UEMF
الجامعة الأورومتوسطية بفاس
EUROMED UNIVERSITY OF FES
UNIVERSITÉ EUROMED DE FÈS

EIDIA
École d'Ingénierie Digitale et d'Intelligence Artificielle

Signal and Slots

Sommaire :

- *Calculator*
- *TrafficLight*
- *DigitalClock*

Objectif :

L'objectif est de combiner le C++ de base avec certaines fonctionnalités de base de QT pour créer quelques applications d'interface utilisateur graphique (GUI) et découvrez comment établir des connexions entre des objets.

Réalisée par :EDDOUKS OUMAYMA

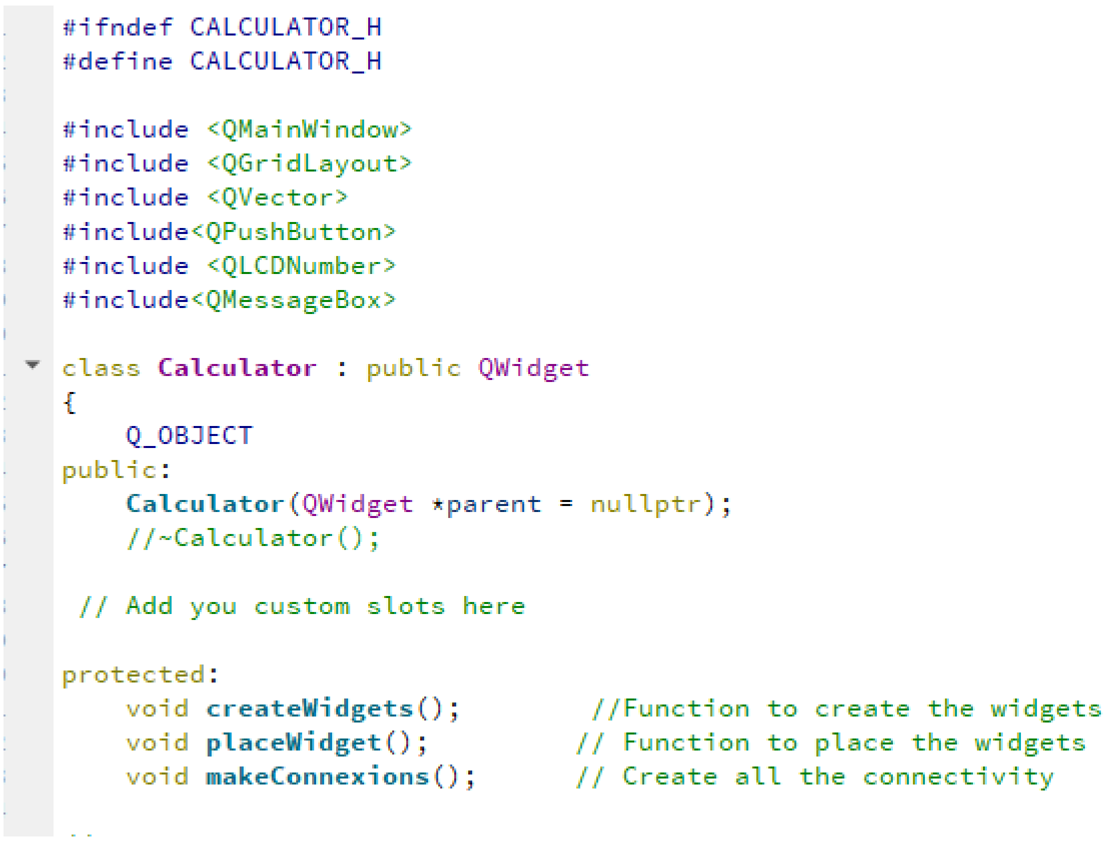
Calculator

- *Cet exercice fait suite pour ajouter une fonctionnalité **interactive** aux widgets **de calculatrice** écrits dans les devoirs précédents.*
  1. *L'objectif est d'utiliser des signaux et des emplacements pour simuler un comportement de calculatrice de base. Les opérations prises en charge sont .`*, +, -, /`*

  2. *améliorer les capacités de la calculatrice :*

- *Ajout du bouton de réinitialisation (C).*
- *Ajoutez un ensemble d'opérations*

