

Dinamikus felületű alkalmazások

Stílusok, időzítő, képek

Felhasználói felület fajtái

- Az alkalmazásaink grafikus felülete alapvetően kétféle lehet:
 - *statikus*: az alkalmazás felületén lévő vezérlőket induláskor rögzítjük, így minden alkalommal ugyanazon vezérlők jelennek meg
 - a vezérlőket a felülettervezővel, vagy közvetlenül a kódban (ablakok konstruktorában) hozzuk létre
 - *dinamikus*: futás közben változhatnak a felületen megjelenő vezérlők
 - a vezérlőket futás közben hozzuk létre és helyezzük el a felületen, és szüntetjük meg: ekkor a hozzákötött eseménykezelő társítások is megszűnnek (a **disconnect** hívódik meg a háttérben).
 - emellett természetesen lehet állandó része is a felületnek rögzített vezérlőkkel

Azonos típusú vezérlők csoportja

- ❑ Azonos típusú vezérlőket célszerű egy közös adatszerkezetbe szervezni, és egy közös eseménykezelőt rendelni hozzájuk.
 - A közös eseménykezelőben meghatározhatjuk, melyik vezérlő váltotta ki, és így tudjuk a viselkedését egyedire szabni.
 - A `sender()` (vagy `QObject::sender()`) művelet adja vissza az esemény küldőjét `QObject` mutatóként.
 - 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!"));
}
```

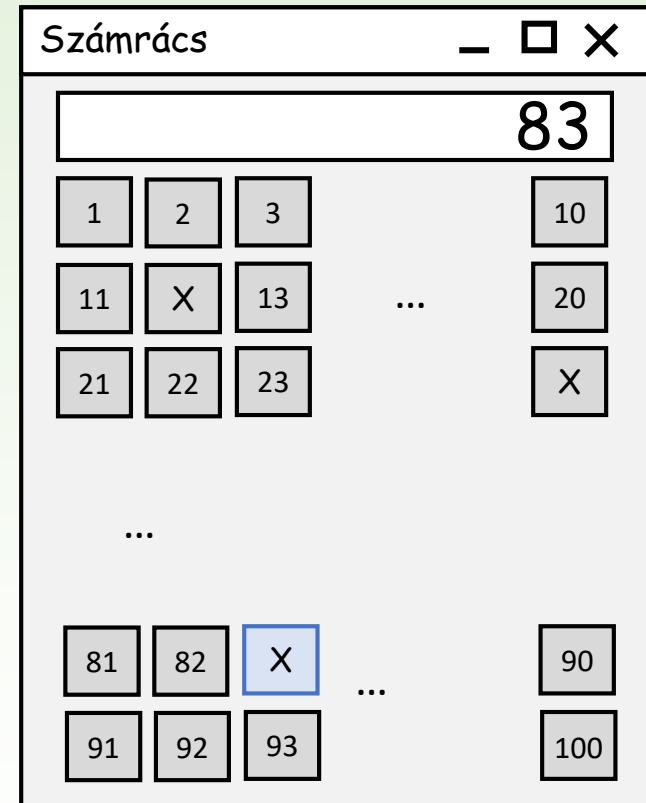
közös eseménykezelő

tudjuk, hogy a küldő objektum egy nyomógomb

1.Feladat

Készítsünk egy olyan alkalmazást, amely egy 10×10-es rácsban száz darab 1-től kezdődően sorszámozott nyomógombot jelenít meg.

Egy nyomógombra kattintva a gomb sorszáma jelenjen meg egy központi kijelzőn, a gombon pedig ettől kezdve az „X” felirat látszódjon. Később az ilyen feliratú gombokra kattintva már ne történjen semmi.



1.Feladat: tervezés

- ❑ A felületen felveszük egy LCD kijelzőt, valamint a nyomógombokat rács elrendezésben, a gombok feliratának (**text**) beállítjuk a sorszámot.
- ❑ A nyomógombokhoz közös eseménykezelőt (**setNumber()**) rendelünk, amely a küldő gomb (**sender()**) sorszámát beírja az LCD kijelzőbe (feltéve, hogy nem ,X' feliratú), a gomb feliratát pedig lecseréli ,X'-re.

| <i>QWidget</i> NumberGridWidget | |
|---|---|
| - | <code>_lcdNumber :QLCDNumber*</code> |
| - | <code>_gridLayout :QGridLayout*</code> |
| - | <code>_vBoxLayout :QVBoxLayout*</code> |
| + | <code>NumberGridWidget(QWidget*)</code> |
| + | <code>~NumberGridWidget()</code> |
| «slot» | |
| - | <code>setNumber() :void</code> |

1.Feladat: megvalósítás

```
NumberGridWidget::NumberGridWidget(QWidget *parent) : QWidget(parent)
{
    setWindowTitle(tr("Számrács"));
    setFixedSize(400,400);

    _lcdNumber = new QLCDNumber();

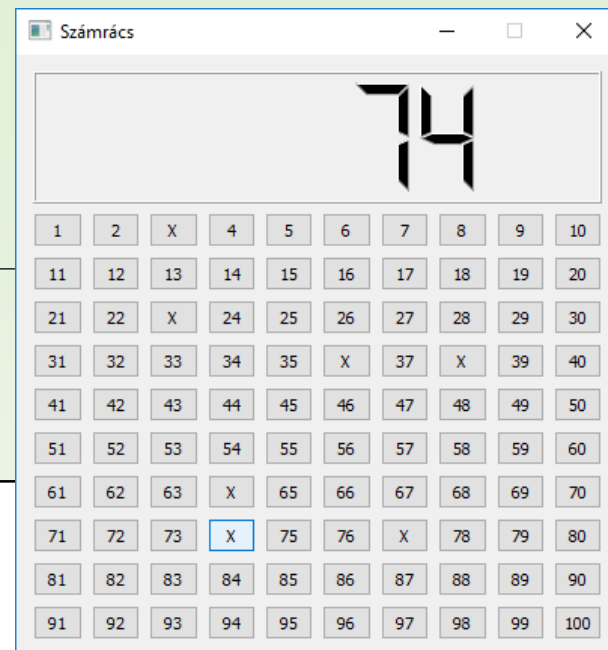
    _gridLayout = new QGridLayout(); // rácsszerkezet a nyomógomboknak
    for (int i = 0; i < 10; ++i) {
        for (int j = 0; j < 10; ++j){
            QPushButton* button = new QPushButton(
                QString::number(i * 10 + j + 1), this);
            _gridLayout->addWidget(button, i, j);
            connect(button, SIGNAL(clicked()), this, SLOT(setNumber()));
        }
    }
    _vBoxLayout = new QVBoxLayout(); // függőleges elhelyezés
    _vBoxLayout->addWidget(_lcdNumber);
    _vBoxLayout->addLayout(_gridLayout);
    setLayout(_vBoxLayout);
}
```

új gomb létrehozása és
a rácsba helyezése

1.Feladat: megvalósítás

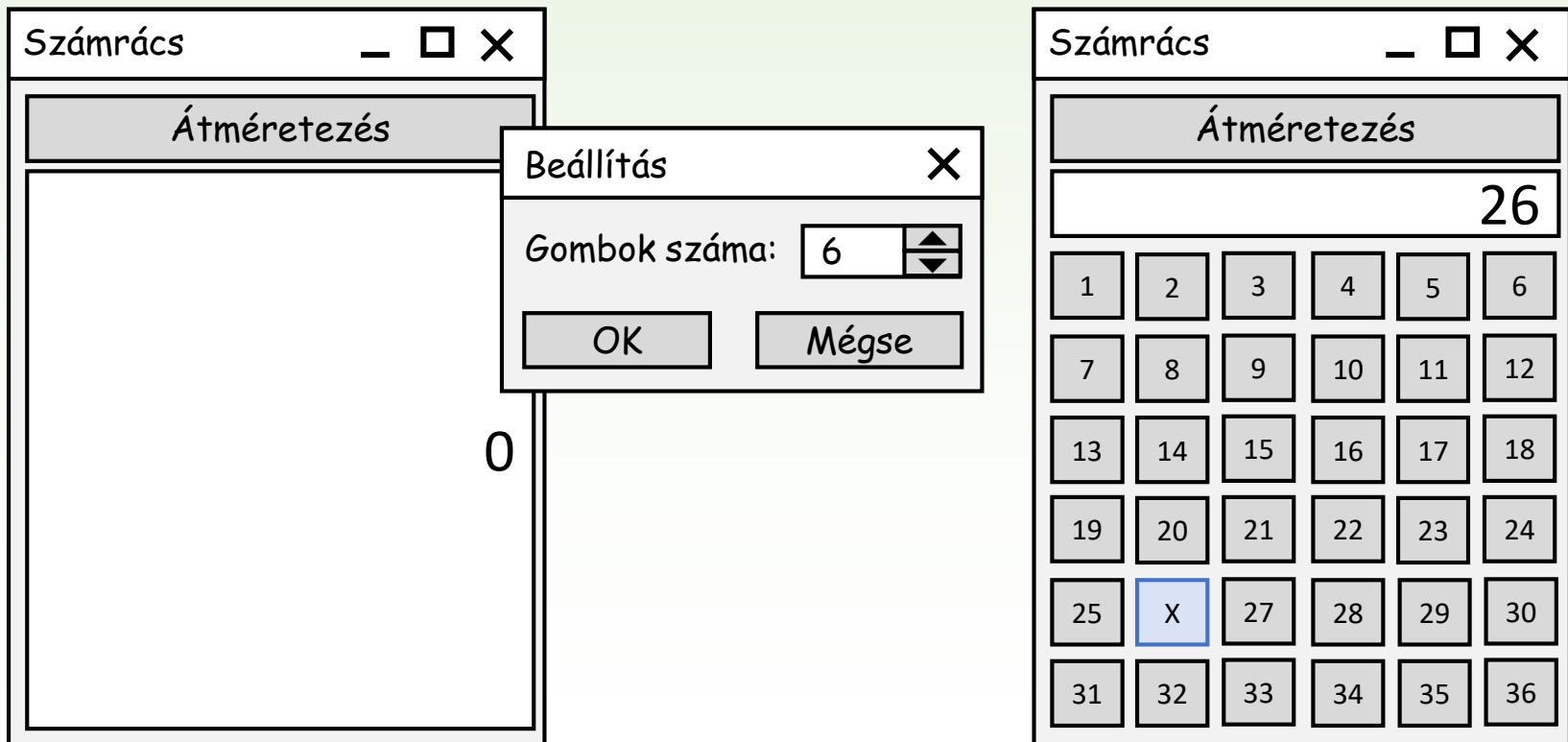
```
void NumberGridWidget::setNumber()
{
    QPushButton* senderButton =
        qobject_cast<QPushButton*>(sender());

