Eötvös Loránd Tudományegyetem
Informatikai Kar

# Eseményvezérelt alkalmazások fejlesztése I

## 3. előadás

## Dinamikus felületű alkalmazások

Giachetta Roberto

http://people.inf.elte.hu/groberto

# Dinamikus felületű alkalmazások

## A felhasználói felület típusai

- Az alkalmazásaink grafikus felülete alapvetően kétféle lehet:
  - *statikus*: az alkalmazás felületén lévő vezérlők rögzítettek, azaz mindig ugyanazon vezérlők jelennek meg
    - a vezérlőket a felülettervezővel, vagy a kódban a konstruktorral hozzuk létre
  - *dinamikus*: futás közben változhatnak a megjelenő vezérlők, illetve tulajdonságaik
    - a vezérlőket kódban, futás közben esemény hatására hozzuk létre és helyezzük el a felületen
    - a változó mellett lehet állandó része is a felületnek rögzített vezérlőkkel

# Dinamikus felületű alkalmazások

## Dinamikusan létrehozott vezérlők

- A futási időben létrehozott vezérlőket ugyanúgy kezeljük, mint a konstrukció során létrehozott vezérlőket
- Egyszerre több, ugyanolyan típusú vezérlőt is létrehozhatunk
  - ezeket célszerű egy adatszerkezetbe szervezni, és közös eseménykezelőt rendelni hozzájuk
  - az eseménykezelőben meghatározhatjuk, melyik vezérlő küldte az eseményt
    - a `sender()` (vagy `QObject::sender()`) művelet visszaadja az esemény küldőjét `QObject` mutatóként
    - ezt konvertálhatjuk megadott altípusra a `qobject_cast<T>` utasítással

# Dinamikus felületű alkalmazások

## Dinamikusan létrehozott vezérlők

- Pl.:

```
QVector<QPushButton*> buttons;
   // gombokat tartalmazó vektor
…
for (…) // gombok létrehozása egy ciklusban
{
   QPushButton* button = new QPushButton(this);
   buttons.append(button); // új gomb hozzá vétele

   connect(button, SIGNAL(clicked()),
           this, SLOT(buttonClicked()));
      // eseménykezelés megvalósítása közös
      // eseménykezelővel
}
```

# Dinamikus felületű alkalmazások

## Dinamikusan létrehozott vezérlők

- Pl.:

```
…
void buttonClicked() // eseménykezelő
{
   QPushButton* senderButton =
      qobject_cast<QPushButton*>(sender());
      // lekérjük az esemény küldőjét gombként

   senderButton->setText("You clicked me!");
      // a külsőt módosítjuk
}
```

# Dinamikus felületű alkalmazások

## Dinamikusan létrehozott vezérlők kezelése

- A dinamikus létrehozást akkor célszerű használni, ha:
  - a felületen sok ugyanolyan típusú, hasonló tulajdonságú vezérlőt kell létrehozni
  - a felületen lévő vezérlők száma változhat
- A dinamikusan létrehozott vezérlőket célszerű elhelyezések segítségével pozícionálni és/vagy méretezni
- Ha változhat a vezérlők száma, a régieket törölni kell (`delete`), amikor levesszük őket a felületről
  - ekkor a hozzá kötött eseménykezelő társítások megszűnnek (a `disconnect` hívódik meg a háttérben)

