Tallinna Ülikool

Informaatika Instituut

# Qt raamistik

Seminaritöö

Autor: Karmo Rosental

Juhendaja: Jaagup Kippar

Tallinn 2012

# Sisukord

**Sissejuhatus** **3**

**Olemasolev materjal** **4**

**Qt raamistik** **5**

1. Tutvustus 5
2. Arendustööriistad 6
3. Arenduskeskonna seadistamine 7
4. Esimene rakendus 9
5. Paigutushaldurid 13
6. Vidinad 16
7. Signaalid ja pesad 21
8. Tõlkimine 24
9. Laadid 27
10. Graafika 30
11. Ülesanded 35
12. Edasiõppimiseks 36

**Õppematerjali testimine** **38**

**Kokkuvõte** **39**

**Kasutatud kirjandus** **40**

**Lisad** **41**

1. Ülesannete võimalikud lahendused 41

# Sissejuhatus

Mida aeg edasi, seda keerulisemaks ja mitmekülgsemaks kipuvad töölaua- ja mobiilsed rakendused minema. Lisandub järjest uusi seadmeid, mis toovad esile ühilduvusprobleemid ning järjest kiireneva elutempo juures oleks vaja luua tarkvara võimalikult väikese ajakuluga. Tekib küsimus, et kuidas seda kõike saavutada ning millise hinnaga? Siin tuleb appi Qt raamistik. See võimaldab kiiresti ja mugavalt arendada mitmekeelseid ja erinevatele platvormile mõeldud rakendusi, olles seejuures täiesti tasuta kasutatav koos kõigi oma lisadega.

Käesoleva töö on õppematerjal, mille eesmärk on tutvustada eesti keelsele lugejaskonnale rakenduste loomist Qt raamistiku abil. Eesmärgi saavutamiseks annab autor ülevaate raamistiku ajaloost, komponentidest ja võimalustest. Pärast ülevaadet näidatakse praktilistes peatükkides, kuidas luua erinevaid lihtsaid graafilisi rakendusi, kasutades Qt raamistiku võimalusi. Iga praktiline peatükk keskendub mingile kindlale Qt raamistiku komponendile, millele toetudes on loodud väike näidisrakendus. Rakenduste lähtekoodid on püütud hoida võimalikult lühikesed, et lugejal oleks sellest lihtsam aru saada ning selles orienteeruda. Iga praktilise peatüki lõpus (v.a. esimese rakenduse) on koostatud paar lihtsalt ülesannet, kinnistamaks saadud teadmisi. Praktilise osa lõpus on kogu õppematerjali põhjal koostatud lugejale kolm pikemat ülesannet, mille võimalikke lahendusi saab vaadata lisadest. Lisaks on antud soovitusi, mida Qt raamistikuga seonduvat peale õppematerjali läbitöötamist edasi uurida ja katsetada.

Käesolev õppematerjal on mõeldud kõigile programmeerimishuvilistele inimestele, kes soovivad teada, kuidas luua graafilisi rakendusi. Õppematerjaliga töötamisel on soovitatav omada algteadmisi programmeerimiskeelest C++ ja astmelistest laadilehtedest (*Cascading Style Sheets* ehk CSS). Kõik näidirakendused on kirjutatud programmeerimiskeeles C++ ning peatükis "Laadid" puutume kokku ka CSSiga. Kuna Qt raamistik on multiplatvormne, siis õppematerjaliga töötamine ei eelda mingi kindla operatsioonisüsteemi olemasolu.

Otsustasin kirjutada valitud teemal, kuna tekkis suur isiklik huvi Qt raamistiku vastu ning teiseks ajendiks oli asjakohase eesti keelse materjali puudumine. Käesolev õppematerjal püüabki anda lugejale eesti keeles baasteadmised Qt raamistikust ja selle rakendamisest.

## Olemasolev materjal

Eesti keeles võib leida uudiseid ja artikleid uuema põlvkonna Qt raamistikuga seoses, kuid materjal raamistikul põhinevate rakenduste loomiseks seni puudus.

Qt rakenduste arendamist käsitlevad inglise keelseid materjal leidub erinevaid. Neist kõige põhjalikum on ametlik Qt dokumentatsioon, mis on leitav aadressilt *http://doc.qt.nokia.com/4.8/*. Kuna raamistik on tohutult suur ja erinevaid tööriistu on väga palju, siis võib algajal kasuta segadusse sattuda ja ta ei tea kust oleks õige alustada ja kuhu edasi liikuda. Juba mõningaste kogemustega Qt arendaja leiab seal sirvides suure tõenäosusega abi kõigeile oma küsimustele. Qt ametlik dokumentatsioon on väga hea vaatamaks missugused funktsioonid kuuluvad mingisse klassi, millistest klassidest mingi klass pärineb, millised on mingi funktsiooni argumendid jne ehk tehnilisemad ja detailsemad küsimused.

