

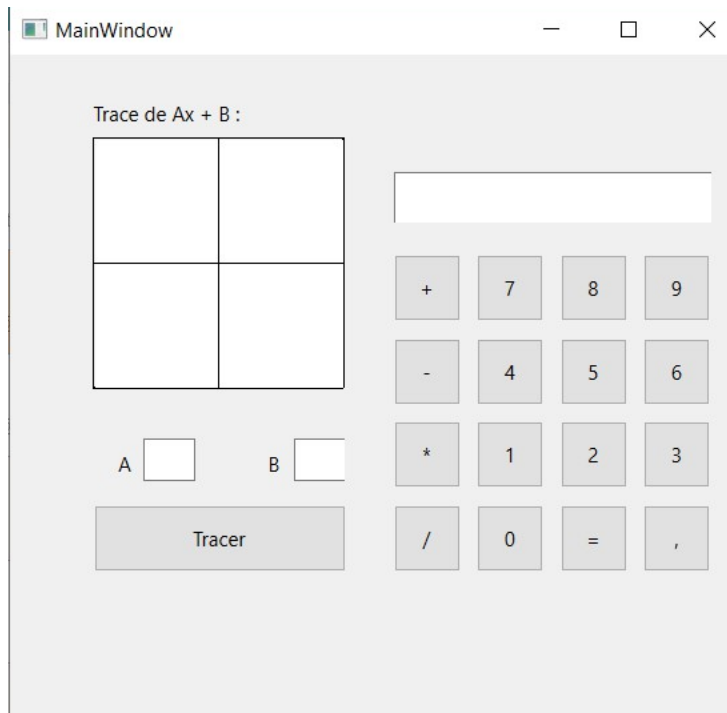
# COMPTE RENDU

Calculatrice – Projet IHM

Félix Baubriaud  
Corentin Prigent

# Interface et diagramme d'objet

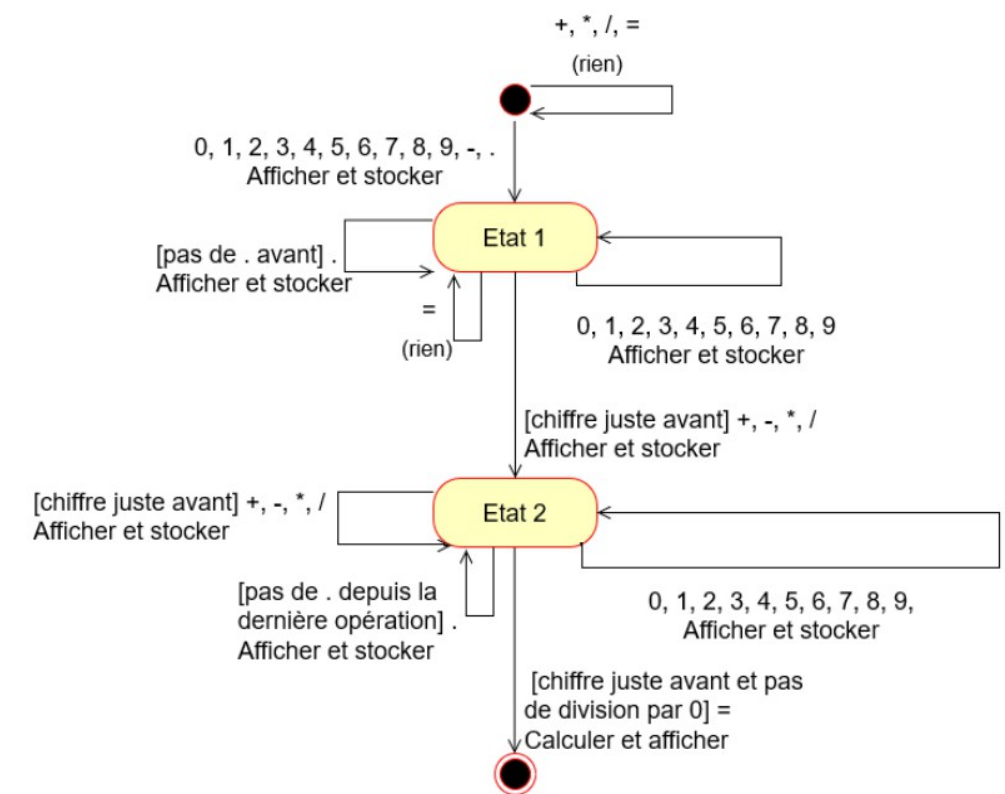
Voici l'interface de notre calculatrice :



Tous les boutons semblent fonctionner. Nous avons interdit certains cas comme mettre plus de deux opérandes, taper = alors qu'il n'y a pas de calcul à faire ou après un opérateur, mettre plusieurs virgules à un nombre et mettre deux opérateurs d'affilé. En revanche, nous avons autorisé de commencer l'opération par un - (pour un nombre négatif), de mettre une virgule sans aucun chiffre avant (,12 est équivalent à 0,12) et la division par 0 affiche une erreur à l'écran.

