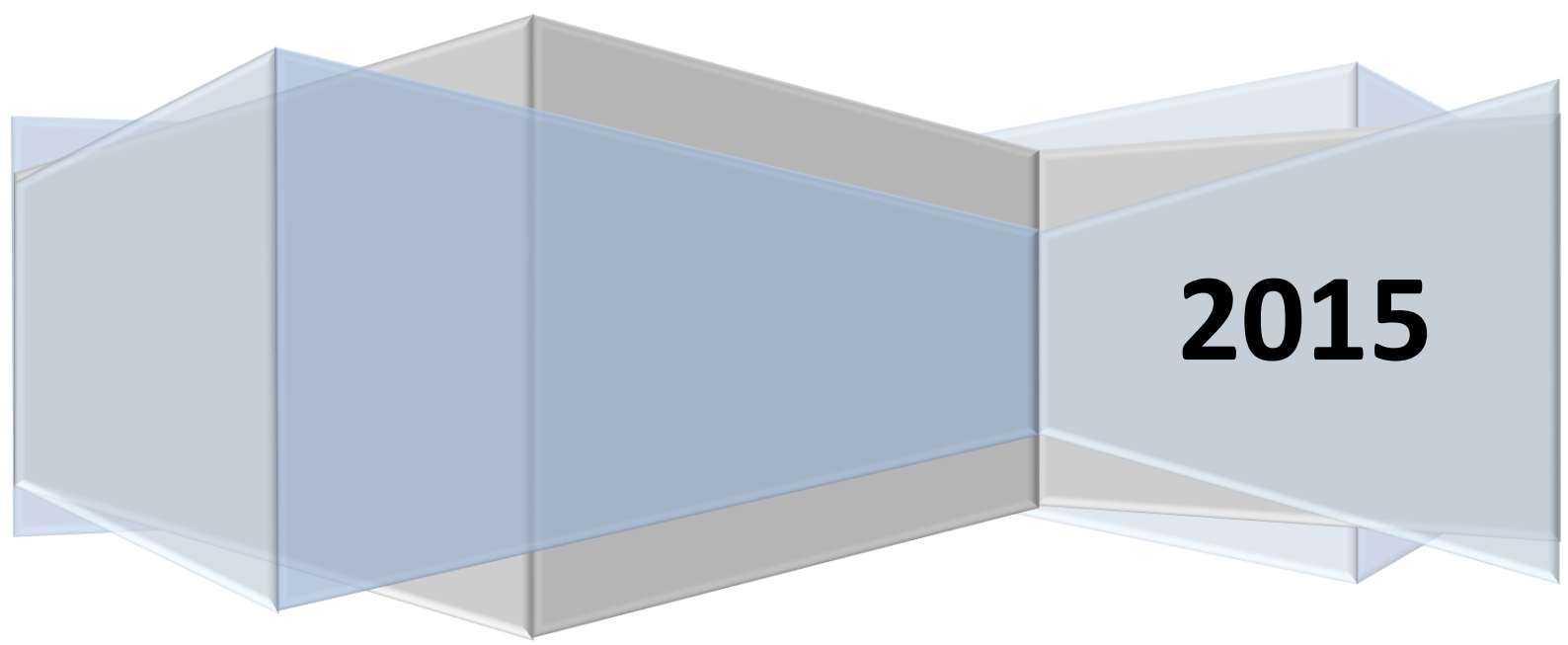


Aide à l'optimisation de planning dans une grande école

Projet C++

Dossier de Conception

FABRE Maxime
LEPOT Florian



2015

Sommaire

Introduction.....	2
I. Présentation des fonctionnalités	3
II. Architecture et fonctionnement du projet.....	4
A. Architecture du projet	4
1. Traitement de données	4
2. Affichage	4
3. Diagramme des classes.....	5
B. Fonctionnement détaillé	6
1. Fenêtre principale	6
2. Gestion des données	7
3. Création de cours	9
4. Gestion des conflits	9
Conclusion	10
Annexe	0

Introduction

Pour ce premier projet en langage C++, nous devons développer un gestionnaire d'emploi du temps respectant divers critères fournis dans le cahier des charges. Que ce soit structurel ou fonctionnels. Ce programme ne nécessitait aucune intégration graphique, cependant nous avons jugé plus simple et plus intuitif pour l'utilisateur de le faire en fenêtre. Nous avons donc utilisé Qt tout au long de ce projet.