Täiesti algajatele hästi sobiv sait asub aadressil *http://sector.ynet.sk/qt4-tutorial/*, mille juures väärib märkimist see, et seal pühendatud peatükk Qt dokumentatsioonidest arusaamisele. Piltidega koodinäiteid ja põnevaid Qt nippe leiab veel The Qt4 tutorial saidilt aadressil *http://zetcode.com/tutorials/qt4tutorial/* ja Learn Qt blogist aadressil *http://www.thelins.se/learnqt/*. Mõlemas blogis on põhjalikult lahti seletatud koodiga rakenduste näiteid. Esimene sobib rohkem päris algajatele, teine sisaldab ka teemasid, mis sobivad uurimiseks rohkem harjutanutele. Näiteid ei ole paraku eriti palju ja need hõlmavad vaid väikese osa Qt raamistiku võimalustest.

Palju huvitavaid videoõpetusi võib leida YouTube portaalist aadressil *http://www.youtube.com/*. Kui peaks esile kerkima mõni spetsiifiline probleem, millest ei saa jagu, siis Stack Overflow aadressil *http://www.stackoverflow.com/* ja Qt Centre foorum aadressil *http://www.qtcentre.org/forum.php* on kohad, kust kindlasti tasub abi küsida. Tuhandetele Qt raamistikuga seotud küsimustele on nendes juba vastused olemas.

Blogipõhiste õpetusportaalide halb külg on see, et postitused ei ole õppijale loogilises järjekorras. Segamini on käsitletud lihtsamaid ja keerulisemaid teemasid. Käesolev õppematerjal püüab õpetada erinevaid teemasid sellises järjestuses, et õppijal oleks võimalikult lihtne edasi liikuda ja seda eesti keeles.

# Qt raamistik

## 1. Tutvustus

Qt raamistik (ingl. k. *Qt framework*) on raamisik kiireks ja mugavaks graafiliste rakenduste loomiseks erinevatele platvormidele ja seadmetele. Qt raamistikul baseeruvaid rakendusi on võimalik kasutada operatsioonisüsteemidel Linux, Mac OS X ja Microsoft Windows. Lisaks toetavad Qt raamistkku veel mobiilsed platvormid nagu Maemo, Meego ja Symbian. Qt loodi 1992. aastal Norra firma Trolltech poolt. Pärast seda, kui Trolltechi omandas Nokia, arendab raamistikku viimane. Töö kirjutamise hetkel (detsembris 2011) kannab uusim Qt versiooninumbrit 4.8. Populaarsemad rakendused, mille kasutajaliides on loodud Qt raamistikule, on KDE töölauakeskond oma kõigi rakendustega, Google Earth, Skype, VirtualBox ja VLC Media Player. Lisaks kasutavad Qt raamistikku oma seadmete tarkvaras firmad nagu ASUS, Samsung ja Sony.

Qt raamistikku kuuluvad mitmed multimeedia moodulid, lõimehaldus, võrgutugi, andmebaasidega suhtlus, XML keelte süntaksianalüsaator, vektorgraafika kuvamine ja loomine, WebKit mootoril põhinev veebibrauser ning moodulid OpenGL (2D/3D graafika rakendusliides) ja OpenVG (kiirendatud vektorgraafika liides) kasutamiseks. Qt raamistikul on ka korralik internatsionaliseerimise ja lokaliseerimise tugi. Versiooniga 4.7 tuli Qt raamistikku QML keele *(Qt Meta Language)* kasutamise võimalus. QML on JavaScriptil põhinev deklaratiivne keel, mis on mõeldud kasutajaliidese-kesksete, eelkõige mobiilsete, rakenduste loomisteks. QML võimaldab disaineritel ja arendajatel teha tihedat koostööd ning luua silmapaistvaid animeeritud kasutajaliideseid.