# Dinamikus felületű alkalmazások

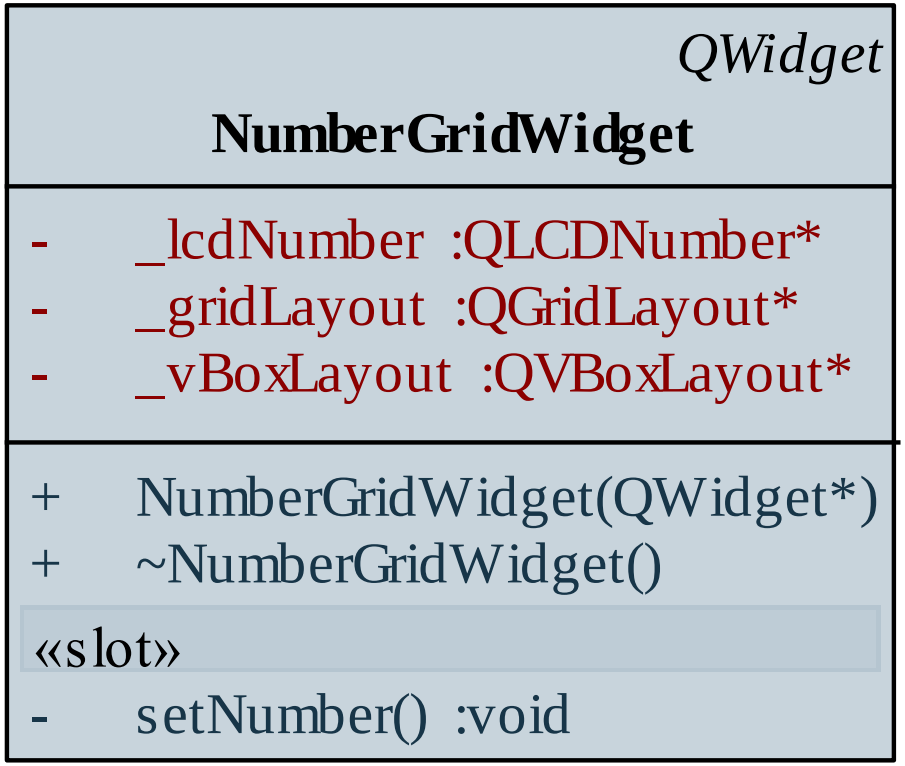
## Példa

*Feladat:* Készítsünk egy alkalmazást, amelyben száz gombot jelenítünk meg egy 10 × 10-es rácsban sorszámozva. Gombra kattintással a sorszám megjelenik egy központi kijelzőn, míg a gombon egy ‚X' felirat. Az ilyen feliratú gombokra kattintva már nem történik semmi.

- a felületen felveszünk egy LCD kijelzőt, valamint a gombokat rács elrendezésben, a gombok feliratának (**`text`**) beállítjuk a sorszámot
- a gombokat közös eseménykezelőhöz (**`setNumber()`**) rendeljük, amely a küldő gomb (**`sender()`**) sorszámát beírja az LCD kijelzőre (amennyiben nem ‚X' feliratú), a gomb feliratát pedig lecseréli ‚X'-re

## Példa

*Tervezés:*



# Dinamikus felületű alkalmazások

## Példa

*Megvalósítás (*`numbergridwidget.cpp`*):*

```
NumberGridWidget::NumberGridWidget(QWidget
    *parent) : QWidget(parent) {
    …
    for (int i = 0; i < 10; ++i) {
       for (int j = 0; j < 10; ++j){
          QPushButton* button = new QPushButton(
            QString::number(i * 10 + j + 1), this);
             // gomb létrehozása
          _gridLayout->addWidget(button, i, j);
             // gomb felvétele az elrendezésre
          connect(button, SIGNAL(clicked()),
                  this, SLOT(setNumber()));
             // eseménykezelő kapcsolat
```

# Dinamikus felületű alkalmazások

## Példa

*Megvalósítás (*`numbergridwidget.cpp`*):*

```
…
void NumberGridWidget::setNumber() {
   QObject* senderObject = sender();
      // küldő objektum lekérdezése
   QPushButton* senderButton =
      qobject_cast<QPushButton*>(senderObject);
      // a küldő típusát konvertálnunk kell

   if (senderButton->text() != "X")
      _lcdNumber->display(senderButton->text());
   senderButton->setText("X");
      // a gombra beállítunk egy X feliratot
}
```