    if (senderButton->text() != tr("X")){
        lcdNumber->display(senderButton->text());
        // megjelenítjük az LCD-n a nyomógomb sorszámát
        senderButton->setText(tr("X"));
        // a nyomógombra beállítunk egy X feliratot
    }
}
```



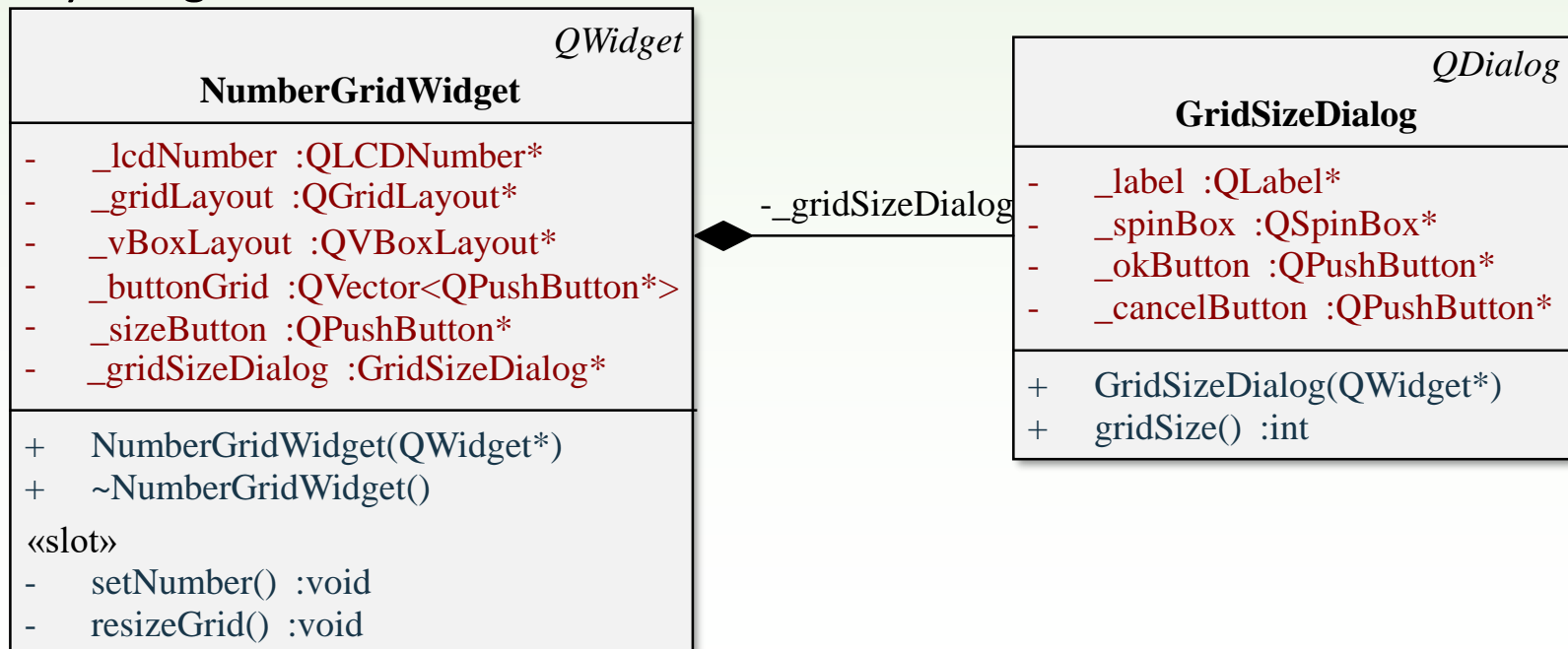
2.Feladat

Az előző feladat nyomógombrácsát futás közben át lehessen méretezni. Kezdetben a rács üres. Egy külön nyomógomb felhoz egy előugró ablakot, amelyben beállíthatjuk az új méretet. Ennek hatására a rács méreteződjön át, és minden nyomógombján látszódjék annak sorszáma.



2.Feladat: tervezés

- Minden átméretezésnél (**resizeGrid()**) töröljük a régi nyomógombokat, és megfelelő mennyiségű új gombot generáljunk.
- A méretet beállító számlálót (**QSpinBox**) a **QDialog**-ból származtatott osztállyal (**GridSizeDialog**) leírt modális ablak tartalmazza. Az örökölt **accept()**, illetve **reject()** szignálokat az OK, illetve a Mégse feliratú nyomógombra kattintással válthassuk ki.



2.Feladat: megvalósítás

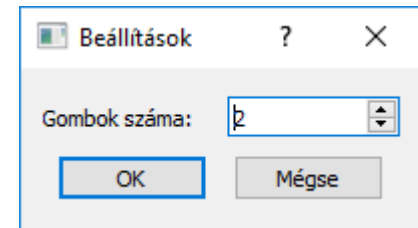
```
GridSizeDialog::GridSizeDialog(QWidget *parent) : QDialog(parent)
{
    setFixedSize(200,80);
    setWindowTitle(tr("Beállítások"));
    setModal(true);

    _label = new QLabel(tr("Gombok száma: "));
    _spinBox = new QSpinBox();
    _spinBox->setRange(2, 10);    // az értékek 2 és 10 közöttiek
    _spinBox->setSingleStep(2);   // lépésköz 2

    _okButton = new QPushButton(tr("OK"));
    _okButton->setFixedSize(75, 23); // a gombok mérete rögzített
    _cancelButton = new QPushButton(tr("Mégse"));
    _cancelButton->setFixedSize(75, 23);

    connect(_okButton, SIGNAL(clicked()), this, SLOT(accept()));
    connect(_cancelButton, SIGNAL(clicked()), this, SLOT(reject()));

    ... // elrendezés
}
```



dialógus metódusai


2.Feladat: megvalósítás

```
NumberGridWidget::NumberGridWidget(QWidget *parent) : QWidget(parent)
{
    setWindowTitle(tr("Számrács"));
    setFixedSize(400,400);

    _sizeButton = new QPushButton(tr("Átméretezés"));
    _lcdNumber = new QLCDNumber();
    _gridLayout = new QGridLayout();
    _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

    _vBoxLayout = new QVBoxLayout();
    _vBoxLayout->addWidget(_lcdNumber);
    _vBoxLayout->addLayout(_gridLayout);
    setLayout(_vBoxLayout);
}
```



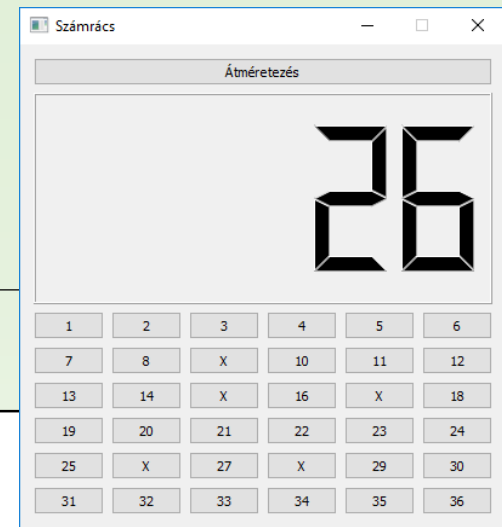
majd a destruktóban
törölni kell

2.Feladat: megvalósítás

a nyomógombok hivatkozásait
egy külön tömbben tároljuk

