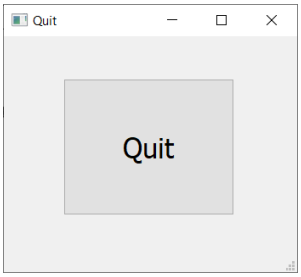
# 1. hét

Cél: Megismerkedni a Qt keretrendszer néhány eszközével és azok használatával.

| https://doc.qt.io/ |
|---|

## *1. Egy egyszerű alkalmazás*

Készítsünk egy olyan alkalmazást, amely egy nyomógombot jelenít meg egy ablakban. Erre rákattintva az alkalmazás álljon le.




*Projekt létrehozása*

A QtCreator elindítása után a *File* → *New file or project…* menüpont kiválasztása teszi lehetővé új projekt létrehozását. A projekt létrehozása több lépésben történik:

1. Válasszuk ki az *Application* sablonok közül a *Qt Widget Application* típusút.
2. Nevezzük el a projektünket (*Quit*) és adjuk meg, hogy hol legyen fizikailag a gépen.
3. A rendszer építéshez használjuk az alapbeállításként felajánlott *qmake*-et.
4. Megadjuk az alkalmazás főablakának jellemzőit. Nevezzük el ennek osztályát *QuitWidget*-nek, amelyet a *QWidget*-ből származtatunk. (Az alapértelmezés a *QMainWindow*, ezért ezt átírjuk.) Az osztály kódja a *quitwidget.h*, illetve *quitwidget.cpp* forrásállományokba kerüljön. Ne felejtsük el a *Generate form* szöveg melletti ellenőrző dobozt (checkbox) bepipálni, mert ez teszi lehetővé, hogy a *QuitWidget* grafikus felületét a *QtDesigner* vizuális tervezővel készíthessük majd el.
5. A projekt többnyelvűvé tételével most nem foglalkozunk.
6. Lehetőség van kit-et választani, ami meghatározza, hogy melyik Qt verzió szerint legyen lefordítva a projekt (több Qt verziót is fel lehet telepíteni egy gépre).
7. Az utolsó lépésen átmehetünk módosítás nélkül.

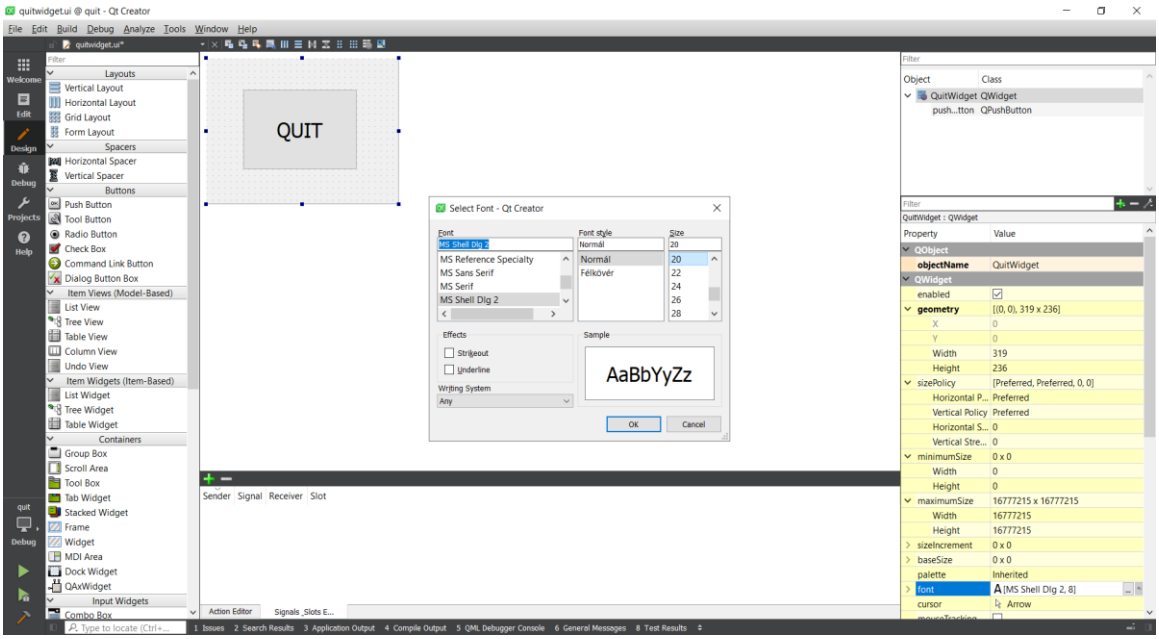
*Finish* gomb.

