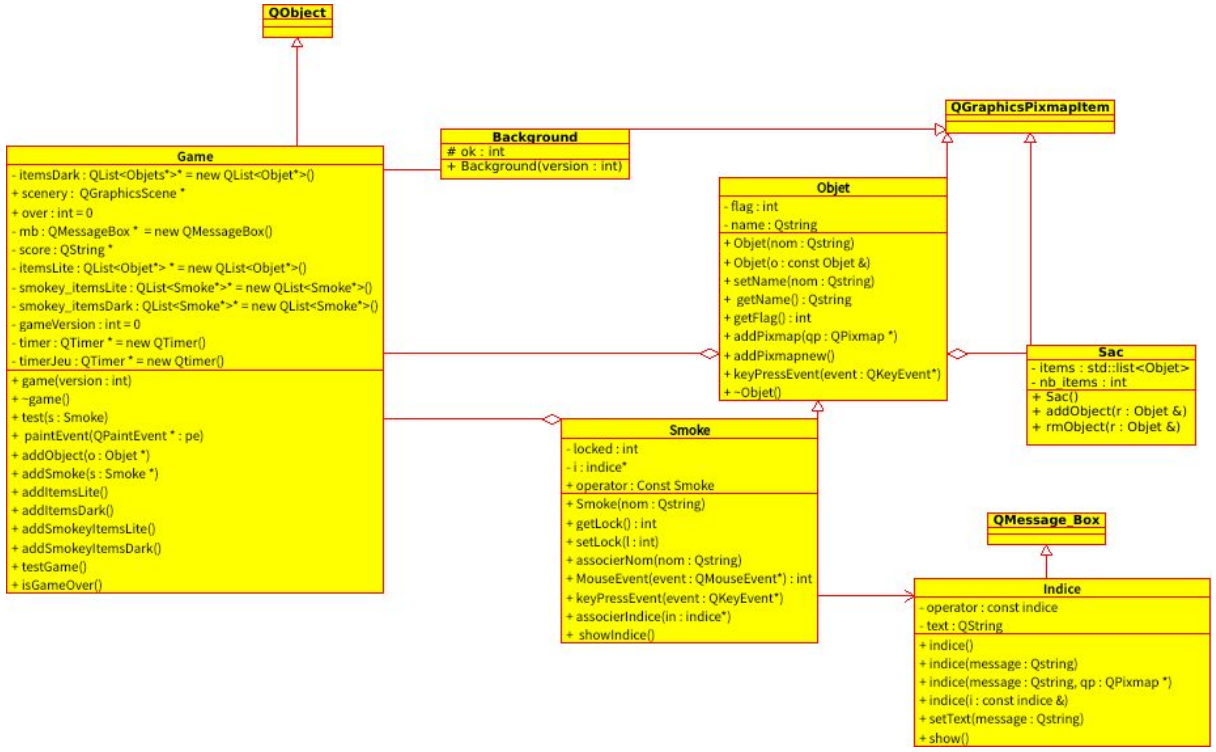
- Une classe indice, qui est un boîte de dialogue spéciale reliable à un objet Smoke pour obtenir des informations sur l'objet. Elle peut aussi prendre en argument des images png pour les afficher dans la boîte de dialogue et ainsi apporter une information supplémentaire à l'utilisateur sur l'objet Smoke désigné



indice relié à l'objet de type Smoke Cigarette

2) Description détaillée du diagramme UML

Voici le diagramme UML relié au projet :



Le diagramme UML ci-dessus contient des classes contenues dans les librairies de QT utilisées pour l'affichage graphique. Nous les avons incluses pour illustrer synthétiquement en un diagramme la nature et le comportement de chaque classe que nous avons créées. C'est pourquoi elles ne sont pas remplies. Elles ont aussi été considérées pour la contrainte des 3 niveaux minimums de hiérarchie.

Pour bien comprendre la modélisation en mode programmation objet de notre jeu, nous allons détailler ci-dessous deux classes clés de notre Jeu.