Qt on kirjutatud programmeerimiskeele C++ standardteeke kasutades ja selles sisaldub spetsiaalne makrode kompilaator (*Meta Object Compiler* ehk moc). Peale C++ keele võib Qt võimalusi kasutada veel Java, Perl, PHP, Python, Ruby, juba mainitud QML ja paljudes teistes keeltes. C++ kompilaatoritest on toetatud GCC *(GNU Compiler Collection)* C++ kompilaator, MinGW ja MSVC *(Microsoft Visual C++)*. Qt raamistikul põhinevaid rakendusi võib luua nii koodi kirjutades, kui visuaalselt komponente paigutades ja redigeerides. Erinevatest arendustööriistadest on kirjutatud järgmises peatükis.

Qt raamistik on kaitstud GNU LGPL *(GNU Lesser General Public License)* versioon 2.1 litsentsiga. See tähendab, et raamistikku lähtekood on vabalt kättesaadav ning igaüks võib

seda alla laadida, levitada ja muuta. Antud versioon sobib nii avatud, kui kinnise lähtekoodiga projektide arendamiseks. Lisaks on Qt arendajad sõlminud lepingu firmaga Digia, kes pakub kommertslitsentsiga Qt raamistikku. Kommertsversiooni nimetus on Qt Commercial. (Nokia Corporation, 2011)

## 2. Arendustööriistad

Qt raamistikul põhinevate rakenduste loomiseks on küllaltki palju erinevateid mooduseid. Järgnevalt on kirjeldatud mõned põhilisemad tööriistad, mis hõlbustavad rakenduste arendamist Qt raamistikus.

**Qt Creator** on multiplatvormne integreeritud arenduskeskkond *(IDE)* mis on kohaldatud Qt arendajate vajadustega. See pakub C++ ja QML koodiredaktorit, integreerimist Qt Designer ja Qt Quick Designer kasutajaliidese *(UI)* disaineritega, projektihalduse tööriistu, silurit *(debugger)*, versioonihaldust, mobiilse kasutajaliidese emulaatorit ning see toetab rakenduste arendamist nii töölaua, kui mobiilsete platvormide jaoks.

**Qt Designer** on võimas multiplatvormne kasutajaliideste ja vormide konstruktor. Qt Designer võimaldab kiiresti disainida ja konstrueerida Qt rakendusi ja komponente. Qt Designer'is loodud vormid näevad välja ja käituvad eelvaates täpselt samamoodi nagu valmis rakenduses. Liidese prototüüpidest on võimalik genereerida vastav C++ või Java kood. Qt Designer'it on võimalik kasutada ka integreerituna Qt Creator arenduskeskkonda.

**Qt Quick Designer** on visuaalne QML failide redaktor, milles kuvatav pilt vastab täielikult tegelikkusele välimusele *(WYSIWYG)*. Qt Quick Designer on mõeldud kiireks Qt rakenduste ja komponentide loomiseks puhtalt lehelt. Tööks võib kasutada nii disaineri, kui redaktori režiimi. Alates Qt Creator versioonist 2.2, on ka Qt Quick Designer integreeritud Qt Creator arenduskeskkonda.

**Qt Linguist** pakub tööriistade komplekti, mis kiirendab rakenduste lokaliseerimist ja internatsionaliseerimist (tõlkimist). Qt toetab ühes rakenduses mitme erineva keele kasutamist. Qt Linguist kogub endasse kokku kõik rakenduses olevad tõlgitavad tekstid, võimaldab lihtsalt neid tõlkida, lisada juurde uusi keeli ja see toetab enamusi maailma alfabeetidest.

**Qt Assistant** on dokumentatsiooni vaataja, mida saab lihtsalt kohaldada ja levitada koos Qt rakendusega. Qt Assistant võimaldab kiiret võtmesõna järgi otsimist, täisteksti otsingut, indekseerimist ja järjehoidmist. Dokumentatsiooni võib hoida nii lokaalses masinas, kui veebis.

**qmake** on multiplatvormne rakenduse kompileerimise abimees, mis lihtsustab arendusprojektide kompileerimise protsessi üle erinevate platvormide. qmake genereerib projekti- ja meikfaile *(makefile)*. qmake suudab luua projekte ka Microsoft Visual Studio tarbeks.

**Qt SDK** kombineerib Qt raamistiku paljude intuiivsete ja võimsate tööriistadega, mis muudavad rakenduste arendamise nii töölaua, kui mobiilsete platvormide jaoks lihtsamaks. Qt SDK komplekti kuuluvad kõik eespool kirjeldatud ja mitmed teised abiprogrammid. (Nokia Corporation, 2011).

**QDevelop** on vabatarkvaraline, multiplatvormne ja mitmekeelne Qt arenduskeskkond. Kui eelnevalt loetletud programmid on ametlikud Qt SDK komplekti kuuluvad arendustööriistad mida arendab Nokia, siis QDevelop on arendatud vabatahtlikke arendajate poolt. (QDevelop, 2011).