A projekt létrejön, a bal oldali navigációs sáv mutatja a létrejött fájljainkat.

*Grafikus felület elkészítése*

A grafikus felhasználói felületeket közvetlenül a kódból is felépíthetjük, de a *QtDesigner* vizuális tervezőt is igénybe vehetjük ehhez. Ez utóbbit a .ui kiterjesztésű (ez most a *quitwidget*.ui) fájlra történő dupla kattintás hozza elő. Középen látjuk az egyelőre még üres felületet (form), bal oldalt megjelennek a felületre elhelyezhető előre definiált vezérlők. Nekünk most csak egy nyomógombot (*PushButton*) kell az egérrel a felületre húzni. Növeljük meg ennek méretét, duplán rákattintva megváltoztathatjuk a feliratát. A *QtCreator*

jobb oldalán megjelenő *Property Editor* az éppen kijelölt vezérlő tulajdonságait mutatja, amelyeket itt változtatni is lehet. Például beállíthatjuk a nyomógomb feliratának stílusát egy felbukkanó (*Select Font*) ablakban, vagy megváltoztathatjuk a nyomógomb automatikusan generált nevét (*pushbutton*) is, amivel a gombra, mint objektumra tudunk hivatkozni a kódban. Megadhatjuk azt is, hogy milyen cím jelenjen meg a nyomógombot tartalmazó ablak fejlécében (*windowTitle*).




*Eseménykezelés*

A nyomógombon kiváltott eseményekhez eseménykezelőket lehet hozzárendelni. Ez megtehető úgy, hogy a *QtDesigner*-ben az egérrel jobbklikkelünk a gombra, majd a *Go to slot…* menüpont kiválasztása után kijelöljük például a *clicked()* szignált. Ennek hatására belekerül a kódba (*QuitWidget*) egy *on_pushButton_clicked()* eseménykezelő metódus. Ha ennek a törzsében a *QWidget* ősosztályától örökölt *close()* metódust hívjuk meg, akkor a gombon kattintás hatására a gombot tartalmazó ablak bezárul, és emiatt az alkalmazás leáll.

Ugyanez el érhető úgy is, ha a *QtDesigner* ablakának alján kiválasztjuk a *Signals Slots Editort*, felveszünk abba egy új sort, és beírjuk a < *sender: pushButton, signal: clicked(), receiver: QuitWidget, slot: close()* > sort.

*Futtatás*

A projekt kész, már csak futtatni kell. Bal alul található egy zöld háromszög ikon, ami fordítja és futtatja a projektet. A Ctrl+R billentyűkombináció szintén ugyanezt teszi.

## 2. Gombvadászat

Készítsünk el egy egyszerű játékot. Az alkalmazás felületén egy nyomógomb jelenjen meg, amelyikre ha rákattintunk, az egy véletlenszerű másik pozícióba ugrik. A cél a gomb minél többször történő megnyomása. Írjuk ki a felület alsó sorában folyamatosan azt, hogy eddig hány kattintásnál járunk.

*Projekt létrehozása*

Hozzunk létre egy *Qt Widget Application* projektet a a *QMainWindow*-ból származtatva.