```
void NumberGridWidget::resizeGrid() {  
    foreach(QPushButton* button, buttonGrid) {  
        _gridLayout->removeWidget(button); // levétele az elrendezésről  
        delete button; // nyomógombok törlése  
    }  
    _buttonGrid.clear(); // nyomógombok tömbjének törlése  
  
    for (int i = 0; i < gridSizeDialog->gridSize(); ++i) {  
        for (int j = 0; j < _gridSizeDialog->gridSize(); ++j) {  
            QPushButton* button = new QPushButton(  
                QString::number(i * _gridSizeDialog->gridSize() + j + 1));  
            _gridLayout->addWidget(button, i, j);  
            _buttonGrid.append(button); // elmentés a tömbbe  
            connect(button, SIGNAL(clicked()), this, SLOT(setNumber()));  
        }  
    }  
}
```

új rács építése



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.

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

a nyomógomb fehér
háttéren piros betűs lesz

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

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

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

a háttérkép tölti ki,
keret megformázva

Alkalmazás szintű stílus

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

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

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

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
}
```

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
```

3.Feladat

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



3.Feladat: tervezés

- ❑ Az ablak egy nyomógombjának színét fogjuk véletlenszerűen váltogatni.
- ❑ Szükség van egy időzítőre (**QTimer**), amely olyan eseménykezelőt futtat (**timeout()**), amelyben a „színváltó” nyomógomb stílusát változtatjuk.
- ❑ Külön nyomógomb szolgál az időzítő elindítására/megállítására (**modifyColor()**).

| <i>QWidget</i> ChangingColorWidget | |
|--|---------------------------------------|
| - | _colorButton :QPushButton* |
| - | _startStopButton :QPushButton* |
| - | _timer :QTimer* |
| + | ChangingColorWidget(QWidget*) |
| + | ~ChangingColorWidget() |
| «slots» | |
| - | modifyColor() :void |
| - | timeout() :void |

3.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(modifyColor()));

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

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

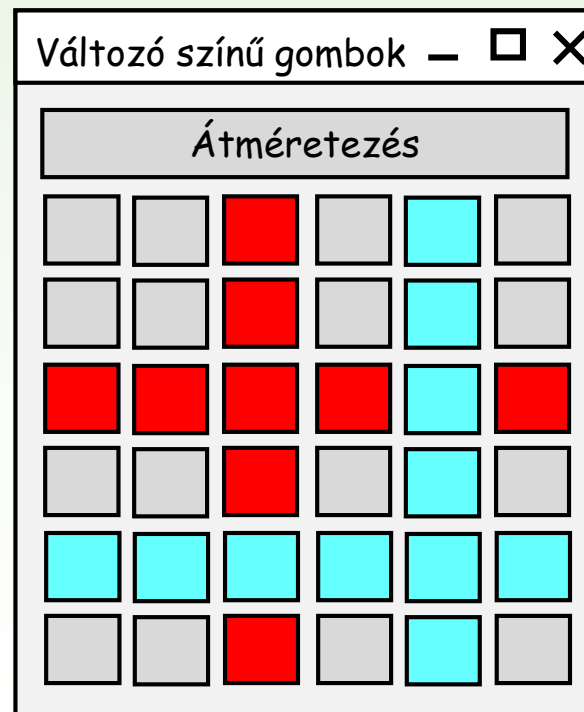
3.Feladat: megvalósítás

```
void ChangingColorWidget::modifyColor() {  
    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  
    }  
}
```

```
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) + ") }");  
}
```

4.Feladat

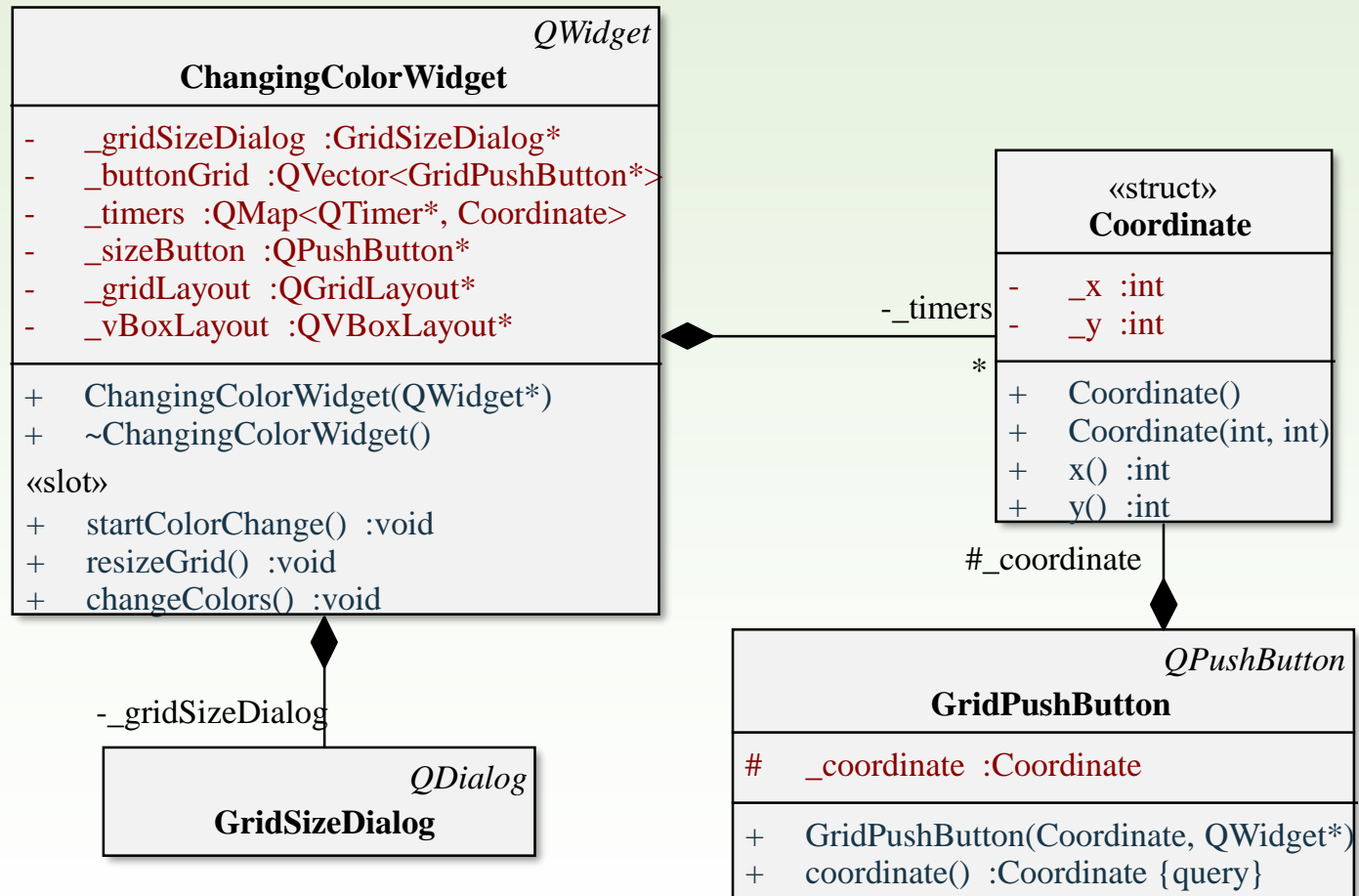
Készítsünk egy nyomógombrácsot, amelyben kattintás hatására indul a színváltó animáció, de nem csak azon a gombon, amelyre kattintottunk, hanem kereszt alakban (a kattintott gomb teljes sorában és oszlopában) minden nyomógomb együttl változtatja a színét.



4.Feladat: tervezés

- ❑ Minden egyes kattintásra egy újabb időzítőt indítunk el.
- ❑ Az időzítőket és a nyomógombokat a sor, oszlop koordinátájuk alapján kapcsoljuk össze, ezért
 - létrehozunk egy koordináta segédtypust (**Coordinate**)
 - az időzítőket a koordinátájukkal egy asszociatív tömbben (**QMap**) tároljuk
 - az időzítőkhöz közös eseménykezelőt kapcsolunk, amely az időzítő koordinátái alapján azonosítja az átszínezendő nyomógombokat
 - a nyomógombok számára létrehozunk egy speciális gombtypust (**GridPushButton**), amely tárolja a koordinátát is
- ❑ Egy külön ablak segítségével végezzük az átméretezést (**GridSizeDialog**). Átméretezéskor ügyelünk arra, hogy minden létező időzítőt leállítsunk, és töröljünk.

4.Feladat: tervezés



4.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);

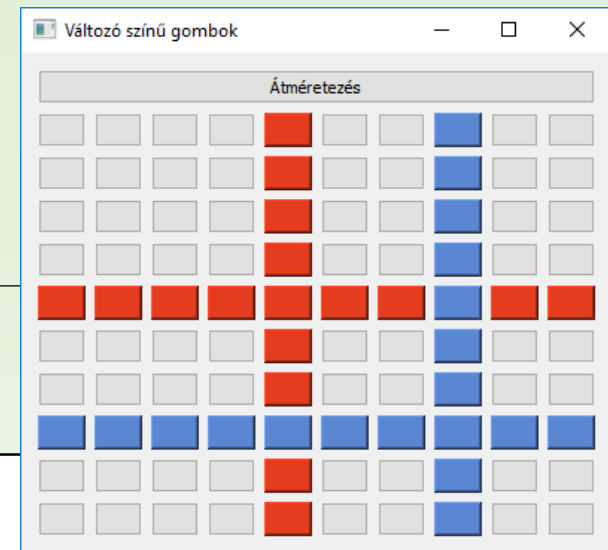
    _gridSizeDialog = new GridSizeDialog();
    connect(_sizeButton,      SIGNAL(clicked()),
            _gridSizeDialog, SLOT(exec()));
    connect(_gridSizeDialog, SIGNAL(accepted()),
            this,             SLOT(resizeGrid()));
    qsrand(QTime::currentTime().msec());
}
```