# Dinamikus felületű alkalmazások

## Példa

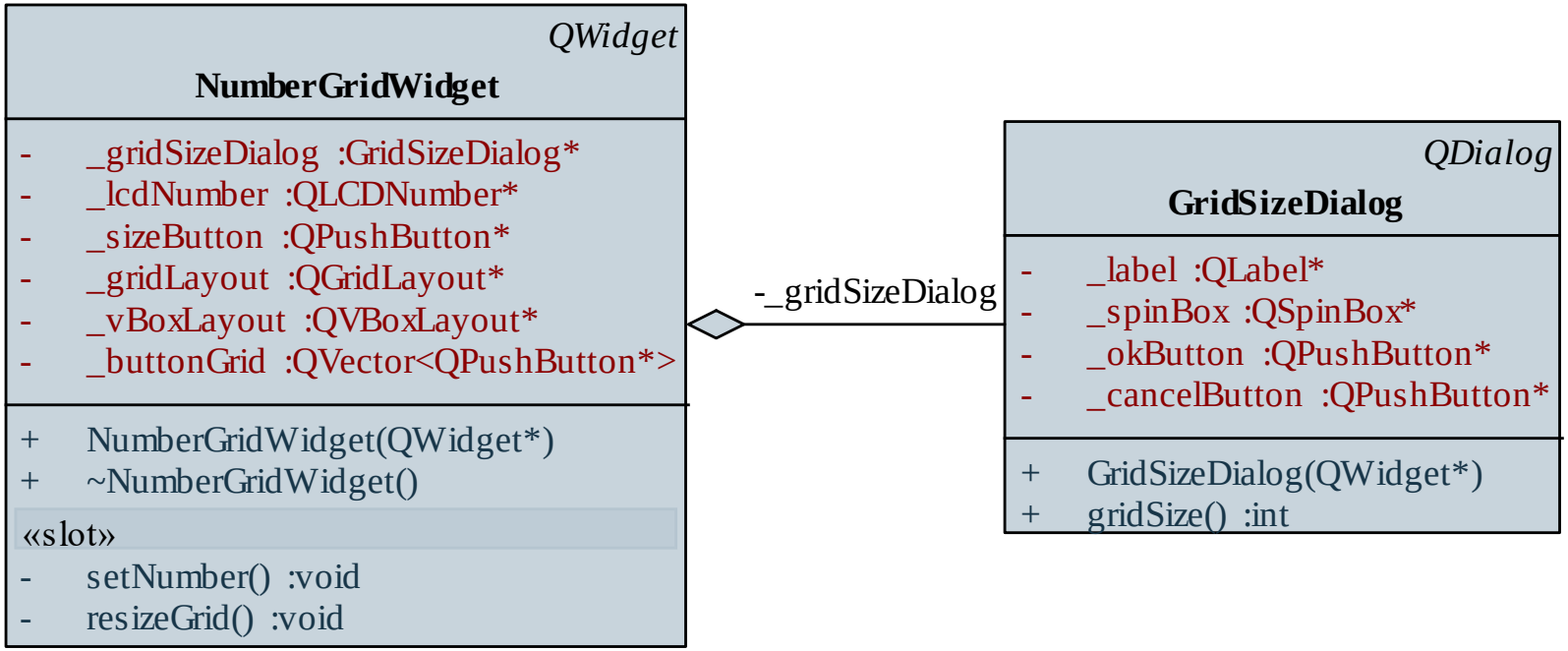
*Feladat:* Módosítsuk az előző feladatot úgy, hogy futás közben át lehessen méretezni a gombrácsot. Kezdetben a rács üres, egy külön gombbal lehet egy előugró ablakot előhozni, amelyben beállíthatjuk az új méretét, és ennek hatására a rács újraméreteződik, és alaphelyzetbe áll.

- minden átméretezésnél (`resizeGrid()`) töröljük a régi gombokat, és megfelelő mennyiségű új gombot generálunk
- a méret beállító ablak egy új osztály (`GridSizeDialog`), amely a `QDialog` leszármazottja, így felhasználhatjuk az `accept()` és `reject()` eseménykezelőjét, megjeleníteni pedig az `exec()` metódussal fogjuk (így blokkolja a háttérben lévő ablakot)

# Dinamikus felületű alkalmazások

## Példa

*Tervezés:*



# Dinamikus felületű alkalmazások

## Példa

*Megvalósítás (*`numbergridwidget.cpp`*):*

```
NumberGridWidget::NumberGridWidget(QWidget
    *parent) : QWidget(parent) {
    …
    _gridSizeDialog = new GridSizeDialog();
    connect(_sizeButton, SIGNAL(clicked()),
       _gridSizeDialog, SLOT(exec()));
       // méretező ablak megjelenítése gombnyomásra
    connect(_gridSizeDialog, SIGNAL(accepted()),
       this, SLOT(resizeGrid()));
       // átméretezés a dialógus elfogadására
}
…
```