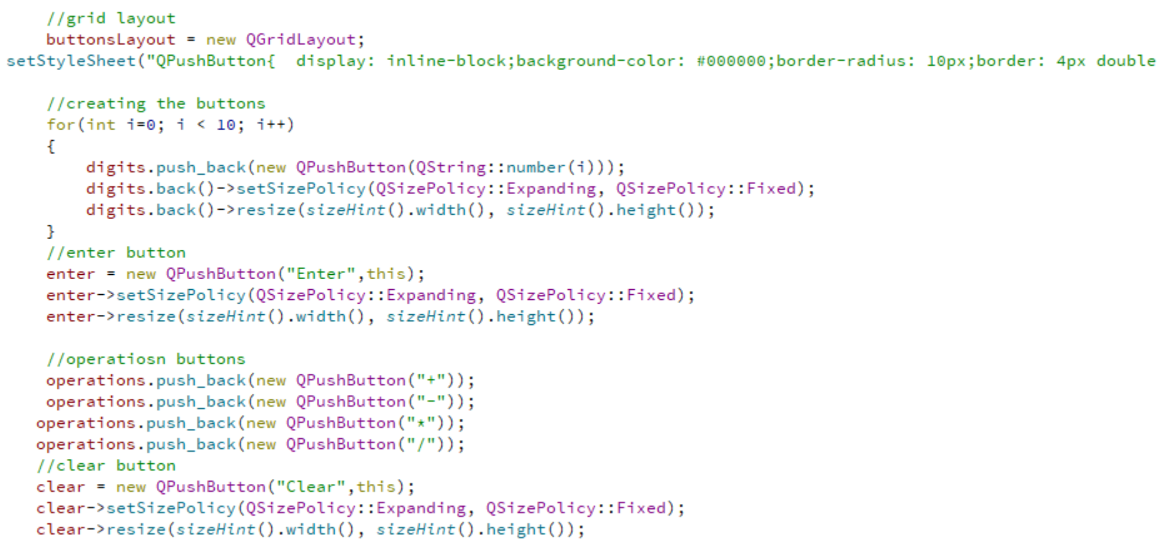
Calculator.h



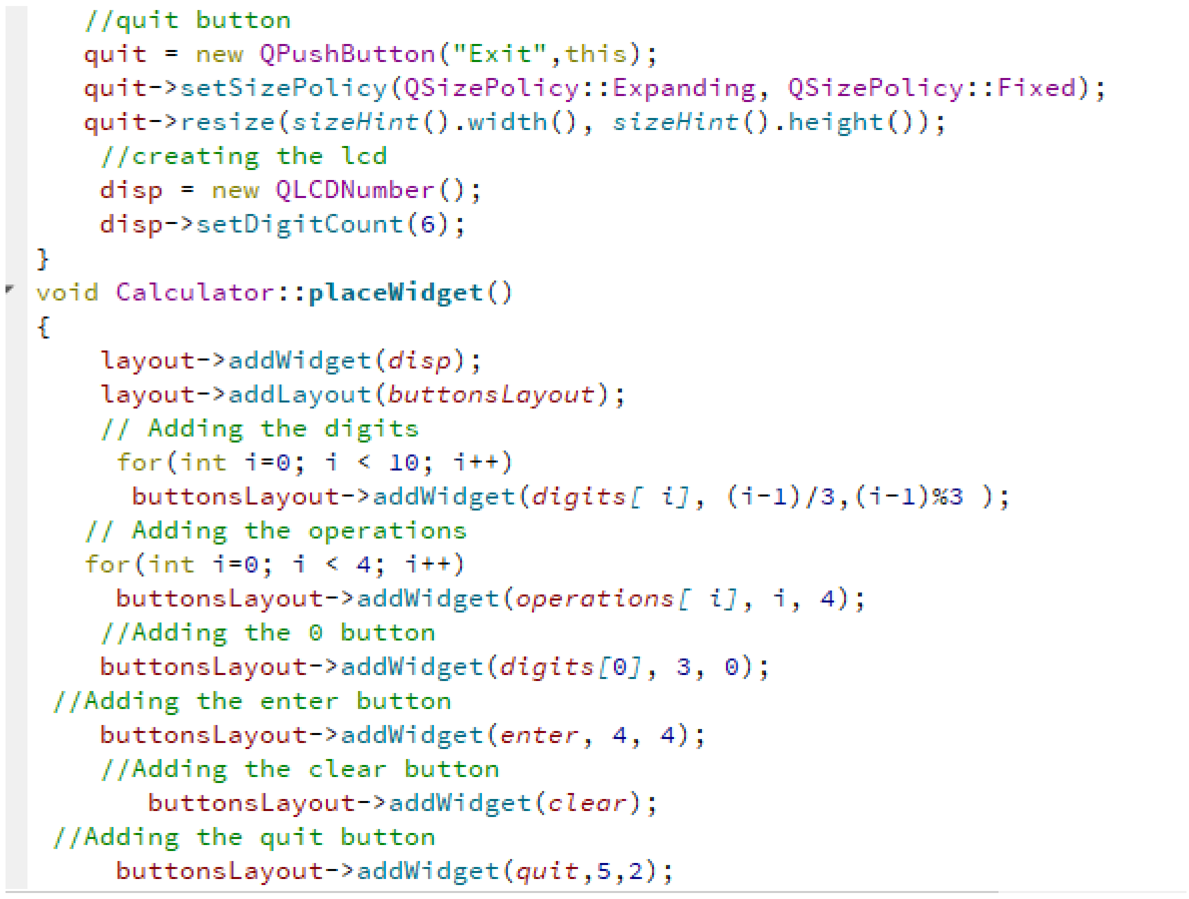
```
#ifndef CALCULATOR_H
#define CALCULATOR_H

#include <QMainWindow>
#include <QGridLayout>
#include <QVector>
#include<QPushButton>
#include <QLCDNumber>
#include<QMessageBox>

class Calculator : public QWidget
{
    Q_OBJECT
public:
    Calculator(QWidget *parent = nullptr);
    //~Calculator();

 // Add you custom slots here

protected:
    void createWidgets();         //Function to create the widgets
    void placeWidget();          // Function to place the widgets
    void makeConnexions();       // Create all the connectivity
protected:
    void keyPressEvent(QKeyEvent *e)override;     //Override the keypress events
public slots:
       void newDigit();
        void changeOperation();
        void Enter();
         void Quit();
         void clearAll();
        //Slot to handle the click on operations
signals:
       void digitClicked(int digit);
private:
       QPushButton *createButton(const QString &text, const char *member);

    QGridLayout *buttonsLayout; // layout for the buttons
    QVBoxLayout *layout;        //main layout for the button
    QVector<QPushButton*> digits;  //Vector for the digits
    QPushButton *enter;            // enter button
    QPushButton *quit;
    QPushButton *clear;
    QVector<QPushButton*> operations; //operation buttons
    QLCDNumber *disp; // Where to display the numbers
    double * left;          //left operand
        double * right;          // right operand
        QString *operation;  // Pointer on the current operation
};
#endif // CALCULATOR_H
```

Calculator.cpp

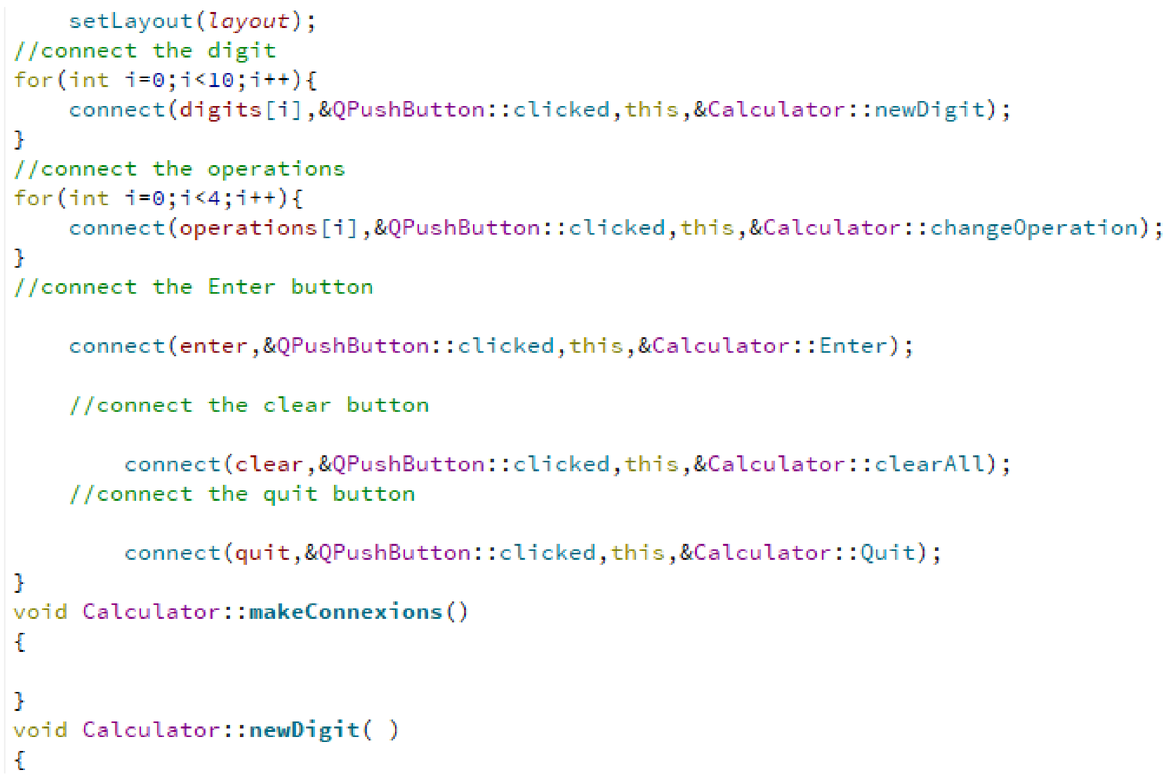
```
#include "calculator.h"
#include <QKeyEvent>
#include <QApplication>
#include <QObject>
#include<QString>
QString evaluate;

Calculator::Calculator(QWidget *parent)
    : QWidget(parent)
{
    createWidgets();
    placeWidget();
    makeConnexions();

    left=nullptr;
    right=nullptr;
    operation=nullptr;
}
void Calculator::createWidgets()
{
    //Creating the layouts
    layout = new QVBoxLayout();
    layout->setSpacing(2);
```



