Dinamikusan változó felületű alkalmazások

Futás közben változó felület,
stílusok, időzítő, képek

Vezérlők megjelenési módja

- Az alkalmazások grafikus felhasználói felületein alapvetően kétféle vezérlővel találkozhatunk:
  - statikus: az alkalmazás induláskor jönnek létre és kerülnek elhelyezésre, és az alkalmazás futása során nem változnak.
    - Ezeket a vezérlőket létrehozhatjuk a felülettervezővel, vagy közvetlenül a kódban, de ekkor az ablakok konstruktorában.
  - dinamikus: az alkalmazás futása közben
    - jönnek létre és kerülnek elhelyezésre,
    - változtatják a helyüket
    - meg is szűnhetnek (ez esetben egy vezérlő eseménykezelő társításai is megszűnnek: a `disconnect` hívódik meg a háttérben).

1.Feladat

Készítsünk egy alkalmazást, amelyik véletlenszerűen változtatja meg minden másodpercben egy ablak színét. A színváltoztatás animációját ki/be kapcsolhatjuk.

Időzítés

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

Példa

```
…
_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
}
```

Szövegkezelés

- Qt-ben a karakterek 16 bites Unicode (UTF8) kódolásúak.
  - Ehhez a **`QObject`** típus biztosít konverziót egy osztályszintű művelettel (**`QObject::trUtf8`**).
- A karakterek kezelését a **`QChar`** típus biztosítja, míg szövegre a **`QString`** típus alkalmazható.
  - kompatibilis a C++ standard könyvtár **`string`** típusával, pl.:

    `QString::fromSTDString(stdstr)`
  - megkülönbözteti az üres és a nem létező szöveget (**`isNull`**, **`isEmpty`**)
  - alkalmas típuskonverziókra, pl.:

    `QString::number(4)`, **`str.toInt()`**

Véletlenszám generátor

- A Qt a C++-ban használt véletlenszám generáláshoz hasonló eszközzel van ellátva.
  - A generátort a **`qsrand(<kezdőérték>)`** függvény indítja, amelyet időfüggő értékkel indítunk (**`QTime::currentTime().msec()`**).
  - Egy véletlen szám előállítására a **`qrand()`** függvény szolgál.

```
qsrand(QTime::currentTime().msec());
…
int n = qrand() % 256;    // 0 .. 255 közötti véletlen szám
```

Stílusok

- A vezérlők megjelenését a **`styleSheet`** tulajdonság segítségével szabályozhatjuk: ezt különböző stílusokkal láthatjuk el.
- A stílusokat CSS (Cascading Style Sheets) szerű leírással adjuk meg szöveg (string) formájában.

a nyomógomb fehér háttéren piros betűs lesz

```
"QPushButton { color: red; background-color: white }"

"QCheckBox:hover:checked { color: white }"

"QLineEdit {
    background-image: url(:/images/bck.png);
    border-width: 1px;
    border-style: solid;
    border-radius: 4px;
}"
```

ha az egérkurzor a kiválasztott kijelölő doboz felett van, akkor fehér színű lesz a szöveg

a hátteret kép tölti ki, keret megformázva

Alkalmazás szintű stílus

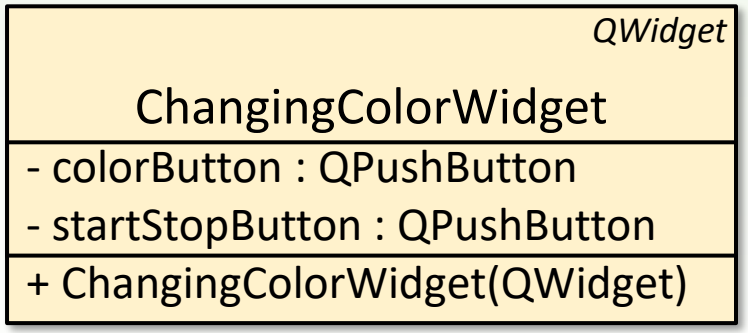
- A szabályozása történhet az egész alkalmazás, illetve bármely vezérlő szintjén.

```
QApplication::setStyleSheet(
     "QPushButton { color:black }
      QPushButton:enabled { color:red }"
);
```