Az alkalmazás főablakát *ButtonHunt*-nak nevezzük. Így a felületnek lesz alul egy ún. státusz sora, ahová az aktuális kattintások számát kiírhatjuk. Helyezzünk el a *QtDesigner* segítségével egy nyomógombot (*QPushButton*) „PUSH ME" felirattal. Állítsunk be a felületnek egy értelmes minimális méretet (*minimumSize*), hogy a nyomógombnak legyen helye ugrálni.

Vegyünk fel a *ButtonHunt* osztályba egy privát adattagot (*points*) a sikeres kattintások számolásához.

*Eseménykezelés*

Készítsünk eseménykezelőt a nyomógomb kattintásához. A nyomógomb új pozícióját a *setGeometry()*. metódussal állítjuk be, és a koordinátákhoz a véletlenszámokat a *qrand()* segítségével generáljuk. Ügyeljünk arra, hogy a gomb ne lóghasson le a felület központi területéről (ez a *centralwidget*). Ehhez szükségünk van a gomb és a központi terület magasságára és szélességére (*height*, *width*).

Növeljük meg eggyel a *points* értékét is, és ezt a főablak státusz sorába írjuk ki. A *QString:number()* konvertál számot sztringgé.

```
void ButtonHunt::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(qrand()%x, qrand()%y, w, h);
   ++points;
   ui->statusbar->showMessage("Points: " + QString::number(points));
}
```

*Konstruktor*

A konstruktorban inicializáljuk a véletlenszám generátort, amely használatához inklúdolnunk kell a *QTime* könyvtárat.

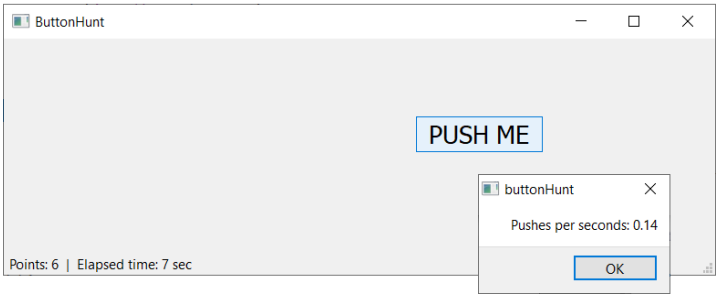
```
   qsrand(unsigned(QTime::currentTime().msec()));
```

A *points* értékét nullára állítjuk, és megjelenítjük a főablak státusz sorában.

```
    points = 0;
    ui->statusbar->showMessage("Points: " + QString::number(points));
```

*Továbbfejlesztés*

Egészítsük ki a programot azzal, hogy számolja és megjeleníti az eltelt időt, a program bezárásakor pedig kiírja, hogy átlagosan hányszor sikerült lenyomni a gombot másodpercenként.




Szükségünk lesz egy *QTime* típusú privát adattagra, amivel az eltelt időt le lehet kérdezni. Használni fogunk egy időzítőt (*QTimer*, amelyet inklúdolni kell), hogy az elindítása után (*start()*) megadott időközönként küldjön egy *timeout()* szignált, amelyhez hozzáösszekötjük a státuszsor frissítését végző tevékenységet (ez az *updateStatusBar()* slot). Felüldefiniáljuk a ButtonHunt örökölt privát *closeEvent()* eljárását, amelyet a program bezárása vált ki.

```
class ButtonHunt : public QMainWindow
{
    Q_OBJECT
public:
    ButtonHunt(QWidget *parent = nullptr);
    ~ButtonHunt() override;
private slots:
    void on_pushButton_clicked();
    void updateStatusBar();
private:
    Ui::ButtonHunt *ui;
    int points;
    QTime _time;
    QTimer _timer;
    void closeEvent(QCloseEvent *) override;
};
```

A konstruktorban a connect utasítással köthetjük össze az időzítő *timeout()* szignálját az *updateStatusBar()* slot-tal. Ez négy paramétert igényel:

1. Az objektum pointere, ami a szignált küldi: a mi esetünkben ez a timer objektum.
2. A kiváltott szignál SIGNAL(szignálnév) formában: SIGNAL(timeout()).
3. Az eseménykezelő slot tulajdonosának pointere: jelen példánkban this.
4. A slot, ami kezeli az eseményt SLOT(slotnév) formában: most SLOT(updateStatusBar()).

```
    connect(&_timer, SIGNAL(timeout()), this, SLOT(updateStatusBar()));
    _time.start();
    _timer.start(1000);
```

Ezután (még mindig a konstruktorban) elindítjuk az időmérést és az időzítőt is. Az időzítő elindításakor a start metódus bemeneti paraméterének segítségével adjuk meg, hogy a *timeout()* szignált az időzítő másodpercenként (1000 milisec) emittálja.

A státuszsor frissítését végző *updateStatusBar()* metódust nemcsak az időzítő szignáljai váltják ki, hanem közvetlenül is meghívjuk az *on_pushButton_clicked()* metódusból. Ehhez ennek a metódusnak a korábbi verziójában az utolsó sort kell lecserélni az *updateStatusBar()* hívására. A *time.elapsed* az eddig eltelt időt adja meg.

```
void ButtonHunt::updateStatusBar()
{
    ui->statusbar->showMessage("Points: " + QString::number(points)
         +" | Elapsed time: "+QString::number(_time.elapsed()/1000)+" sec");
}
```

A programbezárás eseménykezelőjének törzsében létrehozunk egy felugró üzenet-ablakot (*QMessageBox*), ami az eddigi sikeres kattintások számát (*points*) elosztja az eltelt másodpercek számával. (A üzenet-ablak használatához inklúdolnunk kell a *QMessageBox* könyvtárat.) Az üzenet-ablak felfüggeszti a programfutást, amíg le nem „okézzuk”, így nem tud addig bezáródni a program, amíg tudomásul nem vesszük az eredményt.

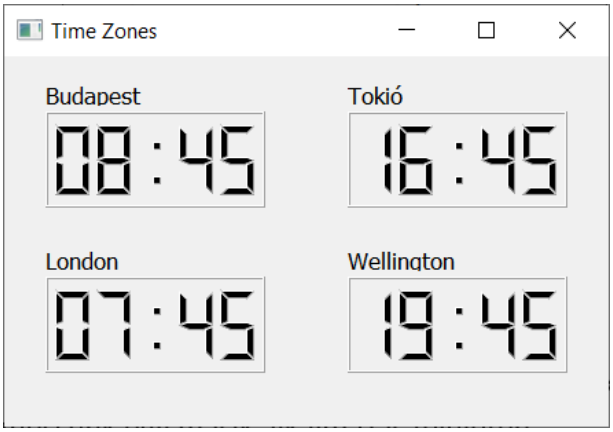
```
void ButtonHunt::closeEvent(QCloseEvent *)
{
    int millisec = _time.elapsed();
    QMessageBox msgBox;
    msgBox.setText("Pushes per seconds: "
           + QString::number(double(points)*1000/millisec,'f',2));
    msgBox.exec();
}
```

A statisztika kiírásában két tizedesjegyet engedünk csak meg. Ezt a *QString::number()* statikus metódusban használt *f* kapcsoló kényszeríti ki.

Az időzítőt indíthatná egy nyomógomb a konstruktor helyett, amelyet a megnyomása után egyből láthatatlanná tennénk annak *setVisible(false)* metódusával, és az ugráló gomb ekkor válna láthatóvá (*setVisible(true)*). Ekkor ügyelni kell arra, hogy az alkalmazás bezárásakor felbukkanó üzenet-ablak szövege akkor is értelmes legyen, ha a játék még nem indult el, és ne végezzünk ekkor nullával való osztást. Ilyenkor írjuk ki inkább azt, hogy „The game has not started yet.”.