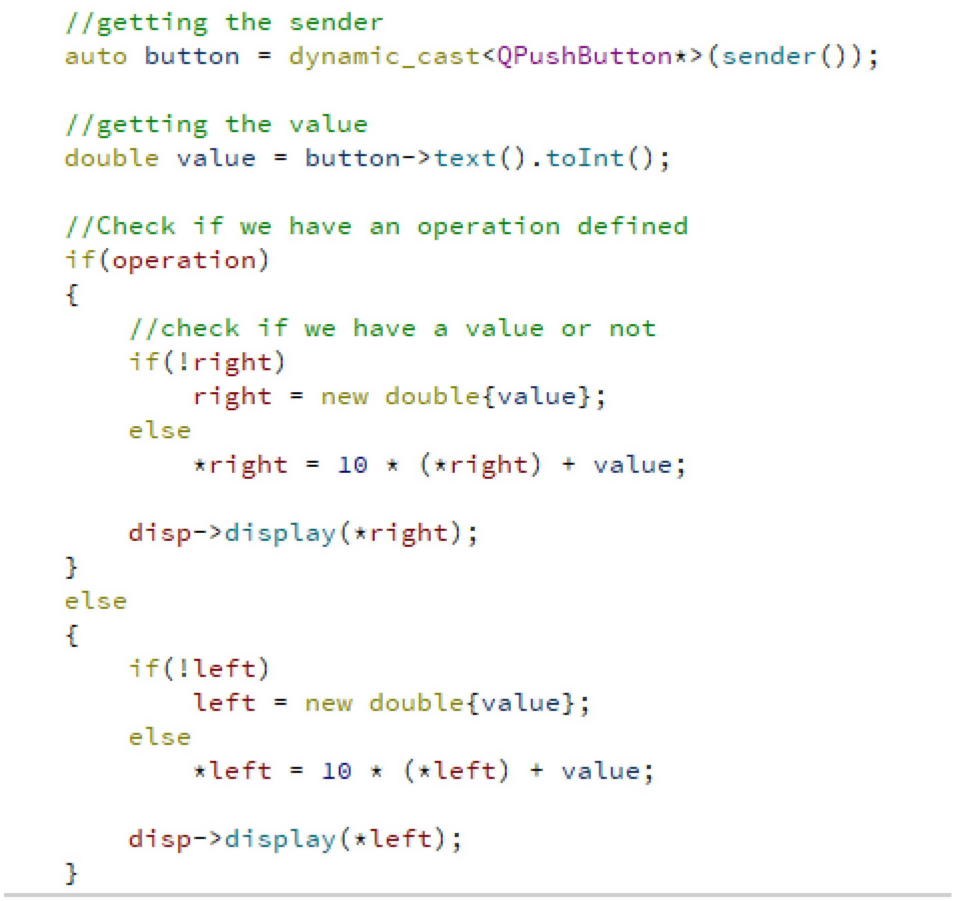
```
    //grid layout
    buttonsLayout = new QGridLayout;
setStyleSheet("QPushButton{  display: inline-block;background-color: #000000;border-radius: 10px;border: 4px double

    //creating the buttons
    for(int i=0; i < 10; i++)
    {
        digits.push_back(new QPushButton(QString::number(i)));
        digits.back()->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
        digits.back()->resize(sizeHint().width(), sizeHint().height());
    }
    //enter button
    enter = new QPushButton("Enter",this);
    enter->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
    enter->resize(sizeHint().width(), sizeHint().height());

    //operatiosn buttons
    operations.push_back(new QPushButton("+"));
    operations.push_back(new QPushButton("-"));
   operations.push_back(new QPushButton("*"));
   operations.push_back(new QPushButton("/"));
   //clear button
   clear = new QPushButton("Clear",this);
   clear->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
   clear->resize(sizeHint().width(), sizeHint().height());
```



```
    //quit button
    quit = new QPushButton("Exit",this);
    quit->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
    quit->resize(sizeHint().width(), sizeHint().height());
     //creating the lcd
     disp = new QLCDNumber();
     disp->setDigitCount(6);
}
void Calculator::placeWidget()
{
     layout->addWidget(disp);
     layout->addLayout(buttonsLayout);
     // Adding the digits
      for(int i=0; i < 10; i++)
       buttonsLayout->addWidget(digits[ i], (i-1)/3,(i-1)%3 );
    // Adding the operations
    for(int i=0; i < 4; i++)
      buttonsLayout->addWidget(operations[ i], i, 4);
     //Adding the 0 button
     buttonsLayout->addWidget(digits[0], 3, 0);
  //Adding the enter button
     buttonsLayout->addWidget(enter, 4, 4);
     //Adding the clear button
        buttonsLayout->addWidget(clear);
  //Adding the quit button
      buttonsLayout->addWidget(quit,5,2);
```



```
    setLayout(layout);
//connect the digit
for(int i=0;i<10;i++){
    connect(digits[i],&QPushButton::clicked,this,&Calculator::newDigit);
}
//connect the operations
for(int i=0;i<4;i++){
    connect(operations[i],&QPushButton::clicked,this,&Calculator::changeOperation);
}
//connect the Enter button

    connect(enter,&QPushButton::clicked,this,&Calculator::Enter);

    //connect the clear button

        connect(clear,&QPushButton::clicked,this,&Calculator::clearAll);
    //connect the quit button

        connect(quit,&QPushButton::clicked,this,&Calculator::Quit);
}
void Calculator::makeConnexions()
{

}
void Calculator::newDigit( )
{
```

