Rapport de projet C++

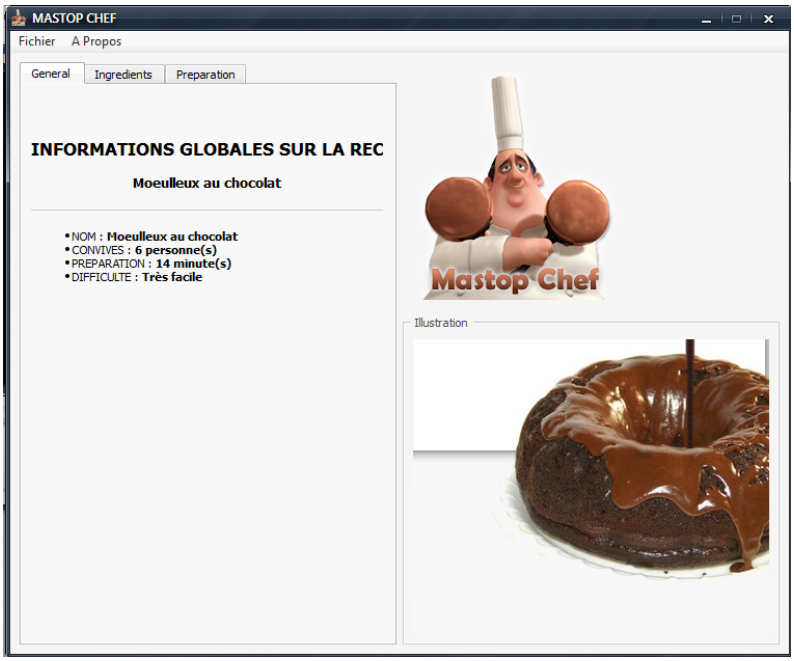
MASTOP CHEF

EFREI
École d'ingénieur
Paris Sud

Sommaire

Introduction

Le projet de C++ de cette année à l'EFREI consistait en un programme d'édition de recettes dynamiques. Le principal objectif de ce projet était la mise en pratique des notions avancées de C++ vues en cours. Nous avons vraiment essayé de coller le plus possible aux consignes du projet, bien que celles-ci s'avèrent parfois hasardeuses. Nous pensons que les enseignants EFREI n'ont pas établis ces consignes par hasard et que ceux-ci s'attendaient donc à plusieurs prises de décision de notre part : ce que nous avons fait !



Structure du projet

Le projet n'est pas très structuré à proprement parlé mais les consignes ont été respectées. L'interface graphique (IHM ou GUI) est mise d'un côté et le traitement des flux et des informations données par l'utilisateur est placé de l'autre. De ce fait, on évite les conflits entre les différents traitements de l'application et le code est donc plus facilement lisible.

L'application finale est fournie en une archive composée de deux dossiers « Exécutables » contenant le programme Windows (portable sur Linux également) ainsi que le dossier « Sources » contenant les codes sources du projet (.cpp et .h).

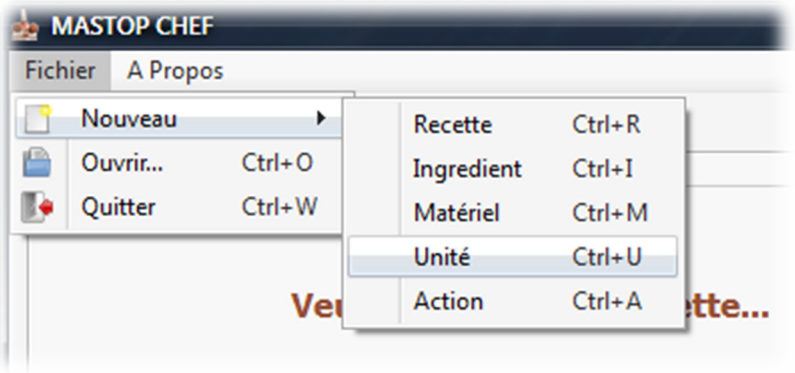
Dernière remarque : le projet utilise Qt (prononcé « Cute ») en version 4.4 (dernière version en release il y a 2 mois). L'application se sert d'un dossier pictures (contenant les images nécessaire à certains affichages de l'IHM car nous n'avons pas utilisé QResources pour ce projet) et un dossier res qui contient tous les fichiers de sauvegardes nécessaires au fonctionnement de l'application (fichiers de recettes, de matériels, d'unités…).