Attention problème rencontré : pour la fonction stof (trouvée sur internet) qui permet de convertir un string en float, le séparateur décimal est la virgule sur les ordinateurs de Polytech, mais sur mon ordinateur par exemple c'est le point. Pour cela, nous avons créé une variable `m_v` qu'on a mis en attribut pour contenir la virgule. Si les calculs ne prennent pas en compte la virgule, il suffit de changer dans le constructeur de MainWindow (dernière ligne) `m_v = ','` par `m_v = '.'`.

Diagramme d'objet :



Notre diagramme d'objet modélise le cas où l'on peut faire autant d'opération que l'on souhaite, alors que notre calculatrice n'effectue qu'une opération à la fois.

## Partie de droite de la calculatrice

Les boutons de la calculatrice sont cliquables et l'évènement est différent selon le bouton :

-Pour les boutons de 0 à 9, quelque soit le cas, on ajoute le chiffre (concaténation) à l'attribut `m_aff` de notre classe, puis on affiche le contenu de `m_aff` à l'écran, `m_aff` étant un `QString` initialisé à une chaîne vide au départ.

-Pour les boutons `+`, `-`, `*` et `/`, on commence par convertir notre `QString` `m_aff` en `string` pour pouvoir utiliser certaines fonctions. Comme pour les chiffres, on ajoute l'opérateur à `m_aff` et on affiche `m_aff` à l'écran mais sous certaines conditions. La première est qu'il ne doit pas déjà y avoir d'opérateur dans `m_aff`, à part s'il s'agit du moins au début de la chaîne. Pour cela, nous avons créé un attribut booléen `m_op` que nous initialisons à `false` et que nous mettons à `true` lorsqu'on met un opérateur (à part s'il s'agit du moins au début de la chaîne). La deuxième condition est qu'il doit y avoir un chiffre avant. On vérifie cela avec la fonction `isdigit`, après s'être assuré que la chaîne `m_aff` est non nulle pour accéder à son dernier élément par `.size`. On a également utilisé la méthode `.at` qui est l'équivalent de l'opérateur `[]` sur les chaînes de caractère.

-Pour le bouton `,` : on convertit le `QString` `m_aff` en `string`. S'il n'y a encore rien d'affiché à l'écran, on peut directement le stocker et l'afficher comme pour les autres boutons. Sinon, on regarde s'il y a une autre virgule avant, en partant de la fin de la chaîne et jusqu'au début ou jusqu'à ce qu'on trouve un opérateur. S'il n'y en a pas, on peut l'afficher.

-Pour le bouton `=` : on convertit le `QString` `m_aff` en `string`. On commence par chercher l'indice de l'opérateur dans la chaîne, puis on vérifie que le dernier caractère est un chiffre. On découpe la chaîne en 3 parties à l'aide de la fonction `substr` : les premier et deuxième opérandes que l'on convertit en `float` avec la fonction `stof`, et l'opérateur. On calcule alors selon l'opérateur le résultat, que l'on stocke dans l'attribut `m_res`. Dans le cas de la division, on regarde si le deuxième opérande vaut 0 et on affiche une erreur dans ce cas. Enfin, on convertit notre résultat en `QString` afin de l'afficher à l'écran et on réinitialise les paramètres pour pouvoir effectuer d'autres opérations.

Tests :