La classe Game

On peut voir que la classe Game est celle qui est reliée à presque toutes les classes. C'est une classe fille de QObject, qui permet d'utiliser des Qtimer. Elle possède des attributs qui valent la peine d'être expliqués en détail.

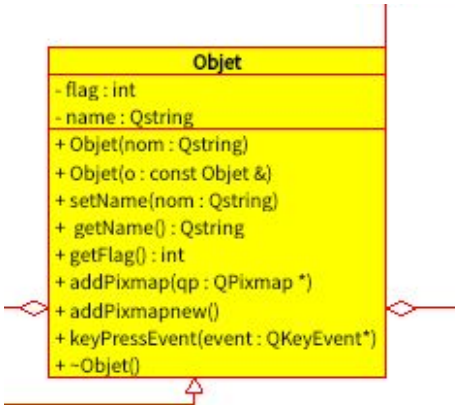
Les pointeurs sur QList<Objects*> *itemsDark* et *itemsLite* sont les listes d'Objets qui vont être affichés dans les fenêtres de jeu, respectivement pour la version Dark et pour la version Lite de game. Les pointeurs sur QList<Smoke*> *smokey_itemsDark* et *smokey_itemsLite* sont les listes de

Smoke qui vont être affichés dans les fenêtres de jeu, respectivement pour la version Dark et pour la version Lite de game. C'est sur cette liste que va être effectuée la vérification *isGameOver().*
Le pointeur sur Qtimer, *timer* est le timer affecté à l'appel périodique de *isGameOver()*.

Le pointeur sur Qtimer *timerJeu* est le timer de jeu de 3 minutes, lancé à la création d'une game.

Les méthodes de Game sont aussi très spécifiques:

- Le constructeur game(int version) est celle qui est appelée dans le main en fonction de la sélection de l'utilisateur. La version de jeu est Lite si version == 0 et Dark si version == 1.
- La fonction addSmoke(Smoke* s) permet d'ajouter un objet de type Smoke à la liste smokey_itemsLite ou smokey_itemsDark, et d'associer un pixmap à cet objet en fonction des ressources (voir la classe objet et sa méthode addPixmapnew())
- La fonction addObject(Objet * o) fait la même chose que addSmoke avec des Objets
- La fonction addSmokeyItemsLite() permet de créer tous les Smoke associés à la version de jeu Lite, et de leur créer puis associer des indices. Elle fait appel à addSmoke(Smoke *s).
- La fonction addSmokeyItemsDark a le même comportement pour une version de jeu Dark.
- La fonction testGame() permet de faire le lien entre le timer et la fonction isGameOver(). En somme, elle permet de faire en sorte qu'elle soit appelée périodiquement



La classe Objet

La classe Objet possède deux attributs importants: *flag* qui se met à 1 dès que l'objet est dans le sac et *name* qui contient le nom de l'objet. L'attribut name permet de comparer deux objets entre eux grâce à leur nom, et d'associer une image à l'objet dans la méthode addPixmapnew() qui va chercher dans les ressources le Pixmap png du même nom.

La méthode keyPressEvent() est une méthode de QGraphicsPixmapEvent dont hérite Objet. Elle est ici réimplémentée afin de tester les touches du claviers qui sont enclenchées lorsque l'objet est sélectionné. C'est

ainsi, par exemple, que nous pouvons faire bouger un objet une fois que les flèches sont enclenchées.

3) Les parties dont nous sommes les plus fières

Le lien effectif entre toutes les classes et l'implémentation de méthodes telles que addPixmapnew() dans la classe Objet qui permet de récupérer le nom de l'objet puis de le lier avec l'image contenue dans les ressources portant le même nom, automatiquement était une fierté d'implémentation.

Le design de l'application et l'ajout de fonctionnalités supplémentaires (un timer de jeu, un score qui s'affiche, une sélection à faire au début du jeu), tant de choses qui ne tenaient qu'à nous d'imaginer, constituent aussi une fierté.

4) Les contraintes implémentées

Pour pouvoir utiliser facilement certaines affectations comme les opérateurs = ou + avec des objets que nous avons créés, nous avons dû redéfinir certains opérateurs. Par exemple, l'opérateur d'affectation