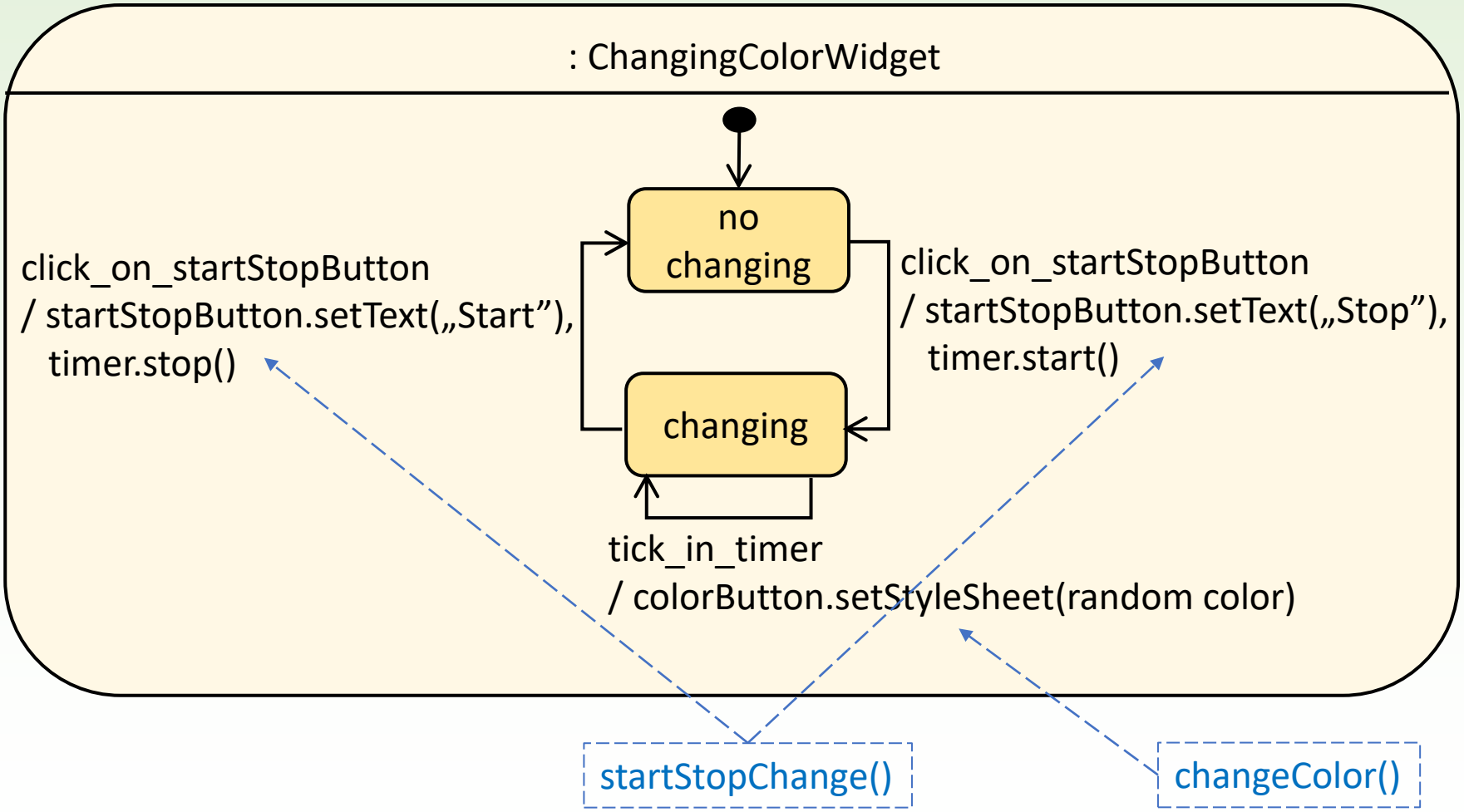
minden engedélyezett nyomógomb piros, minden nem engedélyezett nyomógomb fekete feliratú lesz

1.Feladat: tervezés

- Egy saját ablakon két nyomógomb legyen:
  - az egyik színe fog véletlenszerűen változni,
  - a másik megnyomásával lehet a színváltozást elindítani. illetve megállítani.