4.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 koordinátaival
}
```

4.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

4.Feladat: megvalósítás

```
void ChangingColorWidget::resizeGrid()
{
    foreach(QTimer* timer, _timers.keys()) { // időzítők leállítása és törlése
        timer->stop();
        _timers.remove(timer);
        delete timer;
    }
    foreach(GridPushButton* button, _buttonGrid) { // nyomógombok törlése
        _gridLayout->removeWidget(button);
        delete button;
    }
    _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()));
        }
    }
}
```

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 több vezérlő segítségével is felhelyezhetünk, de alapvetően a

- címke (**QLabel**) szolgál képmegjelenítésre a **pixmap** tulajdonságán keresztül, mely egy **QPixmap** objektumot tud fogadni, pl.:

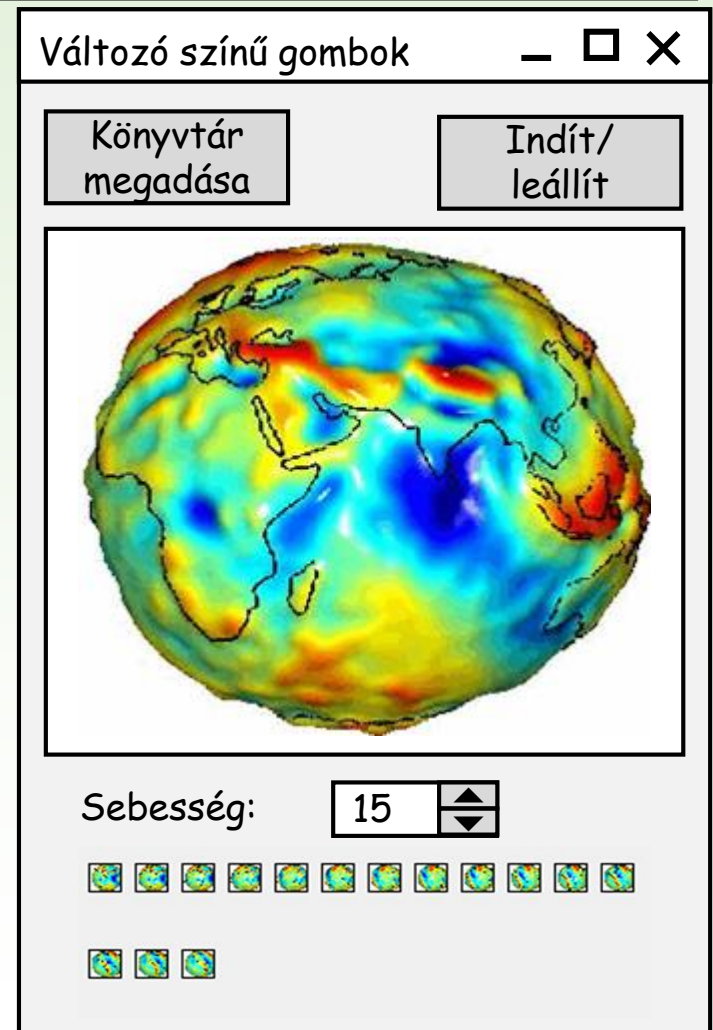
```
QPixmap* pic = new QPixmap("img.bmp");  
label->setPixmap(*pic);
```

- 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));
```

5.Feladat

Készítsünk egy mozgókép megjelenítő alkalmazást, amelyben képek sorozatát tudjuk betölteni (mint filmkockákat), és azokat animációként megjeleníteni. Lehessen szabályozni az animáció sebességét, valamint mutassuk meg, hogy az adott sebesség mellett milyen képkockák fognak majd egymás után megjelenni a következő 1 másodpercben. (Minél lassabb az animáció, annál kevesebb képkocka jelenik majd meg.)



5.Feladat: tervezés

- ❑ A felület statikus részét (képeket betöltő nyomógombot, mozgást indító/leállító nyomógombot, képmegjelenítő 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 periodikusan cserélni őket.

| <i>QWidget</i> MotionPictureWidget | |
|--|--|
| - | _ui :Ui::MotionPictureWidget* |
| - | _smallImageLabels :QVector<QLabel*> |
| - | _images :QVector<QPixmap*> |
| - | _timer :QTimer* |
| - | _currentImage :int |
| + | MotionPictureWidget(QWidget*) |
| + | ~MotionPictureWidget() |
| - | reloadImages() :void |
| - | reloadLabels() :void |
| | «slot» |
| + | loadImages() :void |
| + | startStopMotion() :void |
| + | changeSpeed(int) :void |
| + | changeImages() :void |

5.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()));
}

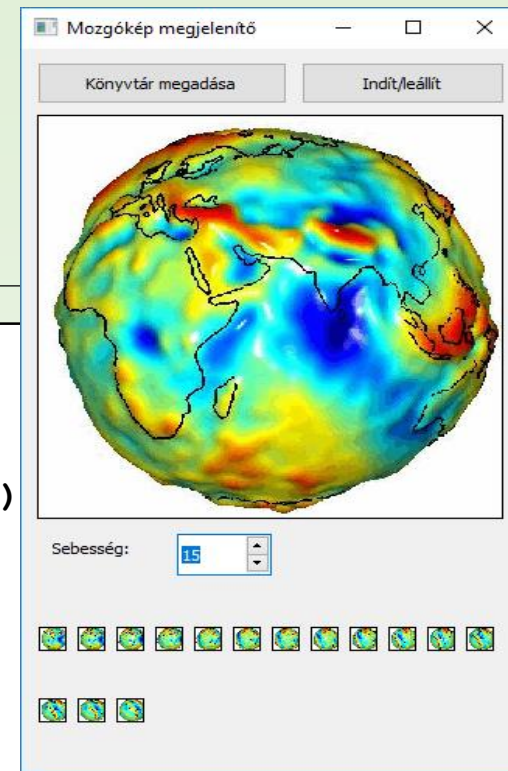
MotionPictureWidget::~MotionPictureWidget()
{
    foreach(QPixmap* image, _images) { delete image; }
    foreach(QLabel* label, _smallImageLabels) { delete label; }
    delete _ui;
}
```

5.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();
}
```



5.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;
        }
        ...
    }
}
```

5.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
    }
}
```

5.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);
    }
}
```

5.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)));
    }
}
```

Elemi grafika

Rajzoló objektum, egér- és billentyű követés

Rajzolósi felület

- ❑ A Qt-ban a grafikus felhasználói felület tartalmát tetszőlegesen „rajzolhatjuk”, ezáltal egyedi megjelenítést adhatunk neki.
 - Mindenre rajzolhatunk, ami a **QPaintDevice** leszármazottja, így tetszőleges grafikus vezérlőre (**QWidget**), képre (**QPixmap**), de akár nyomtató vezérlőre (**QPrinter**) is.
 - A kirajzolást a vezérlőnek a **paintEvent (QPaintEvent*)** metódusa végzi, amely
 - vagy akkor fut le, amikor a rendszer a megjelenítést frissíti,
 - vagy amikor az **update ()** eseménykezelőt közvetlenül hívjuk (pl. időzítővel történő frissítés esetén).
 - és ez a metódus természetesen felüldefiniálható.

