Alkalmazás készítés Qt osztályokkal

A Qt egy keretrendszer C++ kódú, platform független, többnyelvű alkalmazások fejlesztéséhez, amely többek között magába foglal egy a grafikus felületen megjeleníthető vezérlők használatát támogató kiterjedt osztály könyvtárat.

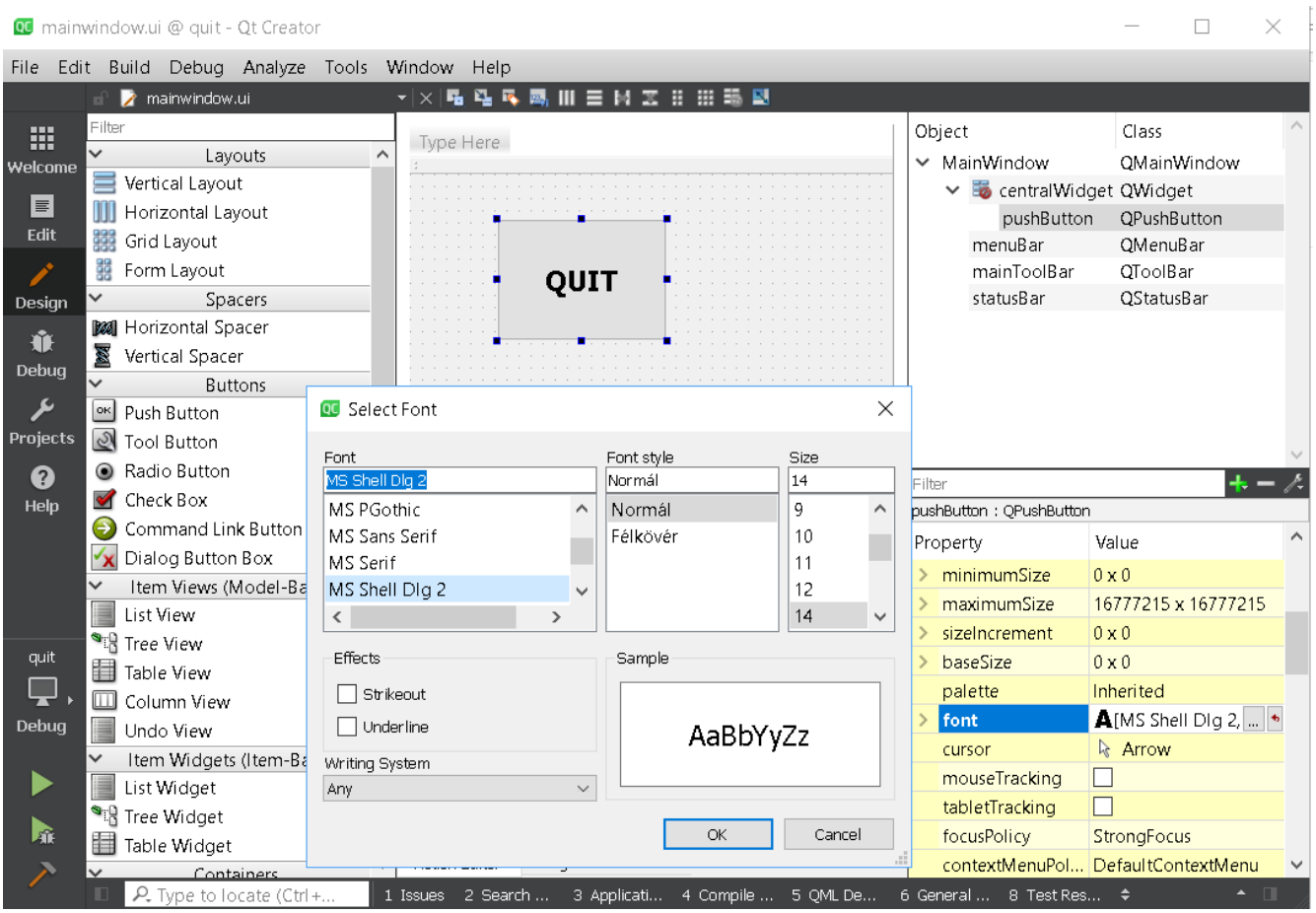
A Qt eszközök honlapja: www.trolltech.com

Cél: Ennek a gyakorlatnak az a célja, hogy megismerkedjünk a Qt néhány eszközével és azok használatával.

