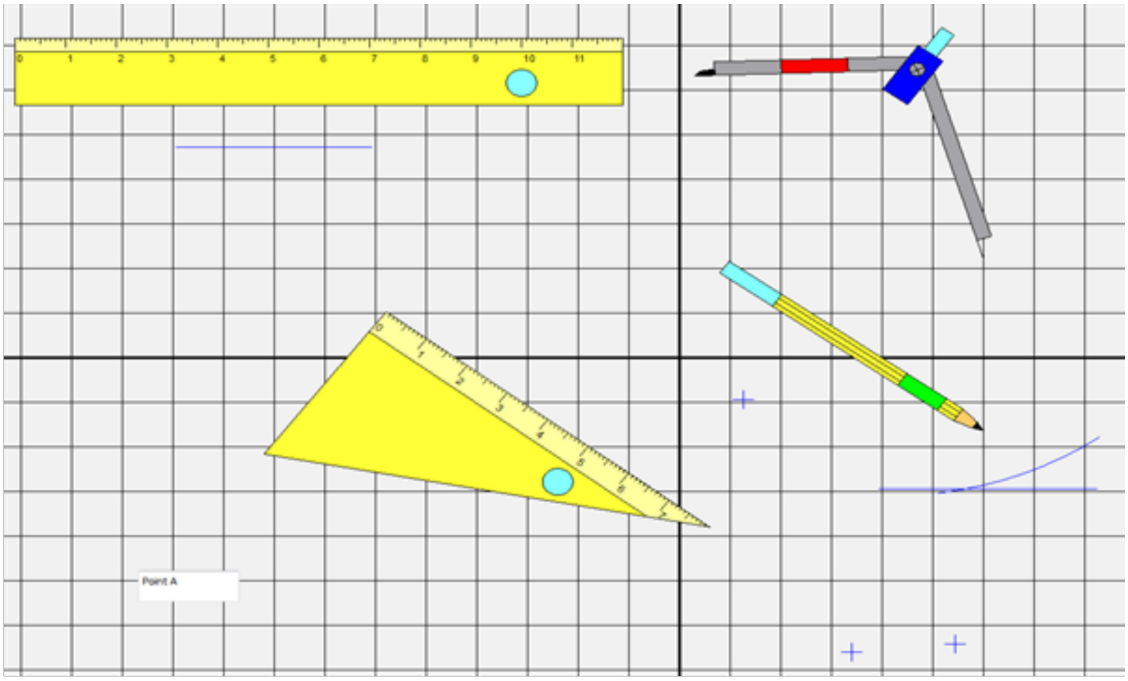
Ravel ADAMONY
Pierre BIOLLEY
David BONNIER
Dylan JACQUIN
Hugo ROCHE

Dossier du Projet tuteuré

Logiciel de Mathématique



2013-2014

Remerciements

A Dominique MONCORGE, qui nous a proposé ce sujet et accompagné tout au long du Projet.

Au personnel du Service de Soins et d'Eduction Spécialisé à Domicile (SSESD) pour leur temps qu'ils nous ont consacré ainsi que leurs expliquations aussi précieuse que pertinente.

Résumé

Notre objectif était la création du module de géométrie d'un logiciel de Mathématiques pour les handicapés. Nous devions parvenir à reconstituer **les conditions réelles de tracé** de figure via la présence des instruments usuels de tracé géométrique : règle, équerre, compas et crayon. Nous avons organisé plusieurs réunions avec les clients (le **SSESD**[1]) qui ont abouti à l'établissement d'un **cahier des charges** qui nous a permis de bien comprendre les fonctionnalités désirées. Nous sommes partis de deux logiciels (**BOMEHC**[2] et **TGT**[3]) dont nous avons extraits les points forts / faibles pour créer un logiciel qui conviendrait au mieux aux clients.

Les instruments cités plus haut sont déplaçables et orientables dans notre logiciel. De plus, nous avons implémenté un algorithme de **magnétisme** qui permet d'avoir la sensation qu'un instrument bute contre autre et donc de reproduire les étapes de création d'une droite perpendiculaire (par exemple) le plus fidèlement possible.

Nous avons par ailleurs implémenté un système de **gestion d'utilisateurs**. Le logiciel ne nécessite pas de se connecter pour démarrer mais permet aux utilisateurs enregistrés de retrouver leurs **paramètres** propres s'ils le font. **Ces paramètres sont définis par un administrateur** pouvant être créé au premier démarrage du logiciel.

Sommaire

Introduction

Nous avons pu, pendant 2 mois, nous confronter à un projet d'envergure : la création d'un logiciel d'apprentissage des Mathématiques pour personnes en situation de handicap. Nous nous sommes plus spécifiquement chargés du module de géométrie de ce logiciel. À l'aide de réunions régulières nous avons pu définir un objectif clair et, grâce à l'établissement d'un cahier des charges, avoir une meilleure idée des attentes des clients. Nous avons travaillé en coopération avec un orthophoniste et trois ergothérapeutes du Service de Soins et d'Education Spéciale à Domicile (SSESD). Nous nous sommes inspirés de deux logiciels existants que nous avons analysés afin de créer un produit qui réponde au mieux aux objectifs.

Nous allons vous présenter le déroulement du projet, en commençant par la présentation du SSESD et des deux logiciels dont nous nous sommes inspirés. Dans un second temps, nous vous présenterons la phase de programmation et le travail qui l'a précédée.

Partie 1

Présentation du contexte

I. Un premier projet : BOMEHC

BOMEHC (Boîte à Outils Mathématiques pour des Élèves en situation de Handicap au Collège) est un logiciel de mathématiques orienté pour des personnes en situation de handicap créé par des étudiants de Lille. Au départ, notre projet tuteuré devait consister à reprendre ce logiciel pour améliorer son module de géométrie, en coopération avec ces derniers. Par faute de délai de ses concepteurs, ce projet n'a pu être réalisé.

II. Mise en place du projet tuteuré avec le SSESD

Le SSESD (Service de Soins et d'Éducation Spéciale à Domicile) est un service qui aide des personnes de 3 à 20 ans atteintes de divers handicap moteur dans leur scolarisation et intégration sociale.