Rajzoló eszköz

- ❑ A rajzolás a `QPainter` típusú rajzoló objektummal végezzük.
 - A konstruktorának átadjuk a rajzfelületet (általában az aktuális vezérlő)
pl. `QPainter painter(this);`
 - Beállítjuk a rajzolási tulajdonságokat (szín, háttérszín, vonaltípus, betűtípus, ...) a `set<paraméter>(<érték>)` metódusokkal, amelynek hatása a következő beállításig tart.

```
painter.setBackground(<kitöltés>); // háttérszín  
painter.setFont(<betűtípus>); // szöveg betűtípusa  
painter.setOpacity(<mérték>); // átlátszóság
```

Alakzatok rajzolása

- ❑ A rajzó eszköznek a **draw**-szerű műveleteivel rajzolhatunk, amelyek alakzatoknál keretet és kitöltést is rajzolnak:

draw<alakzat/szöveg/kép>(<elhelyezkedés, ...>)

- a műveletek sorrendben futnak le, és egymásra rajzolnak
- a rajzolás az alakzat bal felső sarkától indul (kivéve szöveg)

- ❑ Alakzat lehet: pont, vonal, törtvonal, téglalap (lekerekített sarokkal), ellipszis, körcikk, körszelet, körcikk, szöveg, kép, stb.

```
painter.drawRect(10, 30, 50, 30);  
    // 50x30-as téglalap kirajzolása a (10,30) koordinátába  
painter.fillRect(20, 40, 50, 30);  
    // keret nélküli téglalap kirajzolása  
painter.drawText(20, 50, "Hello");  
    // szöveg a (20,50) koordinátába
```

- ❑ Kitöltést a **fill**<alakzat>(...) -szerű művelettel rajzolhatunk.

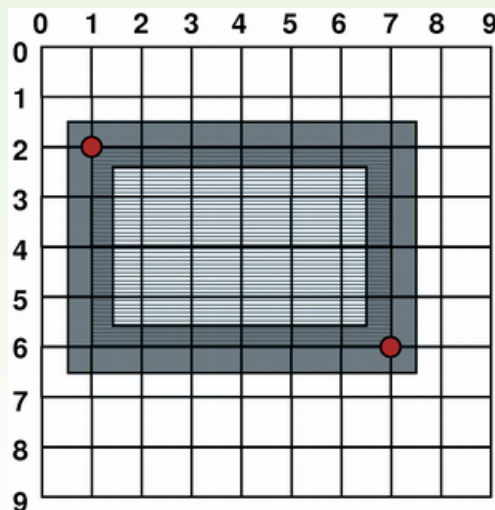
Tollak, ecsetek

- ❑ Külön befolyásolhatjuk az alakzatok kitöltését és keretét.
 - A keretet, szöveget *toll* (**QPen**) segítségével készítjük, amely lehet egyszínű, de tartalmazhat szaggatásokat (dash, dot, dashdot, ...), nyilakat, beállíthatjuk a vonalvégeket (flat, square, round), a vonal találkozásokat (miter, bevel, round)
 - A kitöltést *ecset* (**QBrush**) segítségével készítjük, amely lehet egyszínű, adott mintájú, textúrájú, ...

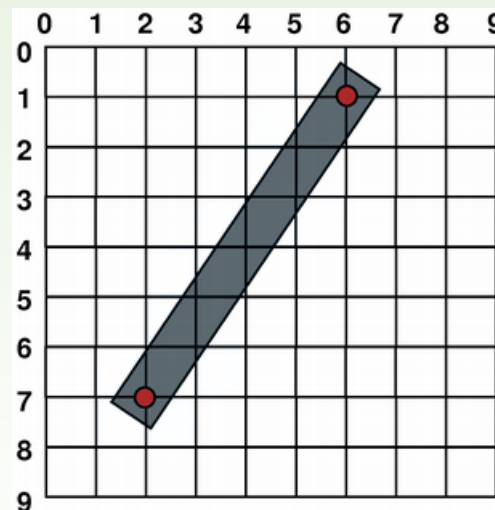
```
painter.setPen(Qt::darkGreen) ;  
    // 1 vastag sötétzöld toll  
painter.setPen(QPen(QColor(Qt::blue), 4, Qt::DotLine)) ;  
    // 4 vastag pöttyös kék toll  
painter.setBrush(QBrush(QColor(250, 53, 38), Qt::CrossPattern)) ;  
    // rácsos vöröses ecset
```

Rajzolás logikai koordinátái

- A rajzolást úgynevezett „logikai” koordináták segítségével végezzük, ezek határozzák meg az alakzat sarokpontjait.



QRect (1 , 2 , 6 , 4)

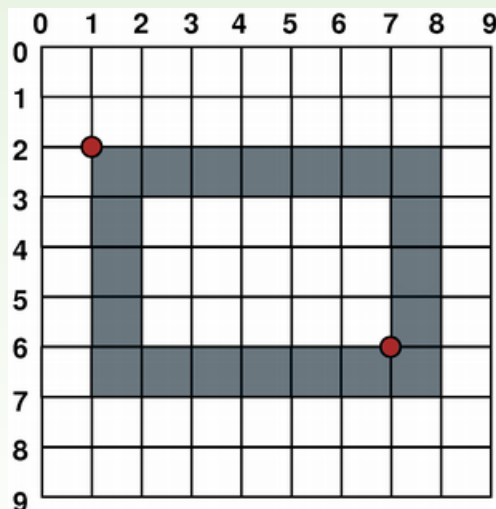


QLine (2 , 7 , 6 , 1)

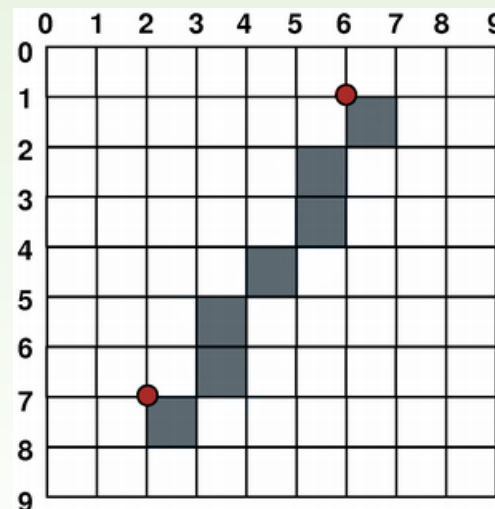
- A rendszer átranzformálja az adatokat „fizikai” koordinátákká (*viewport*)

Rajzolás fizikai koordinátái

- A rajzolási műveletek az alakzatot a megfelelő képpontok koordinátáira (az ablak koordinátáira) igazítják.



`drawRect (1,2,6,4) ;`

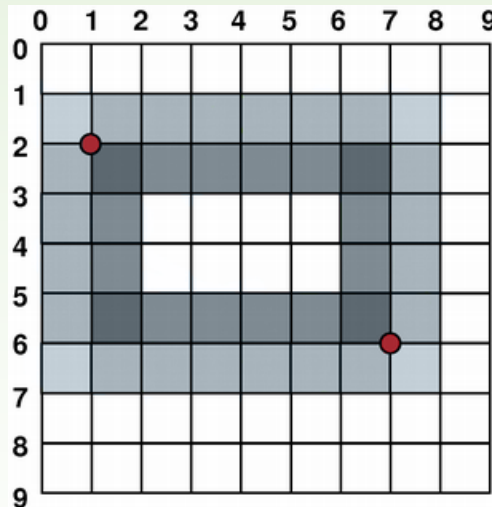


`drawLine (2,7,6,1) ;`

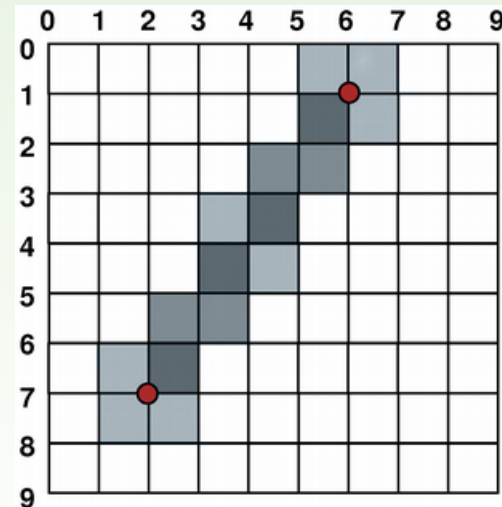
- Amennyiben a toll vastagsága páratlan, jobbra és lefelé tolódik az elhelyezés.

Simítás

- Lehet simítást alkalmazni a rajzoláskor, hogy a logikai koordinátán helyezkedjen el a rajz.



`drawRect(1, 2, 6, 4) ;`

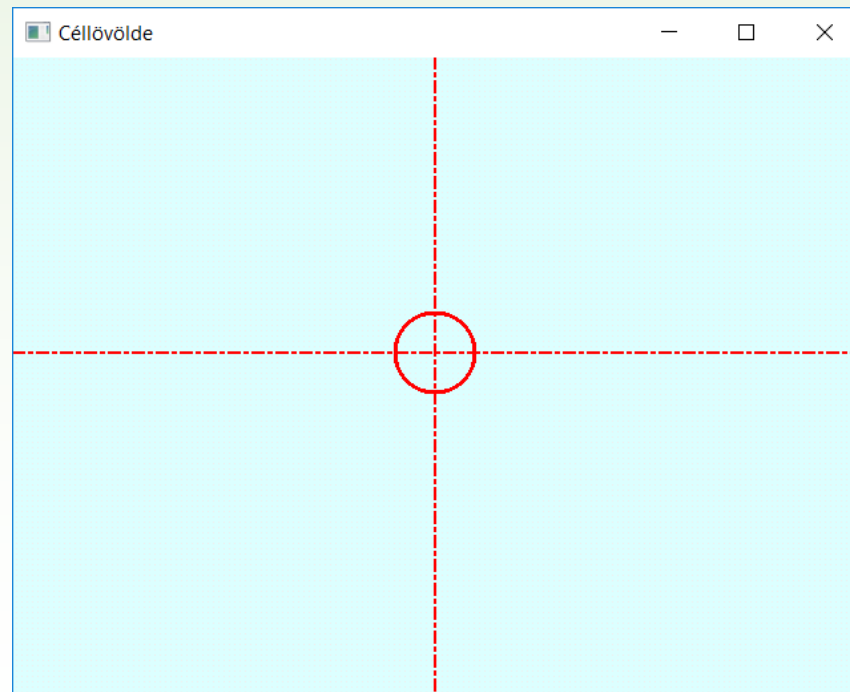


`drawLine(2, 7, 6, 1) ;`

- Ehhez a `setRenderHint(QPainter::Antialiasing)` üzemmódot kell a rajzolóra beállítanunk.