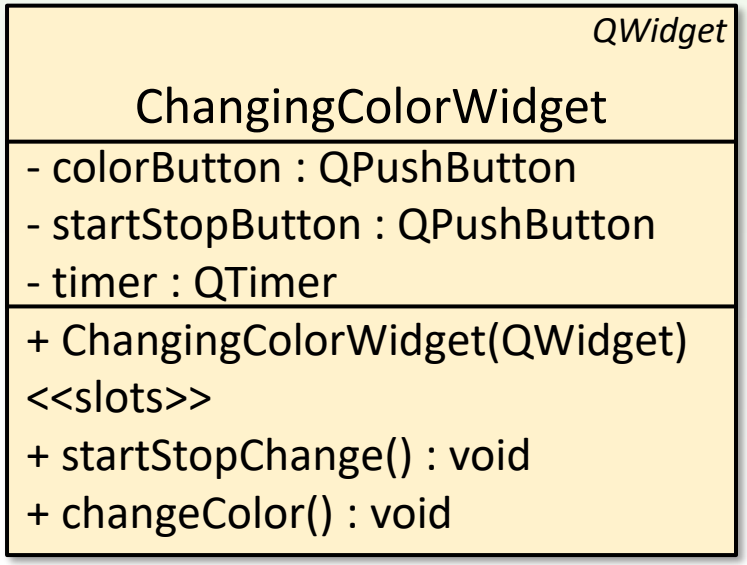


1.Feladat: tervezés



1.Feladat: tervezés

- Kell egy időzítő (**`QTimer`**), amelyiknek **`timeout()`** szignáljához kötjük a „színváltó” nyomógomb színét változtató **`changeColor()`** eseménykezelőt.
- A színváltoztatást elindító, illetve megállító nyomógomb clicked() szignálja aktíválja a **`coloringStartStop()`** eseménykezelőt.



1.Feladat: megvalósítás

```
ChangingColorWidget::ChangingColorWidget(QWidget *parent)
: QWidget(parent)
{
    setWindowTitle(tr("Változó színű gomb"));
    setFixedSize(270, 300);

    _colorButton = new QPushButton("", this);
    _colorButton->setGeometry(0,0,270,270);
    _colorButton->setEnabled(false); // gomb kikapcsolása

    _startStopButton = new QPushButton(tr("Start"), this);
    _startStopButton->setGeometry(0, 270, 270, 30);
    connect(_startStopButton, SIGNAL(clicked()),
                    this, SLOT(coloringStartStop()));

    _timer = new QTimer(this); // időzítő
    connect(_timer, SIGNAL(timeout()), this, SLOT(changeColor()));

    qsrand(QTime::currentTime().msec()); // szám generátor indul
}
```

signal-slot kapcsolatok

1.Feladat: megvalósítás

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

színváltás elindítása/megállítása

```
void ChangingColorWidget::changeColor() // 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) + ") }");
}
```

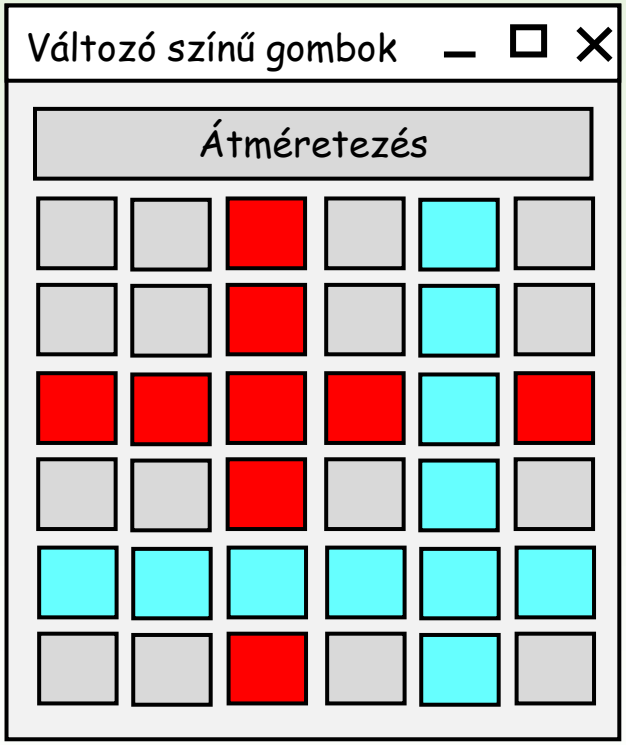
színváltó nyomógomb átszínezése

2.Feladat

Készítsünk nyomógombokból egy rácsot, amelynek méretét futás közben is lehessen majd változtatni.

Amikor a rács egyik gombjára kattintunk, induljon el rajta az előző feladatban már látott színváltó animáció, de ne csak ezen a gombon, hanem ezzel együtt keresztalakban (azaz a kattintott gomb teljes sorában és oszlopában) minden nyomógomb ugyanúgy változtassa a színét.

Újabb kattintásokkal újabb keresztalakzatok nyomógombjai fogják egyszerre változtatni a színüket.