Point très important : un point que nous avons longtemps souligné lors de notre soutenance, c'est que l'intégralité de l'interface graphique a été développé « à la main » sans l'aide de logiciels « drag-n-drop » tels que Qt Creator / Qt Designer. Cela nous a pris beaucoup plus de temps mais, au final, il est évident que notre version est plus optimisée que celles générées par ce genre de générateurs d'interface (ils ont tendance à générer du code en trop ainsi que des fichiers ressources en sus du projet…).

Voici la capture d'écran de l'écran de base de l'application finale :

Raccourcis

Pour une utilisation plus simple, nous avons mis des raccourcis. Ils ont pour objectif un gain de temps non négligeable pour l'utilisateur en cas de forte utilisation du logiciel. Il suffit de taper [Ctrl] + [Initiale de l'action à réaliser] : on aura donc les lettres R pour Recette, I pour Ingrédient, M pour Matériel, U pour Unité ou A pour Action. On retrouve les raccourcis de base comme [Ctrl+O] pour ouvrir un fichier recette ou encore W pour fermer le programme. Ils sont indiqués lors de la sélection de nouveaux éléments :



Affichage : Utilisation Qt

L'utilisation de Qt n'est vraiment pas très compliquée et c'est d'ailleurs pour cette raison que nous avons décidé de coder notre IHM « à la main » sans générateur. En plus de se faire une expérience de base dans la programmation graphique avec Qt, nous nous entraînions au développement C++ professionnel (Qt est utilisé par de nombreux éditeurs de progiciels).

La classe qui a été créée pour gérer l'interface utilisateur est MainFrame. Cette classe fait le lien entre les fenêtres (principales ou modales) et les données (les modèles de données nous dirons ici).

De plus, le sujet nous conseillait vivement d'utiliser la structure MVC (Modèle – Vue – Contrôleur) disponible dans Qt. Nous ne l'avons pas fait, pourquoi ?
Dans un premier temps, nous avons évidemment essayé de nous atteler à ce concept (que, personnellement, je connais bien mais pour la programmation Web en PHP via des Frameworks tels que Symfony ou encore avec la sous couche de Zend) mais nous n'avons pas vraiment réussi.

Le problème venait essentiellement de l'entretien des valeurs d'un modèle une fois la vue posée à l'utilisateur (comment modifier cette vue après… il fallait passer par des adapteurs assez spéciaux et donc la tâche s'avérait délicate). Pour un petit programme de 2500 lignes comme celui-ci, nous nous sommes vite dit que le temps d'implémentation était plus important.

Stockage des données

Le programme Mastop Chef a très souvent besoin de stocker des données sur l'ordinateur hôte. Ces stockages interviennent lorsque l'utilisateur décide d'ajouter un nouvel ingrédient ou encore lorsqu'il enregistre sa recette. Nous avons hésité entre trois solutions :

- Le stockage par base de données (BDD). Nous avons fouillé du côté de MySQL, SQLite ou encore Acces mais l'ennui était toujours le même… Comment « installer » le gestionnaire d'accès. La tâche devenait compliquée. Dommage, nous n'avons pas retenu cette manière de stocker mais elle nous aurait été très utile au niveau de la fonction de recherche du logiciel (en effet, une simple requête SQL sur la vue des recettes aurait permis de rechercher très rapidement et de n'afficher que les résultats voulus en un rien de temps).