Ce projet nous a beaucoup appris sur la manière à réfléchir et visualiser l'algorithme d'un programme avant de se lancer dans le code, ceci dans le but d'éviter d'obtenir un programme trop lourd, non optimisé, et donc, lent. L'enjeu était bien évidemment de se familiariser avec la POO et d'apprendre à structurer un programme de la sorte.

I. Présentation des fonctionnalités

Nous avons voulu intégrer dans ce projet de C++ un maximum de fonctionnalités afin de faciliter la tâche de l'utilisateur. Nous avons voulu également nous rapprocher un maximum de ce qu'un utilisateur pourrait désirer comme fonctionnalités pour la création d'emploi du temps. Nous allons donc détailler les fonctionnalités du programme rapidement, que l'on peut diviser en 5 petites parties :

- Une gestion dynamique
- Une gestion des données
- Un affichage clair
- Possibilités d'ajouter des cours ou de remplacer un cours.
- Gestion de la sauvegarde.

Gestion EdT est un gestionnaire d'emploi du temps, capable de s'adapter à n'importe quelle école. L'affichage est donc entièrement dynamique. Dès l'ouverture du programme, des cours tests sont déjà créés mais ceux-ci peuvent être modifiés à tout moment. En effet, il est facile de supprimer ou d'ajouter de nouvelles matières, salles, promos, groupes. Un menu est dédié pour toutes ces opérations.

Nous avons aussi voulu un affichage clair. Par exemple chaque bloc de cours crée dans le tableau est coloré pour distinguer les bordures. Chaque bloc ne représente évidemment qu'une cellule de tableau. Le cours, la salle et le professeur sont affichés sur chaque bloc de séance. De plus, l'emploi du temps est annuel et décomposé en jours et en semaines. La fenêtre d'affichage affiche l'emploi du temps seulement pour un jour. Il est donc possible de changer de jours et de semaines pour afficher les futurs cours.

La fonctionnalité principale du programme est évidemment la possibilité d'ajouter des cours. On peut ainsi ajouter mais aussi remplacer un cours déjà enregistré. Pour chaque ajout de cours une gestion des conflits est faite. Cela évite que plusieurs cours se retrouvent dans la même salle ou avec le même professeur par exemple. Nous avons donc la gestion des professeurs, des salles, des groupes, des promos et de l'horaire. Si la gestion trouve une erreur elle proposera alors à l'utilisateur d'écraser ou non l'ancien cours qui pose problème. Si l'utilisateur refuse de remplacer l'ancien cours l'opération sera annulée.

Pour terminer, chaque cours crée est enregistré dans un fichier de sauvegarde. Ainsi on peut quitter le programme et revenir, l'emploi du temps sera sauvegarder et charger automatiquement. Quelle que soit le jour ou la semaine. Il y a un fichier .txt de sauvegarde pour chaque jours qui enregistre la semaine, afin d'éviter des sauvegardes trop lourdes.

II. Architecture et fonctionnement du projet

A. Architecture du projet

Etant donné que nous avons utilisé Qt pour l'élaboration de ce projet, nous avons décidé de séparer tout ce qui est traitement de données et affichage. Tout ce qui concerne le traitement de données se situe dans le dossier **classes** et l'affichage dans le dossier **views**. Nous n'avons pas adopté le modèle MVC pour la structure, toutefois la séparation des deux types de fichiers nous a parue nécessaire.

1. Traitement de données

Pour le traitement de données, nous avons plusieurs classes différentes :

- **Promos** : C'est la classe qui gère les promos,
- **Groupes** : Permet de gérer les groupes,
- **Salles** : Sert pour la gestion des salles,
- **Professeurs** : Définit les différents attributs et méthodes pour les professeurs,
- **Matières** : Gère les matières,
- **Heures** : Définit les heures
- **Seances** : C'est la classe qui va gérer les séances. Elle est composée d'un objet de type promo, groupe, salle, professeur, matiere, d'une heure de début et d'une heure de fin.
- **GestionEdT** : C'est la classe qui va gérer l'ensemble de notre système. Elle est composée de toutes les données et permet de rajouter des cours. Plus de détails seront donnés dans la partie **Fonctionnement du projet**.