Après l'annulation du projet BOMEHC, il a fallu trouver un nouveau projet en correspondance avec l'ancien. Le SSESD de Brive-Charensac cherchait un nouveau logiciel de mathématiques pour les personnes en situation de handicap. Il nous a été demandé de tester 2 logiciels pour en tirer leurs avantages et inconvénients : BOMEHC et TGT (Trousse Géo Tracer).

Les principaux avantages de BOMEHC sont qu'il peut gérer plusieurs utilisateurs, il touche aussi bien à l'arithmétique qu'à la géométrie. Ce logiciel est séparé en deux écrans : cahier et exercice. Dans les exercices on peut y retrouver des tableaux pour faire des opérations, des outils pour la géométrie. Le système de cahier est très intéressant car il permet d'incorporer les énoncés ou travaux faits dans l'écran exercices dans l'écran cahier sous forme d'image. Il est aussi possible d'écrire dans celui-ci pour créer des exercices et de les imprimer.

Mais BOMEHC possède aussi de nombreux défauts, pour commencer, le design du logiciel qui est très grossier. Certaines fonctionnalités ne sont pas implémentées, d'autres sont difficiles d'utilisation comme les tableaux. Le gros point faible de ce logiciel est son module de géométrie, qui n'est pas pratique, et peu complet.

TGT est lui un logiciel spécialisé dans la géométrie, on peut manipuler une équerre, une règle, un compas, un rapporteur et un crayon. Chaque instrument est magnétisé avec les autres, ce qui est pratique pour faire des constructions géométriques. Certaines options sont cachées comme le réglage exact de la position, rotation...

Un très gros point faible de ce logiciel est qu'un instrument est une fenêtre. De ce fait, on est très vite perdu et cela créer des bugs. Les boutons sur les instruments sont trop petits, et le fait de devoir maintenir le clic sur ceux ci pour bouger/tourner l'instrument ne convient pas pour un utilisateur avec un handicap moteur. Les options cachées sont difficiles à trouver. Le système de sauvegarde/chargement, n'est quant à lui pas encore au point.

Après avoir testé ces deux logiciels, nous avons rencontré le SSESD afin de

définir leurs attentes. Durant cette réunion, nous avons pu passer en revue BOMEHC et TGT avec eux et se mettre d'accord sur les fonctionnalités à garder, à améliorer ou à implémenter. C'est à l'issue de cette réunion que fut définit le projet définitif : créer un logiciel d'aide aux mathématiques pour les personnes en situation de handicap. Ce logiciel pourra contenir plusieurs modules, et garder le système de cahier de BOMEHC. Ce projet s'étalera sur plusieurs années, pour 2013/2014 il a été convenu de créer la base du logiciel (cahiers, gestion d'utilisateurs, sauvegarde/chargement), ainsi qu'un module de géométrie où l'on pourra manipuler 4 instruments qui sont l'équerre, la règle, le compas et le crayon.

Partie 2

Expérimentation

Notre travail a été divisé en deux parties : tout d'abord l'étude de l'existant et l'organisation du projet puis dans un second temps la réalisation de ce dernier.

I. Etude et organisation

1. Etude de l'existant

Dans un premier lieu, nous avons étudié les logiciels existants chacun de notre côté : TGT et BOMEHC. Puis nous avons départagé les qualités et les défauts de ces logiciels afin de pouvoir plus facilement orienter notre projet.

Nous avons confronté nos idées avec celles du SSESD lors d'une réunion pour mélanger contraintes d'utilisateurs et contraintes de programmeurs de façon à se mettre d'accord sur un projet final et établir un cahier des charges.

2. Organisation

Malgré les conseils de M.Moncorgé, notre façon de nous organiser a été de s'attribuer des tâches puis de nous voir régulièrement sur Skype ou physiquement à l'IUT (tous les deux à trois jours) pour faire le point sur la situation et pour éventuellement s'attribuer de nouvelles tâches si la précédente est terminée.

Finalement, sur nos derniers jours de développement, nous avons décidé de suivre ses conseils en établissant un diagramme de Gantt (en annexe).

Sur ce dernier point, nous avons néanmoins rencontré une difficulté : celle de la répartition des tâches à cinq. En effet, les tâches ne sont pas toutes aussi simples les unes des autres ce qui nous empêcha d'appréhender la durée de ces dernières correctement, causant ainsi du retard au sein de notre travail.

Malgré cette difficulté, nous avons néanmoins pu élaborer une répartition des tâches correcte.

II. Développement

1.QT

Le projet nécessitant la présence d'une interface utilisateur, nous avons immédiatement pensé à Qt car c'est une bibliothèque permettant de créer des interfaces de façon aisée, de plus nous l'avons étudié donc nous la connaissons bien, et pour finir, elle utilise le langage C++ qui est le langage que nous manipulons le mieux. Parallèlement, Qt est une librairie gratuite, mise à jour régulièrement avec une communauté active, ce qui facilite la programmation.

2.UML

Mais avant de commencer le projet, nous avons préféré nous organiser à travers un diagramme UML (en annexe) de façon à aborder méthodiquement le développement même s'il a été modifié en cours de programmation.

Ainsi nous avons agencer notre code en essentiellement deux parties qui sont d'une part les figures et d'autres parts les instruments.

Pour les figures, le travail a surtout consisté en une réadaptation du code existant dans Qt, pour répondre à nos besoins d'héritage, dans un souci de sauvegarde et d'accès des données.

Pour les instruments, il a fallu les créer de toutes pièces. Nous avons alors commencer par une classe instruments abstraite de laquelle héritera tout les instruments. Elle possède les donnés essentielles comme la position de l'instrument son orientation ainsi que son fichier XML (expliqué plus loin). Ensuite nous avons créé des classes qui en héritent : la règle, l'équerre, le crayon et le compas, qui ont chacun des données et fonctions membres spécifiques à leurs rôles.

3.La gestion des utilisateurs

Dans BOMEHC, une gestion d'utilisateur était présente et le SSESD nous a demandé de la garder et d'y ajouter un système d'administrateur.

Nous avons donc créé un premier système de gestion à l'aide de fichiers texte, à chaque fois créé au moment de la création de l'utilisateur, dans lesquels étaient stockés les informations et les paramètres de ceux-ci (identifiant, mot de passe, sensibilité du magnétisme, présence ou non de la grille…).

