Projet IN505 - Compte rendu

Compte rendu du projet d'IN505 de l'UVSQ

Auteurs :

- **DANKOU Mathis**
- **SOURSOU Adrien**

Informations techniques

Structure du projet

```
.
├── data
├── doc
├── inc
├── mainwindow.ui
├── Makefile
├── README.md
├── sfsca.pro
├── sfsca.pro.user
└── src
```

Les fichiers sources sont situés dans src, les headers dans inc, les données utilisateurs dans data et la documentation relative au projet dans doc.

GUI

Nous avons utilisé le framework QT ainsi que le QT Creator pour créer rapidement et de manière "visuelle" et plus instinctive une interface utilisateur.

Notre interface est composé de plusieurs éléments donc :

- **Qapplication**

Conteneur principale et unique de notre application

- **Qmainwindows**

Représente la fenêtre principale et unique utilisée par la Qapplication et est composée de notre SfscaScene, d'un Login et d'un fichier mainwindows.ui; ce fichier contient la mise en page de l'application ainsi que quelque comportements très basiques.

- **SfscaScene**

Nous permets de gérer le circulation du carburant et les modifications des paramètre de simulation, elle est affiché via un QGraphicView généré via le fichier ui.
Elle dérive de la classe QGraphicScene et est composé des Items graphiques à afficher.

- **Items Graphiques**

Utilisés pour l'affichage, ils sont chacun lié via pointeur à une différente partie du système (tank, valve ou motor), ils permettent également de gérer certaines actions si on clique dessus.

Signaux et Slots

Les signaux sont une manières d'informer un changement depuis n'importe lequel des objets QT (ou QT_Object).

Ces signaux peuvent lier à une fonction spécifique, appelée Slots via la fonction "connecte" ainsi que, dès le signal est capté, l'objet possédant le slot l'exécute.

Certaines fonctions tels que ceux qui gère les boutons de la fenêtre se reposent sur ce principe fondamentale de QT.

```
//exemple d'appel à connect
connect(ptr_sender, SIGNAL(event()), ptr_receiver, SLOT(action()));
//exemple d'appel à disconnect
disconnect(ptr_sender, SIGNAL(event()), ptr_receiver, SLOT(action()));
```

Simulation du système de carburant

Le système est un graphe dont les différentes Part sont les nœuds.
Tous les éléments utilisés faisant partie du système héritent d'une classe abstraites Part.

La classe Part possède un nom, un vecteur de pointeur sur Part et un mécanisme de marquage pour effectuer le parcours sans incidence, en particulier les problème de circuit.

La fonction supplyFuel est virtuelle absolue, et est implémenté par les classes filles.

Parcours

Le parcours consiste principalement en un transfert de carburant de voisins en voisins, jusqu'au moment où il n'y a plus de voisins vers qui transférer le carburant.

Le transfert se fait via la fonction SupplyFuel qui si les conditions internes le permettent marque tous les voisins auxquels il a accès.

Le système, après chaque parcours lancé à partir d'un Tank, l'ajoute aux voisins du Tank si celui-ci est marqué après passage.

Système d'authentification

Les données des utilisateurs sont stockées dans des fichiers individuels. Le système d'authentification vise à protéger leurs données.

Le chemin d'accès aux fichiers des utilisateurs est dans le dossier data.

Par conséquent, les mots de passe ne sont jamais écrits en clair dans les fichiers, afin d'éviter l'usurpation d'identité.

On a choisi d'utiliser le SHA-512 comme fonction de hachage, dont on a récupéré une implémention en C++.

Format de fichier utilisateur

Chaque utilisateur possède un fichier *[username].dat* où *[username]* est remplacé par le nom de l'utilisateur.

```
[sha512(username + password)] # hash d'identification
[sha512(hash + sha512(historique))] # empreinte de l'historique
[timestamp],[grade] # une entrée de l'historique (date + note)
# ...
# 0 à n entrées possibles dans l'historique
```

Lors de la création d'un utilisateur, l'historique est vide. Le format à donc cette forme :

```
[sha512 hash]
[sha512 foothprint]
```

Exemple de fichier :

Contenu de *user.dat* :

```
d6bfd9699e316894027d7dcc3a52a9c385031b0e822d8faa3fc856fe677962a2843825e1818d72fe
32ee6bd02a4e065cbdbbb5a318dc02db873d6e8a5a0c0934
c62017b4193a376ab834e6c0c65ff6c12a630dd19663a31daf2ad4cecc1ca310c2a967f37ccad58d
a28ac8d25ef3b9aa0a98fc48a99c088b5b5977fcd2e14884
1578206720,5.27
1578206728,9.69
1578206746,2.42
```

Les notes ne sont pas chiffrées, mais elles ne sont pas modifiables.
Toute altération malveillante du fichier est détectée par le système d'authentification. Si un utilisateur modifie une note ou supprime une ligne, le fichier de l'utilisateur est considéré comme corrompu.