**KDevelop** on samuti vabatarkvaraline, multiplatvormne (toetatud on Linux, Mac OS X, Microsoft Windows ja Solaris) ja mitmekeelne lihtsalt kasutatav arenduskeskkond mis toetab erinevaid programmeerimiskeeli. KDevelopis on ühendatud projektihaldus, koodiredaktor, klassisirvija ja silur ning see toetab ka Qt raamistikul põhinevate projektide loomist. (KDevelop, 2011).

## 3. Arenduskeskonna seadistamine

Järgmistes peatükkides proovime ise luua rakendusi Qt raamistikus ning selle tarbeks seadistame omale vajaliku arenduskeskonna.

Operatsioonisüsteemiks sobib Linux, Mac OS X või Microsoft Windows. Qt rakenduste arendamiseks peavad masinasse olema paigaldatud vähemalt käitusteegid *(runtime libraries)* ja arendusfailid *(development files)*. Graafilise arenduskeskonna kasutamine on arendaja oma vaba valik. Käesolevas õppematerjalis siiski kasutatakse rakenduste loomiseks Qt Creator graafilist arenduskeskkonda, sest see ei nõua kasutajalt keerulist seadistamist

ning toimib kõikides loetletud operatsioonisüsteemides sarnaselt. Sellega on tagatud õppematerjali võimalikult suur kasulikkus lugeja operatsioonisüsteemist sõltumatult. Kogu vajalik tarkvara on võimalik alla laadida Qt SDK komplektina aadressilt *http://qt.nokia.com/downloads/*.

Linuxis on võimalik vajalikud pakid alla laadida lisaks Qt ametlikule veebilehele ka distributsiooni pakihaldurist. Seda võib teha kas graafilise pakihalduri kaudu või käsurealt. Vaja on installeerida Qt Creator, mille sõltuvustena paigaldatakse ka käitusteegid, arendusfailid ja veel mõned abiprogrammid nagu Qt Linguist. Teisisõnu, koos Qt Creatoriga paigaldatakse kogu tarkvara, mida käesoleva õppematerjaliga töötades võib vaja minna. Installeerimine käsurealt erinevates distributsioonitüüpides võib olla veidi erinev. Allpool on toodud installeerimiskäsklused kahes levinud baasdistributsioonis ja nende derivaatides.

- Debian Linux ja selle derivaadid (Ubuntu, Mint jt):

```
sudo apt-get install qtcreator
```

- Fedora Linux ja selle derivaadid (Red Hat Enterprise Linux, CentOS jt):

```
sudo yum install qtcreator
```

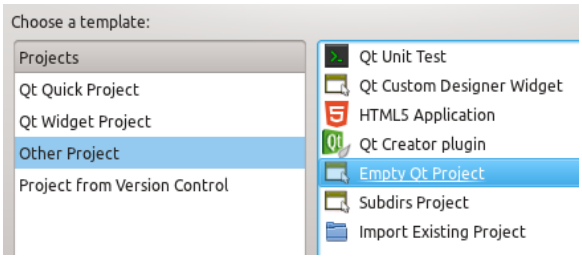
Kui Qt Creator (leiad selle operatsioonisüsteemi programmide menüüst) käivitub ilma probleemideta (joonis 1), oled edukalt paigaldanud arendustarkvara. Enne kui liigume edasi rakenduste loomise juurde, muudame teksti kodeeringu. Seda saa teha valides Qt Creator menüüst *Tools -> Options -> Text Editor -> Behaviour* ning *Default encoding* väärtuseks määra UTF-8. Nii ei teki meil probleeme osade eesti keele tähtedega.

*Joonis 1. Qt Creator.*

## 4. Esimene rakendus

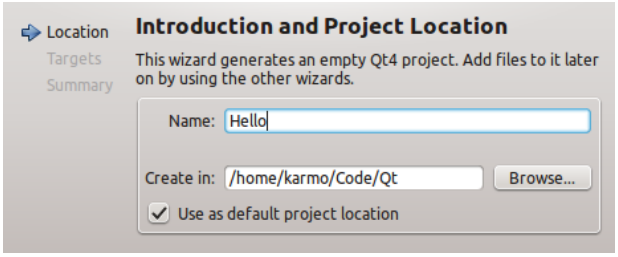
Käesolevas ja edaspidistes peatükkides õpetatakse Qt raamistikul põhinevate rakenduste loomist. Iga peatükk keskendub ühele kindlale Qt raamistiku moodulile. Peatükke on kokku kümme ja soovitav on need läbi töötada sellises järjekorras, nagu need on antud. Juba mingis eelnevas peatükis lahti seletatud funktsionaalsust või koodi enamasti üle ei korrata. Igas praktilises peatükis on kõigepealt esitatud programmi lähtekood, seejärel koodi seletus ning lõpus kuvatõmmis töötavast rakendusest.

