OpenBoard

-

Explication du code source

Installation de Qt 5.5 :

Qt est nécessaire à la compilation du code source de OpenBoard.
Pour la version 1.3 d'OpenBoard, c'est la version 5.5.1 de Qt qui est nécessaire.

Installer QtCreator (optionnel) :

QtCreator est l'IDE idéal pour compiler OpenBoard. Cet IDE est capable de compiler le projet C++ et est capable d'ouvrir les fichiers .ui qui sont vitaux pour le fonctionnement du logiciel.

Télécharger le code source d'OpenBoard :

Il faut télécharger le code source sur GitHub (voir lien ci-dessous) ainsi que le dossier "Openboard-Thirdparty" pour s'assurer du bon fonctionnement de la compilation. Compiler en utilisant QtCreator ou directement avec Qt depuis un terminal (Linux seulement).

Liens :
Github Openboard :
https://github.com/DIP-SEM/OpenBoard

Github Openboard-Thirdparty :
https://github.com/DIP-SEM/OpenBoard-ThirdParty

Wiki Openboard :
https://github.com/DIP-SEM/OpenBoard/wiki

Dans le dossier du code source, on peux constater 3 fichiers/dossiers important :

- /src/ contient les divers fichier .cpp et .h classé par dossier pour mieux s'y retrouver.

- /resources/ contient les fichiers images des différent icônes/curseurs du logiciel OpenBoard, ainsi que des fichiers .ui : Une extension propre à Qt Creator qui définis des affichages “pré-fabriqué” sans code. Respectivement :
- /resources/images/
- /resources/forms/

- /OpenBoard.pro est un fichier à ouvrir avec QtCreator pour ouvrir le projet et le compiler facilement.

Présentation des différentes sections du code :

Les dossiers suivant se trouvent dans le dossier /src/ et constituent le code du programme. Voici une brève description pour la plupart d’entre elles :

adaptors : Contient les fonctionnalités “export/import” d’OpenBoard, ce qui lui permet d’exporter les pages en PDF ou en UBZ (format OpenBoard). Ou encore importer divers fichiers pour les afficher sur les pages. (PDF, Image…)

board : Contient les fonctionnalités propres au mode “Board” (le mode normal au démarrage). Par exemple la barre d’outil fixe ainsi que toute ses actions qui lui sont associés, gérer la couleur du crayon en fonction de la couleur du fond, …

core : Contient le fichier “mère” (main.cpp) qui est exécuté au démarrage d’OpenBoard. Les autres fichiers regroupent la plupart des fonctions et classes utilisées par toutes les autres branches du code, les préférences utilisateurs, et la gestion de l’affichage en plein écran.

desktop : Contient les fonctionnalités du mode “bureau”, tel que l’affichage de la palette flottante d’outils (sélectionner outil, bouger la palette, replier/déplier la palette) et le fait de pouvoir dessiner sur son bureau.

domain : Contient le code qui gère les différent objets sur le tableau comme les dessins, les objets PDF, le texte, et tout autre objets. Cela comprend aussi la sélection de ces divers objets, l’affichage des options qui leurs sont propres (comme change la police pour l’objet texte).

gui : Comprend les fonctionnalités de toutes les palettes du logiciel : Les palettes flottantes des différents modes, les palettes “dépliables” du mode board (à gauche et à droite) et autre objet “apparaissant” comme le clavier virtuel.

pdf - pdfMerger : Permet l’affichage des objets PDF sur le tableau (cet objet étant complexe, il mérite sont propres dossier).

tools : Comprend la gestion des divers outils, certains sont simple, mais d'autres sont assez compliqué comme la "règle" qui a sont propre fichier .cpp

Les outils les plus simples ne sont pas géré par cette catégorie "tools", le crayon par exemple est géré par la catégorie "domain".

Explications des modifications que j'ai apportées :

(Attention, lors de la création de nouvelle variable dans un fichier .cpp ne pas oublier de la déclarer dans le fichier .h)

Nouveaux curseurs pour le crayon et la règle

Le curseur de base pour les outils crayon et souris était une "croix", ce bitmap n'est même pas une image, mais un curseur généré par Qt.

J'ai donc crée ou modifié deux images qui conviennent le mieux pour ces deux outils et ajouté ces images dans le dossier /resources/images/cursors sous le format 'png'

Ensuite dans le fichier gui/UBResources.cpp :

On voit que chaque icône est associés à une variable d'outils, ici penCursor et rulerCursor.