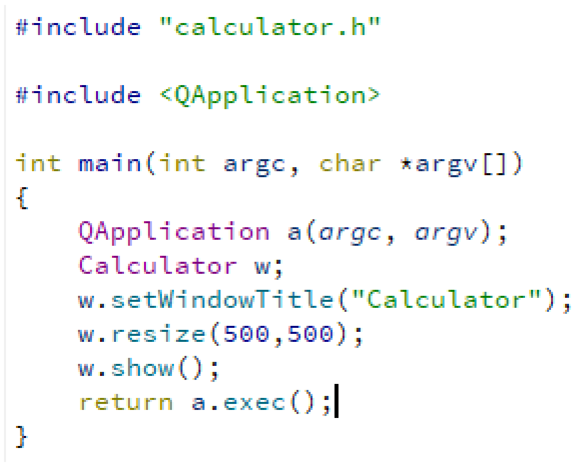


```
    //getting the sender
    auto button = dynamic_cast<QPushButton*>(sender());

    //getting the value
    double value = button->text().toInt();

    //Check if we have an operation defined
    if(operation)
    {
        //check if we have a value or not
        if(!right)
            right = new double{value};
        else
            *right = 10 * (*right) + value;

        disp->display(*right);
    }
    else
    {
        if(!left)
            left = new double{value};
        else
            *left = 10 * (*left) + value;

        disp->display(*left);
    }
```

```
}
void Calculator::changeOperation()

{
    //Getting the sender button
       auto button = dynamic_cast<QPushButton*>(sender());

       //Storing the operation
       operation = new QString{button->text()};

       //Initiating the right button
       right = new double{0};

       //reseting the display
       disp->display(0);
}

void Calculator::Enter(){
    if(*operation=="+"){

        disp->display(*left+(*right));*left=*left+(*right);
    }else if(*operation=="-"){
        disp->display(*left-(*right));*left=*left-(*right);
   } else if (*operation=="*"){
            disp->display(*left*(*right));*left=*left*(*right);
   } else{
```

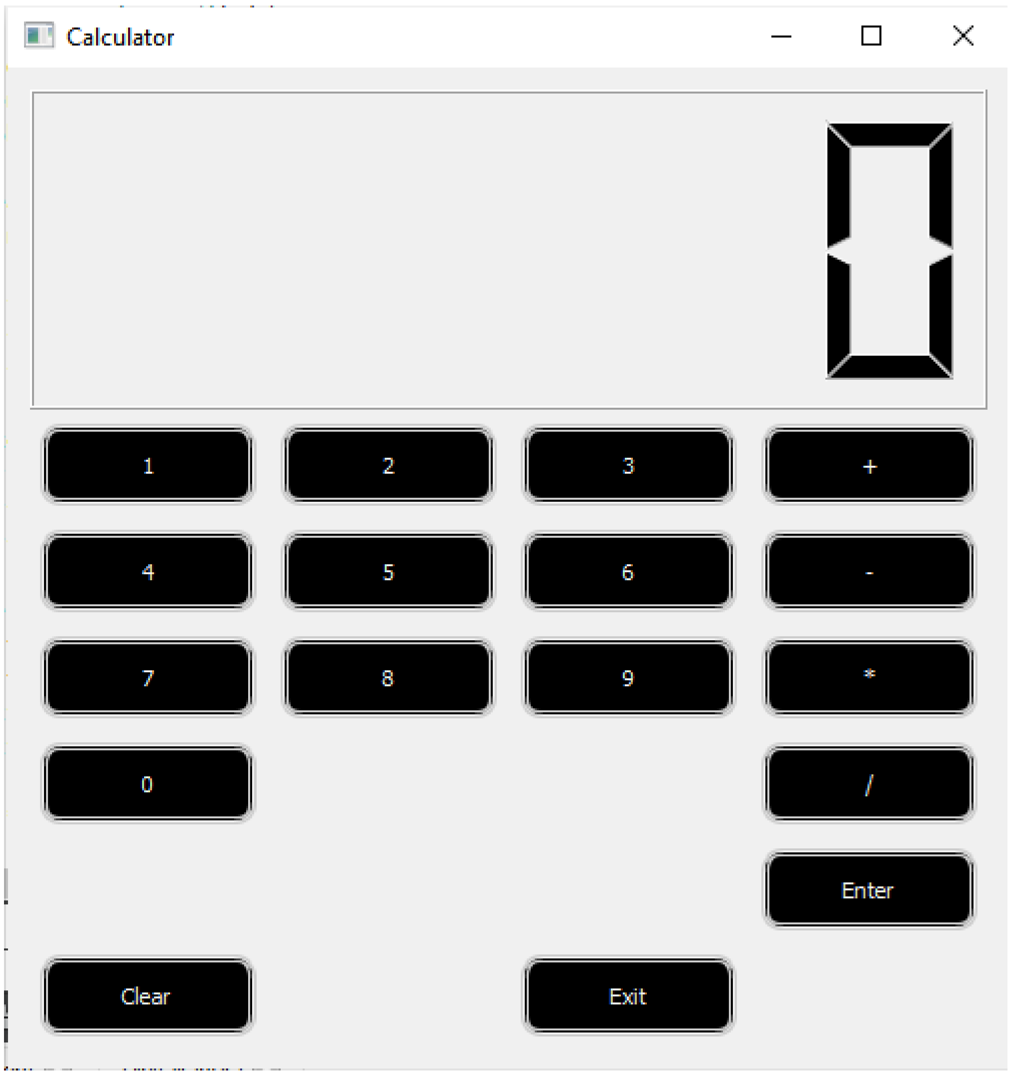
```
                disp->display(*left/(*right));*left=*left/(*right);}
 //  waitingForOperand = true;
}

void Calculator::clearAll()
{
disp->display("");
evaluate ="";

}
void Calculator::Quit(){
    QMessageBox messageBox;
        messageBox.setWindowTitle(tr("Calculator"));
        messageBox.setText(tr("Do you really want to quit?"));
        messageBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);
        messageBox.setDefaultButton(QMessageBox::No);
        if (messageBox.exec() == QMessageBox::Yes)
            qApp->quit();
}
void Calculator::keyPressEvent(QKeyEvent *e)
{
    //Exiting the application by a click on space
    if( e->key() == Qt::Key_Escape)
        qApp->exit(0);
    //You could add more keyboard interation here (like digit to button)
}
```

Calculator main



```
#include "calculator.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Calculator w;
    w.setWindowTitle("Calculator");
    w.resize(500,500);
    w.show();
    return a.exec();
}
```