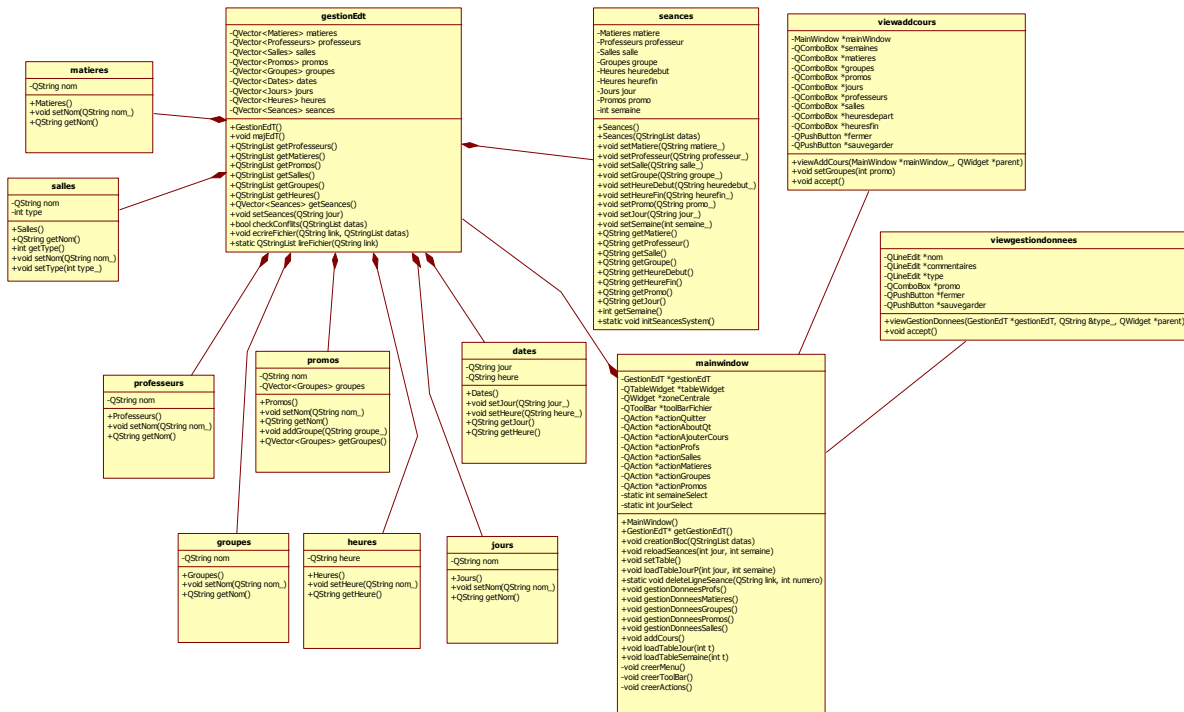
2. Affichage

L'affichage est séparé en 3 classes différentes :

- **MainWindow** : C'est la fenêtre principale sur laquelle l'utilisateur travaillera, elle est composée de différents éléments que nous verrons plus tard.
- **viewGestionDonnees** : C'est la fenêtre qui va nous servir à gérer les données (Promos, groupes, professeurs, etc..).
- **viewAddCours** : C'est par cette fenêtre que l'utilisateur va pouvoir ajouter des cours en indiquant les différents paramètres voulus.

3. Diagramme des classes

(Disponible en Annexe 1 pour que ce soit plus lisible. Egalement disponible sous format image dans le dossier doc du projet)



