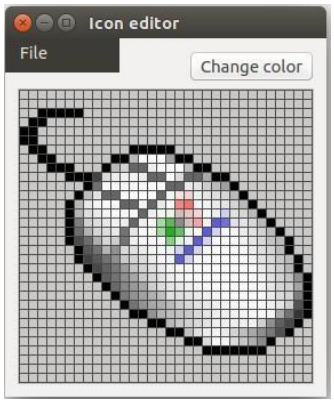
A 4. gyakorlaton egy ikonszerkesztő alkalmazást készítünk el egyrétegű architektúrában.

Funkcionalitások

- Az alkalmazás jelenítsen meg egy PNG formátumú képet „pixelezve”. Jobbegérgombbal való kattintásra egy adott pixelt színezzünk feketére, balegérgomb-kattintásra tegyük színtelenné a pixelt. Kattintás után a kurzor mozgatására az érintett pixelekre is végezzük el a megfelelő műveletet.

- Legyen lehetőség az ecset színének megváltoztatására egy színpalettáról kiválasztva az új színt.

- A felhasználónak legyen lehetősége új képet betölteni és a betöltött képet elmenteni.



Képmegjelenítő

Egy görgethető képmegjelenítőt szeretnénk készíteni. Ehhez hozzuk létre az `IconEditor` osztályt, ez származzon a `QWidget` osztályból. Az osztály egy példánya tárolja a megjelenítendő képet (`QImage`), a kép útvonalát (`QString`), az ecset színét (`QColor`), valamint a kép nagyításának mértékét egész szám formájában. Legyen lehetőség a kép, az útvonal, a nagyítás és a szín lekérésére és beállítására.

Az ecset színét a `setPenColor(const Qcolor& newColor)` metódus végezze. A nagyítást a `setZoomFactor(int newZoom)` metódussal végeztessük el, amely a nagyítás beállítása után meghívja a QWidget beépített `update()` és `updateGeometry()` metódusait, hogy frissítse a képet. A nagyítás ne lehessen kisebb, mint 1. Kezdetben az ecset színe legyen fekete, a nagyítás értéke pedig 8. A képet állítsuk 32 x 32-es méretűre, a formátuma pedig legyen `QImage::Format_ARGB32`. Az üres képet töltsük ki szürke színnel:

```
image = QImage(32, 32, QImage::Format_ARGB32);
image.fill(qRgba(0, 0, 0, 0));
```

A pixelek színezését a `mousePressEvent(QMouseEvent* event)` és a `mouseMoveEvent(QMouseEvent* event)` eseménykezelők hívása váltja ki. Előbbi a gombnyomást, utóbbi az gombnyomás utáni mozgatást kezeli. A bal egérgomb az ecset színének megfelelően kiszínezi az aktuális pixel(eke)t, a jobb egérgomb eltávolítja a színezést. Mindezt a következőben bemutatott `setImagePixel(const QPoint &pos, bool opaque)` eljárás segítségével.

```
void IconEditor::mousePressEvent(QMouseEvent* event)
{
    if (event->button() == Qt::LeftButton)
    {
        setImagePixel(event->pos(), true);
    }
    else if (event->button() == Qt::RightButton)
    {
        setImagePixel(event->pos(), false);
    }
}
void IconEditor::mouseMoveEvent(QMouseEvent* event)
{
    if (event->buttons() & Qt::LeftButton)
    {
        setImagePixel(event->pos(), true);
    }
    else if (event->buttons() & Qt::RightButton)
    {
        setImagePixel(event->pos(), false);
    }
}
```

A pixelek színének beállítását a képmegjelenítőben definiált `setImagePixel(const QPoint& pos, bool opaque)` metódussal végezhetjük el, ami a zoom értéke alapján kiszámolja a képkoordinátákat, ahova mutat a pos változó, majd amennyiben a képen található ilyen koordinátájú pixel (`if (image.rect().contains(i, j)) { … }`), a képben felülírja azt a pixelt vagy a toll színével, vagy a háttérszínnel (`image.setPixel(i, j, penColor().rgba())`), az opaque változótól függően. Végül frissíti a megjelenítőnek azt a részét, ami módosult: `update(pixelRect(i, j))`.

A kép frissítését a `paintEvent(QPaintEvent* event)` eseménykezelő végzi. Ha a nagyítás értéke 3 vagy annál nagyobb, egy `QPainter` típusú és `painter` nevű objektum segítségével rajzoljunk vízszintes és függőleges vonalakat a képre, hogy kirajzoljuk a pixeleket. Ennél kisebb nagyítás esetén a rácsot nincs értelme kirajzolni.

```
if (zoom >= 3)
{
    painter.setPen(palette().windowText().color());

    for (int i = 0; i <= image.width(); ++i)
    {
        painter.drawLine(zoom * i, 0,
                         zoom * i, zoom * image.height());
    }

    for (int j = 0; j <= image.height(); ++j)
    {
        painter.drawLine(0, zoom * j,
                         zoom * image.width(), zoom * j);
    }
}
```

A rács kirajzolása után színezzük újra az egér által érintett pixeleket. Az átvett eseményből kinyerhetjük, hogy mely pixeleket kell kiszínezni. A pixeleket a képmegjelenítő `pixelRect(int i, int j)` metódusával (következő metódus) kérhetjük le egy-egy `QRect` objektum formájában.

```
for (int i = 0; i < image.width(); ++i)
{
    for (int j = 0; j < image.height(); ++j)
    {
        QRect rect = pixelRect(i, j);
        if (event->region().intersects(rect))
        {
            QColor color = QColor::fromRgba(image.pixel(i, j));
            painter.fillRect(rect, color);
        }
    }
}
```

A képmegjelenítő `pixelRect(int i, int j)` metódusa `QRect` objektumként visszaadja azt a téglalapot, ami tartalmazza az (i,j) koordinátájú képpixelt. A QRect konstruktora a bal felső sarok koordinátáit kéri el, illetve a szélességét és magasságát a téglalapnak. Itt figyeljünk arra, hogy amennyiben van rács rajzolva a képre, azok a vonalak maradjanak ki, tehát egy pixellel lejjebb legyen a sarok és ne legyen olyan széles és magas!