Nous nous sommes vites rendus compte que cela était trop peu pratique et gourmand en place (un fichier texte multiplié par x élèves…). Nous avons donc opté pour le format XML. Ce format utilise des balises à la manière du HTML. Nous nous retrouvons donc maintenant avec un seul fichier pour tous les utilisateurs, ce qui est beaucoup plus pratique que la version précédente. Voici ce que contient le fichier par défaut :

```
<utilisateurs>
	<firstUse>true</firstUse>
	<utilisateur>
		<identifiant>admin</identifiant>
		<password>admin</password>
		<admin>true</admin>
	</utilisateur>
</utilisateurs>
```

Comme vous pouvez le voir, les utilisateurs sont stockés entre les balises <utilisateur> </utilisateur> contenues dans la première balise du fichier : <utilisateurs> </utilisateurs>. Ici un seul utilisateur est créé et est administrateur pour permettre aux clients de créer d'autres utilisateurs et d'accéder à toutes les fonctionnalités du logiciel (la création et la suppression d'utilisateur ne sont disponibles que pour un administrateur).

Vous pouvez aussi remarquer une balise <firstUse></firstUse>. Cette balise permet, comme son nom l'indique de savoir si c'est la première fois que le logiciel est lancé sur le poste en question. Si tel est le cas, alors nous lançons l'enregistrement et si l'utilisateur choisi de s'enregistrer, il sera administrateur (on ne laisse pas le choix ici).

Par la suite, si l'on veut créer un utilisateur, il faudra être administrateur et spécifier si l'utilisateur que l'on crée le sera aussi (on laisse le choix). Lorsqu'un nouvel utilisateur est créé, nous parcourons notre fichier XML afin de savoir si l'identifiant spécifié existe déjà (pour éviter les doublons). Si l'identifiant n'est pas trouvé lors de la vérification, nous cherchons la ligne qui contient « </utilisateurs> » et la remplaçons par :

```
	<utilisateur>
		<identifiant>identifiant saisi</identifiant>
		<password>mot de passe saisi</password>
		<admin>true ou false suivant la valeur saisie</admin>
	</utilisateur>
</utilisateurs>
```

Ainsi notre fichier aura la forme suivante :

```
<utilisateurs>
	<firstUse>true</firstUse>
	<utilisateur>
		<identifiant>admin</identifiant>
		<password>admin</password>
		<admin>true</admin>
	</utilisateur>
	<utilisateur>
		<identifiant>identifiant saisi</identifiant>
		<password>mot de passe saisi</password>
		<admin>true ou false suivant la valeur saisie</admin>
	</utilisateur>
</utilisateurs>
```

Notre utilisateur est donc bien créé et rajouté à la suite des autres. Si nous souhaitons le supprimer, c'est plus facile. Il suffit pour cela d'utiliser le parseur (analyseur syntaxique permettant le décodage d'un fichier XML très simplement) d'XML que propose Qt. Ce parseur propose une architecture en arbre : notre balise <utilisateurs> est la balise « root » (racine de l'arbre), les balises <firstUse> et <utilisateur> sont des « childs » (branches) et les balises comme <identifiant> ou encore <password> sont des « grandChilds » (feuilles). Pour supprimer l'utilisateur dont l'identifiant est saisi par la personne qui utilise le logiciel, il suffit de regarder chaque balise <identifiant> de chaque child jusqu'à ce que les identifiants concordent.

Il n'y a ensuite plus qu'à supprimer le child en question. Ceci supprimera directement toute la balise <utilisateur>.

Pour la connexion, il faut saisir un identifiant et un mot de passe. Après avoir vérifié si l'identifiant existait bien en effectuant un premier parcours du fichier XML, nous allons vérifier que le mot de passe concorde bien avec l'identifiant (en regardant les contenus des balises <identifiant> et <password> du *child* concerné). Ensuite les paramètres (s'ils existent) seront chargés et un booléen prendra la valeur de la balise <admin> pour savoir à quelles fonctionnalités l'utilisateur désigné à accès.

4.Le dessin des instruments grâce au fichier XML

Tous les instruments sont créés à partir d'un fichier XML qui contient leur position, orientation et des constantes propres à certains instruments comme la longueur d'une branche de compas, de la mine du crayon...

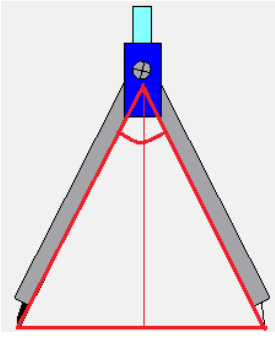
La règle est composée simplement de deux rectangles : l'un, au dessus, plus petit, où sont présentent les graduations. C'est aussi ce rectangle qui délimite la zone qui peut être rendue transparente. Une second rectangle est placé sous ce dernier pour définir le corps de la règle. La hauteur et la largeur de la règle sont des constante du fichier XML, les tailles des rectangles sont ensuite calculé à partir de ces deux valeurs, ce qui permet de modifier la taille de la règle juste en modifiant le fichier correspondant.

L'équerre est elle composée d'un triangle et d'un quadrilatère. Comme la règle, le quadrilatère est placé sur le triangle, elle contient les graduations et délimite la zone transparente. Pour trouver les coordonnées de la jonction entre la droite séparant les deux figures et l'hypoténuse, il suffit d'appliquer le théorème de Thalès étant donné que l'on connaît toutes les dimensions de l'équerre. Pour modifier la taille de celle ci, le fichier XML contient également la hauteur et la largeur, tout le reste est calculé dans le code.