1. Diagramme des classes

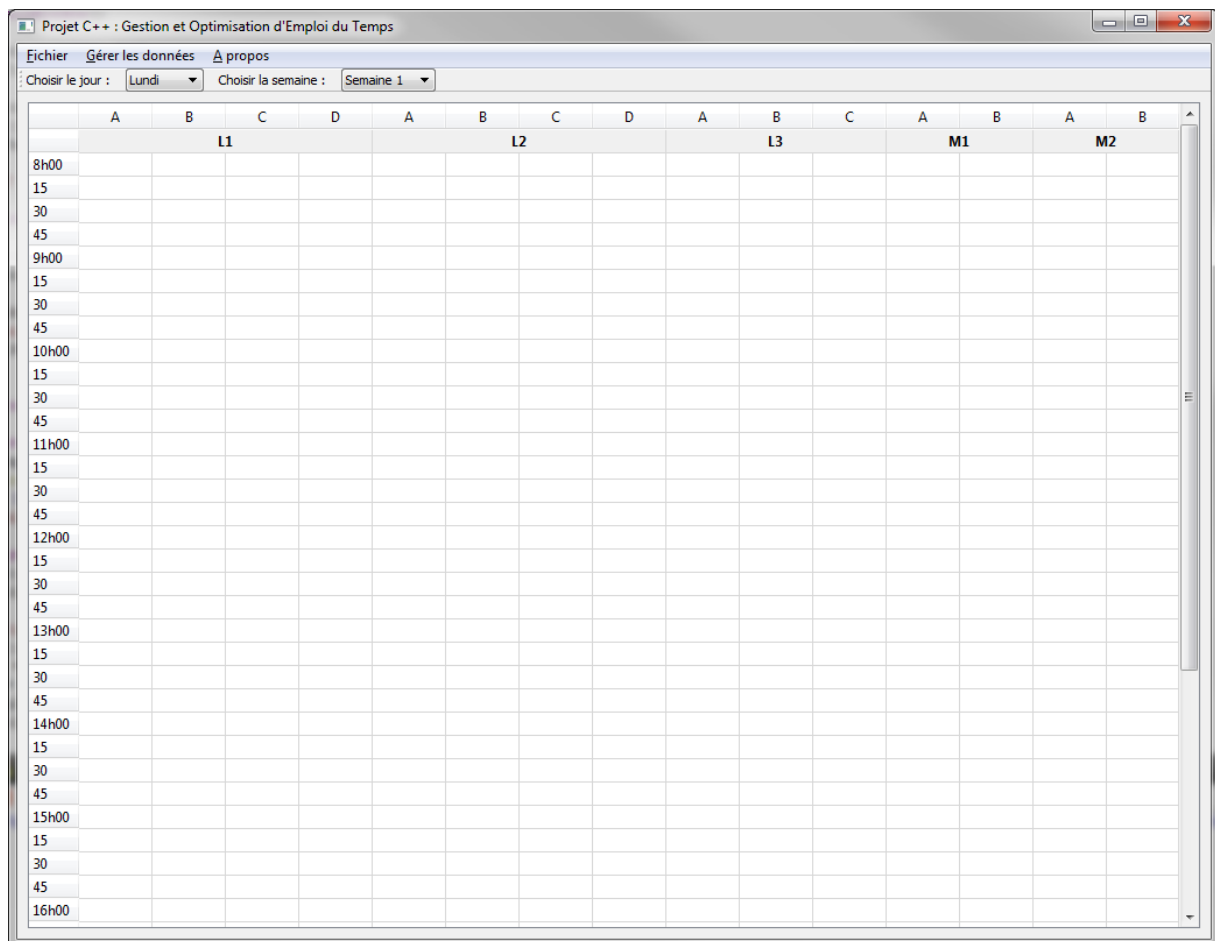
B. Fonctionnement détaillé

Nous allons étudier le fonctionnement du programme dans l'ordre d'utilisation et d'apparition. Nous commençons donc par l'affichage de la fenêtre principale.

1. Fenêtre principale

Lorsqu'on lance le programme, celui-ci commence par créer les différents éléments dont nous aurons besoin dans le programme. En effet, il convertit les données se trouvant dans les fichiers textes en objets des différentes classes, et crée un objet **gestionEdT** qui va nous servir pour toute la gestion de l'emploi du temps, et dans lequel nous aurons des tableaux contenant toutes les données.