Alustamiseks ava programm Qt Creator ning loo uus projekt (*Ctrl+N* või *File -> New File or Project...*). Avanenud aknast vali vasakult poolt *Other Project* ning paremalt poolt *Empty Qt Project* (joonis 2). Ka kõikides järgnevates praktilistes peatükkides tuleb teha projekti loomisel samad valikud.




*Joonis 2. Projekti malli valimine.*

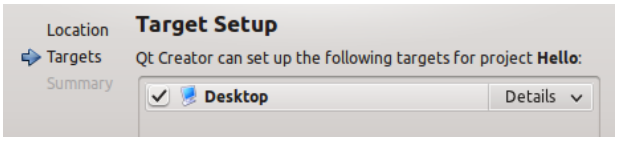
Kirjuta avanevasse aknasse projekti nimi ning vali asukoht, kuhu projekti kataloog salvestatakse (joonis 3). Märkides ära *Use as default project location*, pakutakse edaspidi seda asukohta vaikimisi kõikide uute projektide puhul. Vajuta *Next*.




*Joonis 3. Projekti nimi ja asukoht.*

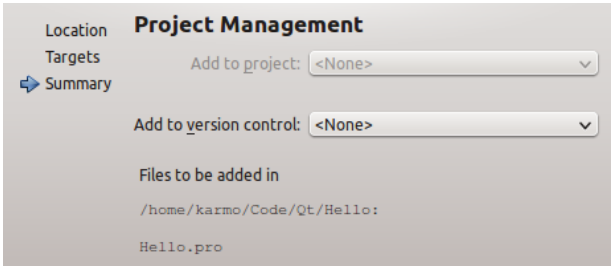
Teisel vahelehel saab valida rakenduse sihtplatvormid (joonis 4). Meie projekti puhul peaks seal olema ainult üks vaikimisi märgitud valik *Desktop*, mis tähendab, et rakendus

kompileeritakse töölaua masinate peal töötamiseks. Teiste platvormide jaoks tuleks valida mõni muu projekti mall, kuid sellel siinkohal pikemalt ei peatu. Vajuta *Next*.



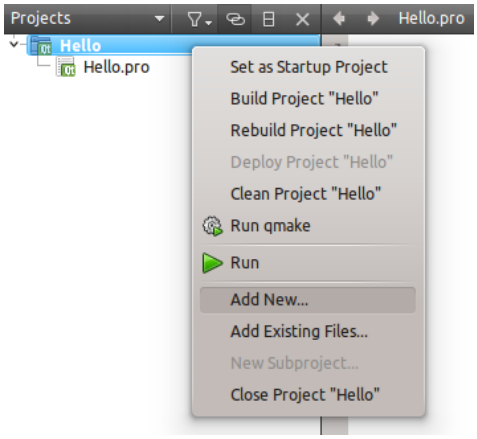

*Joonis 4. Projekti sihtplatvormid.*

Kolmandal ja viimasel vahelehel pakutakse projektihaldust (joonis 5). Versioonihaldusele antud juhul ei keskendu. Seega jäta kõik nii nagu on ja vajuta *Finish*.




*Joonis 5. Projektihaldus.*

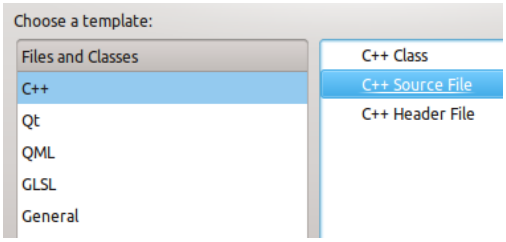
Nüüdseks oleme loonud uue tühja projekti, kuhu lisame esimese ja ainukese koodifaili. Faile saab projekti lisada, valides projekti failipuus projekti nimel paremakliki menüüst *Add New...* (joonis 6). Juba valmis faile (pildid, videod, dokumendid jms) saab lisada samast menüüst valides *Add Existing Files...*.