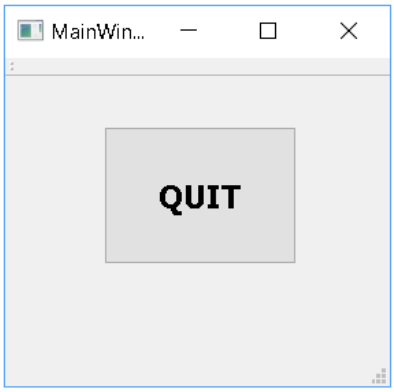
Objektumok közötti kommunikáció (quit)

A Qt Creator alkalmazás elindítása után a *File → New file or project…* menüpont kiválasztása teszi lehetővé új projekt létrehozását. Ebben a félévben főként *Qt Widget Application* típusú alkalmazások fejlesztésével foglalkozunk.
A következő lapon elnevezzük a projektet (quit) és megadjuk, hogy hol legyen fizikailag a gépen.
A Next gomb megnyomása után lehetőség van kit-et választani, ami nagyjából azt jelenti, hogy melyik Qt verzió szerint legyen lefordítva a projekt (több Qt verziót fel lehet telepíteni egy gépre). Ez a dokumentáció a Qt 4.9 alapján dolgozik, de az első alkalmazásoknál nagyjából mindegy, melyik verzió van kijelölve.
A következő lap a fájlok létrehozásához szükséges, itt lényeges, hogy a *Generate form* szöveg mellett legyen pipa. Az alap osztálynak most a *QMainWindowt* választjuk.
A következő lapon átmehetünk módosítás nélkül. *Finish* gomb.

A projekt létrejön, a bal oldali menü mutatja a jelenlegi fájljainkat. A Formunknak a QtDesigner biztosít grafikus kezelőfelületet, ezt a .ui kiterjesztésű (mainwindow.ui) fájlra duplán kattintva lehet előhozni. Ekkor bal oldalt megjelennek az egyes vezérlők, amiket a formra fel lehet pakolni. Nekünk csak egy *PushButton*-ra van szükségünk, ezt egérrel rá lehet vonszolni a formra és a méretét be lehet állítani. Duplán rákattintva a feliratát is lehet módosítani. A gombot kijelölve a Qt creator jobb oldalán megjelenik a Property Editor, ahol a kijelölt gomb különböző tulajdonságait lehet beállítani.



A gombhoz lehet hozzárendelni különböző eseménykezelőket, amik közül most a kattintás az érdekes számunkra. Ennek előhívása: egérrel jobbklikk a gombra, *Go to slot…* menüpont, majd a *clicked()* esemény kiválasztása.
A megírandó eljárás a *mainwindow.cpp* fájlba kerül, a *MainWindow* osztály részeként. Az eljárásban csak a *close()* metódust kell meghívni.
A projekt kész, már csak futtatni kell. Bal alul található egy sima zöld háromszög, ami futtatja a projektet. A Ctrl+R billentyűkombináció szintén ugyanezt végzi.