On peut alors afficher la fenêtre principale suivante :



2. Fenêtre principale

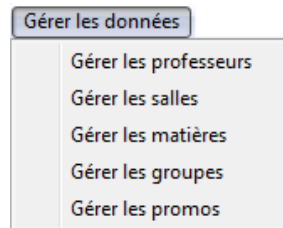
On peut voir qu'il y a différents menus. Pour commencer, dans la barre d'outils on a accès à deux menus :

- **Fichier** : Dans lequel on pourra ajouter un cours ou quitter le logiciel.
- **Gérer les données** : Dans ce menu on pourra choisir des données à modifier, que ce soit promos, groupes, matières, etc..

Il y a également une barre d'outils grâce à laquelle l'utilisateur peut changer de jours et de semaines au niveau de la vue et de l'affichage de l'emploi du temps, au moyen de menus déroulants.

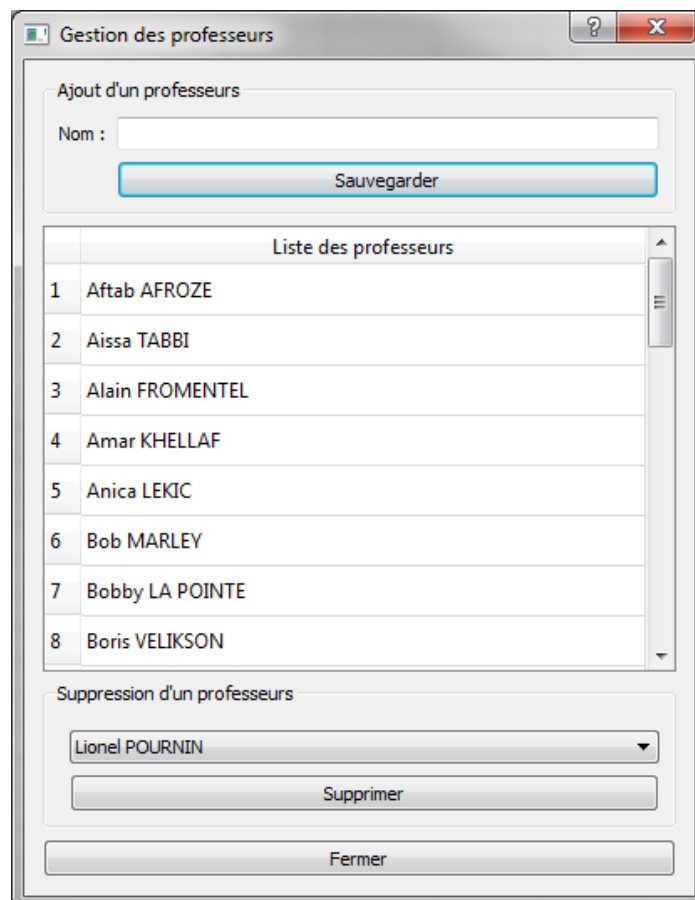
2. Gestion des données

Ajoutons maintenant des données à notre emploi du temps, en allant dans le menu « Gérer les données » :



3. Menu de Gestion de Données

Lorsqu'on va choisir un de ces menus, cela va créer un objet de type **viewGestionDonnees** et va ainsi pouvoir afficher la fenêtre qui nous permettra d'effectuer diverses actions. Le fonctionnement est le même pour toutes les données, c'est d'ailleurs la même classe qui sera utilisée à chaque fois. Nous n'allons donc étudier qu'un exemple, ici, la gestion des professeurs.



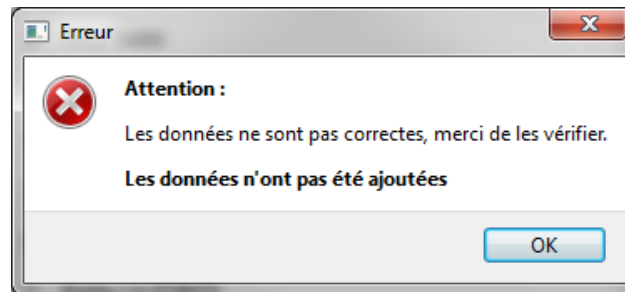
4. Gestion des professeurs

La fenêtre **viewGestionDonnees** va permettre de créer un professeur, de voir la liste des professeurs déjà existants ou d'en supprimer. Toutes ces actions seront effectuées au moyen de méthodes qui vont être appelées lors du clic sur un bouton.