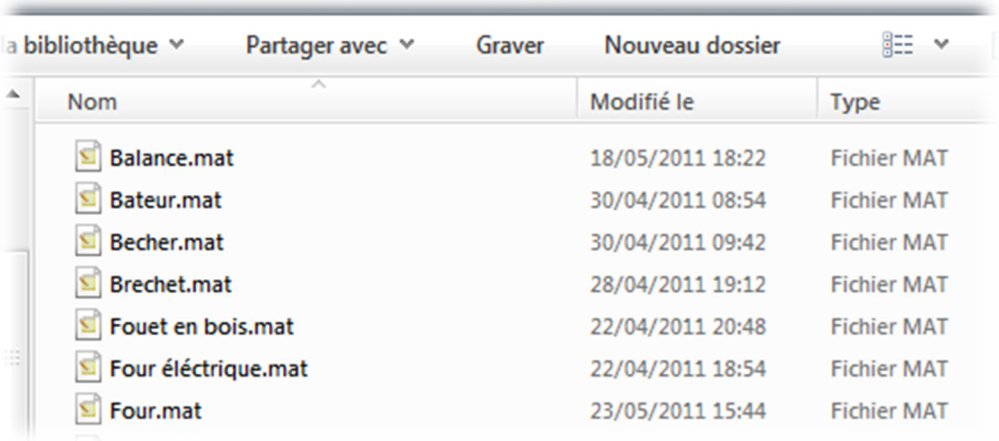
- Le stockage en XML. Le concept du XML est très pratique (j'en fais beaucoup en PHP d'ailleurs) mais m'a posé beaucoup de difficulté ici. Pourquoi ? Il s'agissait simplement de problème de compilation (le compilateur ne voyait pas que j'avais ajouté le « += Xml au fichier du projet .pro). Encore une fois, c'est dommage car ce système de stockage s'avère très rentable au niveau des ressources utilisées.

- Le stockage en fichier texte. C'est cette solution, plus que basique que nous avons retenue finalement. Il s'avère au final qu'elle est adaptée à la taille de notre application. Cette une technique très rapide et surtout très flexible au niveau du développement.

A noter que nous avons voulu passer par la sérialisation des classes afin d'enregistrer nos données directement (pas besoin de traiter et de mettre en forme les données puisque ici c'est une classe entière que l'on conserve dans un fichier). Cependant, encore un fois c'est une méthode plus que lourde à mettre en place et le manque de temps ne nous a pas permis d'aller plus loin dans cette voix.
J'ai été surpris de voir la complexité de mise en place alors que le code en PHP pour serialiser une classe est :

```
$objet = new Class ;
$data = serialize($objet);
```

Concernant la structure du stockage, hors mis pour le stockage des différentes unités, nous avons décidé de stocker les ingrédients, les recettes et les outils dans des fichiers séparés : cela revenait au plus simple pour nous (une boucle foreach offerte par Qt 4 et le tour est joué pour le parcours de fichier dans un répertoire). Tous ces fichiers sont classés et ordonnés dans le des répertoires du dossier res.



Dans chaque fichier, les données sont stockées en brutes. Pour lire les données et les formater correctement, on passe alors par des méthodes issues de QString. Ce sont des méthodes de parsage de chaînes de caractères comme join ou encore split qu'on applique sur des motifs de scission.
Voilà tout ce que nous pouvons dire sur le stockage des données par l'application.

Exemple d'utilisation : La fenêtre « Recette »

L'affichage est généré de la façon suivante, il y a différents types d'affichage à générer, nous allons commencer de haut en bas, chaque affichage du programme est créé avec les mêmes fonctions (à peu de chose près). Nous allons étudier le cas d' l'ajout de recette.

Tout d'abord il nous a fallu créer la fenêtre en elle-même grâce à la méthode **QDialog()**, la définir comme Modal : **addRecipeDialog->setModal(true);** c'est-à-dire que lorsqu'elle est ouverte on ne peut pas communiquer avec la fenêtre principal. La description se fera de haut en bas :