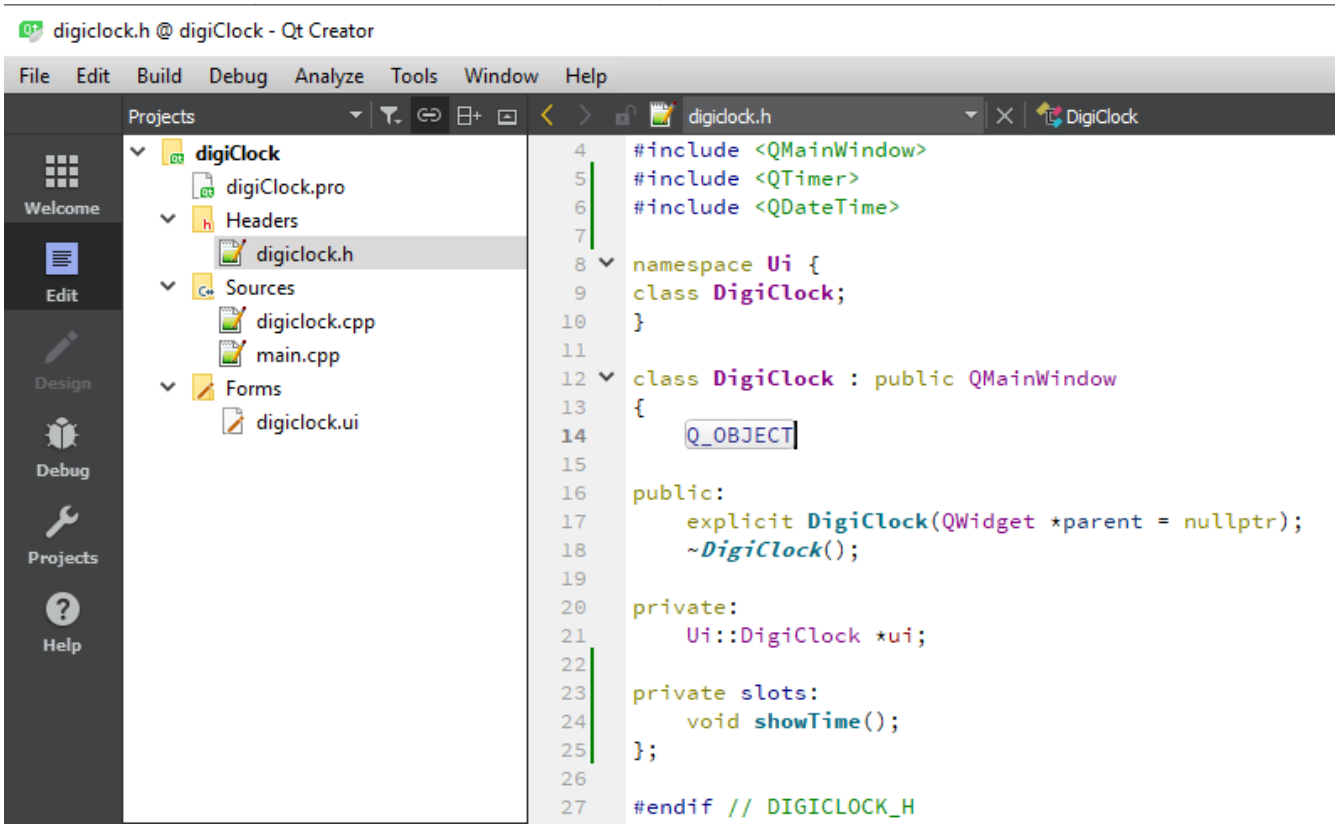


Digitális óra (digiClock)

A következő projektben egy digitális órát hozunk létre.

Ehhez a grafikus szerkesztőben feltehetünk egy LCD kijelzőt (LCD Number) a Formra, a többit kézileg kell hozzáadnunk.

A header fájlban include-oljuk a QTimer és QDateTime library-ket! Utána az osztályon belül hozzunk létre egy privát eseménykezelő eljárást (showTime), ami az aktuális időt jeleníti meg az LCD kijelzőn. Ezt az eseménykezelőt a *private slots* csoportba soroljuk.