4.2\*4

+

7

8

9

-

4

5

6

\*

1

2

3

/

0

=

,

→ on tape = →

16.8

+

7

8

9

-

4

5

6

\*

1

2

3

/

0

=

,

-7\*.01

+

7

8

9

-

4

5

6

\*

1

2

3

/

0

=

,

→ on tape = →

-0.07

+

7

8

9

-

4

5

6

\*

1

2

3

/

0

=

,

20/6

+

7

8

9

-

4

5

6

\*

1

2

3

/

0

=

,

→ on tape = →

3.33333

+

7

8

9

-

4

5

6

\*

1

2

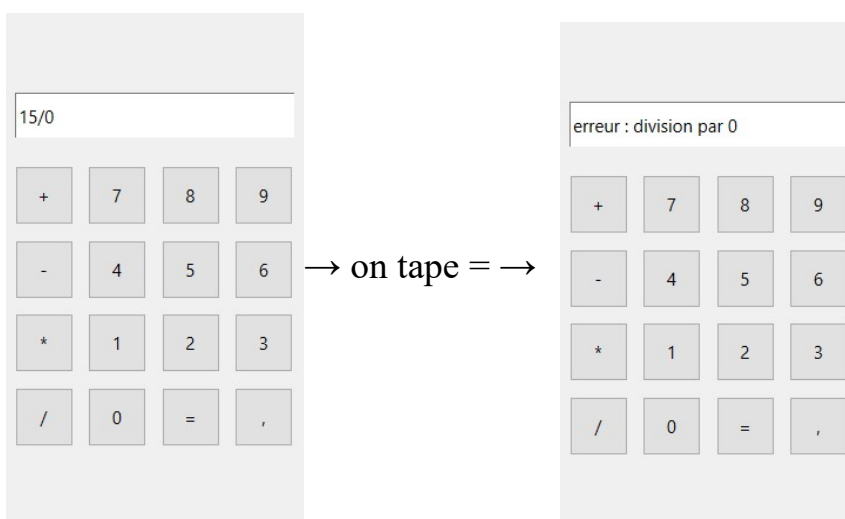
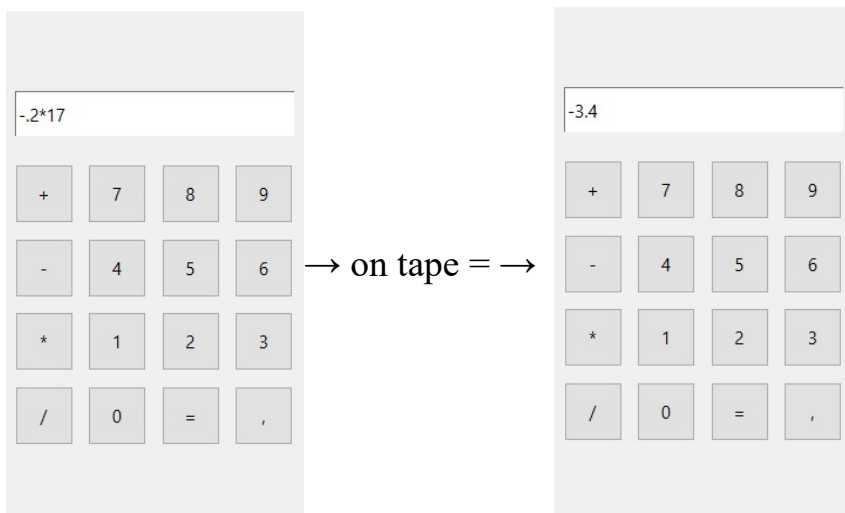
3

/

0

=

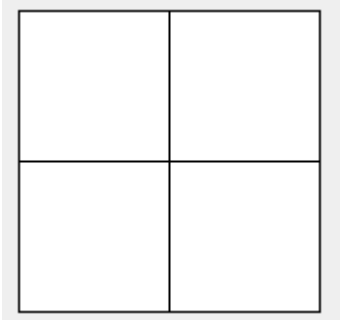
,



# Partie gauche : graphique

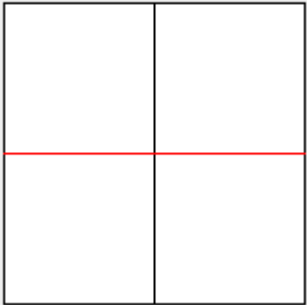
## Introduction

Écran graphe :



Taille 150x150, carré de position (50,50) à (200,200). Sur un repaire (i,-j) i.e. Pour tout (x,y) de  $\mathbb{R}^2$ ,  $(x,y) \sim (x,-y)$  sur  $\mathbb{R}'$ . Et d'élément neutre (125,-125) donc  $\mathbb{R}''((125,-125),(i,-j))$  i.e. (x,y) sur  $\mathbb{R}^2 \sim (x+125,-y-125)$  sur  $\mathbb{R}''$

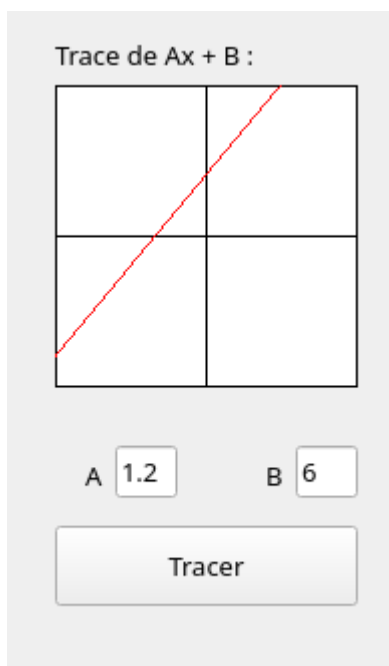
Trace de  $Ax + B$  :

A 2x2 grid of squares, representing a coordinate system. A red horizontal line is drawn across the middle of the grid, passing through the center of the four squares.