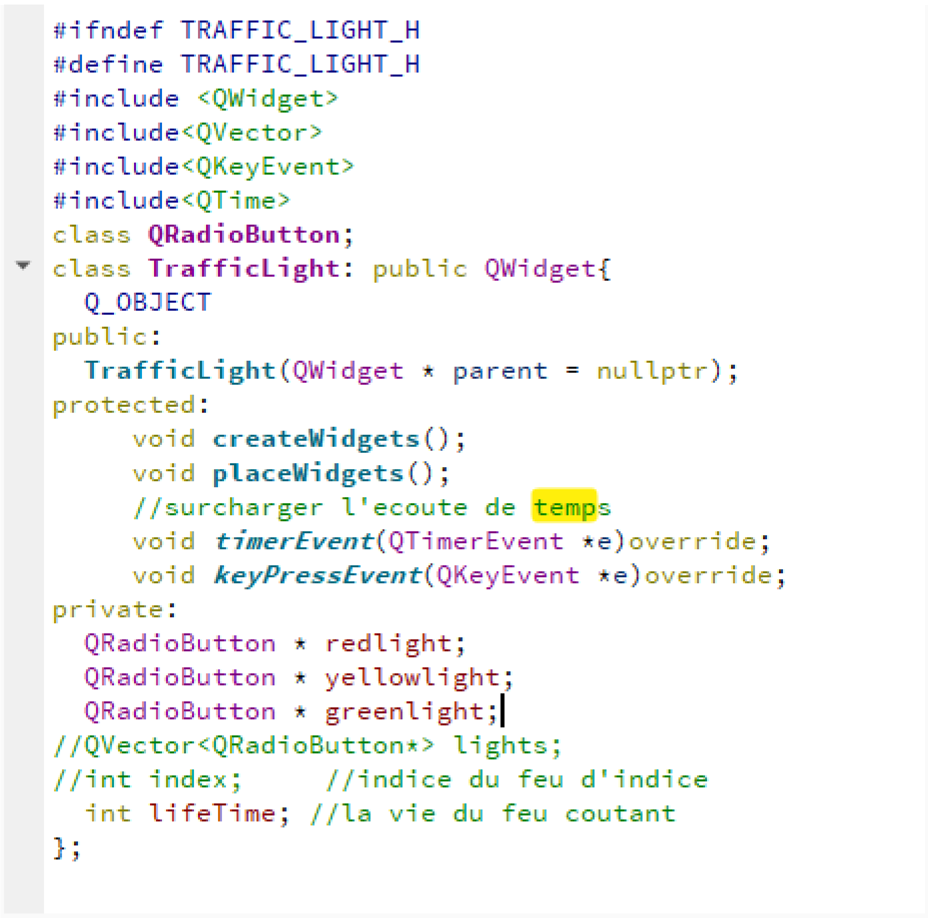
Resultat



Feu de signalisations

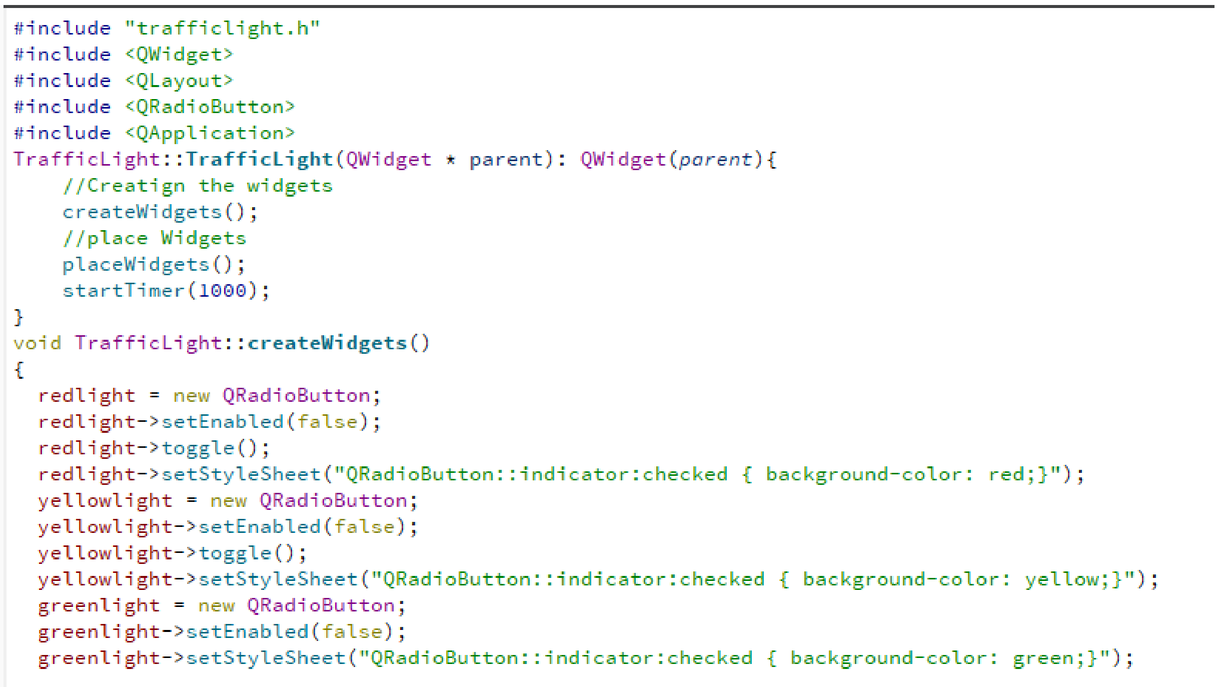
- ***Dans cet exercice, nous utiliserons le QTimer pour simuler un*** *feu de circulation*
- *On ajoute quelques fonctions afin de changer la couleur du feu* ***après chaque second*** *dans l'ordre suivant :* `Red -> Yellow-> Green`

`TrafficLight.h`

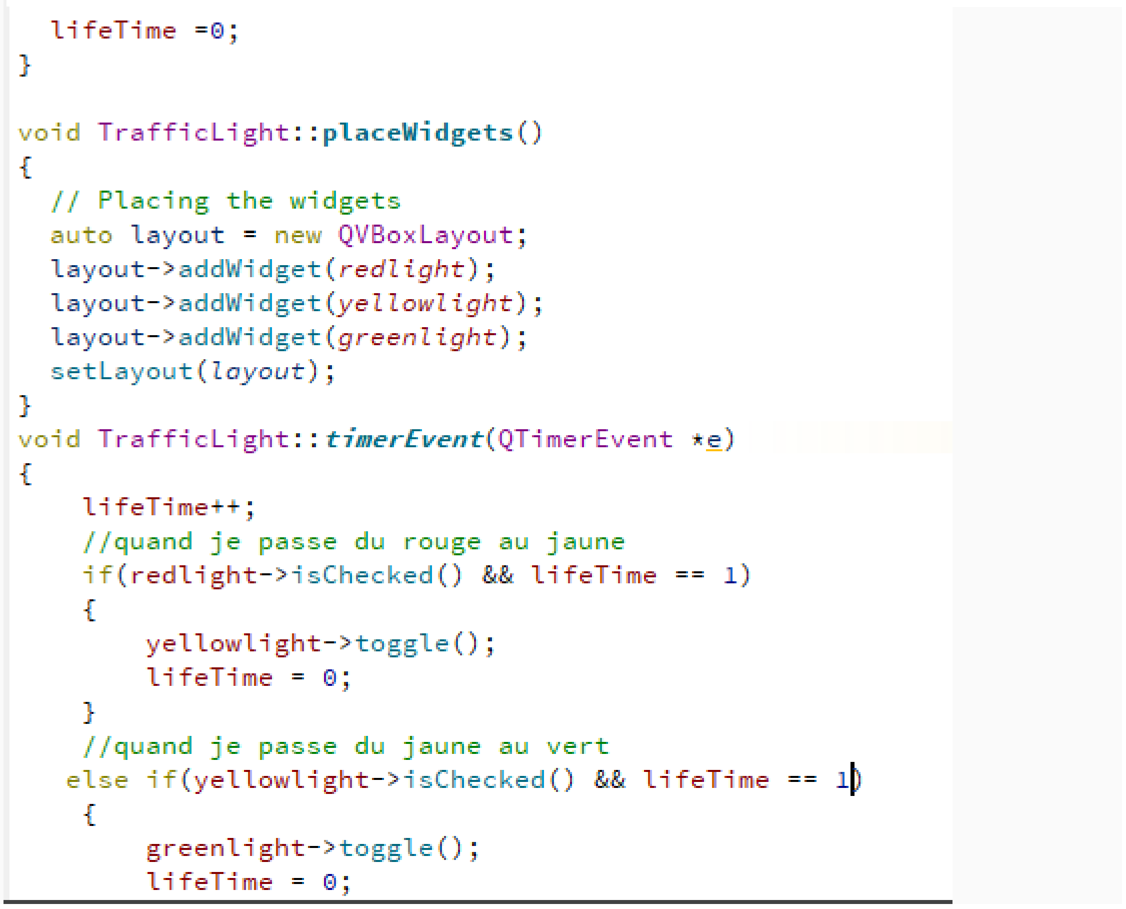


```
#ifndef TRAFFIC_LIGHT_H
#define TRAFFIC_LIGHT_H
#include <QWidget>
#include<QVector>
#include<QKeyEvent>
#include<QTime>
class QRadioButton;
class TrafficLight: public QWidget{
  Q_OBJECT
public:
  TrafficLight(QWidget * parent = nullptr);
protected:
     void createWidgets();
     void placeWidgets();
     //surcharger l'ecoute de temps
     void timerEvent(QTimerEvent *e)override;
     void keyPressEvent(QKeyEvent *e)override;
private:
  QRadioButton * redlight;
  QRadioButton * yellowlight;
  QRadioButton * greenlight;
//QVector<QRadioButton*> lights;
//int index;     //indice du feu d'indice
  int lifeTime; //la vie du feu coutant
};
```