A forrásfájlban (digiClock.cpp) megírjuk az eseménykezelő törzsét úgy, hogy az órát és a percet elválasztó kettőspont két másodpercenként villanjon:

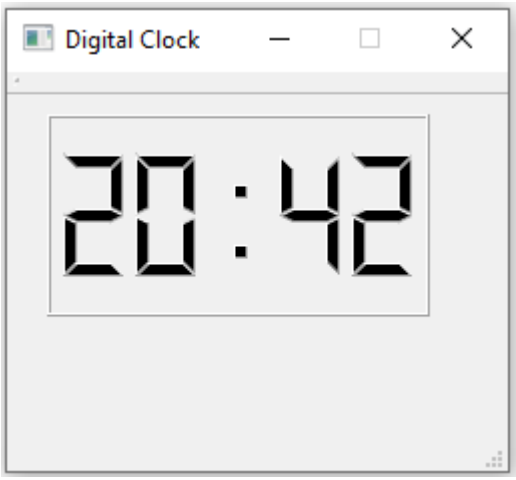
```
void DigiClock::showTime()
{
    QTime time = QTime::currentTime(); //aktuális idő
    QString text = time.toString("hh:mm"); //sztringgé alakítás
    if ((time.second() % 2) == 0) text[2] = ' '; // villogó elválasztó
    ui->lcdNumber->display(text); // megjelenítés
}
```

A következő feladatunk az, hogy létrehozzunk és konfiguráljunk egy időzítőt, ami másodpercenként jelet küld. A jel hatására lefut a showTime() eseménykezelő, ami frissíti az LCD kijelzőt.

Ezt a projekt osztály konstruktorában tehetjük meg.

```
DigiClock::DigiClock(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::DigiClock)
{
    ui->setupUi(this);
    QTimer *timer = new QTimer(this); //időzítő létrehozása
    connect(timer,SIGNAL(timeout()),this,SLOT(showTime()));
        //a szignál és az eseménykezelő összekapcsolása
    timer->start(); // az időzítő elindítása
}
```

Az utolsó szépítés a form méretének fixálása (ezt úgy lehet elérni, hogy a maximum és a minimum size property-ket egyformára állítjuk), valamint a fejlécen megjelenő szöveg módosítása (windowTitle property).



Digitális órák különböző időzónákkal (digiClock_timezone)

A következőkben létrehozunk egy módosított digiClock projektet (digiClock2), hogy önálló vezérlőként felpakolhassuk majd egy másik formra.
Eddig a digiClock osztály a QMainWindow osztályból öröklődött. Ezt most módosítjuk, hogy úgy lehessen kezelni, mint egy átlagos LCD kijelzőt. Hozzunk létre egy új projektet (digiClock2), most a generate form mellől szedjük ki a pipát!

A header és forrás fájlokban a QMainWindow helyett a QLCDNumber library-t include-oljuk és a digiClock2 osztály szülője is a QLCDNumber osztály legyen.

Az előző digiClock osztályból másoljuk ki a hasznos részeket, a showTime() deklarációját és kódját, valamint a konstruktor kódját (de itt az ui-re vonatkozó részeket töröljük).

digiclock2.h:

```
#include <QLCDNumber>
#include <QTimer>
#include <QDateTime>

class digiClock2 : public QLCDNumber
{
    Q_OBJECT

public:
    digiClock2(QWidget *parent = nullptr);

private slots:
    void showTime();
};
```

digiClock2.cpp:

```
digiClock2::digiClock2(QWidget *parent)
    : QLCDNumber(parent)
{
    QTimer *timer = new QTimer(this); //időzítő létrehozása
    connect(timer,SIGNAL(timeout()),this,SLOT(showTime()));
        //a szignál és az eseménykezelő összekapcsolása
    timer->start(); // az időzítő elindítása
}

void DigiClock::showTime()
{
    QTime time = QTime::currentTime(); //aktuális idő
    QString text = time.toString("hh:mm"); //sztringgé alakítás
    if ((time.second() % 2) == 0) text[2] = ' '; // villogó elválasztó
    display(text); // megjelenítés
}
```

Mivel a szülő osztályunk most a QLCDNumber, ezért a display metódust elég önmagában meghívni. Ha most futtajuk a projektet, az LCD kijelzőnk minimális méretű és kitölti a teljes formot. A konstruktorban lehetőségünk van kézzel szépítgetni a formot pl. a következő két utasítás hozzáadásával:

```
    setWindowTitle(tr("Digital Clock"));
    resize(150, 60);
```

A következő lépésben kiegészítjük a digiClock2 osztályt egy új privát *int* típusú adattaggal (timeZone) és annak szetterével, hogy az időzónát lehessen kívülről változtatni.