1.Feladat

Készítsünk egy alkalmazást, amelyben egy célkeresztet helyezünk az ablak közepére. A célkeresztet két szaggatott-pöttyözött piros vonallal és egy piros körrel jelenítjük meg, a háttérrel pöttyös világoskék ecsettel festjük ki.



1.Feladat: megoldás

- ❑ Felüldefiniáljuk az ablak **paintEvent** metódusát, létrehozunk benne egy rajzoló objektumot (**painter**).
- ❑ Először kitöltjük a háttérét a **fillRect** utasítással, majd meghúzzuk a függőleges és vízszintes vonalakat (**drawLine**), végül a közepére állítunk egy ellipszist (**drawEllipse**).
- ❑ A rajzolások közben megfelelően állítjuk a tollat és az ecsetet (az ecsetet kikapcsoljuk az ellipszis rajzolása előtt).

1.Feladat: megvalósítás

```
void CrosshairWidget::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);
    QPen dashDotRedPen(QBrush(QColor(255,0,0)),2, Qt::DashDotLine);
    QPen solidRedPen(QBrush(QColor(255, 0, 0)), 3);
    QBrush blueBrush(QColor(230, 255, 255), Qt::Dense1Pattern);

    painter.setBrush(blueBrush);
    painter.fillRect(0, 0, width(), height());
    painter.setPen(dashDotRedPen);
    painter.drawLine(0, height() / 2, width(), height() / 2);
    painter.drawLine(width() / 2, 0, width() / 2, height());
    painter.setPen(solidRedPen);
    painter.drawEllipse(width()/2 - 30, height()/2 - 30, 60, 60);
}
```

pontozott-szaggatott vonalú piros toll

piros toll

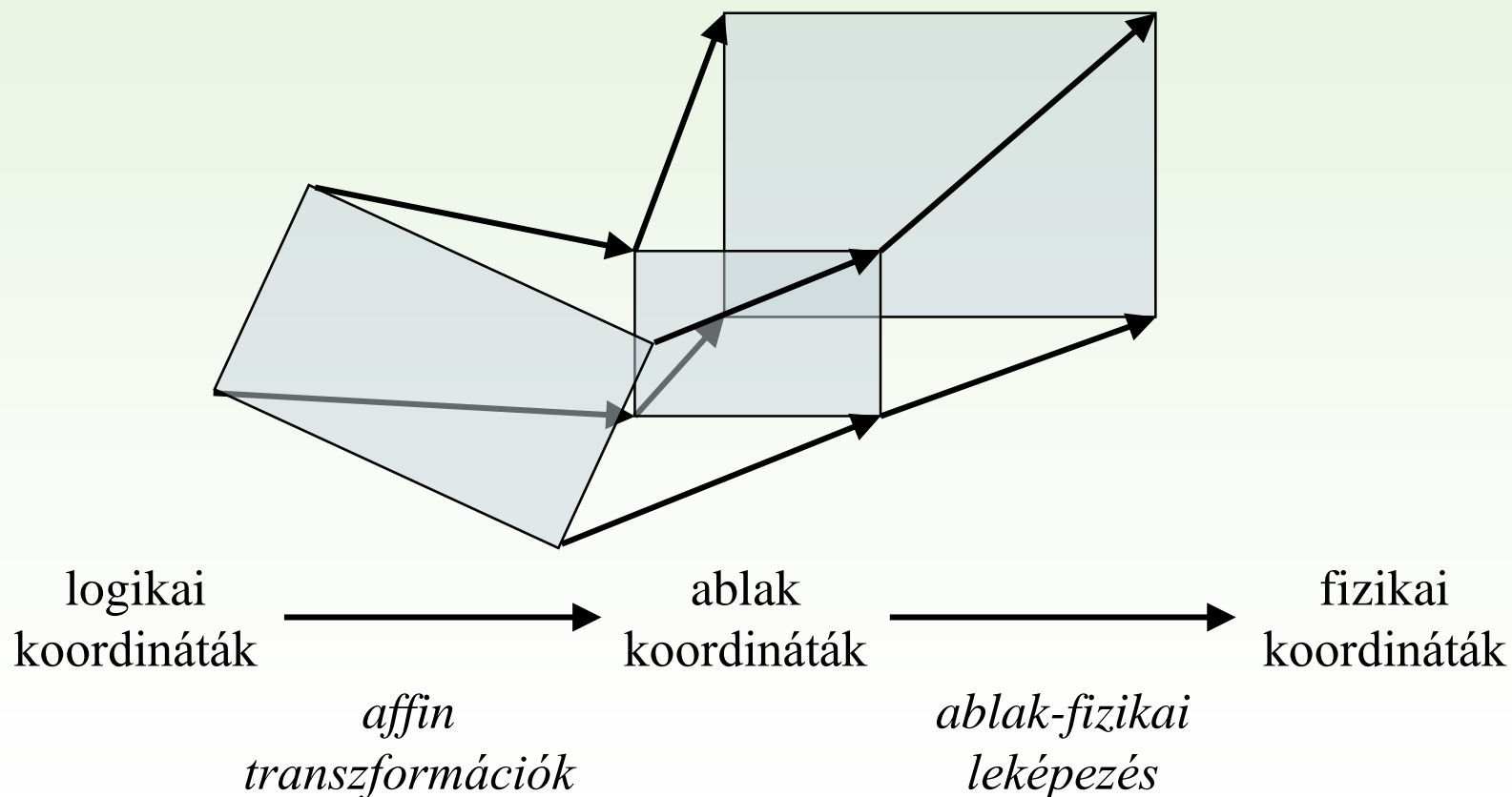
pöttyös világoskék ecset

Transzformációk

- ❑ Alapból a rajzoló objektum a megadott vezérlő koordinátarendszerében dolgozik, de lehetőségünk van ennek affin transzformálására (**worldTransform**).
 - forgatás (**rotate**(*<szög>*))
 - méretezés (**scale**(*<vízszintes>*, *<függőleges>*))
 - áthelyezés (**translate**(*<vízszintes>*, *<függőleges>*))
 - ferdítés (**shear**(*<vízszintes>*, *<függőleges>*))
- ❑ Így egy újabb szint (transzformáció) épülhet be a fizikai (**viewport**) koordináták és a logikai koordináták közé (a logikai koordináták először az affin transzformációkkal ablak koordinátákká, majd fizikai koordinátákká változnak.)

Logikai – ablak – fizikai koordináták

- Minden leképezés transzformációs mátrixok alkalmazásával történik.

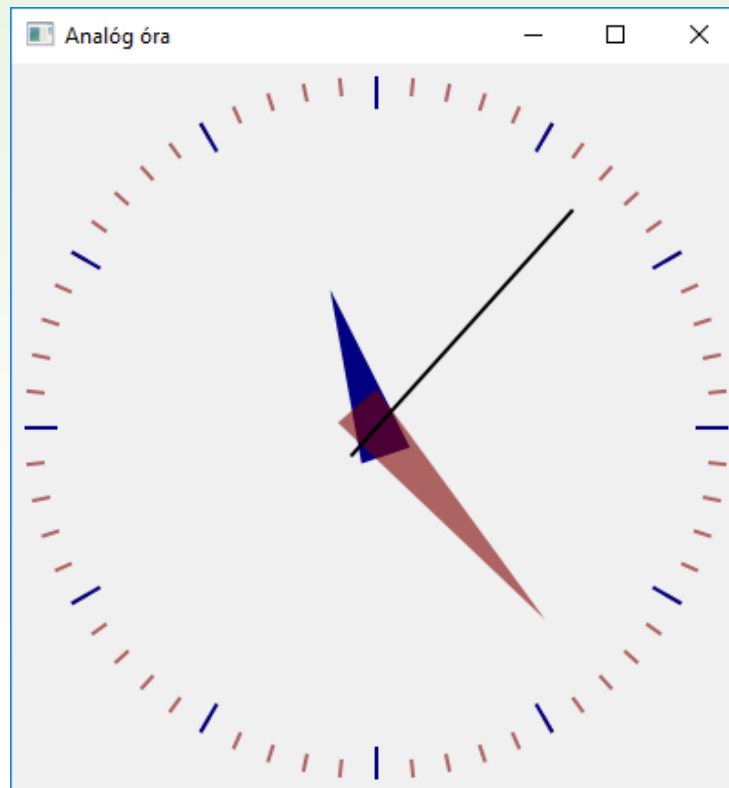


További rajzolási lehetőségek

- ❑ A háttérét külön állíthatjuk (**background**), ekkor a teljes rajzfelület változik, a rárajzolt tartalom külön törölhető (**erase()**).
- ❑ Amennyiben több tulajdonság beállítását is elvégezzük a rajzolás során, lehetőségünk van korábbi beállítások visszatöltésére.
 - A **save()** művelettel elmenthetjük az aktuális állapotot, a **restore()** művelettel betölthetjük az utoljára mentettet.
- A rajzolás tartalmát megvághatjuk téglalap (**clipRegion**), vagy egyéni alakzat (**clipPath**) alapján.
- Több rajzot is összeilleszthetünk különböző műveleti sémák szerint (**compositionMode**).