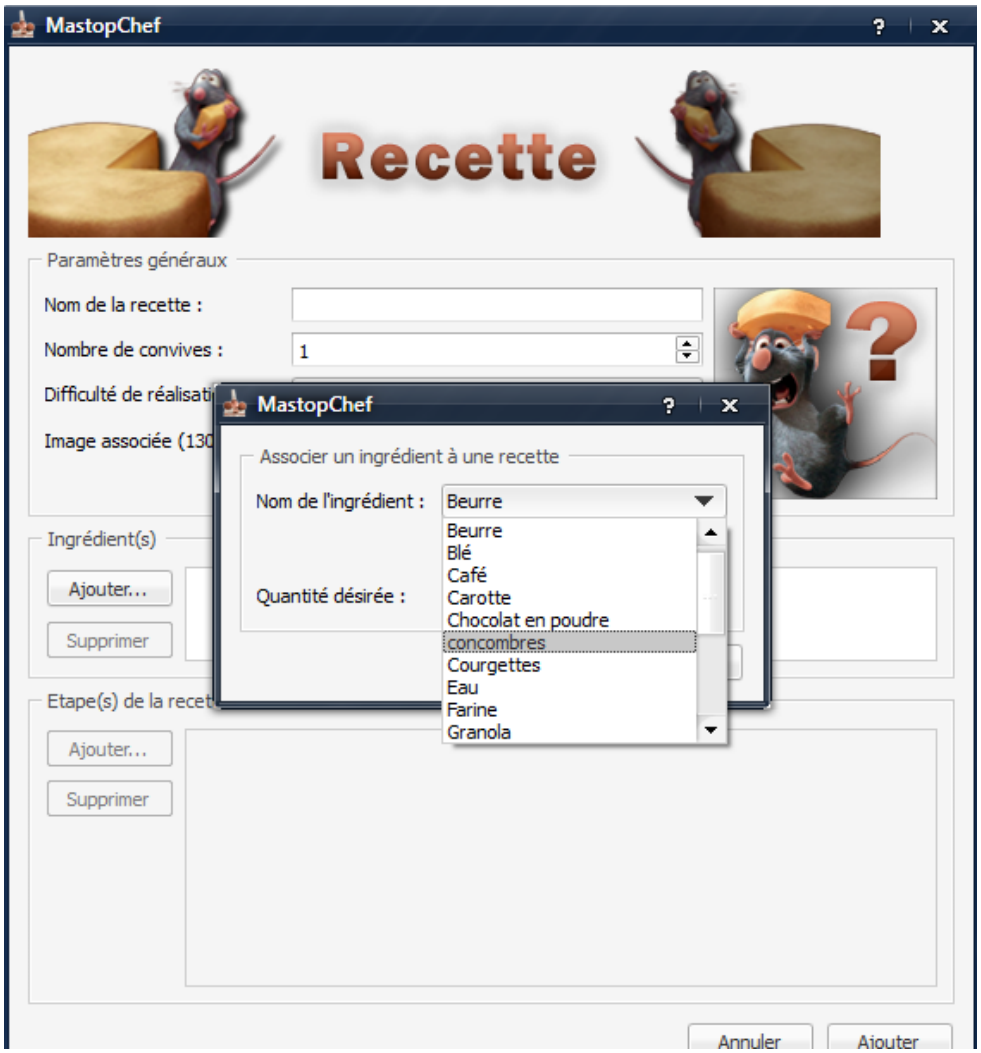
Tout d'abord, l'image générée : on créer un nouveau label à qui on attribuera une image, en lui donnant le cheminement de celle-ci via la fonction **sepPixmap(QPixmap(« picture/etc.. »))**. L'ajout du composant se faisant via la méthode **addWidget()**. Le layout est ensuite placée vers le haut via **setAlignment(Qt::AlignTop)**.

Ensuite la création d'un groupe grâce à **QGroupBox()** à qui on attribue le titre « paramètres généraux », puis celle-ci est rajouté avec la même fonction cité précédent.
Dans ce groupe de composants, on rajoute, après avoir assigné un layout, tout son contenant à savoir « nom de la recette » etc. Pour les créer, on utilise **addItem()** qui permet d'afficher le texte demandé ainsi que son espace utilisateur pour la saisi variant selon le type de données à saisir, pour le nom de la recette il s'agit **QLineEdit** permettant à l'utilisateur de saisir le nom qu'il veut.

Concernant le nombre de convives on laisse l'utilisateur choisir un chiffre compris entre 1 et 99 personnes grâce à **QSpinBox()**, la définition du domaine étant réalisé à l'aide de **setMinimum()** et **setMAximum()**. La difficulté de réalisation est défini avec un menu déroulant ou l'utilisateur peut choisir son niveau de difficulté : « très facile », « plutôt facile », etc grâce à **QcomboBox().** Pour associer une image à la recette, on rajoute un bouton permettant d'aller chercher directement l'image, ce bouton est généré avec **QPushButton()**. Cette méthode de la classe MainForm se chargera ensuite de copier l'image sélectionnée directement dans le repertoire res/recipes/pictures de l'application.