# Dinamikus felületű alkalmazások

## Példa

*Megvalósítás (*`numbergridwidget.cpp`*):*

```
void NumberGridWidget::resizeGrid() {
    foreach(QPushButton* button, buttonGrid) {
        // korábbi gombok
        _gridLayout->removeWidget(button);
            // levétel az elrendezésről
        delete button; // vezérlő törlése
    }
    _buttonGrid.clear(); // mutatók törlése

    for (int i = 0; i < gridSizeDialog->gridSize();
         ++i) {
    …
}
```

# Dinamikus felületű alkalmazások

## Stílusok

- A vezérlőket különböző stílusokkal láthatjuk el, amelyek szabályozzák annak megjelenését a **`styleSheet`** tulajdonság segítségével
- A stílusok kezelése szintaktikailag hasonló a CSS leíráshoz
  - megadása szöveges formában történik az objektum, a típus és a beállítás megadásával, pl.:

```
QPushButton { color: red;
              background-color: white }
   // a gomb fehér háttéren piros betűs lesz
QCheckBox:hover:checked { color: white }
   // amennyiben ki van választva a
   // kijelölő doboz, és az egér rajta van,
   // akkor fehér színű lesz a szöveg
```

# Dinamikus felületű alkalmazások

## Stílusok

```
QLineEdit {
    background-image: url(:/images/bck.png);
    border-width: 1px;
    border-style: solid;
    border-radius: 4px; }
    // a hátteret kép tölti ki, keret megformázva
```

- a szabályozása történhet az egész alkalmazás, illetve bármely vezérlő szintjén, pl.:

```
QApplication::setStyleSheet("
    QPushButton { color:black }
    QPushButton:enabled { color:red }");
    // minden engedélyezett gomb piros feliratú,
    // minden nem engedélyezett gomb fekete
    // feliratú lesz a programban
```

# Dinamikus felületű alkalmazások

## Időzítés

- Sokszor nem a felhasználó által vezérelten, hanem rögzített módon, adott időközönként szeretnénk lefuttatni egy tevékenységet, erre szolgál az *időzítő* (`QTimer`)
  - a `start(<intervallum>)` eseménykezelő indítja az időzítőt (az intervallumot ezred másodpercben adjuk meg), a `stop()` leállítja
  - az idő leteltekor kiváltja a `timeout` eseményt, majd újra elindítja a visszaszámlálást
  - lehetőség van egyszeri kiváltásra is (`singleShot(…)`)
  - lekérdezhető az állapota (`active`, `singleShot`)
  - egyszerre tetszőleges sok időzítőt használhatunk

# Dinamikus felületű alkalmazások

## Időzítés

- Pl.:

```
…
_timer = new QTimer(); // időzítő
connect(_timer, SIGNAL(timeout()),
        this, SLOT(updateTime()));
_timer->start(1000); // időzítő indítása
…

void updateTime() // eseménykezelő
{
   _time--;
   _textBox->setText(QString::number(_time));
   // 1 másodpercenként frissül a szöveg
}
```