2.Feladat

Készítsünk egy analóg órát, amely mutatja az aktuális időt.




2.Feladat: megoldás

- ❑ Az aktuális idő mutatósához időzítőt használunk és mindig lekérdezzük az aktuális időt (`QTime::currentTime()`).
- ❑ Az óra és perc mutatókat háromszöggént rajzoljuk ki (`drawConvexPolygon`, némi áttetszéssel), és a megfelelőhelyre forgatjuk (`rotate`), hasonlóan forgatjuk a többi jelölőt és mutatót, de azok már vonalak lesznek.
- ❑ Az egyszerűbb forgatás és helyezés érdekében eltoljuk (`translate`) és méretezzük (`scale`) a koordináta-rendszert, hogy az ablak közepén legyen az origó.

2.Feladat: megvalósítás

```
AnalogClockWidget::AnalogClockWidget(QWidget *parent)
    : QWidget(parent) {
    setWindowTitle(tr("Analóg óra"));
    resize(400, 400);
    QTimer *timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(update()));
    timer->start(1000);
}
```

az update hívja a paintEvent-t



```
void AnalogClockWidget::paintEvent(QPaintEvent *){
    // mutatók geometriája:
    QPoint hourTriangle[3] =
        { QPoint(7, 8), QPoint(-7, 8), QPoint(0, -40) };
    QPoint minute Triangle[3] =
        { QPoint(7, 8), QPoint(-7, 8), QPoint(0, -70) };
    // mutatók és vonások színe:
    QColor hourColor(0, 0, 128);           // RGB
    QColor minuteColor(128, 0, 0, 150);    // RGB és áttetszőség
    ...
}
```

2.Feladat: megvalósítás

```
...
QPainter painter(this);
painter.setRenderHint(QPainter::Antialiasing);
painter.translate(width() / 2, height() / 2);
painter.scale(height() / 200.0, height() / 200.0);
// percjelek rajzolása:
painter.setPen(minuteColor);
for (int j = 0; j < 60; ++j){
    painter.drawLine(92, 0, 96, 0);
    painter.rotate(6.0); // forgatás 6 fokkal
}
// órajelek rajzolása:
painter.setPen(hourColor);
for (int i = 0; i < 12; ++i){
    painter.drawLine(88, 0, 96, 0);
    painter.rotate(30.0); // forgatás 30 fokkal
}
...
```

origó az ablak közepére

A vásznat forgatjuk

2.Feladat: megvalósítás

```
...  
    // nagymutató:  
    painter.save(); // rajzolási tulajdonságok elmentése  
    painter.setPen(Qt::NoPen); // nincs toll  
    painter.setBrush(hourColor); // ecset színe  
    painter.rotate(30.0 * ((time.hour() + time.minute() / 60.0)));  
    painter.drawConvexPolygon(hourTriangle, 3); // poligon  
    painter.restore(); // rajzolás tulajdonságok visszaállítása  
  
    // percmutató:  
    ...  
    // másodpercmutató:  
    ...  
}
```

Egérkezelő műveletek

- ❑ Az egérkezelés (követés, kattintás lekérdezése) bármely vezérlő területén elvégezhető, műveletek felüldefiniálásával
- ❑ Egy saját widget négy öröklött eseménykezelőt írhat felül:
 - **mousePressEvent** : egér lenyomása
 - **mouseReleaseEvent** : egér felengedése
 - **mouseMoveEvent**: egér mozgatása
 - **mouseDoubleClickEvent** : dupla kattintás
- ❑ Minden eseménykezelő **MouseEvent** paramétert kap, amely tartalmazza az egér pozícióját lokálisan (**pos()**) és globálisan (**globalX()**, **globalY()**), illetve a használt gombot (**button()**).
- ❑ Az egérkövetés alapértelmezetten csak lenyomott gomb mellett működik, de ez átállítható állandóra a **mouseTracking** tulajdonság állításával.

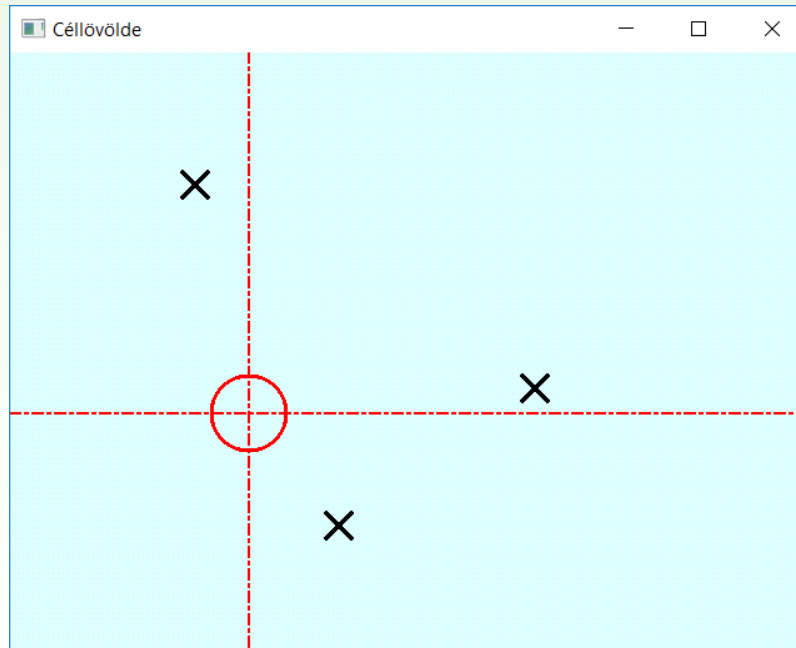
Billentyűzetkezelő műveletek

- ❑ Egy saját widget négy öröklött eseménykezelőt írhat felül:
 - **keyPressEvent** : billentyű lenyomása
 - **keyReleaseEvent** : billentyű felengedése
- ❑ Minden eseménykezelő **QKeyEvent** paramétert kap, amely tartalmazza az érintett billentyűt (**key**).

3.Feladat

A célkereszt megjelenítő programunkban kövesse az egérkurzort a célkereszt mindaddig, amíg az egérkurzor az ablak felett van.

Az egérfül vagy a szóköz billentyű lenyomásával lehessen lőni úgy, hogy a lövés helyén egy fekete X jelenik meg.



3.Feladat: megoldás

- ❑ Beállítjuk állandóra (`setMouseTracking(true)`) az egérkövetést
- ❑ Felüldefiniáljuk az egérkövetés eseményt úgy, hogy az elmentse az aktuális kurzor-pozíciót (a `mouseLocation`-t).
- ❑ Felüldefiniáljuk az egér/billentyű lenyomás eseményeket úgy, hogy minden kattintásnál elmentse az újabb lövések célkereszt-pozíciót a korábbiak mellé egy vektorba (`hitPoints`).
- ❑ Mindhárom eseménykezelő frissítse a kijelzőt (`update()`).
- ❑ Kirajzoláskor (`paintEvent`) az elmentett pontokat is kirajzoljuk.