La sous partie « Ingrédient(s) est créer de la même manière, les bouton sont cette fois placés sous forme de coordonnées avec un **QGridLayout()**. Les éléments sont ensuite blittés avec la fonction **addWidget()** dans laquelle on donne les informations concernant la position du bouton : 0,0 pour le « ajouter… », 1,0 pour « Supprimer ». L'ordre est ainsi : colonne, ligne, rowspan, colspan.

La case pour le listing des éléments est générée avec un **QListWidget()**, puis on rajoute les coordonnées nécessaires avec addWidget, à savoir sa position, comme pour les bouton précédents, mais aussi sa taille : 4,1 pour celle-ci.



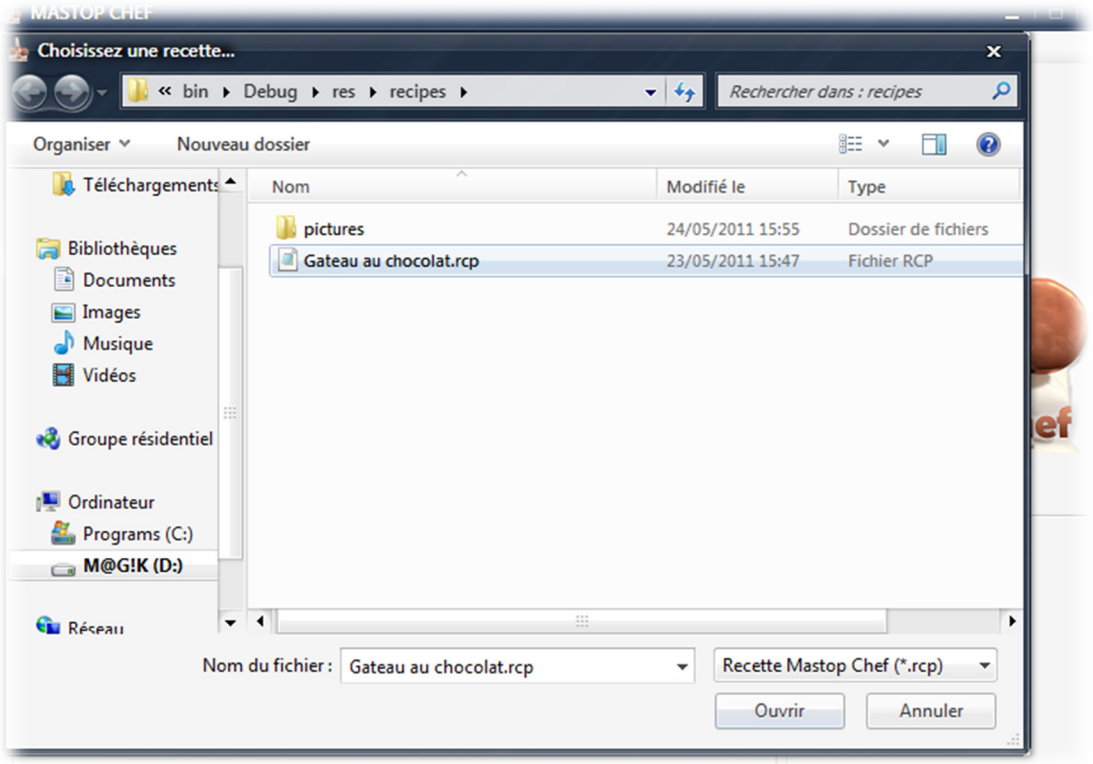
La gestion des événements

La gestion des événements est gérer à partir de la méthode Qt connect().

Voici les différents champs à remplir :

```
connect(type_de_bouton, SIGNAL (type de signal()), variable, SLOT(action_a_effectuer())) ;
```

Lecture d'une recette



La fenêtre d'ouverture s'affiche via le menu > ouvrir (ou encore Ctrl+O). Cette boîte de dialogue permet de sélectionner une recette depuis le répertoire des recettes (recipes/). Nous avons décidé de ne pas charger toutes les recettes au démarrage. Il s'agit de bon sens ; si il y a beaucoup de données, ce chargement n'améliorera pas les performances de l'application. Les fichiers .rcp (recipe) sont les seuls sélectionnables.