- **Ajout d'un professeur**

Pour ajouter un professeur, nous n'avons besoin que d'entrer son nom. En appuyant sur « Sauvegarder », la méthode **ajouterData()** va être appelée. Elle va vérifier que le nom entré n'est pas vide, et donc qu'on peut enregistrer le nouveau professeur. L'enregistrement va se faire au moyen de **QFile** et de **QTextStream** pour l'enregistrement dans le fichier, mais nous allons aussi créer un nouvel objet professeur dans le tableau de données de **gestionEdT** (Qui est l'objet contenant toutes les données de l'emploi du temps).

La sauvegarde va renvoyer un message pour confirmer l'ajout du professeur ou en cas d'erreur :



5. Message d'erreur de la Gestion des Données

- **Liste des professeurs**

L'utilisateur a également accès à la liste des professeurs pour pouvoir voir ceux déjà existants.

- **Supprimer un professeur**

L'utilisateur peut aussi supprimer un professeur. Il lui suffit de le sélectionner dans la liste déroulante et de cliquer sur « Supprimer ». Cette action va appeler une autre méthode qui va permettre, à la fois de supprimer l'objet correspondant, mais également de le supprimer des fichiers sources.

Une fois nos données de base enregistrées, nous pouvons commencer à créer des cours.

3. Création de cours

La création de cours va commencer par instancier un objet de classe **viewAddCours**. Celui-ci va créer une fenêtre dans laquelle l'utilisateur va pouvoir choisir les différents paramètres de création pour son cours.

Lorsque l'utilisateur va cliquer sur « Sauvegarder » pour ajouter le cours, une nouvelle méthode va être appelée et va effectuer différentes actions :

- Création d'un objet de type **Seance** qui va être enregistré, encore une fois, dans un tableau de **gestionEdT**,
- Enregistrement du cours dans le fichier source,
- Affichage du cours dans le tableau de la fenêtre principale pour le visualiser.

Toutefois, l'ajout d'un cours peut parfois créer des conflits, que ce soit au niveau des salles, des groupes, etc.. Et une méthode est appelée juste avant l'enregistrement pour gérer les conflits.

4. Gestion des conflits

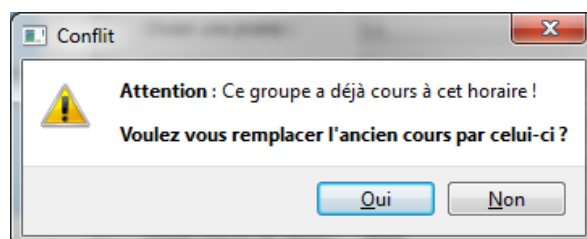
En tout premier lieu, notre programme gère et réagit devant les conflits suivants :

- Un groupe a deux cours au même moment,
- Un professeur a deux cours au même moment,
- La salle est déjà utilisée.

La gestion des conflits se fait dans la méthode **checkConflits** de la classe **gestionEdT** et son fonctionnement est simple.

Comme toutes les séances sont enregistrées dans des entrées d'un **QVector** de type **Seances**, dans l'objet **gestionEdT**, nous parcourons toutes les séances existantes et nous vérifions si, à une même heure et pour le même groupe, il y a des données déjà existantes.

Si oui, le système propose à l'utilisateur d'effectuer un arbitrage. Il pourra choisir de remplacer le cours existant déjà par le nouveau ou décider de garder l'ancien, par le biais d'une boîte de dialogue :



Bien évidemment la boîte de dialogue évolue en fonction du conflit.

Conclusion

Pour finir sur ce projet, nous avons essayé de respecter au maximum le cahier des charges, sans pouvoir apporter d'autres améliorations que l'interface graphique car nous ne disposons pas de suffisamment de temps pour cela.