3.Feladat: tervezés

| <i>QWidget</i> CrosshairWidget | |
|--|---|
| - | hitPoints :QVector<QPoint> |
| - | mouseLocation :QPoint |
| + | CrosshairWidget(QWidget*) |
| + | ~CrosshairWidget() |
| # | keyPressEvent(QKeyEvent*) :void |
| # | mouseMoveEvent(QMouseEvent*) :void |
| # | mousePressEvent(QMouseEvent*) :void |
| # | paintEvent(QPaintEvent*) :void |

3.Feladat: megvalósítás

```
CrosshairWidget::CrosshairWidget(QWidget *parent): QWidget(parent)
{
    setWindowTitle(tr("Céllövölde"));
    setMouseTracking(true); // állandóan követjük az egeret
}

void CrosshairWidget::paintEvent(QPaintEvent *)
{
    QPainter painter(this);

    QPen dashDotRedPen(QBrush(QColor(255,0,0)), 2, Qt::DashDotLine);
    QPen solidRedPen(QBrush(QColor(255, 0, 0)), 3);
    QBrush blueBrush(QColor(230, 255, 255), Qt::Dense1Pattern);

    painter.fillRect(0, 0, width(), height(), blueBrush);
    painter.setPen(QPen(Qt::black, 4)); // fekete, 4 vastag toll
    ...
}
```

3.Feladat: megvalósítás

```
void CrosshairWidget::paintEvent(QPaintEvent *)
{
    ...
    foreach(QPoint point, hitPoints) {
        painter.drawLine(point.x() - 10, point.y() - 10,
                          point.x() + 10, point.y() + 10);
    }
    painter.setPen(dashDotRedPen);
    painter.setBrush(Qt::NoBrush); // ecset eltávolítása
    painter.drawLine(0, mouseLocation.y(), width(), mouseLocation.y());
    painter.drawLine(mouseLocation.x(), 0,
                      mouseLocation.x(), height());
    painter.setPen(solidRedPen);
    painter.drawEllipse(mouseLocation.x()-30, mouseLocation.y()-30,
                        60, 60);
}
```


3.Feladat: megvalósítás

```
void CrosshairWidget::mouseMoveEvent(QMouseEvent*event)
{
    mouseLocation = event->pos();
    update(); // képernyő frissítése
}
void CrosshairWidget::mousePressEvent(QMouseEvent *event)
{
    hitPoints.append(event->pos()); // új pont felvétele
    update(); // képernyő frissítése
}
void CrosshairWidget::keyPressEvent(QKeyEvent *event)
{
    if (event->key() == Qt::Key_Space) { // ha szóközt ütöttünk
        hitPoints.append(mouseLocation); // új pont felvétele
        update(); // képernyő frissítése
    }
}
```

Kurzorkezelés

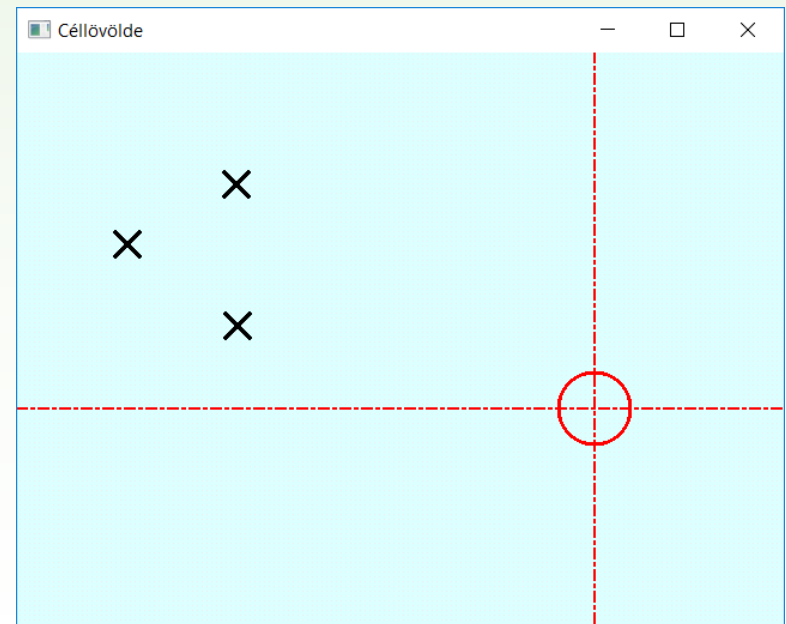
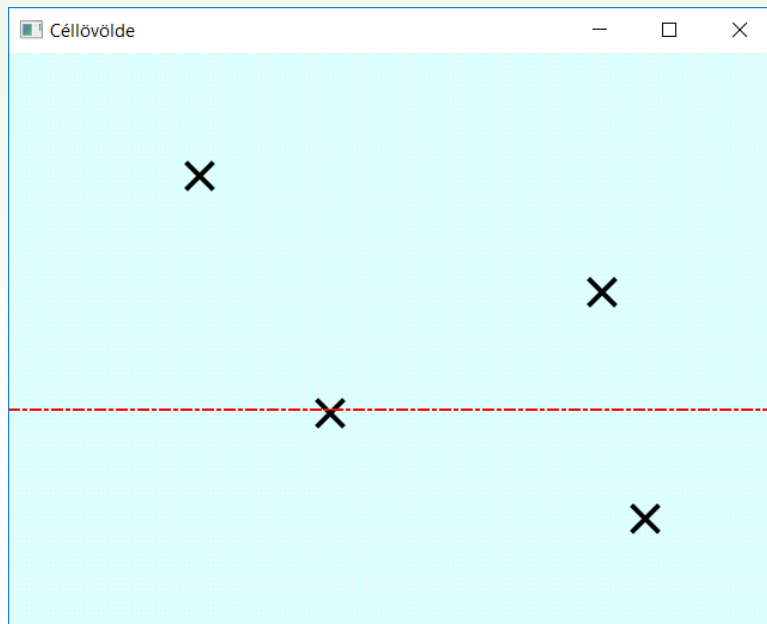
- ❑ Az egérkezelő műveletektől függetlenül is bármikor használhatjuk az egérpozíciót, kurzorkezelés (`QCursor`) segítségével.
 - A kurzor mindig az egérpozícióval egybeeső helyen van, amely lekérdezhető, és beállítható (`QCursor::pos()`).
 - A kurzornak módosítható a kinézete (pl. nyíl, kéz, homokóra, ...), vagy beállítható tetszőleges kép.

```
widget.setCursor(QCursor(Qt::CrossCursor)); // kereszt  
widget.setCursor(QCursor(Qt::BusyCursor)); // homokóra
```

- ❑ A kurzortól lekért pozíció globális, de minden vezérlőnél van lehetőségünk leképezni a lokális koordinátarendszerbe a `QWidget::mapFromGlobal(<pozíció>)` művelettel.

4.Feladat

Módosítsuk a célkereszt megjelenítő programunkat úgy, hogy az egérkurzor maga is egy kereszt legyen, amely takarásban ugyan, de akkor is kövesse az egérkurzor mozgását, ha az az ablakon kívül van.



4.Feladat: megoldás

- ❑ A konstruktorban módosítjuk a kurzormegjelenést.
(`setCursor(Qt::CrossCursor)`)
- ❑ Mivel nincs egérkövetés, nem tudunk egéreseeményre reagálva rajzolni, ezért időzítő segítségével meghatározott időközönként (0.01 másodperc) frissítjük a képernyőt, ahol mindig lekérjük az aktuális kurzorpozíciót a rajzolásnál.
- ❑ Az egér/billentyű lenyomás eseményét megtartjuk, ebben továbbra is felvesszük az újabb lövések célkereszt-pozíciót a korábbiak mellé a `hitPoints` vektorba.

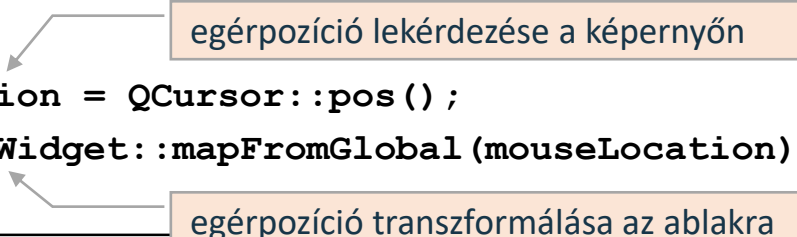
4.Feladat: tervezés

| CrosshairWidget <i>QWidget</i> | |
|---------------------------------------|-------------------------------------|
| - | hitPoints :QVector<QPoint> |
| - | timer :QTimer* |
| + | CrosshairWidget(QWidget*) |
| + | ~CrosshairWidget() |
| # | keyPressEvent(QKeyEvent*) :void |
| # | mousePressEvent(QMouseEvent*) :void |
| # | paintEvent(QPaintEvent*) :void |

4.Feladat: megvalósítás

```
CrosshairWidget::CrosshairWidget(QWidget *parent) : QWidget(parent) {  
    setWindowTitle(tr("Céllövölde"));  
  
    timer = new QTimer(this);  
    connect(timer, SIGNAL(timeout()), this, SLOT(update()));  
    timer->start(10);  
  
    setCursor(Qt::CrossCursor); // kurzor beállítása célkeresztre  
}
```

```
void CrosshairWidget::paintEvent(QPaintEvent *)  
{  
    ...  
    QPoint mouseLocation = QCursor::pos();  
    mouseLocation = QWidget::mapFromGlobal(mouseLocation);  
    ...
```



The diagram illustrates the process of mapping mouse coordinates. An arrow points from the text box 'egérpozíció lekérdezése a képernyőn' to the `QCursor::pos()` call in the code. Another arrow points from the text box 'egérpozíció transzformálása az ablakra' to the `QWidget::mapFromGlobal(mouseLocation)` call.

egérpozíció lekérdezése a képernyőn

egérpozíció transzformálása az ablakra