# Dinamikus felületű alkalmazások

## Példa

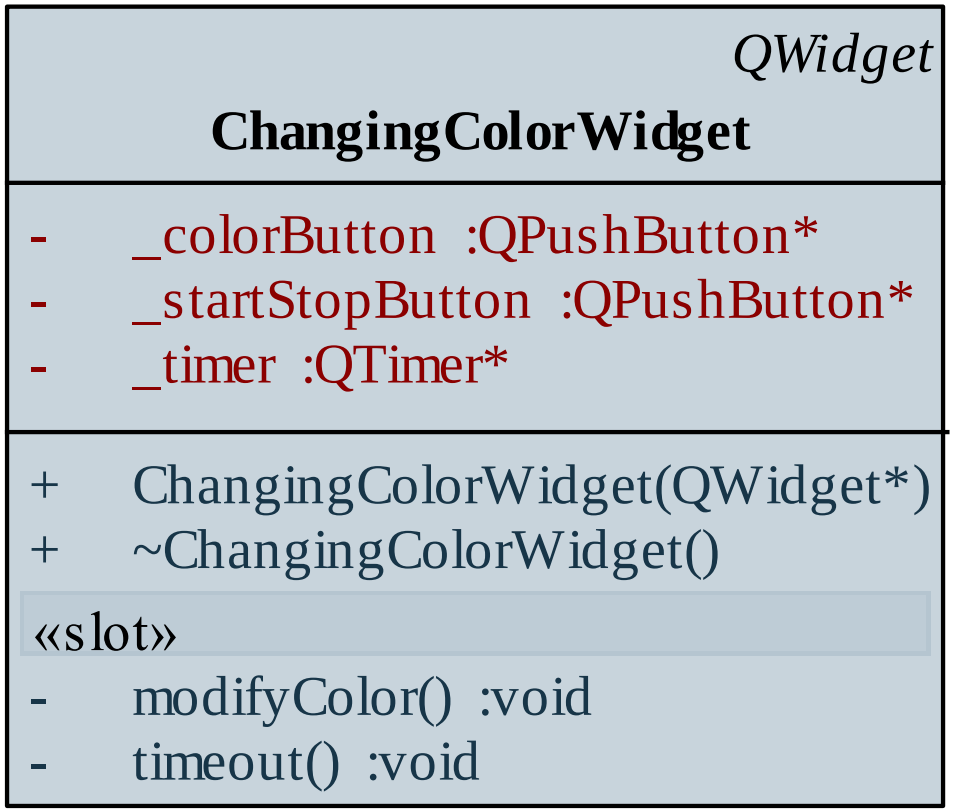
*Feladat:* Készítsünk egy alkalmazást, amelyben véletlenszerűen változtatjuk egy gomb színét másodpercenként. Egy külön gombbal ki/be kapcsolhatjuk az animációt.

- a felületen (**`ChangingColorsWidget`**) két gombot helyezünk el, az egyikkel indítjuk/állítjuk le az időzítőt
- az időzítő (**`timer`**) egy eseménykezelőt futtat (**`timeout`**), amelyben a gomb stílusát változtatjuk
- ehhez véletlen számokat generálunk a **`qsrand(<kezdőérték>)`** és **`qrand()`** függvények segítségével, amit időfüggő értékkel indítunk (**`QTime::currentTime().msec()`**)

# Dinamikus felületű alkalmazások

## Példa

*Tervezés:*



# Dinamikus felületű alkalmazások

## Példa

*Megvalósítás (*`changingcolorwidget.cpp`*):*

```
…
void ChangingColorWidget::modifyColor() {
    if (!_timer->isActive())
    {
        // ha az időzítő nem fut
        _startStopButton->setText(tr("Stop"));
        _timer->start(1000); // elindítjuk
    }
    else { // ha fut
        _startStopButton->setText(tr("Start"));
        _timer->stop(); // leállítjuk
    }
}
```

# Dinamikus felületű alkalmazások

## Példa

*Megvalósítás (*`changingcolorwidget.cpp`*):*

```
void ChangingColorWidget::timeout()
{  // időzített eseménykezelő

   // stílus beállítása véletlen számok
   // segítségével:
   _colorButton->setStyleSheet("QPushButton {
      background-color: rgb(" +
      QString::number(qrand() % 256) + "," +
      QString::number(qrand() % 256) + "," +
      QString::number(qrand() % 256) + ") }");
}
```

# Dinamikus felületű alkalmazások

## Példa

*Feladat:* Az előző feladat alapján készítsünk egy gombrácsot, amelyben kattintás hatására indul a színváltó animáció, de nem csak az adott gombon, hanem egy kereszt alakban (a teljes sorban és oszlopban). A rács méretét külön ablakban állíthatjuk be.