A showTime() metódust annyiban módosítjuk, hogy a beállított időzóna szerint mutassa az időt, ezt az addSecs függvénnyel érhetjük el:

```
    QString text = time.addSecs(3600*_timeZone).toString("hh:mm");
```

Most következik az új form létrehozása, amire a digiClock2 vezérlőkből felpakolunk párat.

File → New File or Project… → Qt → Qt Designer Form Class

1. ablak: Widget template kiválasztása

2. ablak: Menjen a digiClock2 könyvtárába

3. ablak: Adja hozzá a digiClock2.pro projekthez

Utána felpakolunk a frissen elkészített formra egy sima LCD kijelzőt. A kijelzőre jobbklikk, *Promote to…* kiválasztása. A *Promoted class name*-hez beírjuk, hogy digiClock2, ügyelve, hogy a header fájl neve jól jelenjen meg. Az *Add* gomb lenyomásával hozzáadtuk a választható vezérlők közé a digiClock2-t a Promote gomb pedig végrehajtja az LCD kijelző módosítását.

Újabb LCD kijelző elhelyezése után a *Promote to* > már egyből felajánlja a digiClock2-t.

Az LCD kijelzőknek adjunk beszédes neveket, pl. lcdBudapest és lcdTokyo.

Ezután a form forrásfájljában (form.cpp) a konstruktorban beállíthatjuk pl. a Tokyo-hoz tartozó időzónát:

```
Form::Form(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Form)
{
    ui->setupUi(this);
    ui->lcdTokyo->setTimeZone(8);
}
```