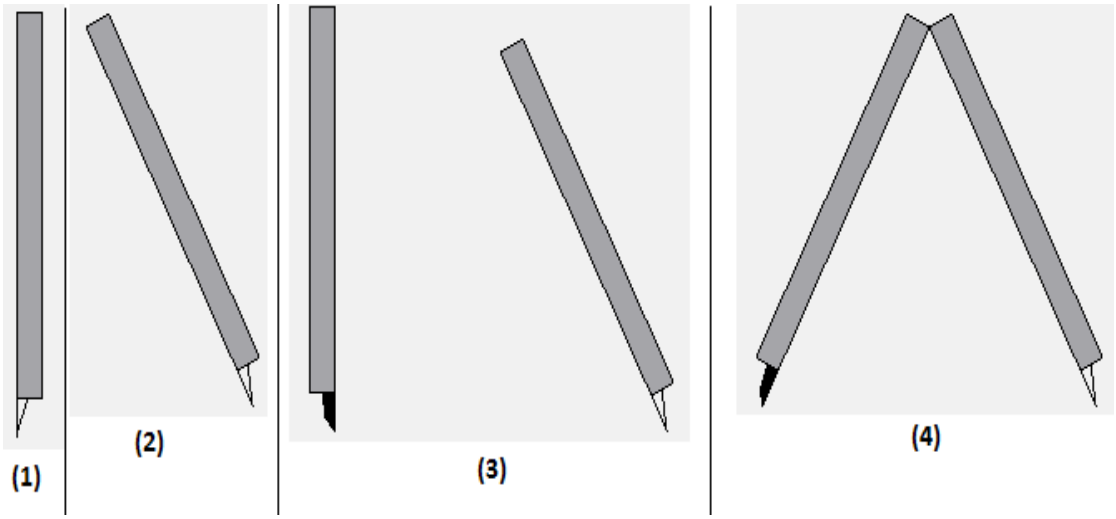
Le crayon contient un triangle et un quadrilatère, composant la pointe et un rectangle, constituant le corps. Le triangle de la pointe représente la mine, il peut être rendu transparent. Le quadrilatère correspond au bois coupé d'un crayon. On retrouve deux droites dans le corps du crayon, ce qui donne un léger effet que celui-ci est hexagonal. Dans le fichier XML, on peut retrouver la hauteur du crayon, sa largeur, ainsi que la hauteur de la pointe.

Enfin le compas est un peu plus complexe. Les deux branches sont des rectangles, la pointe est un triangle, la mine un quadrilatère. La base est composée de deux rectangles, et la vis d'un cercle avec deux droites. Le plus compliquer pour modéliser cet instrument a été de réaliser correctement l'écartement : lorsque l'on commence à l'écarter il faut que la pointe reste fixe, que les deux branches s'orientent de la bonne façon et se replacent. Il faut aussi que la base se repositionne par rapport aux branches et que la vis tourne proportionnellement à l'écartement.

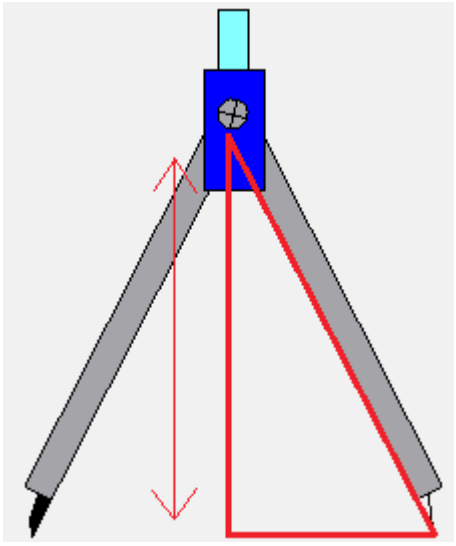
Pour la réalisation de l'orientation des branches, il faut calculer l'angle créé par l'écartement entre les deux branches et le diviser par deux car on ne cherche que l'orientation d'une branche. Cela se fait grâce à la trigonométrie, connaissant la longueur des branches et de l'écartement.



Une fois l'angle d'orientation calculé, on peut placer la première branche (1), puis la tourner de la valeur trouvée (2). Pour l'autre, il suffit de la déplacer de la valeur d'écartement (3) et de la tourner de l'opposé de l'angle (4).



L'abaissement de la base est calculé grâce au théorème de Pythagore dans le triangle formé par une branche, le demi écartement et la hauteur cherchée.



Une fois la hauteur connue, il est simple de placer la base.

Enfin, pour la vis, seul les deux segments sont tournés de la moitié de l'angle trouvé.

Dans le fichier XML du compas, on retrouve comme constantes la longueur / largeur des branches ainsi que la hauteur des mines et de la base.

5.Interactions avec la souris

Le SSESD a fait remarquer quelque chose d'intéressant à propos du logiciel TGT: pour déplacer les instruments avec la souris, il utilisait un système de "Drag and Drop" (Cliquer/Glisser) : l'utilisateur devait maintenir le clic enfoncé lorsque le curseur était sur l'instrument puis déplacer la souris afin de le déplacer. Ils nous ont fait part du fait que ce système, certes utilisé par une grande quantité de logiciels et par toutes les interfaces de systèmes d'exploitation, s'avère peu pratique aux mains de personnes souffrant d'un handicap moteur.

Nous leur avons donc proposé un autre système d'interactions : l'utilisateur pourra cliquer une première fois sur l'instrument pour pouvoir le sélectionner. Une fois sélectionné, l'instrument suivra le curseur et si l'utilisateur clique une seconde fois, l'instrument sera désélectionné et arrêtera de suivre le curseur. De même pour les boutons de rotation, de tracé et d'écartement du compas : prenons l'exemple du bouton de rotation : un premier clic pour faire passer l'instrument en mode "rotation", puis un autre clic pour arrêter la transformation. L'idée leur a convenu et nous avons donc pu implémenter ce système.

A. Explication globale

L'implémentation de ce système n'a pas été si simple : en effet avant même de le coder une première difficulté s'est manifestée : celle de la détection de l'instrument via le clic de la souris. Étant donné que chaque instrument a été dessiné composant par composant en dur (par exemple, la fonction de dessin du compas dessine un rectangle bleu foncé pour faire la base, un rectangle bleu clair pour la tige avec le bouton de rotation, un triangle noir pour faire la mine, etc.), à première vue il était impossible de savoir sur quel instrument on avait cliqué. Puis nous avons trouvé une solution : celle de récupérer la couleur du pixel sur lequel on a cliqué.

En effet, nous avons remarqué que chaque instrument était composé de peu de couleurs. Donc en fonction de la couleur du pixel sur lequel on clique, il est possible de savoir sur quel instrument ou bouton on a cliqué à condition que chaque couleur utilisée ne soit utilisée qu'une seule fois. De ce fait, pour pouvoir utiliser une couleur deux fois, il a fallu utiliser deux valeurs RGB voisines (par exemple le jaune saturé de la règle est de valeur RGB (255, 255, 51) alors que celui de l'équerre est de RGB (255, 255, 50)).