Conclusion

Nos bases en langage C++ sont donc confirmées je pense. Nous avons enfin pu réellement nous tourner vers le développement modulaire et surtout la conception orientée objet. La suite, qu'est-ce que c'est ? Et bien selon le directeur des études de l'EFREI, la suite c'est l'apprentissage du Java : un langage totalement conçu en POO qui va très certainement nous donner du fil à retordre.

Nous avons été heureux de coder ce projet qui, au final, s'avère plutôt bien réussi je trouve. Il ne nous manquait que quelques fonctions supplémentaires pour répondre parfaitement au cahier des charges du projet. Nous sommes donc contents de vous présenter notre version en mode graphique du projet des Master Chefs !

mec dis aussi qu'on a pas respecté le modèle mvc car, selon nous, c'est un modele très pratique mais seulement pr les projets de grande envergure, pr le notre tu peux dire que le code était encore lisible donc ça passe

[23/05/2011 20:59:12] Christophe de Batz: tu peux dire aussi qu'on a essayé mais que la modification interne des données dans les composants graphiques s'avèrait très compliqué (il faut repasser par l'interface du modèle avant de pouvoir modifier les données ...)

[23/05/2011 20:59:33] Christophe de Batz: enfin une dernière chose, on a utilisé des fichiers textes et non xml

[23/05/2011 21:00:06] Christophe de Batz: car pr le xml j'ai essayé, en vain... lol nan mais tu peux dire que le xml s'avérait un peu complexe à mettre en place par rapport au fichier texte

[23/05/2011 21:00:23] Christophe de Batz: et trop "encombrant" pour le peu de donnée que va brasser l'application

[23/05/2011 21:00:38] Christophe de Batz: une dernière dernière chose

[23/05/2011 21:01:11] Christophe de Batz: on avait eu l'idée de serialiser les données des classes pour les enregistrées dans des fichiers et enfin les desérialiser pour récupérer les données

[23/05/2011 21:01:38] Christophe de Batz: mais les classes d eserialisations avec Qt sont vraiment pas simple d'utilisation et par manque de temps, la solution a également été abandonnée, mm si on a pas mal cherché de ressource dessus

[23/05/2011 21:01:42] Christophe de Batz: ça va ? oO

[23/05/2011 21:02:04] BERTRANB: tkt je vais te resaucissonner ca

[23/05/2011 21:02:07] BERTRANB: ca va le faire

[23/05/2011 21:05:22] Christophe de Batz: mvc (modèle - vue - contrôleur) => c'est une structuration de projet très atypique permettant de séparer les données, leur traitement et l'interface utilisateur (ihm ou gui comme tu veux!) après c'est un concept très puissant, apparue grace à la POO (prog orienté objet avec les classes et tout et tout). tu rajouteras aussi que QT ne propose qu'un modèle de base MV (modele vue) il n'y a pas de "côté" controlleur dans QT, il est a géré dans la vue directement !

[23/05/2011 21:07:08] BERTRANB: kk

[23/05/2011 21:11:09] Christophe de Batz: je pense que t'as assez de truc pr une saucisse de 3 pages là

[23/05/2011 21:12:01] BERTRANB: lool

[23/05/2011 21:12:04] BERTRANB: ouai ya moyen =)

[23/05/2011 21:56:28] Christophe de Batz: tu t'en sors ma grosse ?

[23/05/2011 21:56:37] Christophe de Batz: tu veux d'autres pasteques pr ta soupe ?

[23/05/2011 21:58:01] BERTRANB: j'ai pas encore commencer ..

[23/05/2011 21:58:09] BERTRANB: impssible de bosser l'UML -_-

[23/05/2011 22:09:11] Christophe de Batz: pareil

[23/05/2011 22:09:34] BERTRANB: le cours sert a rien j'ai limpréssion non?

[23/05/2011 22:09:43] Christophe de Batz: bah c pas super koi

[23/05/2011 22:09:45] Christophe de Batz: mais bon

[23/05/2011 22:10:01] BERTRANB: j'ai retiré aucune information pr le moment xD