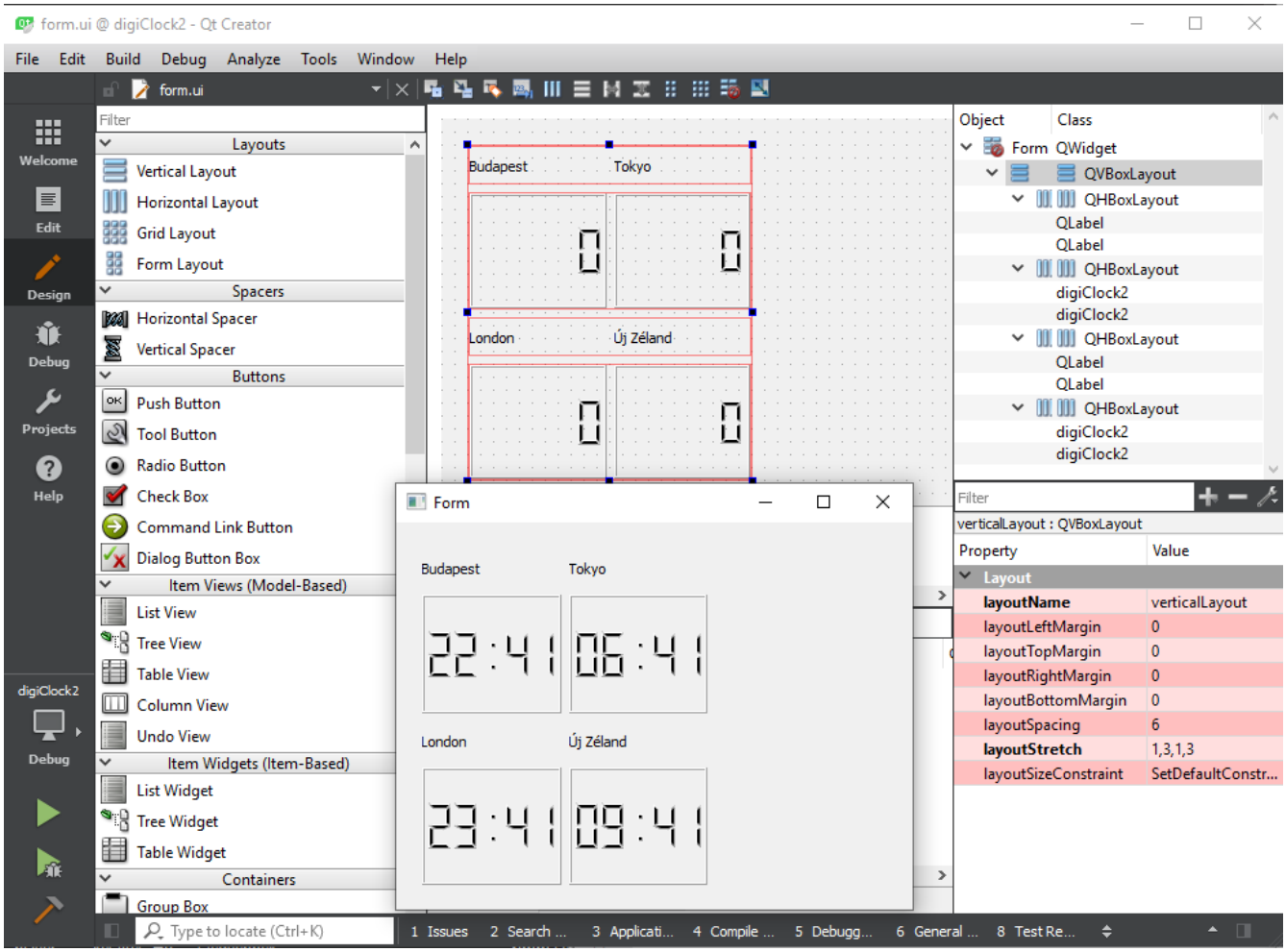
Már csak azt kell elérni, hogy futtatásnál egy darab digitális óra helyett az új formunk jelenjen meg, ehhez a main.cpp-t módosítjuk a következőre (include módosítás!):

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

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

    Form w;
    w.show();
    return a.exec();
}
```

Hasonló módon hozzunk létre új kijelzőket Londonnak (időzóna: -1) és Új Zélandnak (időzóna: 11)!

Tegyünk fel címkéket (label) az egyes órák fölé, hogy lehessen tudni, melyik óra melyik várost jelöli. Az egyes label-ekre duplán kattintva lehet a benne lévő szöveget módosítani.

Rendezőkbe (Vertical és Horizontal Layout) bepakolva az egyes vezérlőket érjük el, hogy az egyes kijelzők pont egymás alá ill. mellé kerüljenek. Először az egy sorban lévőket külön-külön Horizontal Layoutokba helyezzük, majd az így létrejött 4 layout-ot egy Vertical Layout-ba tesszük. A layouton belül drag-and-drop módszerrel érhető el, hogy az egyes elemek helyet cseréljenek. A különböző magasságok eléréséhez a Vertical Layout layoutStretch property-jét kell csak átállítani a megfelelő arányszámokra.



Kitekintő - a qmake eszközről és a main.cpp-ről

A qmake eszköz segítségével könnyen és gyorsan összeállíthatjuk a projektünket. A qmake a fejlesztés alatt álló alkalmazás fájljai alapján automatikusan elkészíti a projektet leíró fájlt (qmake -project), majd a projekt leíró fájlban található információk alapján automatikusan elkészíti a projekt összeállításához szükséges Makefile-t. (qmake -hello.pro). A Makefile-ba automatikusan belekerülnek az adott projekt elkészítéséhez szükséges fordítási, szerkesztési funkciók. Ha nyomkövetést is szeretnénk, akkor ezt a projekt leíró fájlban külön jelezni kell (CONFIG += qt debug).

"Hello Qt" (hello) alkalmazás, kézi fordítás

Feladat: Készítsünk egy futtatható "Hello Qt" alkalmazást Qt osztályok segítségével, „kézi programozással".

hello.cpp