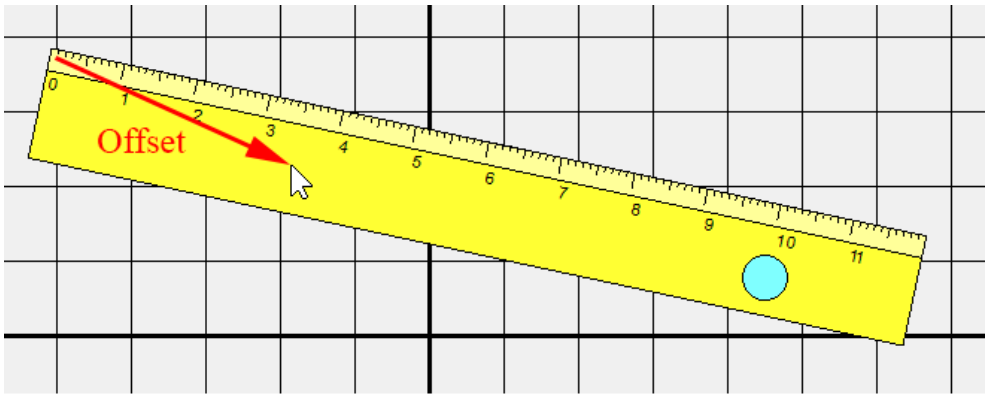
Ainsi en fonction de la couleur récupérée, on effectue différentes actions.

B. Partie commune à tous les déplacements

Chaque instrument possède quatre booléens : un pour le déplacement, un pour la rotation, un pour l'écartement ainsi qu'un pour le tracé. En fonction de la couleur du pixel sur lequel on a cliqué, on met ces booléens soit à "true", soit à "false". Puis lorsque l'on bouge la souris, en fonction de la valeur de ces booléens, on effectue différentes actions (si le booléen de rotation est sélectionné, alors lorsque l'on bouge la souris, l'instrument s'inclinera). Par contre, si on clique sur un pixel blanc ou si on clique alors que l'un de ces booléens est à "true", alors tous les booléens de l'instrument sélectionné passeront à "false".

C. Déplacement des instruments

Comparé aux autres algorithmes, celui du déplacement est assez simple: lorsque l'on clique sur un instrument, on met le booléen de déplacement à "true" et on crée un vecteur "offset" allant de la position de l'instrument à celle du clic de la souris.



Puis à chaque fois que l'on déplace la souris, on soustrait ce vecteur "offset" à la position du curseur afin d'avoir la nouvelle position de l'instrument et on met à jour la position de l'instrument à cette valeur.

D. Rotation des instruments

L'algorithme de la rotation des instruments est lui aussi relativement simple: lorsque l'on clique sur le bouton de rotation, on a juste à mettre le booléen de rotation à "true". Plus lorsque l'on déplace la souris, on calcule l'angle entre :

->la droite passant par le point aux coordonnées de l'instrument et par le celui aux coordonnées du curseur.

->la droite horizontale passant par le point aux coordonnées de l'instrument.

Enfin on met à jour la valeur de l'orientation l'instrument à la valeur précédemment calculée.

E. Écartement du compas

De façon théorique l'algorithme de l'écartement du compas est assez simple. En effet lorsque l'on clique sur le bouton, on récupère la valeur de l'écartement au moment où l'on clique [*appelons – le E*] ainsi que la position du clic.

Puis lorsque l'on déplace la souris, on crée une droite passant par les deux points du compas (la mine et la pointe) puis on calcule deux projetés orthogonaux sur cette droite : celui du point aux coordonnées de l'endroit où l'on a cliqué [P1] et celui du point aux coordonnées du curseur [P2].

Ensuite on calcule deux distances : celle de la pointe du compas à celui des deux projetés [*D1 et D2*]. On fait la différence entre ces distances et on l'ajoute à l'écartement au moment où l'on a cliqué et on met la valeur de l'écartement à cette différence.

Appelons le nouvel écartement N.
On a : $N = E + (D1-D2)$.

Le problème avec cet algorithme est qu'il ne marche pas avec tous les angles : en effet si l'orientation des instruments entre 90 et 120 degrés, l'écartement se fait à l'envers. Nous avons dû donc remplacer le "+" de la formule ci-dessus par un "-". Mais même avec ce remplacement, cet algorithme est largement perfectible.

F. Tracé du crayon

Tout d'abord, il est important de préciser que le SSESD a demandé à ce que le tracé libre au crayon soit impossible. Nous avons donc décidé que tracé du crayon ne soit possible que s'il est magnétisé à un instrument.

La difficulté lors de l'élaboration de cet algorithme était de faire en sorte qu'il puisse donner la possibilité à l'utilisateur de repasser ses lignes sans que l'on ait à ajouter de lignes superflues dans le tableau de figures.

On a donc créé un booléen indiquant si c'est la première fois que l'on touche l'instrument sur lequel on se magnétise, et à chaque fois que l'on déplace la souris en "mode tracé", on regarde s'il est à "true":

-Si c'est le cas l'est alors on crée une nouvelle ligne dont les coordonnées de ses deux points sont égale à celles du crayon et on le fait passer à "false".

-Si ce n'est pas le cas, alors on modifie la dernière ligne du tableau de figures (qui est donc la ligne que l'on est en train de tracer avec le crayon) en fonction de la position du crayon par rapport à la ligne que l'on est en train de tracer.

G. Bilan

La principale difficulté lors du développement de ce système d'interaction avec la souris est l'absence d'outil déjà existant. Cette partie a donc été assez longue, mais au final nous y sommes parvenus grâces à nos connaissances en mathématiques.

Ce système est certes pratique, mais manque de versatilité. En effet si l'on veut par exemple un futur programmeur décide de changer la couleur des instruments, il sera contraint de faire correspondre les couleurs choisies aux couleurs déclenchant les fonctions relatives aux interactions de la souris.