`TrafficLight.cpp`



```
#include "trafficlight.h"
#include <QWidget>
#include <QLayout>
#include <QRadioButton>
#include <QApplication>
TrafficLight::TrafficLight(QWidget * parent): QWidget(parent){
    //Creatign the widgets
    createWidgets();
    //place Widgets
    placeWidgets();
    startTimer(1000);
}
void TrafficLight::createWidgets()
{
  redlight = new QRadioButton;
  redlight->setEnabled(false);
  redlight->toggle();
  redlight->setStyleSheet("QRadioButton::indicator:checked { background-color: red;}");
  yellowlight = new QRadioButton;
  yellowlight->setEnabled(false);
  yellowlight->toggle();
  yellowlight->setStyleSheet("QRadioButton::indicator:checked { background-color: yellow;}");
  greenlight = new QRadioButton;
  greenlight->setEnabled(false);
  greenlight->setStyleSheet("QRadioButton::indicator:checked { background-color: green;}");
```

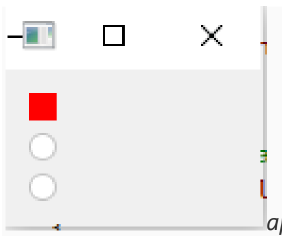


```
  lifeTime =0;
}

void TrafficLight::placeWidgets()
{
  // Placing the widgets
  auto layout = new QVBoxLayout;
  layout->addWidget(redlight);
  layout->addWidget(yellowlight);
  layout->addWidget(greenlight);
  setLayout(layout);
}
void TrafficLight::timerEvent(QTimerEvent *e)
{
    lifeTime++;
    //quand je passe du rouge au jaune
    if(redlight->isChecked() && lifeTime == 1)
    {
        yellowlight->toggle();
        lifeTime = 0;
    }
    //quand je passe du jaune au vert
   else if(yellowlight->isChecked() && lifeTime == 1)
    {
        greenlight->toggle();
        lifeTime = 0;
        lifeTime = 0;
    }
    //quand je passe du vert au rouge
   else if(greenlight->isChecked() && lifeTime == 1)
    {
        redlight->toggle();
        lifeTime = 0;
    }
}
void TrafficLight::keyPressEvent(QKeyEvent *e)
{
//if(e->key() == Qt::Key_Escape)
    //  qApp->exit();
  //else if (e->key() ==Qt::Key_R)
     //  redlight->toggle();

   //else if(e->key() == Qt::Key_Y)
    //   yellowlight->toggle();

  //else if(e->key() ==Qt::Key_G)
    //  greenlight->toggle();
}
```

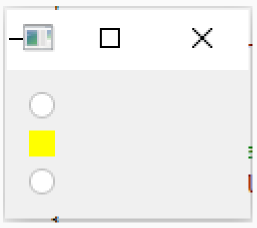
TrafficLight main

```
#include <QApplication>
#include "trafficlight.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);


    //Creating the traffic light
    auto light = new TrafficLight;


    //showing the trafic light
    light->show();

    return a.exec();
}
```