*Joonis 6. Uue faili lisamine projekti.*

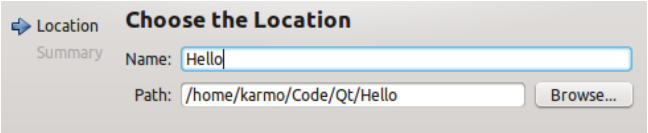
Koodifaili tüübiks vali vasakult poolt *C++* ning paremalt poolt *C++ Source File* (edaspidi *lähtefail*) (joonis 7). Kui õppematerjalis on kästud luua *päisefail*, siis tuleks valida paremalt

*C++ Header File*. Hetkel jääme aga joonisel oleva aktiivse valiku juurde.



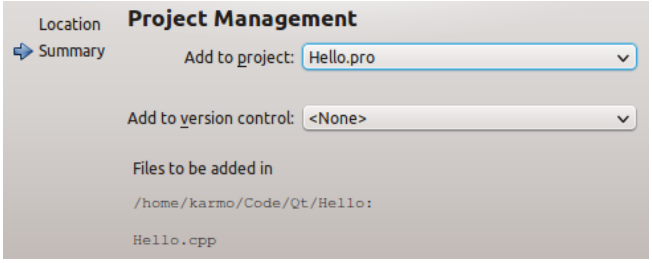

*Joonis 7. Faili tüübi valik.*

Kirjuta avanenud aknasse faili nimi ja vali kataloog kuhu fail salvestatakse (joonis 8). Praegu ja ka edaspidi jätame faili kataloogi nii nagu vaikimisi pakutakse.



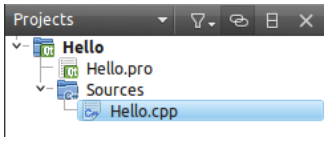

*Joonis 8. Faili nimi ja asukoht.*

Teisel vahelehel saab määrata millisesse versioonihaldusesse fail kuulub (joonis 9). Kuna me projektis versioonihaldust ei kasutanud, siis jätame selle ka siin valimata.




*Joonis 9. Faili projektihaldus.*

Näeme, et uus tühi C++ koodifail on projekti failipuusse juurde tekkinud (joonis 10). Kliki sellel, et paremale tekiks tühi tekstiväli kuhu saab hakata koodi kirjutama.




*Joonis 10. Projekti failipuu.*

Avanenud tühja koodivälja kirjuta järgmine koodijupp:

```
#include <QtGui>

int main(int argc, char *argv[])
{
   QApplication app(argc, argv);
   QMainWindow mainWindow;
   mainWindow.setWindowTitle("Tere!");
   mainWindow.show();
   return app.exec();
}
```

Järgnevalt selgitan täpsemalt mida antud kood teeb. Päises kaasame *QtGui* mooduli, milles sisalduvad kõik vajalikud klassid, mida meie rakendus kasutab. Antud rakenduse puhul on tarvis *QApplication* ja *QMainWindow* klasse. Nende kahe mooduli asemel on päises kaasatud aga hoopis *QtGui* mooduli, et ei peaks kõiki sõltuvuses olevaid graafilise kasutajaliidese mooduleid eraldi välja tooma. Põhifunktsioonis defineerime *QApplication*-tüüpi *app* objekti ning *QMainWindow*-tüüpi *mainWindow* objekti. *QApplication* klass haldab rakenduse ressursse ja seadeid ning on tarvilik kõikide graafiliste Qt programmide puhul. Kuna Qt aktsepreerib mõningaid käivitusargumente ja -väärtuseid, siis saab neid objekti loomisel ette anda. *QApplication* objektiga saab suhelda ükskõik millisest rakenduse klassist globaalse *qApp*-nimelise viida kaudu. *QMainWindow* klass pakub võimalusi rakenduse peaakna loomiseks. Peaakna teksti saab muuta *setWindowTitle* funktsiooni kaudu. Kuna Qt vidinad *(widgets)* ei ole vaikimisi nähtavad, siis funktsioon *show* muudab vidina nähtavaks koos kõikide tema alamvidinatega.

Rakenduses on kasutatud *std::string* andmetüüpi akna pealkirja jaoks. Soovitav on kasutada selle asemel *QString* objekti, sest:

- *QString* on *unicode* ja võimaldab kasutada palju rohkem erinevaid sümboleid kui 8-bitine *std::string*
- *QString* on paljudel juhtudel kiirem kui *std::string*
- *QString* on funktsionaalsem kui *std::string*

(wysota, 2008).