De plus, les boutons sont de couleur unie. Certes, cela facilite la détection par couleur au détriment de l'intuitivité de ces boutons. Cela contraint à préciser à l'utilisateur dans l'aide que par exemple, le bouton bleu clair sert à orienter l'instrument.

Ce système permet tout de même de pouvoir manipuler les instruments avec une certaine aisance et ce de façon à pouvoir reproduire les mêmes manipulations que pourrait faire un élève avec des instruments physiques.

6.Le magnétisme

Lors des réunions, le SSESD a émis l'idée qu'il serait intéressant que les outils soient magnétisés entre eux. Aussi, nous avons dû créer un algorithme réalisant ce magnétisme.

Le principe : lorsque les outils sont proches, ils se collent comme le ferait un écolier avec sa règle et son équerre.

Tout d'abord, quelque chose que nous avons remarqué : c'est que tous les outils ne se magnétisent pas de la même manière. Exemple : le crayon est ponctuel, tandis que l'équerre est composée de deux segments qui peuvent se magnétiser. Aussi l'algorithme a dû être réadapté pour chacun des outils que sont la règle, l'équerre, le crayon et le compas.

Ensuite, il a fallu savoir ce que l'on avait à disposition pour la réalisation de notre algorithme. Ici nous avions :

-L'évènement du déplacement d'un outil.
-La position de cet outil, ainsi que son orientation.
-La position et l'orientation des autres outils.
-La position des segments déjà instanciés par l'utilisateur.

Une fois ces données récupérées, nous avons pu commencer l'algorithme qui se déroule en trois étapes. Pour plus de clarté, l'explication se fera au travers de l'exemple du magnétisme de la règle.

Déjà, la question se pose : à quoi se magnétise la règle ?
->La règle se magnétise à l'équerre ainsi qu'aux segments tracés.

On se rend rapidement compte que l'équerre peut être considérée comme deux segments.

Pour considérer que la règle est proche d'un segment, deux éléments sont à prendre en compte : la différence d'orientation ainsi que la distance qui les sépare.

Intéressons-nous tout d'abord à l'orientation : la règle a comme donnée membre un angle qui est celui entre le segment que forme la règle et le vecteur horizontal (1,0). Pour l'équerre, cette donnée membre est également présente. Il suffit de faire la différence entre les deux pour savoir leur différence d'orientation. Pour l'autre segment de l'équerre, il suffit d'ajouter 90 degrés pour connaitre son angle. Par contre pour les segments tracés, cet angle est à calculer. Pour cela, nous créons le vecteur directeur du segment et grâce au produit scalaire, nous calculons l'angle que forme ce vecteur avec le vecteur horizontal.

NB : la différence d'angle entre l'équerre et la règle doit être d'environ 180 degrés car les outils doivent se faire face et ne pas être l'un sur l'autre tandis que la différence entre la règle et les segments tracés doit être d'environ 180 degrés ou d'environ 0 degré car la règle peut être des deux côtés du segment sans problème.

Maintenant, intéressons-nous à la distance qui les sépare. Il faut savoir que dans un plan, la distance entre deux segments n'existe pas sauf si ils sont parallèles. Par conséquent, nous scindons la règle en trois différents points : ses deux extrémités et son milieu. Nous calculons la distance de chacun de ces points avec le segment avec lequel on cherche à être magnétisé. Pour cela, nous faisons un projeté orthogonal du point sur le segment et grâce à Pythagore, nous récupérons la distance. Par ailleurs, nous vérifions que les coordonnées du projeté orthogonal font bien partie du segment.

Nous gardons en mémoire la distance qui les sépare ainsi que la différence de rotation de façon à pouvoir comparer ces données avec les prochaines trouvées pour garder les meilleures. Nous gardons également les éventuelles nouvelles coordonnées et nouvelle orientation. Par ailleurs, nous ne sauvegardons ces données que si elles sont inférieures à des taux prédéterminés de façon à se magnétiser que si la règle est proche de la droite testée. De plus, nous passons un booléen à « true » pour signifier que nous nous magnétisons.

Une fois toutes les segments testés, nous remplaçons les coordonnées et l'orientation de la règle par celles gardées en mémoire si le booléen est à « true ».

Variantes :
-Pour le crayon et le compas, nous ne testons pas l'orientation mais que la distance.
-Pour l'équerre, l'algorithme se fait sur ses deux droites orthogonales.

7.Gestion de fichier

La sauvegarde peut être fait classiquement en sauvegardant sous ou faisait une sauvegarde simple. Si l'on a pas déjà défini un emplacement de sauvegarde et que l'on souhaite enregistrer, cela nous renvoie sur une fenêtre nous demandant le chemin sous lequel l'utilisateur souhaite enregistrer le fichier.

Les données sauvegardées sont celles présentent dans les cahiers. Tout est conservé : la taille, couleur des caractères, et les images de la géométrie. Les informations sont transcrites en HTML puis sauvegardées dans un fichier texte.

Le HTML est très pratique pour l'enregistrement de ce genre de données, car c'est un langage de mise en forme, ce qui convient parfaitement pour enregistrer notre fichier qui contient différentes tailles, couleurs de caractères ou encore des images.

L'ouverture du fichier décode le HTML du fichier texte ouvert et affiche les données sauvegardées dans les cahiers respectifs.

8. Le format SVG

Pour enregistrer la partie géométrie, tous les tracés (arc et ligne), nous avons utilisé le format SVG (Scalable Vector Graphics). C'est un format d'image vectorielle basé sur le XML, il s'adapte à toutes les dimensions d'image sans pixellisation.

Le cahier est exporté et sauvegardé en html automatiquement par Qt. Le html lit automatiquement le svg en tant qu'image, c'est pour cela que nous l'avons choisi. Pour rouvrir un fichier html il faut garder le fichier SVG sinon il manquera l'export du dessin.

Pour exporter en SVG, il faut donner un paramètre au QPainter. Nous nous sommes heurtés au problème qu'on essayait de tracer et de générer le format SVG en même temps. Le problème est que pour les deux, il faut le même paramètre, donc on ne peut pas faire les deux en même temps. Nous avons donc fait un tableau qui contient toutes les figures et une fonction qui prend en paramètre un Qpainter.