Azonos vezérlők csoportja

- Azonos vezérlők csoportját célszerű mindig kódból létrehozni. Ezek nagy száma esetén ugyanis a felülettervező használata kényelmetlen.
- Ráadásul, ha ezeket nem a program indulásakor kell létrehozni, és/vagy a futás során meg is szűnhetnek, majd újra létrejöhetnek, azaz számuk és elhelyezkedésük dinamikusan változik, akkor ezek megadására a felülettervező alkalmatlan.
- Azonos típusú vezérlőket célszerű közös adatszerkezetbe szervezni, és a közös funkciójukat hozzájuk kapcsolt közös eseménykezelőkbe tenni.
- Egy közös eseménykezelőben meghatározhatjuk, melyik vezérlő váltotta ki, így egyedire tudjuk szabni a viselkedését.
  - A **`sender()`** (vagy **`QObject::sender()`**) művelet adja vissza **`QObject`** mutatóként egy eseményt jelző szignál küldőjét.
  - Ezt konvertálhatjuk megadott altípusra a **`qobject_cast<T>`** utasítással.

Példa

```
QVector<QPushButton*> buttons;

for (…) // gombok létrehozása egy ciklusban
{
    QPushButton* button = new QPushButton(this);
    buttons.append(button); // új gomb hozzávétele
    connect(button, SIGNAL(clicked()), this, SLOT(buttonClicked()));
}

void buttonClicked() // eseménykezelő
{
    QObject* senderObject = sender(); // küldő objektum lekérdezése

    QPushButton* senderButton =
            qobject_cast<QPushButton*>(senderObject);
            // a küldő típusát konvertálnunk kell
    senderButton->setText(tr("You clicked me!"));
}
```