```
#include <QApplication>
#include <QLabel>
```

```
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QLabel *label = new QLabel("Hello Qt!");
    label->show();
    return app.exec();
}
```

A fájl elejére beillesztjük a programban használt osztálykönyvtárat. A főprogramban (*main()*) először a verem tetején létrehozunk egy QApplication (alkalmazás) objektumot (*app*). Ez az objektum kezeli az alkalmazás erőforrásait. Ezután létrehozunk egy, a kívánt szöveget megjelenítő QLabel (címke) widget[1]-et (*label*). Az így előkészített widget-et (*űrlapot, ablakot, vezérlőt)* megjelenítjük a képernyőn (*show()*).
Alapértelmezésben az újonnan létrehozott vezérlők nem láthatók, megjelenítésükről (*show()*) nekünk kell gondoskodni. Ennek oka az, hogy ha a vezérlőelemet a háttérben (memóriában) készítjük el és csak ezután jelenítjük meg, akkor elkerülhető a folyamatos frissítésből adódó villogás. A program végén meghívjuk az alkalmazás *exec()* metódusát. Az *exec()* meghívásával az alkalmazás "készenléti állapotba" kerül, és a neki szóló eseményekre várakozik. A futó alkalmazás folyamatosan feldolgozza a hozzá érkező eseményeket mindaddig, amíg be nem zárjuk az alkalmazás főablakát. A főablak bezárásakor minden főablakhoz rendelt vezérlő törlődik a memóriából.

A QLabel konstruktorában megadott szövegre alkalmazhatjuk a HTML stílusú formázást is.

```
QLabel *label = new QLabel("<h1><i>Hello</i><font color=red> Qt!</font></h1>");
```

A „Hello Qt" fordítása/futtatása qmake eszközzel

1. Hozza létre a hello alkönyvtárat.
2. Gépelje be a fenti programot a hello.cpp fájlba és mentse el azt a hello alkönyvtárba.
3. Legyen a hello alkönyvtárban.
4. A qmake -project paranccsal állítsa elő a platform független projekt leíró fájlt (hello.pro).
5. A qmake hello.pro paranccsal állítsa elő a projekt platform függő make[2] fájlját.
6. A make paranccsal szerkessze össze a projektet.
7. Futtassa a programot. ./hello

A projekt leíró fájl tartalma:

```
TEMPLATE = app
```

[1] *widget: a felhasználói felületen látható elem*

[2] : A Makefile egy olyan script fájl, amely az adott projekt elemeinek lefordítását, összeszerkesztését végzi.

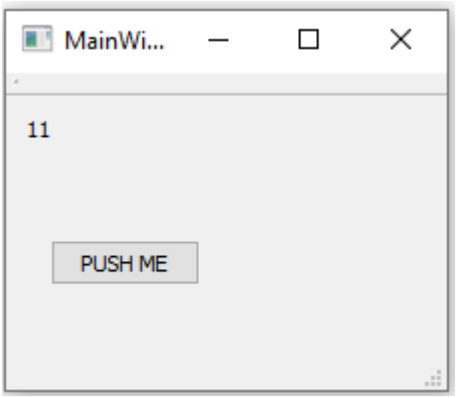
```
TARGET = DEPENDPATH
+= .
INCLUDEPATH += .

# Input
SOURCES += hello.cpp
```

Gombvadászat

A következő feladat egy egyszerű játék létrehozása. A formon megtalálható egy gomb, ami véletlenszerű pozícióba ugrik, amikor lenyomjuk.
A program egy címkében számolja, hogy hányszor sikerült lenyomni.

Ehhez helyezzünk el a formon egy labelt, a benne lévő szöveg legyen 0, illetve egy gombot. A formnak adjunk meg egy értelmes minimális méretet, hogy a gombnak legyen helye ugrálni. Hozzunk létre eseménykezelőt a gomblenyomáshoz.
Az eseménykezelőben a C++-ból ismert random generátor segítségével adjunk meg új pozíciót a gombnak, és a labelben található számot növeljük meg. Az új pozíciót úgy adjuk meg, hogy ne lóghasson le a formról semelyik irányban sem. A random generátorhoz include-oljuk az stdlib.h-t és a konstruktorban inicializáljuk (srand(time(NULL));). A form látható részét úgy hívják, hogy centraWidget, ennek a méreteire vagyunk kíváncsiak. A form mérete általában nem azonos a centralWidget méretével, csak ha külön beállítjuk.