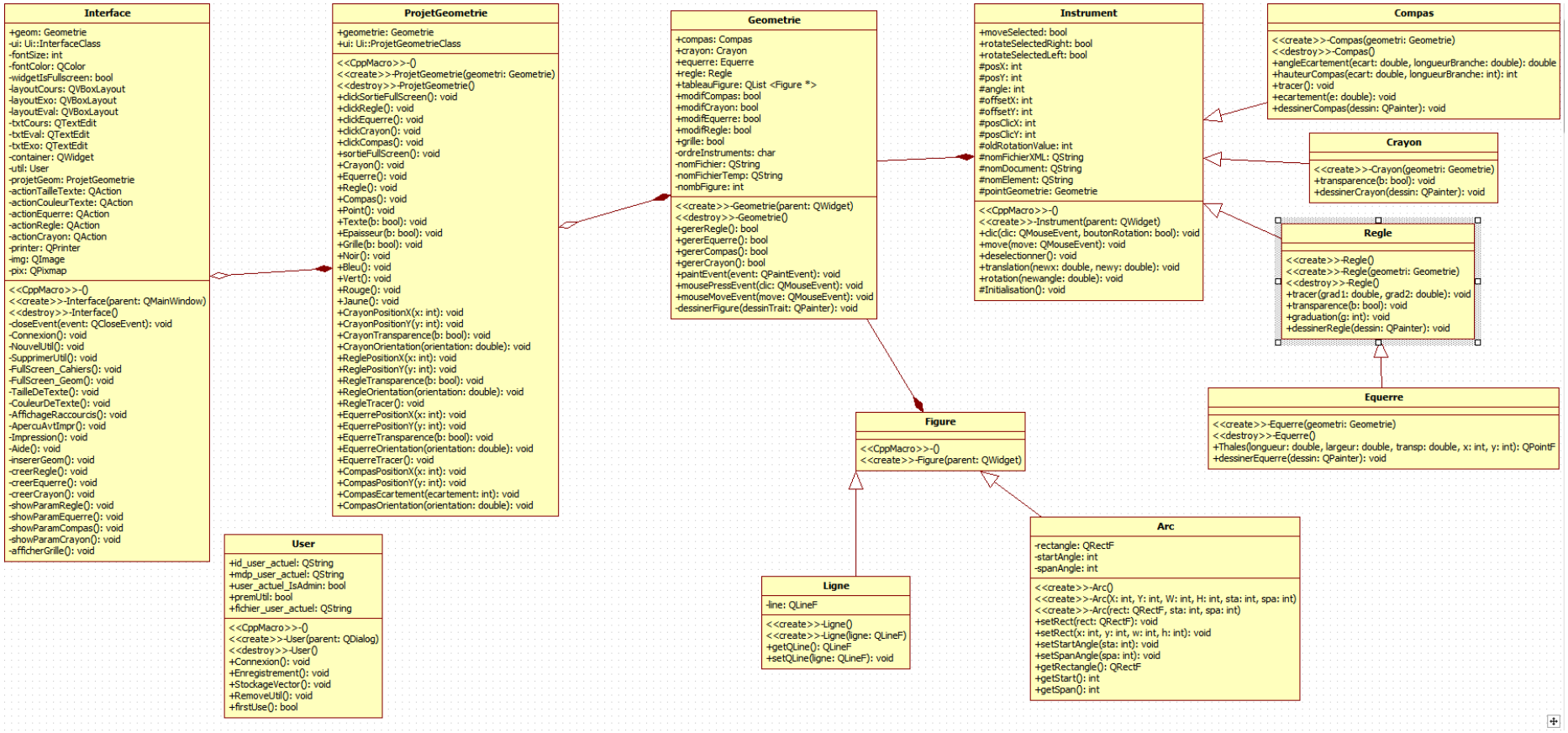
Cette fonction trace toutes les figures avec le QPainter en paramètre. Elle est appelée soit pour générer le format SVG, soit pour afficher les figures à l'écran. Les deux actions sont complètement séparées. L'action pour générer le format SVG crée le fichier avec la date du jour, l'heure et les minutes pour pouvoir sauvegarder à chaque exportation sur un nouveau fichier.

Conclusion

Pour conclure, ce projet constitue le projet le plus important que nous avons eu à faire pendant nos deux années à l'IUT. Comme vous avez pu le lire, la programmation a été précédée d'une longue phase de réunions, de discussions pour bien comprendre ce que les clients attendaient de nous. En partant de l'étude de deux logiciels existants et des remarques des clients par rapport à ces logiciels, nous avons établi un cahier des charges pour clarifier les objectifs. Nous avons ensuite particulièrement travaillé l'aspect réaliste des tracés et interactions entre les instruments (magnétisme) durant la phase de programmation. Par ailleurs, il fallait réussir à rendre le logiciel simple d'utilisation, le public visé étant en situation de handicap. Ceci a rajouté beaucoup de contraintes qu'il nous a fallu prendre en considération.

C'était la première année où ce projet était proposé, le but étant d'arriver à un logiciel de mathématiques complet (partie arithmétique, partie géométrie…). Nous nous sommes chargés de la partie géométrie et de la gestion d'utilisateurs, le projet sera reconduit l'année prochaine et les années qui suivent pour être amélioré.