On modifie penCursor pour qu'il affiche une image png (il suffit de copier les paramètres d'un autre outil comme le marqueur par exemple et modifier le chemin vers l'image du crayon).

On crée aussi une nouvelle variable rulerCursor pour afficher une image à l'outil règle.

Pourquoi créer une variable pour la règle alors que le crayon existait déjà ?
Parce que le crayon et la règle partageait le même icône "croix". On veut qu'ils ait leurs propres icônes dorénavant.

Maintenant que les images sont pris en compte par le logiciel il faut lui dire d'utiliser ces curseurs au bon moment !

Pour cela nous allons modifier le fichier board/UBBoardView.cpp :

C'est la méthode void UBBoardView::setToolCursor (int tool) qui nous intéresse ici, elle prend en paramètre un outils sous la forme d'un int et change le curseur.

Cette méthode traite chaque cas avec un switch. Il suffit d'assigner l'outil a sa variable stocké dans les ressources.

Voilà, le curseur devrait maintenant changer en fonction de l'outil !

Ajouter une pastille de couleur de prévisualisation pour les outils crayon, marqueur et règle

Pour aider l'utilisateur, on peux faire apparaître une pastille de prévisualisation à la couleur et à la taille actuellement sélectionnée.

Il suffit pour cela de modifier un seul fichier : domain/UBGraphicsScene.cpp

De base, un objet de cercle existe déjà pour l'outil marqueur, il s'appelle "mMarkerCircle". Nous allons tout d'abord le cloner en "mPenCircle", ainsi que toute ses méthodes qui lui sont associés à savoir : createMarkerCircle(), hideMarkerCircle() et drawMarkerCircle() de façon à ce que l'outil crayon possède lui aussi un cercle de prévisualisation comme le marqueur.

Pour ensuite "remplir" ces cercles de la couleur correspondante à celle selectionnée, on utilise les lignes suivante :

```
mPenCircle->setBrush(QBrush(UBSettings::settings()->currentPenColor()))
mMarkerCircle->setBrush(QBrush(UBSettings::settings()->currentMarkerColor()))
```

Dans les méthodes drawPenCircle et drawMarkerCircle.

Il n'y a plus qu'a dire quand afficher les pastilles : mPenCircle pour les outils crayon et règle et mMarkerCircle pour l'outil marqueur.

A faire dans la méthode "inputDeviceMove".

Faire en sorte que le clic long pour afficher une sous-palette en mode "bureau" devienne un clic simple, à condition que l'outil soit déjà sélectionné

Pour cela il faudra modifier le fichier desktop/UBDesktopAnnotationController.cpp

Ici chaque sous-palettes d'outils a ses méthodes :

Pour le crayon par exemple c'est penActionPressed( ) qui est executé lors de la pression sur l'outil en mode "bureau" et penActionReleased( ) qui est executé lorsque le bouton de l'outil est "relaché". Ainsi ces deux méthode forme le clic sur le bouton de l'outil crayon. Pour es autres outils les noms des méthodes sont les même, il suffit de remplacer "pen" par "marker" ou encore "eraser"...

Dans ces méthodes, on peux voir qu'un timer démarre lors de la pression, si le timer reste activé pendant un temps donné avant le relâchement du clic, alors la sous-palette s'affiche. Si nous voulons changer ça en clic simple, il suffit tout d'abord d'enlever tout ce qui peux avoir un rapport avec les timers car nous n'en avons plus besoin, et de créer un booléen par outils qui se mettra à 'true' lorsque l'outil est sélectionné et les autres se mettrons à 'false'.

Ainsi on sait toujours quels outils est sélectionné, il n'y a plus qu'a faire un peu de logique et dire lors de la pression sur un outils : "Si l'outil est déjà sélectionné (boolean à true) alors on affiche la sous-palette".

Pour afficher une sous-palette spécifique à un outils il suffit d'utiliser la méthode togglePropertyPalette(palette) qui prend en paramètre la sous-palette à afficher.

Voici le nom des sous-palettes de chaque outils :

Crayon : mDesktopPenPalette
Marqueur : mDesktopMarkerPalette
Gomme : mDesktopEraserPalette

Par exemple pour afficher la sous palette du crayon :
togglePropertyPalette(mDesktopPenPalette);

Ouvrir la sous palette "page" et "effacer" du mode board sans clic long, juste un clic simple

Il faut enlever le timer liés aux boutons et directement afficher la sous-palette pour chacun des boutons, en somme c'est presque la même chose que pour les sous-palettes des outils du mode bureau (point précédant).

Cela ce passe ici : board/UBBoardPaletteManager.cpp