nyomógombokat tároló konténer

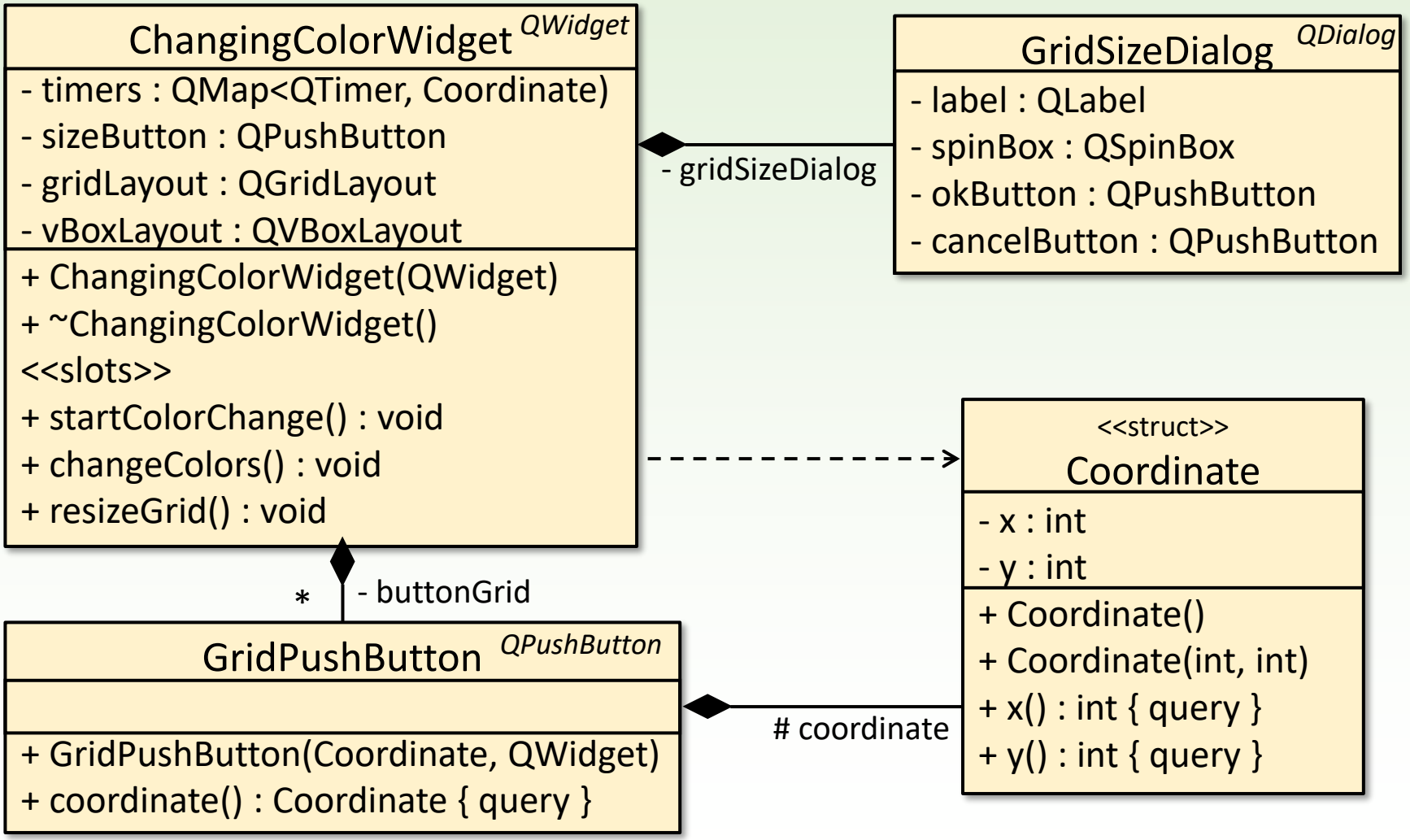
közös eseménykezelő

tudjuk, hogy a küldő objektum egy nyomógomb

2.Feladat: tervezés

- Minden színváltós nyomógombra történő kattintásra egy újabb időzítőt indítunk el.
- Az időzítőket és a színváltó nyomógombokat a sor, oszlop koordinátájuk alapján kapcsoljuk össze, ezért
  - létrehozunk egy koordináta segédtípust (**`Coordinate`**)
  - a színváltó nyomógombok egy speciális osztály (**`GridPushButton`**) példányai lesznek, amelyek tárolják a koordinátát is
  - az időzítőket egy asszociatív tömbben (**`QMap`**) tároljuk úgy, hogy az időzítő lesz a kulcs, a hozzátartozó érték pedig a koordináta
  - az időzítőkhöz közös eseménykezelőt kapcsolunk, amely az időzítő koordinátái alapján azonosítja az átszínezendő nyomógombokat
- Az átméretezést kiváltó nyomógom lenyomásakor egy dialógus ablak nyílik meg (**`GridSizeDialog`**), amelyben az új méret adható meg. Átméretezésnél ügyeljünk arra, hogy létező időzítőkett leállítsuk, és töröljük.

2.Feladat: tervezés



2.Feladat: megvalósítás

```
ChangingColorWidget::ChangingColorWidget(QWidget *parent)
: QWidget(parent)
{
    setWindowTitle(tr("Változó színű gombok"));
    setBaseSize(270, 300);
    _sizeButton = new QPushButton(tr("Átméretezés"));
    _gridLayout = new QGridLayout();
    _vBoxLayout = new QVBoxLayout();
    _vBoxLayout->addWidget(_sizeButton);
    _vBoxLayout->addLayout(_gridLayout);
    setLayout(_vBoxLayout);

    _gridSizeDialog = new GridSizeDialog();
    connect(_sizeButton,      SIGNAL(clicked()),
            _gridSizeDialog, SLOT(exec()));
    connect(_gridSizeDialog,      SIGNAL(accepted()),
            this,                 SLOT(resizeGrid()));
    qsrand(QTime::currentTime().msec());
}
```

signal-slot kapcsolatok

2.Feladat: megvalósítás

```
void ChangingColorWidget::resizeGrid()
{
   foreach(QTimer* timer, _timers.keys()) {
        timer->stop();                  // időzítők leállítása és törlése
        _timers.remove(timer);
        delete timer;
   }
   foreach(GridPushButton* button, _buttonGrid) {
        _gridLayout->removeWidget(button);
        delete button;                  // nyomógombok eltávolítása és törlése
   }
   _buttonGrid.clear();
   for (int i = 0; i < _gridSizeDialog->gridSize(); ++i) {
        for (int j = 0; j < _gridSizeDialog->gridSize(); ++j){
             GridPushButton* button = new GridPushButton(Coordinate(i, j));
             _gridLayout->addWidget(button, i, j);
             _buttonGrid.append(button);
             connect(button, SIGNAL(clicked()),
                     this   , SLOT(startColorChange()));
        }
   }
}
```

signal-slot kapcsolatok

2.Feladat: megvalósítás

```
void ChangingColorWidget::startColorChange()
{
    GridPushButton *button = qobject_cast<GridPushButton*>(sender());
    Coordinate coordinate = button->coordinate();

    QTimer* timer = new QTimer(this);
        // új időzítő létrehozás a küldő nyomógombhoz
    connect(timer, SIGNAL(timeout()), this, SLOT(changeColors()));
    timer->start(1000);   // új időzítő elindítása
    _timers.insert(timer, coordinate);
        // új időzítő elhelyezése a map-ben a koordinátáival
}
```

signal-slot kapcsolatok

2.Feladat: megvalósítás

```
void ChangingColorWidget::changeColors()
{ // adott centrumú nyomógombok átszínezése
    QString styleSheet = "QPushButton { background-color: rgb("
        + QString::number(qrand()%256) + ","
        + QString::number(qrand()%256) + ","
        + QString::number(qrand()%256) + ") }";

    Coordinate coordinate = timers[qobject_cast<QTimer*>(sender())];
    foreach(GridPushButton* button, _buttonGrid){
        if( button->coordinate().x()==coordinate.x()
                || button->coordinate().y()==coordinate.y()){
            button->setStyleSheet(styleSheet);
        }
    }
}
```