L'empreinte de l'historique (la deuxième ligne) permet d'éviter la triche.
Cette sécurité n'est pas infaillible, cependant elle est suffisament performante pour le projet.

Le hash de l'historique de l'utilisateur est modifié à chaque sauvegarde du fichier si de nouvelles entrées ont été ajoutées.

Interface

Le système d'inscription et de connexion contient une gestion d'erreurs. Par exemple, si un utilisateur s'inscrit sur l'interface, son mot de passe qu'il doit fournir doit contenir un nombre minimum de caractères, ainsi que des caractères spéciaux, une majuscule, une minuscule et un chiffre.

En cas de formulaire non conforme, une erreur est spécifiée sur l'interface.
Les utilisateurs ne peuvent pas modifier leur mot de passe car la méthode n'a pas été implémentée.

Manuel utilisateur

Lors de l'exécution du programme, l'utilisateur peut soit se connecter, soit créer un nouveau compte, puis se connecter avec.

Une fois connecté, l'utilisateur peut lancer une simulation qui génère aléatoirement une situation. Aucune situation n'est par défaut résolue, sinon cela n'aurait aucun intérêt pour le calcul du score, qui serait toujours de 10/10.

L'utilisateur peut générer une nouvelle situation en appuyant de nouveau sur le bouton Start. Le score sera abandonné, et le chronomère sera réinitialisé.

Lorsque l'utilisateur a trouvé comment alimenter les 3 moteurs, un message affiche le temps pris pour résoudre le problème et une note de 0 à 10 est générée.

La note est calculée à partir du chronomètre. Nous laissons 2 secondes de marge avant de diminuer linérairement la note au fur et à mesure que le temps passe.

L'utilisateur peut afficher l'historique de ses scores via un bouton sur l'interface.

La moyenne est affichée dans les statistiques.

Des situations peuvent êtres configurées en appuyant sur les différents éléments affichés à l'écran. Par exemple, un réservoir de carburant peut être vider en appuyant sur ce dernier.

Instructions d'utilisation

Pré-requis

Pour compiler les fichiers sources sous Linux, le projet utilise les dépendances suivantes :

- **qmake**
- **qt5**

Les paquets peuvent être installés via la commande suivante sous une distribution Ubuntu :

```
sudo apt-get install qt5-default
```

Conseils d'utilisation

Pour compiler le projet, il suffit d'utiliser le Makefile en utilisant la commande :

```
make
```