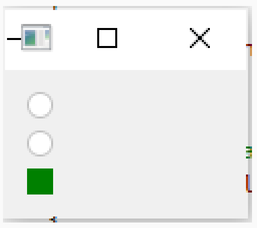
Résultat



après 1 second

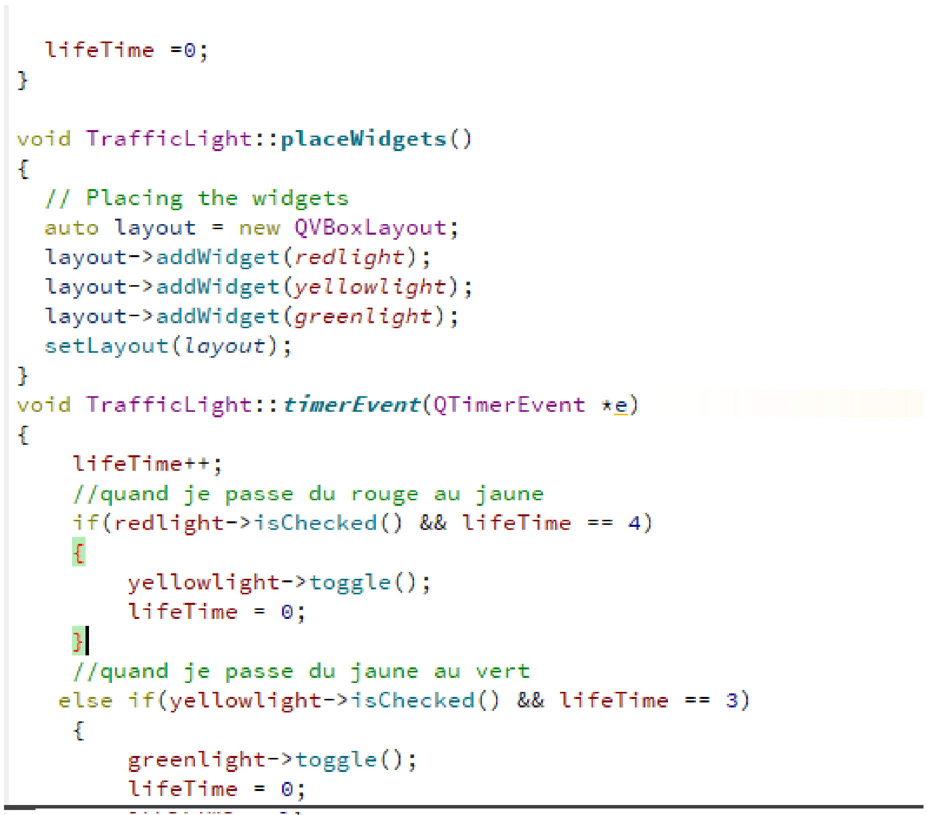


après 1 second

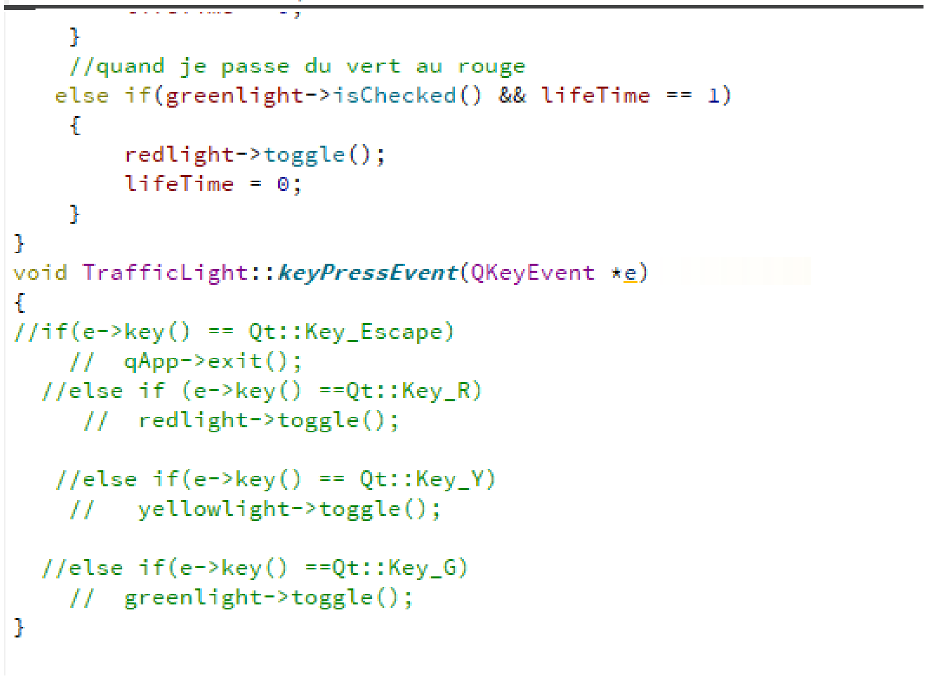


- *on change le lifeTime entre les feus de l’ordre suivant:*

```
Red ---------------> Yellow----------------> Green-------------→
        4 second après          3second après           1 second après
```

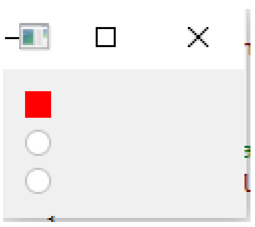


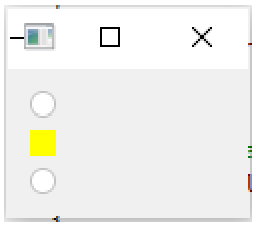
```
  lifeTime =0;
}

void TrafficLight::placeWidgets()
{
  // Placing the widgets
  auto layout = new QVBoxLayout;
  layout->addWidget(redlight);
  layout->addWidget(yellowlight);
  layout->addWidget(greenlight);
  setLayout(layout);
}
void TrafficLight::timerEvent(QTimerEvent *e)
{
    lifeTime++;
    //quand je passe du rouge au jaune
    if(redlight->isChecked() && lifeTime == 4)
    {
        yellowlight->toggle();
        lifeTime = 0;
    }
    //quand je passe du jaune au vert
   else if(yellowlight->isChecked() && lifeTime == 3)
    {
        greenlight->toggle();
        lifeTime = 0;
```

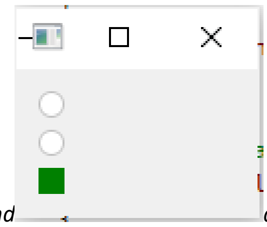


```
    }
    //quand je passe du vert au rouge
   else if(greenlight->isChecked() && lifeTime == 1)
    {
        redlight->toggle();
        lifeTime = 0;
    }
}
void TrafficLight::keyPressEvent(QKeyEvent *e)
{
//if(e->key() == Qt::Key_Escape)
    //  qApp->exit();
  //else if (e->key() ==Qt::Key_R)
     //  redlight->toggle();

   //else if(e->key() == Qt::Key_Y)
    //   yellowlight->toggle();

  //else if(e->key() ==Qt::Key_G)
    //  greenlight->toggle();
}
```

Résultat


après4 second

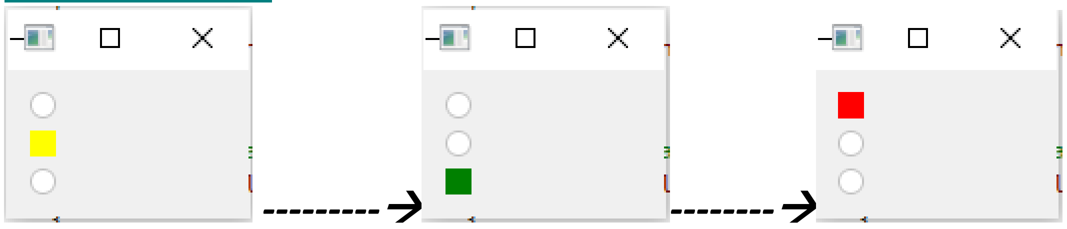

après3second


après1second

- *concernant le Key, lorsqu'on donne une couleur précise au Key(Y, G ou R) , l'ordre change débutant par la valeur du Key en respectant l'ordre précedent* : Red -> Yellow-> Green

```
void TrafficLight::keyPressEvent(QKeyEvent *e)
{
if(e->key() == Qt::Key_Escape)
      qApp->exit();
  else if (e->key() ==Qt::Key_R)
      redlight->toggle();

   else if(e->key() == Qt::Key_Y)
      yellowlight->toggle();

 else if(e->key() ==Qt::Key_G)
      greenlight->toggle();
}
```