## *3. Digitális órák különböző időzónákkal*

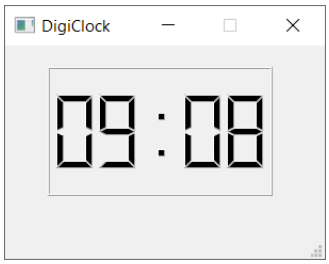
Készítsünk egy olyan alkalmazást, amelyben több – különböző időzónákba eső – város pontos idejét lehet majd egyszerre megjeleníteni.




A megoldás érdekében először egyetlen digitális órát kijelző alkalmazást készítünk, majd lehetővé tesszük, hogy ilyenből négyet elhelyezhessünk egy másik alkalmazás felületén.

*Digitális óra projekt*

A digitális óra alkalmazás (*digiClock*) osztályát most a *QWidget*-ből származtatjuk, és *DigiClock*-nak nevezzük, ennek kódjait tartalmazó fájlok neve pedig *digiclock* lesz.




A grafikus szerkesztővel (*QtDesigner*) felteszünk a felületre egy LCD kijelzőt (*QLCDNumber*). Az ablak méretét beállítjuk és rögzítjük. (Ez utóbbit el lehet úgy elérni, hogy a *maximumSize* és a *minimumSize* tulajdonságokat egyformára állítjuk.)

A *digiClock.h* fájlba inklúdoljuk a *QTimer* könyvtárat, és privát adattagként felveszünk egy hivatkozást egy időzítő (*QTimer*) objektumra. A *DigiClock* osztály *private slots* csoportjában elhelyezünk egy eseménykezelő eljárást: *showTime()*. Ennek feladata lesz az aktuális időt az LCD kijelzőn megjeleníteni.

A *DigiClock* konstruktorában kötjük az időzítőhöz a *showTime()* eseménykezelőt, és az időzítőt 1 másodperces tikkelésre kényszerítjük.

```
DigiClock::DigiClock(QWidget *parent) : QLCDNumber(parent)
{
    ui->setupUi(this);
    connect(&_timer, SIGNAL(timeout()), this, SLOT(showTime()));
    //a szignál és az eseménykezelő összekapcsolása
   _timer.start(1000); // az időzítő elindítása
}
```

A *showTime()* az aktuális rendszeridőt jeleníti (*display()*) meg az LCD kijelzőn. Ehhez a *digiClock.cpp* fájlba kell inklúdolni a *QTime* könyvtárat. Az órát és a percet elválasztó kettőspontot csak a páratlan másodpercekben jelenítjük meg. Ennek hatásaként a pontos idő kijelzésében a kettőspont másodpercenként villogni fog:

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

*Később felhasználható vezérlő készítése a digitális óra projektből*

Készítsünk egy új projektet (*digiClock2*) az előző mintájára, de a létrehozáskor a *Generate form* mellől szedjük ki a pipát. (A felhasználói felületet most a kódból fogjuk megszerkeszteni.) A *DigiClock2* osztály őse a *QLCDNumber* osztály legyen azért, hogy később úgy tudjuk majd kezelni, mint egy általános LCD kijelzőt. Ezért a fej- és forrás fájlokban a *QLCDNumber* könyvtárat inklúdoljuk (a *QWidget* helyett).

A *DigiClock2* osztály deklarációja – az ősosztályától eltekintve – majdnem olyan, mint a *DigiClock* osztályé, csak tartalmaz egy időzónát mutató privát *int* típusú adattagot (*timeZone*), és annak a szetterét.

```
private:
   int _timeZone;
public:
   void setTimeZone(int t) { _timeZone = t;}
```

A *DigiClock2* osztály konstruktora az *ui->setupUi(this)* hívás helyett maga állítja be az ablak méretét (*resize()*) és nevét (*setWindowTitle()*). Az időzónát nullára inicializálja.