- minden egyes kattintásra egy új időzítőt indítunk el, így tetszőlegesen sok időzítő lehet a programban
- az időzítőket és a gombokat a koordináták (sor, oszlop index) alapján tudjuk összekapcsolni, ezért létrehozunk egy koordináta segédtípust (`Coordinate`)
- az időzítőket egy asszociatív tömb (`QMap`) segítségével tároljuk el a hozzátartozó koordinátával egyetemben

# Dinamikus felületű alkalmazások

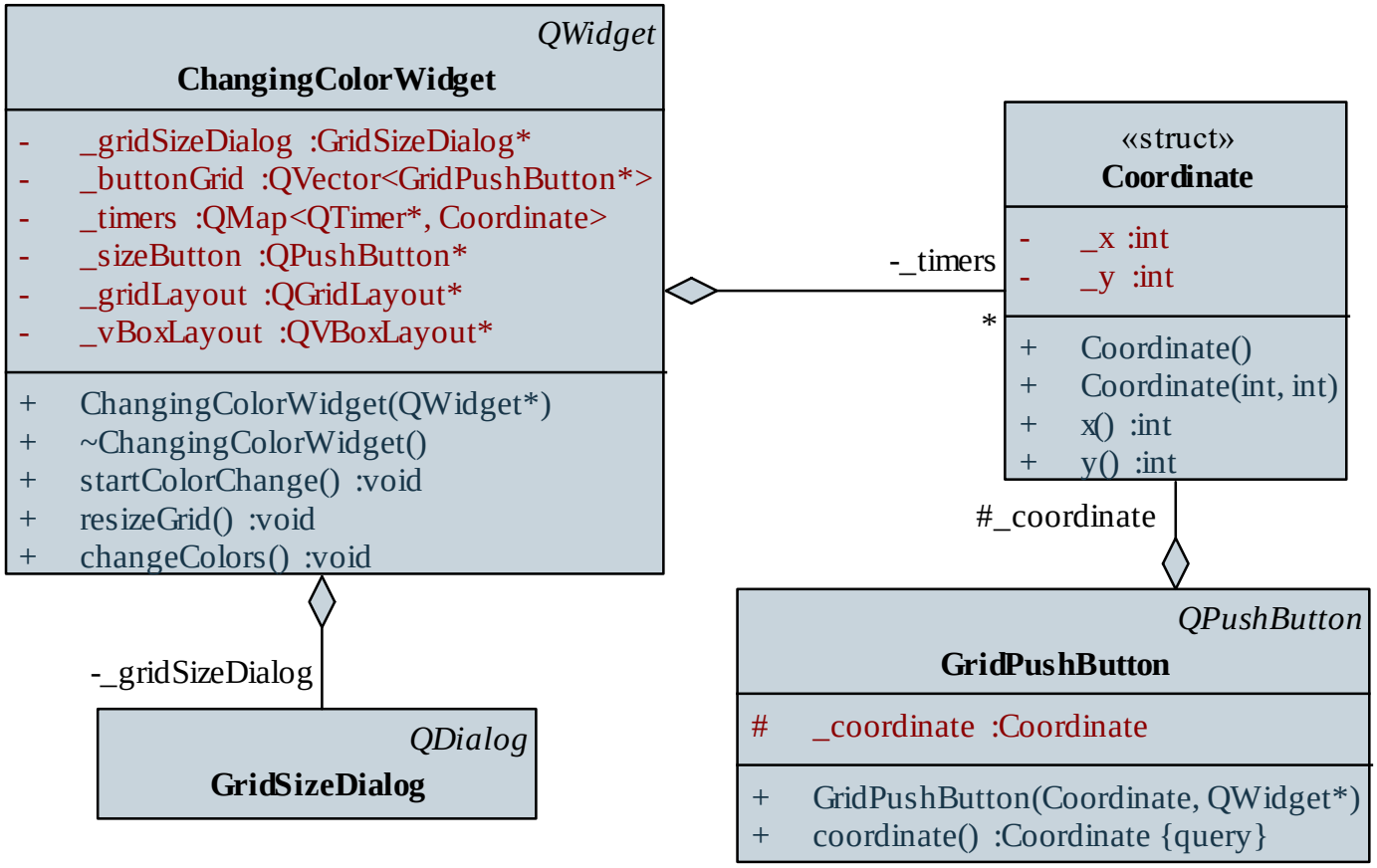
## Példa

- a gombok esetén is el kell tárolnunk a koordinátát, ezért létrehozunk egy speciális gombtípust (`GridPushButton`), amely tárolja a koordinátát is
- az időzítőkhöz közös eseménykezelőt kapcsolunk (`changeColors`), amelyben beazonosítjuk a küldő időzítőt (`sender()`), visszakeressük a hozzátartozó koordinátát, majd ez alapján átszínezzük a megfelelő gombokat
- egy külön ablak segítségével végezzük az átméretezést (`GridSizeDialog`), de ügyelnünk kell arra, hogy minden létező időzítőt leállítsunk, és kitöröljünk

# Dinamikus felületű alkalmazások

## Példa

*Tervezés:*



# Dinamikus felületű alkalmazások

## Példa

*Megvalósítás (*`changingcolorwidget.cpp`*):*

```
void ChangingColorWidget::startColorChange()
{
    // szükségünk van a küldő gomb koordinátájára
    GridPushButton *button =
       qobject_cast<GridPushButton*>(sender());
    Coordinate coordinate = button->coordinate();

    // létrehozunk egy új időzítőt, amit azonnal el
    // is indítunk
    QTimer* timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()),
            this, SLOT(changeColors()));
    timer->start(1000);
```

# Dinamikus felületű alkalmazások

## Példa

*Megvalósítás (*`changingcolorwidget.cpp`*):*

```
        // és elmentünk a megadott koordinátával
        _timers.insert(timer, coordinate);
    }

    …

    void ChangingColorWidget::changeColors()
    {
        …