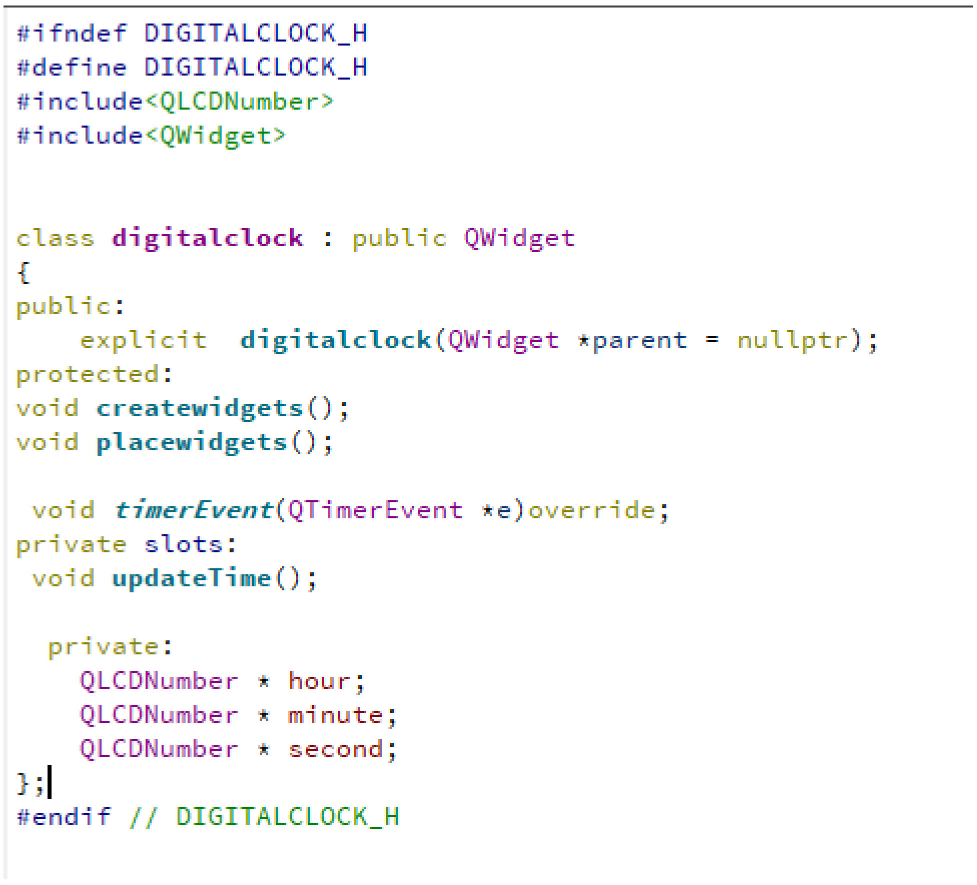
exemple Key=Y



DigitalClock

- *Une horloge numérique est un type d'horloge qui affiche l'heure numériquement,*

Digitalclock.h



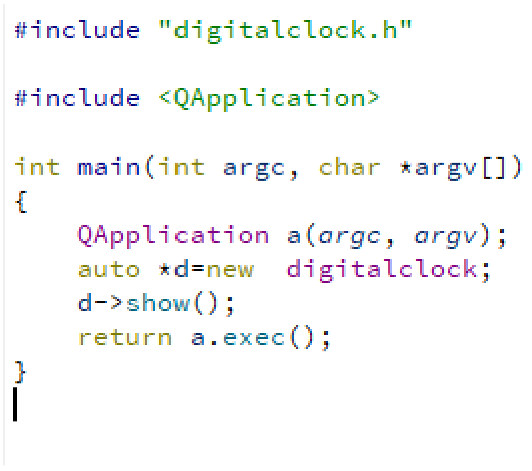
```
#ifndef DIGITALCLOCK_H
#define DIGITALCLOCK_H
#include<QLCDNumber>
#include<QWidget>


class digitalclock : public QWidget
{
public:
    explicit  digitalclock(QWidget *parent = nullptr);
protected:
void createwidgets();
void placewidgets();

 void timerEvent(QTimerEvent *e)override;
private slots:
 void updateTime();

  private:
    QLCDNumber * hour;
    QLCDNumber * minute;
    QLCDNumber * second;
};
#endif // DIGITALCLOCK_H
```

digitalclock.cpp

```
#include "digitalclock.h"
#include<QHBoxLayout>
#include<QTime>
#include<QTimer>
#include<QTimerEvent>
digitalclock::digitalclock(QWidget * parent): QWidget(parent)
{
    createwidgets();
    placewidgets();

    startTimer(1000);
    setWindowTitle(tr("DiditalClock"));
}
void digitalclock::createwidgets(){
    hour=new QLCDNumber;
    hour->setDigitCount(2);
    minute=new QLCDNumber;
   minute->setDigitCount(2);
   second=new QLCDNumber;
   second->setDigitCount(2);
   //aficher le temps courant
    updateTime();
}
void digitalclock::placewidgets(){
  QLayout *layout=new QHBoxLayout;
  setLayout(layout);
```

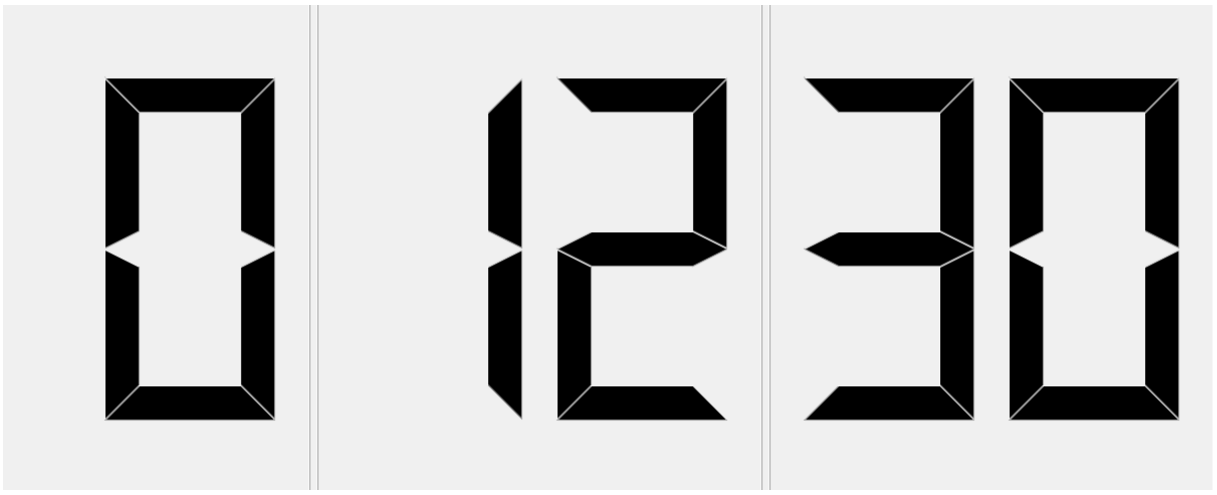
```
  layout->addWidget(hour);
  layout->addWidget(minute);
  layout->addWidget(second);
}
void digitalclock::updateTime(){

    //obtenir le temps actuel
    auto T=QTime::currentTime();
    hour->display(T.hour());
    minute->display(T.minute());
    second->display(T.second());
}
void digitalclock::timerEvent(QTimerEvent *e){
    updateTime();
}
```

digitalcloock main



```
#include "digitalclock.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    auto *d=new  digitalclock;
    d->show();
    return a.exec();
}
```

Résultat



Fin.

EDDOUKS OUMAYMA

Année universitaire : 2021 – 2022