A *showTime()* metódus kódja abban tér el a *DigiClock* osztálybelitől, hogy az aktuális időhöz hozzáadja (*addSecs()*) az időzóna okozta eltérést is. Mivel az ősosztályunk most a *QLCDNumber*, ezért most a display metódust elég önmagában meghívni.

digiclock2.cpp:

```
#include "digiclock2.h"
#include <QTime>

DigiClock2::DigiClock2(QWidget *parent) : QLCDNumber(parent)
{
    setWindowTitle(tr(""));
    resize(150, 60);
    connect(&_timer, SIGNAL(timeout()), this, SLOT(showTime()));
     //a szignál és az eseménykezelő összekapcsolása
    _timer.start(1000); // az időzítő elindítása
    _timeZone = 0;
}

 void DigiClock2::showTime()
{
    QTime time = QTime::currentTime(); //aktuális idő
    QString text = time.addSecs(3600*_timeZone).toString("hh:mm");
         //az időzóna szerinti aktuális idő sztringgé alakítása
    if ((time.second() % 2) == 0) text[2] = ' '; // villogó elválasztó
    display(text); // megjelenítés
}
```

*Az eredeti feladat megoldása*

Most hozzáadunk a *digiClock2* projektünkhöz egy új felületet (*Form*), amire majd a *DigiClock2* típusú vezérlőkből négyet fogunk felpakolni.

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

Válasszuk a *Widget* template-et, és ezután fogadjuk el a felajánlott beállításokat.

A felületen (*form.ui*) elhelyezzünk egy sima LCD kijelzőt. Ha erre jobbklikkelünk, és a *Promote to…* menüpontot választjuk, majd a *Promoted class name*-hez beírjuk a *DigiClock2*-t, akkor az *Add* gomb lenyomásával hozzáadhatjuk a választható vezérlők közé a *DigiClock2*-t, a *Promote* gomb lenyomásával pedig az LCD kijelzőnk egy *digiClock*-szerű kijelzővé válik.

Az újabb LCD kijelzők elhelyezése után a *Promote to* > már egyből felajánlja a *DigiClock2*-t. Az LCD kijelzőknek adjunk beszédes neveket, pl. lcdBudapest és lcdTokyo.

Tegyünk fel címkéket (*QLabel*) az egyes órák fölé, hogy lehessen tudni, melyik óra melyik városhoz tartozik. Ezekre duplán kattintva is lehet a bennük lévő szöveget módosítani.

Elrendezőkbe (*Vertical* és *Horizontal Layout*) szervezve az egyes vezérlőket elérhetjük, hogy a kijelzők pont egymás alá ill. mellé kerüljenek. Először az egy sorban lévőket külön-külön *Horizontal Layoutokba* helyezzük, majd az így létrejött négy elrendezőt egy *Vertical Layout*-ba tesszük. Egy elrendezőn belül drag-and-drop módszerrel lehet az egyes elemeket felcserélni. A különböző magasságok eléréséhez a *Vertical Layout layoutStretch* tulajdonságát kell csak átállítani a megfelelő arányszámokra.

A *Form* konstruktorában (*form.cpp*) beállíthatjuk az egyes városok időzónáit a *DigiClock2* szetterével. London órája hozzánk képest eggyel hátrébb (-1), Tokióé nyolccal (8), Wellingtoné tizeneggyel (11) előrébb jár.

```
Form::Form(QWidget *parent) : QWidget(parent), ui(new Ui::Form)
{
    ui->setupUi(this);
    ui->lcdTokyo->setTimeZone(8);
    ui->lcdLondon->setTimeZone(-1);
    ui->lcdWellington->setTimeZone(11);
}
```

Már csak azt kell elérni, hogy futtatásnál az új formunk jelenjen meg. Ehhez a main.cpp-t kell módosítani:

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

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

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