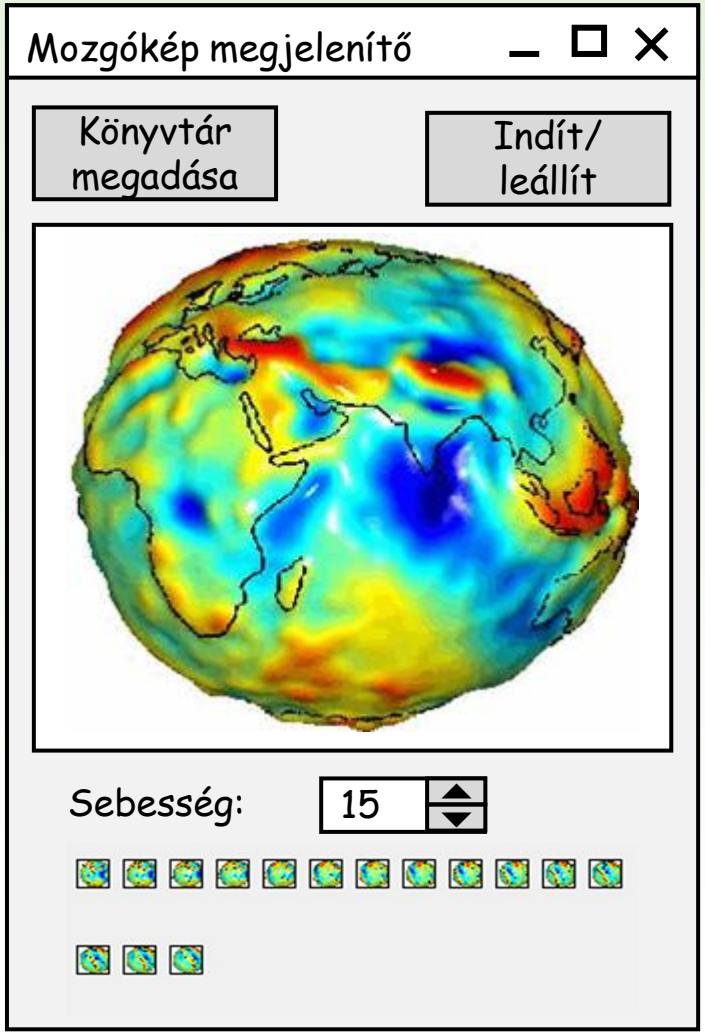
lekérjük a timers-től az időzítő koordinátáit

3.Feladat

Készítsünk egy mozgókép megjelenítő alkalmazást, amelybe képek (filmkockák) sorozatát kell tudjuk betölteni, és a filmet ciklikusan ismétlődve úgy vetítsük le, hogy a filmkockákat egymás után megjelenítjük.

A lejátszás sebességét szabályozhassuk azzal, hogy megadjuk az 1 másodperc alatt egymás után megjelenítendő filmkockák számát.

Egy külön képsorozattal (ennek hossza a sebességtől függ) mindig mutassuk meg a soron következő másodpercben vetítésre kerülő filmkockákat.



Képek kezelése

QImage: pixel szintű manipulációhoz
QPixmap: képek felületi megjelenítésére
QBitmap: monokróm képek kezelésére
QPicture: képre történő rajzolást biztosít

- A Qt támogatja a legtöbb megszokott képformátumot: BMP, GIF, JPEG, PNG, … . Ezeket dinamikusan is betölthetjük az alkalmazásba, de a stílus megadásánál közvetlenül is beállíthatjuk.
- A képek **QImage**, **QPixmap**, **QBitmap**, **QPicture** objektumok.
- A képeket a felületre többféle vezérlő segítségével is felhelyezhetünk, de alapvetően a
  - címke (**QLabel**) szolgál képmegjelenítésre a **pixmap** tulajdonságán keresztül, mely egy **QPixmap** objektumot tud fogadni, pl.:

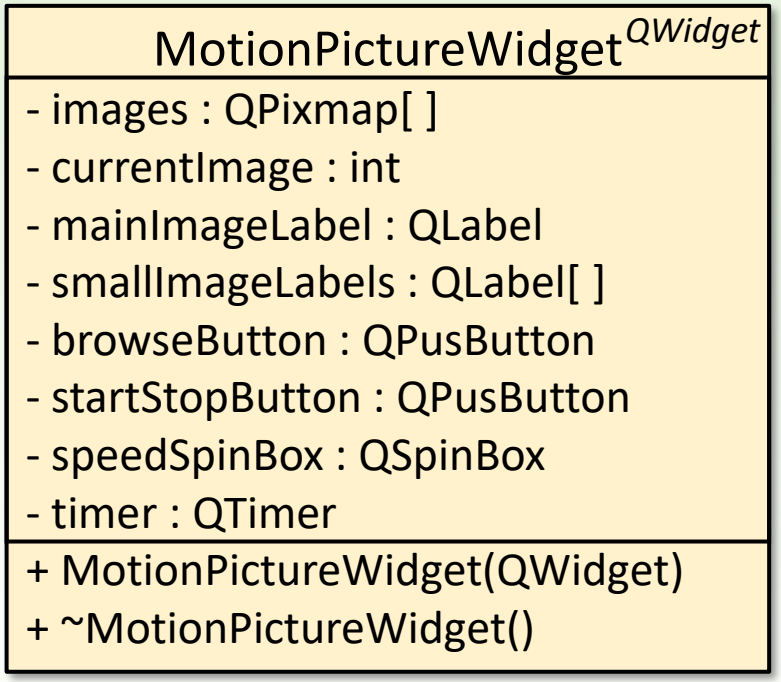
```
QPixmap* pic = new QPixmap("img.bmp");
label->setPixmap(*pic);
```

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

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

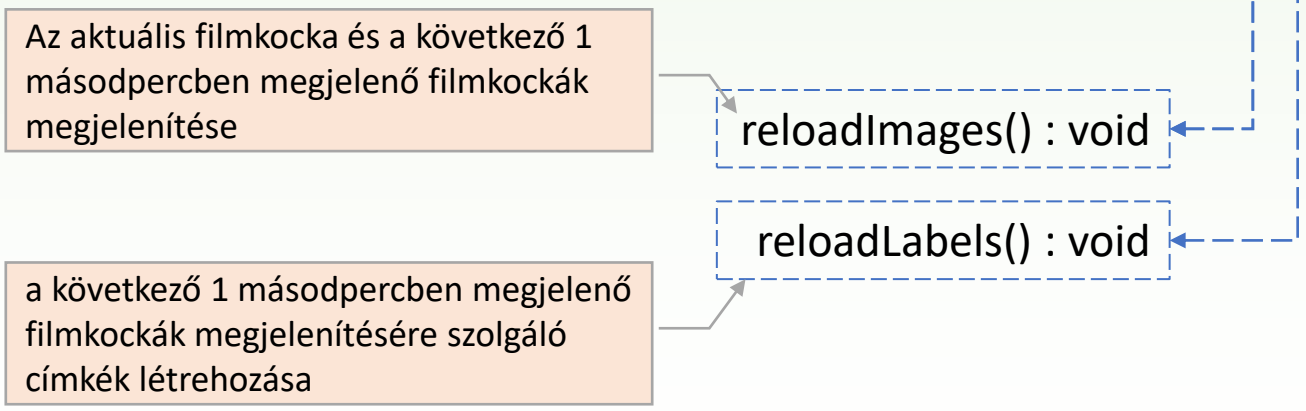
3.Feladat: tervezés

- A felület statikus részét (képeket betöltő nyomógombot, mozgást indító/leállító nyomógombot, kép-megjelenítő címkét, sebességállító számlálót) a QtDesigner-rel készítjük el. Dinamikusan kerülnek viszont fel a felületre a következő másodpercben megjelenő kis képek, hiszen ezek száma a sebességtől függ.
- Eltároljuk a betöltött képeket (**`images`**), valamint a generált címkéket (**`smallImageLabels`**), és időzítő segítségével fogjuk periodikusán cserélni őket.



3.Feladat: eseménykezelés

sender	signal	reciever	slot
browseButton	clicked()	: MotionPictureWidget	loadImages()
startStopButton	clicked()	: MotionPictureWidget	startStopMotion()
speedSpinBox	valueChanged(int)	: MotionPictureWidget	changeSpeed(int)
timer	timeout()	: MotionPictureWidget	changeImages()