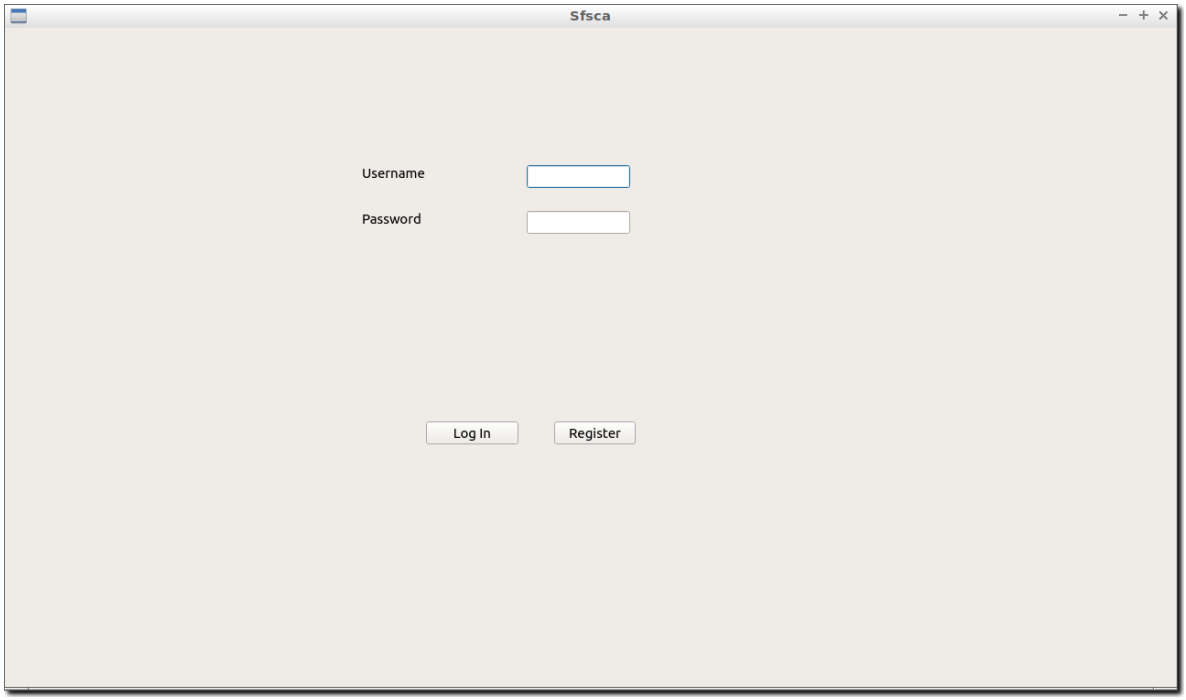
Si le Makefile n'est pas présent :

```
qmake sfsca.pro -spec linux-g++
```

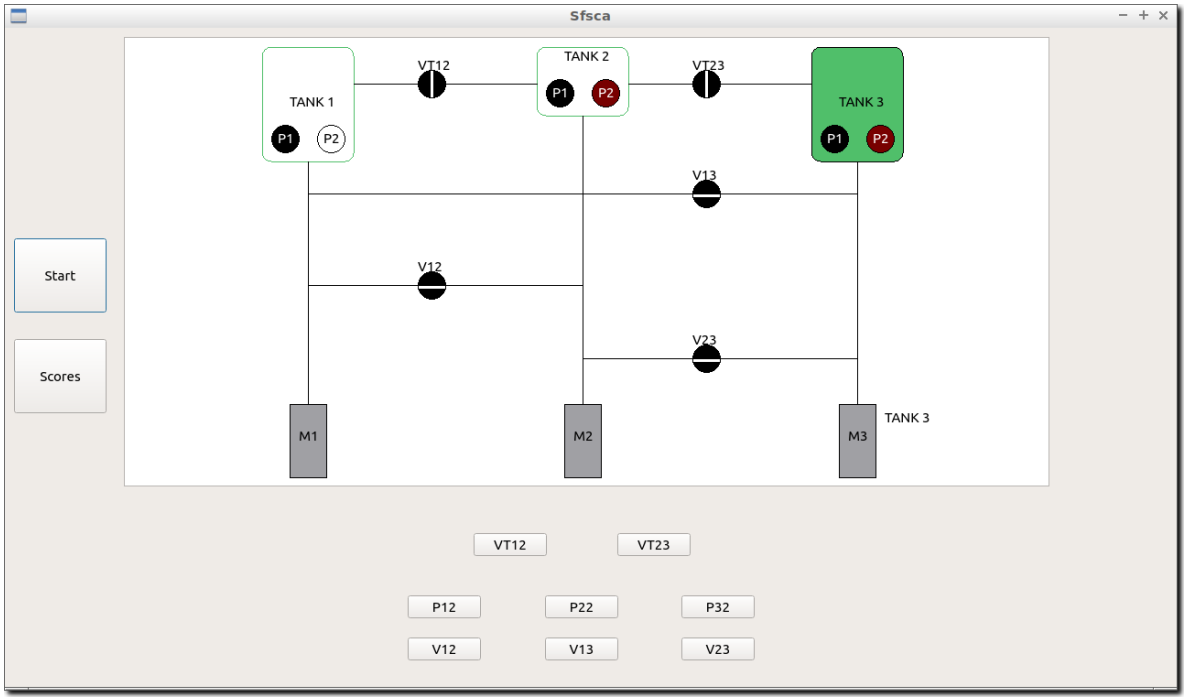
Une fois compilé, le programme peut être lancé avec la commande :

```
./sfsca
```

Captures d'écrans



Interface d'authentification



Interface en cours de simulation

Remerciements

Merci à *Olivier Gay* pour son implémentation du SHA-512