        // megkeressük, melyik koordinátán kell
        // módosítani a színeket
        Coordinate coordinate =
            timers[qobject_cast<QTimer*>(sender())];
```

# Dinamikus felületű alkalmazások

## Példa

*Megvalósítás (*`changingcolorwidget.cpp`*):*

```
    // megkeressük az átszínezendő gombokat
    foreach(GridPushButton* button, _buttonGrid)
    {
       if (button->coordinate().x() ==
           coordinate.x() || …)
       {
          // az adott sorban és oszlopban
          button->setStyleSheet(styleSheet);
          // lecseréljük az összes gomb
          // megjelenését
        }
     }
}
```

# Dinamikus felületű alkalmazások

## Képek kezelése

- Nem csak színeket, képeket is elhelyezhetünk a felületen vagy a stílusban, vagy külön betöltve az alkalmazásban
- A Qt támogatja a legtöbb megszokott képformátumot (BMP, GIF, JPEG, PNG, …), a lista tovább bővíthető
- A képeket objektumként is kezeljük, erre több típus szolgál:
  - **`QImage`**: elsősorban I/O műveletekre és pixel szintű manipulációra optimalizál
  - **`QPixmap`**: elsősorban a képek felületi megjelenítésére szolgál
  - **`QBitmap`**: monokróm képek kezelésére
  - **`QPicture`**: képre történő rajzolást biztosít

# Dinamikus felületű alkalmazások

## Képek megjelenítése

- A képeket a felületre több vezérlő segítségével is felhelyezhetünk, vagy rajzolhatjuk
  - alapvetően a címke (**`QLabel`**) szolgál a képmegjelenítésre a **`pixmap`** tulajdonságon keresztül, mely egy **`QPixmap`** objektumot tud fogadni, pl.:

```
QPixmap* pic = new QPixmap("img.bmp");
label->setPixmap(*pic); // kép beállítása
```

  - a kép a címkén eredeti méretben jelenik meg, ha a címke mérete rögzített, akkor a képet megvágja
  - a képet lekicsinyíthetjük a **`scale(`*`<szélesség>`*`,`** *`<magasság>`*`, …)` művelettel, pl.:

```
label->setPixmap(pic->scale(50,50));
```