Annexes


Interface
+geom: Geometrie
-ui: Ui::InterfaceClass
-fontSize: int
-fontColor: QColor
-widgetIsFullscreen: bool
-layoutCours: QVBoxLayout
-layoutExo: QVBoxLayout
-layoutEval: QVBoxLayout
-txtCours: QTextEdit
-txtEval: QTextEdit
-txtExo: QTextEdit
-container: QWidget
-util: User
-projetGeom: ProjetGeometrie
-actionTailleTexte: QAction
-actionCouleurTexte: QAction
-actionEquerre: QAction
-actionRegle: QAction
-actionCrayon: QAction
-printer: QPrinter
-img: QImage
-pix: QPixmap
<<CppMacro>>-()
<<create>>-Interface(parent: QMainWindow)
<<destroy>>-Interface()
-closeEvent(event: QCloseEvent): void
-Connexion(): void
-NouvelUtil(): void
-SupprimerUtil(): void
-FullScreen_Cahiers(): void
-FullScreen_Geom(): void
-TailleDeTexte(): void
-CouleurDeTexte(): void
-AffichageRaccourcis(): void
-ApercuAvtImpr(): void
-Impression(): void
-Aide(): void
-insererGeom(): void
-creerRegle(): void
-creerEquerre(): void
-creerCrayon(): void
-showParamRegle(): void
-showParamEquerre(): void
-showParamCompas(): void
-showParamCrayon(): void
-afficherGrille(): void
ProjetGeometrie
+geometrie: Geometrie
+ui: Ui::ProjetGeometrieClass
<<CppMacro>>-()
<<create>>-ProjetGeometrie(geometri: Geometrie)
<<destroy>>-ProjetGeometrie()
+clickSortieFullScreen(): void
+clickRegle(): void
+clickEquerre(): void
+clickCrayon(): void
+clickCompas(): void
+sortieFullScreen(): void
+Crayon(): void
+Equerre(): void
+Regle(): void
+Compas(): void
+Point(): void
+Texte(b: bool): void
+Epaisseur(b: bool): void
+Grille(b: bool): void
+Noir(): void
+Bleu(): void
+Vert(): void
+Rouge(): void
+Jaune(): void
+CrayonPositionX(x: int): void
+CrayonPositionY(y: int): void
+CrayonTransparence(b: bool): void
+CrayonOrientation(orientation: double): void
+ReglePositionX(x: int): void
+ReglePositionY(y: int): void
+RegleTransparence(b: bool): void
+RegleOrientation(orientation: double): void
+RegleTracer(): void
+EquerrePositionX(x: int): void
+EquerrePositionY(y: int): void
+EquerreTransparence(b: bool): void
+EquerreOrientation(orientation: double): void
+EquerreTracer(): void
+CompasPositionX(x: int): void
+CompasPositionY(y: int): void
+CompasEcartement(ecartement: int): void
+CompasOrientation(orientation: double): void
Geometrie
+compas: Compas
+crayon: Crayon
+equerre: Equerre
+regle: Regle
+tableauFigure: QList <Figure *>
+modifCompas: bool
+modifCrayon: bool
+modifEquerre: bool
+modifRegle: bool
+grille: bool
-ordreInstruments: char
-nomFichier: QString
-nomFichierTemp: QString
-nombFigure: int
<<create>>-Geometrie(parent: QWidget)
<<destroy>>-Geometrie()
+gererRegle(): bool
+gererEquerre(): bool
+gererCompas(): bool
+gererCrayon(): bool
+paintEvent(event: QPaintEvent): void
+mousePressEvent(clic: QMouseEvent): void
+mouseMoveEvent(move: QMouseEvent): void
-dessinerFigure(dessinTrait: QPainter): void
Instrument
+moveSelected: bool
+rotateSelectedRight: bool
+rotateSelectedLeft: bool
#posX: int
#posY: int
#angle: int
#offsetX: int
#offsetY: int
#posClicX: int
#posClicY: int
#oldRotationValue: int
#nomFichierXML: QString
#nomDocument: QString
#nomElement: QString
#pointGeometrie: Geometrie
<<CppMacro>>-()
<<create>>-Instrument(parent: QWidget)
+clic(clic: QMouseEvent, boutonRotation: bool): void
+move(move: QMouseEvent): void
+deselectionner(): void
+translation(newx: double, newy: double): void
+rotation(newangle: double): void
#Initialisation(): void
Compas
<<create>>-Compas(geometri: Geometrie)
<<destroy>>-Compas()
+angleEcartement(ecart: double, longueurBranche: double): double
+hauteurCompas(ecart: double, longueurBranche: int): int
+tracer(): void
+ecartement(e: double): void
+dessinerCompas(dessin: QPainter): void
Crayon
<<create>>-Crayon(geometri: Geometrie)
+transparence(b: bool): void
+dessinerCrayon(dessin: QPainter): void
Regle
<<create>>-Regle()
<<create>>-Regle(geometri: Geometrie)
<<destroy>>-Regle()
+tracer(grad1: double, grad2: double): void
+transparence(b: bool): void
+graduation(g: int): void
+dessinerRegle(dessin: QPainter): void
Equerre
<<create>>-Equerre(geometri: Geometrie)
<<destroy>>-Equerre()
+Thales(longueur: double, largeur: double, transp: double, x: int, y: int): QPointF
+dessinerEquerre(dessin: QPainter): void
Figure
<<CppMacro>>-()
<<create>>-Figure(parent: QWidget)
Arc
-rectangle: QRectF
-startAngle: int
-spanAngle: int
<<create>>-Arc()
<<create>>-Arc(X: int, Y: int, W: int, H: int, sta: int, spa: int)
<<create>>-Arc(rect: QRectF, sta: int, spa: int)
+setRect(rect: QRectF): void
+setRect(x: int, y: int, w: int, h: int): void
+setStartAngle(sta: int): void
+setSpanAngle(spa: int): void
+getRectangle(): QRectF
+getStart(): int
+getSpan(): int
Ligne
-line: QLineF
<<create>>-Ligne()
<<create>>-Ligne(ligne: QLineF)
+getQLine(): QLineF
+setQLine(ligne: QLineF): void
User
+id_user_actuel: QString
+mdp_user_actuel: QString
+user_actuel_IsAdmin: bool
+premUtil: bool
+fichier_user_actuel: QString
<<CppMacro>>-()
<<create>>-User(parent: QDialog)
<<destroy>>-User()
+Connexion(): void
+Enregistrement(): void
+StockageVector(): void
+RemoveUtil(): void
+firstUse(): bool

Nom	Date de début	Date de fin
Format SVG	27/02/14	28/02/14
Sauvegarder/Charger	01/03/14	01/03/14
Insérer une image dans le cahier	01/03/14	02/03/14
Copier/Couper/Coller	03/03/14	03/03/14
Magnétisme	27/02/14	01/03/14
Tracé Crayon	02/03/14	02/03/14
Gomme/Viewport	27/02/14	02/03/14
Aperçu/Impression	27/02/14	02/03/14
Doc utilisateur/programmeur	01/03/14	02/03/14
Rédaction du rapport/Préparation de la soutenance	27/02/14	20/03/14

Légende :

Couleur	*Membres*
	David
	Pierre/Ravel
	Dylan
	Hugo
	L'équipe entière