3.Feladat: megvalósítás

```
MotionPictureWidget::MotionPictureWidget(QWidget *parent)
: QWidget(parent), _ui(new Ui::MotionPictureWidget)
{
    _currentImage = 0;
    _ui->setupUi(this);
    connect(_ui->browseButton,     SIGNAL(clicked()),
            this,                  SLOT(loadImages()));
    connect(_ui->startStopButton, SIGNAL(clicked()),
            this,                  SLOT(startStopMotion()));
    connect(_ui->speedSpinBox,     SIGNAL(valueChanged(int)),
            this,                  SLOT(changeSpeed(int)));
    _timer = new QTimer(this);
    connect(_timer, SIGNAL(timeout()), this, SLOT(changeImages()));
}

MotionPictureWidget::~MotionPictureWidget()
{
    foreach(QPixmap* image, _images) { delete image; }
    foreach(QLabel* label, _smallImageLabels) { delete label; }
    delete _ui;
}
```

signal-slot kapcsolatok

3.Feladat: megvalósítás

```
void MotionPictureWidget::loadImages()
{
     // könyvtár megnyitó dialógusablak
     QString dirName = QFileDialog::getExistingDirectory(this,
          tr("Könyvtár megnyitása"), "", QFileDialog::ShowDirsOnly);

     if (!dirName.isNull()) {
          if (_timer->isActive()) _timer->stop();
          QDir dir(dirName);
          dir.setFilter(QDir::Files);
          dir.setSorting(QDir::Name);
          QFileInfoList fileInfos = dir.entryInfoList();
          foreach(QPixmap* image, _images) { delete image; }
          _images.clear();
          foreach(QFileInfo fileInfo, fileInfos) {
               // könyvtárbeli képek betöltése
               QPixmap* image = new QPixmap(fileInfo.absoluteFilePath());
               if (!image->isNull()) _images.append(image);
               else delete image;
          }
          …
```

a kiválasztott mappa fájljai

Az **images** felöltése a kiválasztott mappa képeivel.

3.Feladat: megvalósítás

```
void MotionPictureWidget::loadImages()
{
    …
    if (!dirName.isNull()) {
        …
        if (_images.size() > 0) {
            _ui->speedSpinBox->setMinimum(1);
            _ui->speedSpinBox->setMaximum(_images.size());
            _ui->speedSpinBox->setValue(_images.size());
        } else {
            _ui->speedSpinBox->setMinimum(0);
            _ui->speedSpinBox->setMaximum(0);
            _ui->speedSpinBox->setValue(0);
            QMessageBox::warning(this, tr("Hiba!"),
                    tr("A könyvtár nem tartalmazott képeket!"));
        }
        reloadLabels();
        reloadImages();

        _currentImage = 0; // az első képpel kezdünk
    }
}
```

speedSpinBox tulajdonságainak beállítása

A kis képeket megjelenítő címkéket hozza létre a **smallImageLayout**-ban

ez felel a képek megváltozásáért

3.Feladat: megvalósítás

```
void MotionPictureWidget::reloadLabels() // kis képek címkéi
{
     foreach(QLabel* label, _smallImageLabels) {
          _ui->smallImageLayout->removeWidget(label);
          delete label;
     }
     _smallImageLabels.clear();

     for (int i = 0; i < _ui->speedSpinBox->value(); i++) {
          QLabel* label = new QLabel();
          label->setFixedSize(18,18); // rögzített méretű lesz a címke
          label->setFrameShape(QFrame::Box); // egyszerű keret
          _ui->smallImageLayout->addWidget(label, i / 12, i % 12);
          _smallImageLabels.append(label);
     }
}
```

Megszűnteti, majd megadott számban újra létrehozza azokat a kis címkéket, amelyek mindig a soron következő másodpercben levetítésre kerülő képéket fogjak mutatni.

3.Feladat: megvalósítás

```
void MotionPictureWidget::reloadImages()
{
    if (_images.size() > 0) { // amennyiben vannak képek
        // nagy kép beállítása:
        _ui->mainImageLabel->setPixmap(
                _images[_currentImage]->scaled(QSize(298, 298)));

        // kis képek beállítása:
        for (int i = 0; i < _smallImageLabels.size(); i++)
            _smallImageLabels[i]->setPixmap(
                _images[(_currentImage + i + 1) %
                    _images.size()]->scaled(QSize(18,18)));
    }
}
```

Az aktuális filmkocka elhelyezése a nagy címke felületén.

Az aktuális filmkocka után következő adott számú filmkockát helyezi el a kis címkék felületén.

3.Feladat: megvalósítás

```
void MotionPictureWidget::startStopMotion()
{
    if (!_timer->isActive() && _images.size() > 0) {
        _timer->start(1000 / _ui->speedSpinBox->value());
    } else // ha most fut az időzítő, leállítjuk {
        _timer->stop();
    }
}
void MotionPictureWidget::changeSpeed(int value)
{
    if (_images.size() > 0 && _timer->isActive()) {
        _timer->stop(); _timer->start(1000 / value);
    }
    reloadLabels();
    reloadImages();
}

void MotionPictureWidget::changeImages() {
    if (_images.size() > 0)
        _currentImage = (_currentImage + 1) % _images.size();
    reloadImages();
}
```

vetítés indítása/megállítása

vetítés sebességének átállítása

a következő kép