# Dinamikus felületű alkalmazások

## Példa

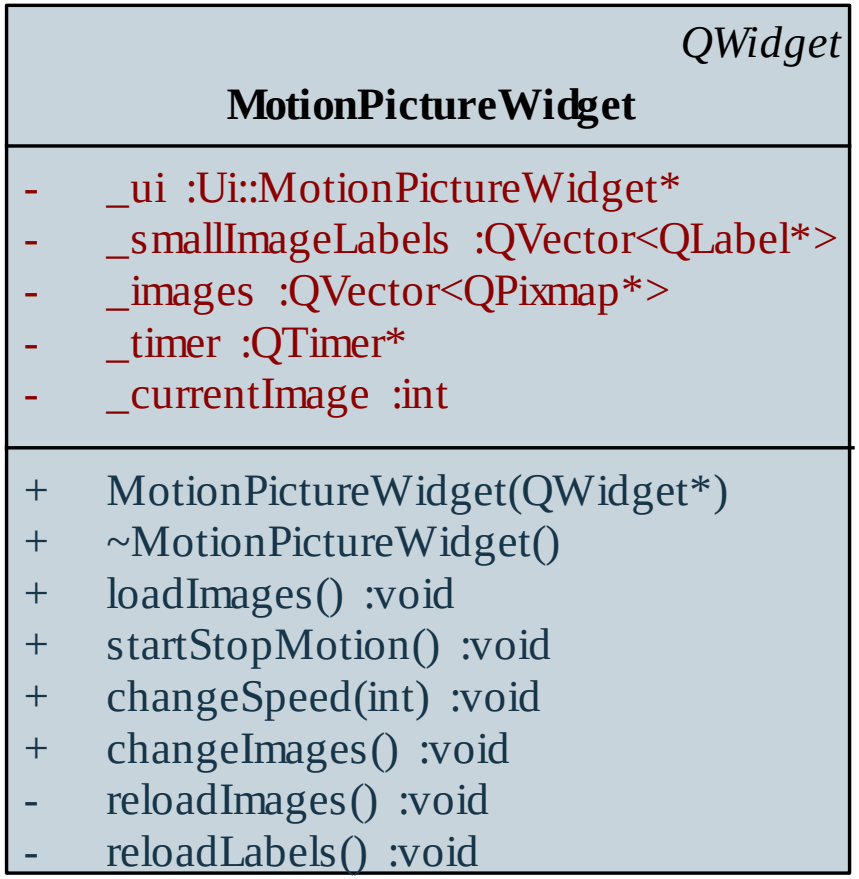
*Feladat:* Készítsünk egy mozgókép megjelenítő alkalmazást, amelyben képek sorozatát tudjuk betölteni (mint filmkockákat), és megjeleníteni azt animációként. Lehessen szabályozni az animáció sebességét, valamint lehessen látni, hogy a következő 1 másodpercben milyen képkockák jelennek meg.

- a felületnek lesz statikus (betöltő gomb, indító/leállító gomb, megjelenítő címke, sebességállító), valamint dinamikus része (egy másodpercnek megfelelő képek), előbbit a felülettervezővel készítjük
- eltároljuk a betöltött képeket (`images`), valamint a generált címkéket (`smallImageLabels`), és időzítő segítségével fogjuk periodikusán cserélni őket

# Dinamikus felületű alkalmazások

## Példa

*Tervezés:*



# Dinamikus felületű alkalmazások

## Példa

*Megvalósítás (*`motionpicturewidget.cpp`*):*

```
void MotionPictureWidget::loadImages(){
    …
    foreach(QFileInfo fileInfo, fileInfos){
        QPixmap* image = new QPixmap(
            fileInfo.absoluteFilePath());
        // képek betöltése
        if (!image->isNull()) // amennyiben képfájl
        _images.append(image);
            // felvesszük a képek közé
        else
            delete image; // különben töröljük
    }
    …
```

# Dinamikus felületű alkalmazások

## Példa

*Megvalósítás (*`changingcolorswidget.cpp`*):*

```
void MotionPictureWidget::reloadImages(){
   if (images.size() > 0) { // amennyiben van kép
      _ui->mainImageLabel->setPixmap(
        _images[_currentImage]->scaled(298, 298));
         // nagy kép beállítása
      for (int i = 0;
           i < _smallImageLabels.size(); i++)
         _smallImageLabels[i]->setPixmap(
            _images[(_currentImage + i + 1) %
            _images.size()]->scaled(18,18));
            // kis képek beállítása
   }
}
```