```
void MainWindow::on_pushButton_clicked()
{
    int w=ui->pushButton->width();
    int h=ui->pushButton->height();
    int x=ui->centralWidget->width()-w;
    int y=ui->centralWidget->height()-h;
    ui->pushButton->setGeometry(rand()%x,rand()%y,w,h);
    ui->label->setNum(ui->label->text().toInt()+1);
}
```



Gombvadászat - folytatás

A következőkben egészítsük ki a programot úgy, hogy számolja az eltelt időt és a program bezárásakor írja ki, hogy átlagosan hányszor sikerült lenyomni a gombot másodpercenként.
Ehhez első körben szükségünk lesz egy QTime típusú privát adattagra (include!), amit a konstruktorban el is indítunk.

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    srand(time(NULL));
    _time.start();
}
```

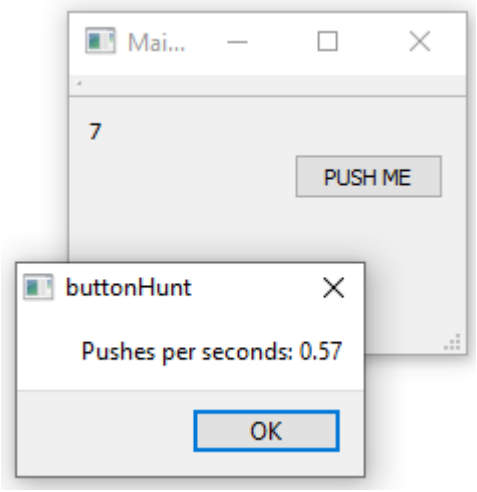
A következő lépés a programbezárás eseménykezelőjének megírása. Ehhez létrehozzuk (felüldefiniáljuk) a privát closeEvent() eljárást.

```
private:
    Ui::MainWindow *ui;
    QTime _time;
    void closeEvent(QCloseEvent *event);
```

A törzsében pedig létrehozunk egy felugró ablakot (QMessageBox, include!), ami a label tartalma+1-et elosztja az eltelt másodpercek számával. A QMessageBox nagy előnye, hogy megakasztja a programfutást, amíg le nem okézzuk, tehát nem tud addig bezáródni a program, amíg tudomásul nem vesszük az eredményt.

A kiírást megszépítendő, két tizedesjegyet engedünk csak meg, ezt az *f* kapcsolóval érjük el.

```
void MainWindow::closeEvent(QCloseEvent *event)  {
    int millisec = _time.elapsed();

    QMessageBox msgBox;
    msgBox.setText("Pushes per seconds: " + QString::number((
        ui->label->text().toDouble()+1)*1000/millisec,'f',2));
    msgBox.exec();
}
```



A programot lehet tovább is szépíteni azzal, hogy az eltelt időt jelezzük egy kijelzőn, illetve a fölső sávba nem engedjük ugrani a gombot, hogy a labelt ne takarja el. Az időzítőt egy gomb indíthatja el a konstruktor helyett, amit a megnyomása után egyből láthatatlanná tehetünk a saját setVisible(false) metódusával, illetve az ugráló gomb ekkor válhatna láthatóvá (setVisible(true)). Arra kell ilyenkor figyelni, hogy a felugró ablak szövegét felhasználóbaráttá tegyük, mert el nem kezdett játéknál nem értelmes az eddig kitalált szöveg, illetve nullával is osztanánk, ami nem túl szerencsés.