Koodinäide *QString* kasutamiseks:

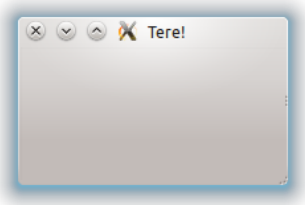
```
// Latin
mainWindow.setWindowTitle(QString("Tere!"));
// UTF-8
mainWindow.setWindowTitle(QString::fromUtf8("Mõääöü!"));
```

Rakenduse kompileerimiseks ja käivitamiseks vajuta *Ctrl+R* või vali ülevalt menüüst *Build -> Run*. Rakenduse käivitamisel peaks avanema väike aknake nagu on kujutatud joonisel 11. Kompileeritud rakendus salvestatakse uude kataloogi, mille nimi on *%PROJEKT%-build-%PLATVORM%*, kus *%PROJEKT%* on projekti nimi ja *%PLATVORM%* on sihtplatvormi nimetus (antud juhul *desktop*).

Linuxis saab rakendusi kompileerida lihtsalt ka käsurealt, ilma Qt Creatori või muu graafilise arenduskeskkonna abita. Selleks tuleb avada käsurida ning liikuda kataloogi, kus asuvad rakenduse lähtefailid. Seejärel tuleb sisestada üksteise järel kolm käsklust:

```
qmake -project
qmake
make
```

Esimene käsklus genereerib projekti konfiguratsioonifaili, teine käsklus tekitab meikfaili vastavalt konfiguratsioonile ning kolmas käsklus loeb sisse meikfaili ja kompileerib rakenduse. Esimest käsklust on vaja sisestada vaid enne kõige esimest kompileerimist ning siis, kui projekti on tekkinud juurde uusi lähte- või päisefaile. Üldjuhul piisab lihtsalt käskudest *qmake* ja *make*. Käsurealt kompileerimise puhul salvestatakse kompileeritud rakendus vaikimisi samasse kataloogi lähtefailidega.




*Joonis 11. Esimene rakendus.*

## 5. Paigutushaldurid

Järgmise rakenduse pühendame erinevatele võimalustele Qt vidinaid rakenduses paigutada. Qt rakendustes enamasti ei paigutata vidinaid staatilistele positsioonidele, vaid automaatne asetus jäetakse paigutushaldurite hooleks. Paigutushalduri ülesanneteks on:

- Objektide positsiooni määramine
- Akende vaikimisi suurused
- Akende miinimumsuurused

- Suuruste muutuste käsitlemine
- Automaatsed uuendused peale erinevaid toiminguid:
    - Objekti kirja suuruse või teksti muutmine
    - Objektide näitamine või peitmine
    - Objektide eemaldamine

Kuna võimalusi objektide paigutamiseks on niivõrd palju, et kõigest ei jõua siin kirjutada, siis keskendume ainult kõige lihtsamatele ja levinumatele paigutushalduritele.

Loo uus projekt nagu eelmises peatükis ning lisa sellesse kolm faili: lähtefailid *main.cpp* ja *mainWindow.cpp* ning päisefail *mainWindow.h*. Praegu ja edaspidi paneme suuremad rakenduse osad nagu aknad eraldi failidesse. Lisaks lähtefailidele kasutame ka päisefaile. Seda selleks, et kood oleks võimalikult modulaarne. Demorakenduse lähtekood on järgmine:

*main.cpp*:

```
#include <QtGui>
#include "mainWindow.h"

int main(int argc, char *argv[])
{
   QApplication app(argc, argv);
   Window *mainWindow = new Window;
   mainWindow->show();
   return app.exec();
}
```

**NB!** Järgnevates näidisrakendustes saab *main.cpp* fail olema täpselt samasuguse koodiga, juhul kui seda pole eraldi välja toodud.

*mainWindow.h*:

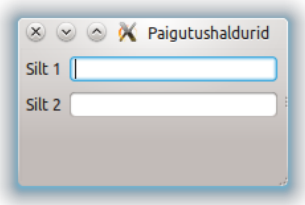
```
#include <QtGui>

class Window: public QMainWindow
{
   public:
   Window();
};
```