A  B

Tracer

A et B sont mis directement à 0 par défaut ce qui donne une droite constante du type  $y=0$



1 graduation = 5 “emplacements” sur l’interface graphique.  
i.e. graphe 15x15 pour l’affichage



## Développement

Bouton et line edits permettant la mise en marche du graphe :

```
void MainWindow::on_coeff_textEdited(const QString &coeff)
{
    m_coeff = coeff.toFloat();
}
```

```
void MainWindow::on_affine_textEdited(const QString &affine)
{
    m_affine=affine.toFloat();
}
```

```
void MainWindow::on_Tracer_clicked()
{
    bl=true;
    repaint();
}
```

- m\_coeff attribut de la classe MainWindow correspond à A et il est converti de QString à Float
- m\_affine attribut de la classe MainWindow correspond à B et idem pour la conversion
- bl booleen attribut de classe MainWindow est le trigger permettant de tracer la droite

Méthode paintEvent:

Interface graphique du graphe sans la droite :

```
QPainter painter(this);
painter.drawText(50,40,QString("Trace de Ax + B :"));
painter.drawText(65,250,QString("A"));
painter.drawText(155,250,QString("B"));
QPainter myline(this);
myline.drawRect(50,50,150,150);
myline.fillRect(51,51,149,149,QBrush(Qt::white));
myline.drawLine(125,50,125,200);
myline.drawLine(50,125,200,125);
```

La droite :

```
if (bl)
{
    float Ax = 50; //Point sur le bord gauche
    float Ay = 75*m_coeff-m_affine*5+125; //obtenu avec y = ax + b

    if (Ay<50 || Ay>200) //si on depasse du graphe
    {
        Ay = 50; //Point sur le bord du haut
        Ax = (75-m_affine*5)/m_coeff + 125; //obtenu avec y = ax + b
        if (Ax<50 || Ax>200)
        {
            Ay = 200; //Point sur le bord du bas
            Ax = (-75-m_affine*5)/m_coeff + 125; //1 unite = 5
        }
    }

    float Bx = 200; //Point sur le bord droit
    float By = -75*m_coeff-m_affine*5+125; //1 unite = 5

    if (By<50 || By>200) //si on depasse du graphe
    {
        By = 200; //Point sur le bord du bas
        Bx = (-75-m_affine*5)/m_coeff + 125; //1 unite = 5
        if (Bx<50 || Bx>200) //si on depasse du graphe
        {
            By = 50; //Point sur le bord du haut
            Bx = (75-m_affine*5)/m_coeff + 125; //obtenu avec y = ax + b
        }
    }

    myline.setPen( QPen(Qt::red, 1) );
    myline.drawLine(Ax,Ay,Bx,By);
    bl=false;
}
```

Tant que 'Tracer' n'est pas "poussé" bl=false ce qui n'active pas la condition dans paintEvent, la fonction repaint() de Tracer permet de réinitialiser cette méthode avec bl=true cette fois-ci.

L'élément neutre est le point (125,125) dans le sens de l'interface Qt, mais (125,-125) dans le sens d'un repère orthonormé classique donc pour  $x'=x-125$  et  $y'=y+125$  on arrive du coup à (0,0) sur R' avec  $x=125$  et  $y=-125$ . De plus, le sens de repère est (i,-j) donc  $x''=x'$  et  $y''=-y'$  sur R''. R'' est notre graphe actuel. Donc  $y=-A*(x-125)+125-B$  et  $x=(-(y+B)+125)/A+125$

Pour tracer les droites on a pris les points aux extrémités donc pour x, on a :

- $x=50, y=A*75+125-B$
- $x=200, y=-A*75+125-B$

Pour y, on a :

- $y=50, x=(75-B)/A+125$
- $y=200, x=-(75+B)/A+125$

On choisit les extrémités en regardant si la courbe dépasse.

On a pris 1 unité = 5 ce qui ne change pas le coefficient directeur mais l'ordonnée à l'origine est multiplié par 5.

# Conclusion

Ce TP nous a permis de nous familiariser avec le logiciel Qt Creator. C'est aussi un bon exercice de C++ sur les classes, chaîne de caractère, conversion... Nous avons été bloqué à plusieurs reprises, mais en recherchant dans le cours et sur internet nous avons réussi à résoudre nos problèmes. Nous voulions au départ permettre à l'utilisateur faire plusieurs opérations en une seule ligne (d'où le diagramme d'objet), mais cela avait l'air compliqué. En effet, dans le cas où l'on mélange les opérateurs +, - avec \* et /, il faut d'abord calculer les multiplications et divisions ce qui complique l'implémentation. En ce qui concerne le graphique, nous avons mis du temps à réussir à afficher la courbe à l'appui du bouton "Tracer", puis nous avons réussi à l'aide d'un booléen. Nous avons choisi d'utiliser des conditions pour que la courbe ne dépasse pas, mais il aurait peut-être été plus simple de cacher ce qui dépasse en mettant un fond de couleur. Pour finir, ce travail va sans doute nous être utile pour l'implémentation du jeu Tetris.