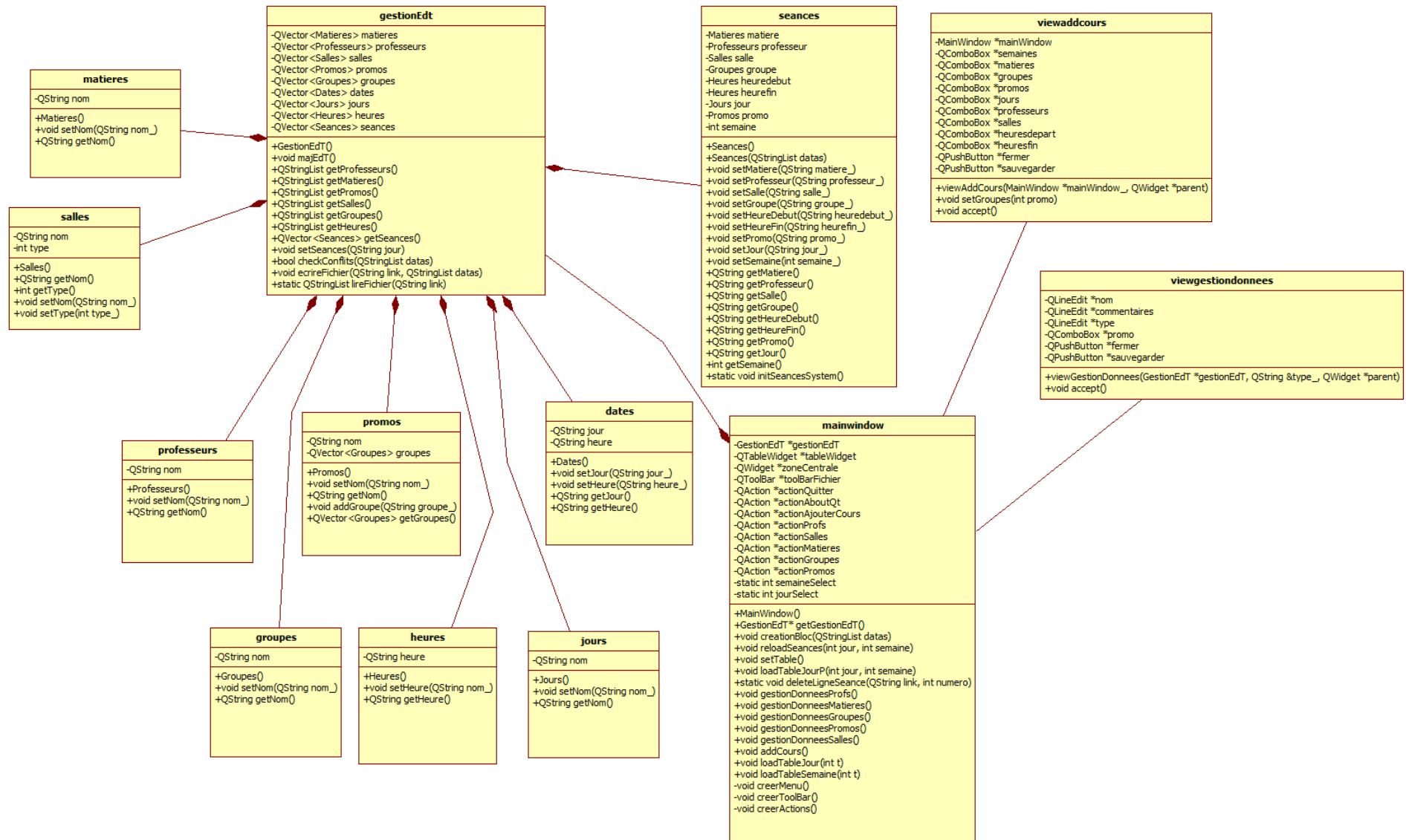
Il y a plusieurs améliorations que nous aurions aimé mettre en place avec plus de temps :

- Gérer les jours fériés, vacances et disponibilités des professeurs,
- Placement automatique des cours en fonction du volume horaire de la matière,
- Ne permettre qu'à certains professeurs d'enseigner telle ou telle matière,
- Avoir des vues différentes en fonction de l'utilisateur : Professeur, Administrateur, Eleve, etc..

Nous avons aussi beaucoup appris quant à l'utilisation du langage C++ et de la POO en général, et cela nous a montré que ce langage avait d'énormes possibilités.

Ce projet nous a également permis d'apprendre à utiliser Qt qui simplifie beaucoup d'éléments et qui nous a permis d'effectuer un affichage agréable et facile d'utiliser.

Annexe



6. Diagramme des Classes