A `saveImage()` metódus létrehoz egy `QImageWriter` példányt, amellyel ezután megpróbáljuk kiírni a képet a megadott útvonalra. Ha nem sikerül a mentés, a hibaüzenetet felugró ablakban jelenítjük meg:

```
bool IconEditor::saveImage()
{
   QImageWriter writer(fileName);

   if (!writer.write(image))
   {
       QMessageBox::information(this, tr("Cannot write
```

```
            %1: %2").arg(QDir::toNativeSeparators(fileName)),
            writer.errorString());
        return false;
    }

    const QString message = tr("Wrote
\"%1\"").arg(QDir::toNativeSeparators(fileName));
    return true;
}
```

A `loadImage(const QString& file)` egy `QImageReader` objektumot hoz létre, amely a paraméterként megadott útvonalról megpróbálja betölteni a képet. Amennyiben ez sikeres, meghívjuk a képet és az útvonalat beállító metódusokat, egyébként egy felugró ablakban jelezzük a hibát:

```
bool IconEditor::loadImage(const QString& file)
{
    QImageReader reader(file);
    const QImage newImage = reader.read();

    if (newImage.isNull())
    {
        QMessageBox::information(this,
            tr("Cannot load %1: %2").arg(QDir::toNativeSeparators(file)),
                "Cannot load image!");
        return false;
    }
setIconImage(newImage);
setFileName(file);
return true;
}
```

Színváltoztatás

A színváltoztató paletta számára hozzuk létre a `SelectColorButton` osztályt. Ez az osztály származzon a `QPushButton` osztályból. Az osztály egy példánya tároljon egy mutatót a képmegjelenítőre, és az ecset színét. Legyen lehetőség a szín lekérésére. Az osztály definiálja a `changeColor()` eseménykezelőt, ami a gomb kattintására váltódik ki. A metódus nyisson meg egy `QColorDialog` példányt, amelyből kiválasztva egy színt megváltoztathatjuk az ecset színt:

```
QColor newColor = QColorDialog::getColor(color, parentWidget());
```

A changeColor() hívja meg a színt beállító `setColor()` metódust, ami a képmegjelenítőben is megváltoztatja az ecset színét.

Az alkalmazás főablaka

Hozzunk létre egy `MainWindow` nevű osztályt, a `QWidget`ből származik. Az osztály egy példánya tárol egy példányt a képmegjelenítőből, egy színváltoztató gombot, továbbá egy függőleges elrendezőt (`QVBoxLayout`), egy görgethető felületet (`QScrollArea`), egy menüsort (`QMenuBar`), egy ehhez tartozó menüpontot (`QMenu`) és két `QAction` példányt, amelyek a mentés és a betöltés menüpontjai lesznek a főmenü alatt. A MainWindow két eseménykezelőt definiál, amelyek a mentésért és a betöltésért felelősek.

A nézet létrehozásához a képmegjelenítőt adjuk át a QScrollAreának, így görgethetővé téve a képet. A QScrollArea `setWidgetResizable`() metódusával beállíthatjuk, hogy a függőleges és vízszintes görgők a betöltött kép méretével együtt változzanak.

```
    scrollArea = new QScrollArea(this);
    scrollArea->setWidget(&iconEditor);
    scrollArea->viewport()->setBackgroundRole(QPalette::Dark);
    scrollArea->viewport()->setAutoFillBackground(true);
    scrollArea->setWidgetResizable(true);
```

Az elrendezés első sorának jobb szélén a színváltoztató gomb jelenjen meg, a második sorba pedig a QScrollArea kerüljön.

```
layout = new QVBoxLayout(this);
layout->addWidget(changeColor, 1, Qt::AlignRight);
layout->addWidget(scrollArea);
```

A `save` és `load` akciók `triggered()` eseményét kössük össze a megfelelő eseménykezelőkkel (`saveImage()` és `loadImage()`), és adjuk az akciókat a `file` menüponthoz (addAction metódus), utóbbit pedig adjuk a menüsorhoz (addMenu metódus).

A `saveImage()` eseménykezelő meghívja a képmegjelenítő mentésért felelős metódusát, ha van betöltve kép, egyébként egy felugró ablakban jelzi, hogy nincs kép betöltve.

```
void MainWindow::saveImage()
{
    if (!iconEditor.getFileName().isEmpty())
    {
        iconEditor.saveImage();
    }
    else
```

```
    {
        QMessageBox::information(this, "No image",
                                 "No image is loaded!");
    }
}
```

A `loadImage()` megnyit egy fájldialógust (`QFileDialog`), amiből kiválaszthatunk egy PNG formátumú fájlt, majd meghívja a képmegjelenítő betöltésért felelős metódusát:

```
void MainWindow::loadImage()
{
    QString filePath = QFileDialog::getOpenFileName(this,
                       tr("Open image"),"",tr("PNG (*.png)"));

    if (!filePath.isNull() && iconEditor.loadImage(filePath))
    {
        save->setEnabled(true);
    }
}
```

A teszteléshez az *images* mappában található néhány PNG kép.