*mainWindow.cpp*:

```
#include "mainWindow.h"

Window::Window(): QMainWindow()
{
   QWidget *widget = new QWidget();
```

```
    //QHBoxLayout *layout = new QHBoxLayout;
    //QVBoxLayout *layout = new QVBoxLayout;
    //QGridLayout *layout = new QGridLayout;
    QFormLayout *layout = new QFormLayout;
    QLabel *label1 = new QLabel(QString::fromUtf8("Silt 1"));
    QLabel *label2 = new QLabel(QString::fromUtf8("Silt 2"));
    QLineEdit *lineEdit1 = new QLineEdit();
    QLineEdit *lineEdit2 = new QLineEdit();
    /* QHBoxLayout, QVBoxLayout:
    layout->addWidget(label1);
    layout->addWidget(lineEdit1);
    layout->addWidget(label2);
    layout->addWidget(lineEdit2); */
    /* QGridLayout:
    layout->addWidget(label1, 0, 0);
    layout->addWidget(lineEdit1, 0, 1);
    layout->addWidget(label2, 1, 0);
    layout->addWidget(lineEdit2, 1, 1); */
    layout->addRow(label1, lineEdit1);
    layout->addRow(label2, lineEdit2);
    widget->setLayout(layout);
    setCentralWidget(widget);
    setWindowTitle(QString::fromUtf8("Paigutushaldurid"));
}
```

*main.cpp* lähtefailis on kood rakenduse käivitamiseks. *mainWindow.h* on päisefail, mis sisaldab peaakna klassi definitsiooni. *mainWindow.cpp* lähtefailis on defineeritud peaakna klass, kus toimub peaaknal asetsevate vidinate loomine. Et paigutushaldurit saaks kasutada, on vaja luua kõigepealt konteinervidin. Konteinervidinaks on siin *QWidget*-tüüpi vidin *widget*. Koodinäites on kasutatud QFormLayout paigutushaldurit, mis paneb vidinad kahe kaupa ühele reale. Näites on vidinateks silt (*QLabel*-tüüpi objekt) ning tekstisisestuskastike (*QLineEdit*-tüüpi objekt). Igal real kummagist üks. *QFormLayout* puhul lisatakse vidinad paigutushaldurisse *addRow* meetodit kasutades. Sellel on kaks argumenti, mis tähistavad vastavalt vasak- ja parempoolset lisatavat vidinat. Koodi välja kommenteeritud osades on näha, kuidas luua ja kasutada ka teisi tüüpi paigutushaldureid. *QHBoxLayout* ja *QVBoxLayout* paigutavad vidinaid vastavalt horisontaalselt ühte ritta või vertikaalselt ühte veergu. Vidinaid lisatakse neisse *addWidget* meetodi kaudu ning millel on üks argument ja selleks ongi lisatav vidin. *QGridLayout* paigutab vidinad võrestikulaadselt. Sellesse lisatakse vidinaid samuti *addWidget* meetodi kaudu, kuid argumente on kokku kolm. Esimene argument on lisatav vidin ning teine ja kolmas on vastavalt rea ja veeru number, kuhu vidin paigutatakse. Tähelepanu tuleks pöörata sellele, et rea ja veeru numbrid algavad nullist mitte ühest. Paigutushaldur seotakse konteinervidinaga *setLayout* meetodit kasutades. Kõige lõpuks seame konteinervidina peaakna tsentraalseks vidinaks. Peaakna objekt luuakse ja kutsutakse välja *main.cpp* lähtefailist meetodiga *show*. Käivitatud rakendus peaks sarnanema joonisel 12 kujutatuga.




*Joonis 12. Paigutushalduri näiterakendus.*

**Ülesanded**

1. Lisa rakendusse juurde veel kolmas rida, kus on silt "Silt 3" ning tekstisisestuskastike.
2. Muuda rakendust nii, et nii kolm silti, kui ka kolm tekstisisestuskastikest oleks eraldi vertikaalsete paigutushaldurite sees ning need paigutushaldurid omakorda ühe horistontaalse paigutushalduri sees.

## 6. Vidinad

Vidinad on peamised Qt kasutajaliideste elemendid. Vidinad võivad kuvada mingeid andmeid ja informatsiooni, võtta vastu sisendväärtusi või olla konteineriks teistele vidinatele. Vidinat, mis ei kuulu ühegi teise vidina koosseisu, nimetatakse aknaks. Vidinad näevad erinevates operatsioonisüsteemides vaikimisi pisut erinevad välja, kuid mõõdud ja paigutus sellest ei muutu. Peatükis "Laadid" räägime lähemalt Qt vidinatele omapärase välimuse andmisest. Selles peatükis aga vaatleme, milliseid erinevaid Qt vidinaid on olemas, ja kuidas neid kasutada rakenduste loomisel.

*main.cpp* fail on siin ja edaspidi täpselt samasugune nagu paigutushaldurite peatükis. *mainWindow.h* kood on siin samuti identne eelmise peatüki omaga.

*mainWindow.cpp*:

```
#include "mainWindow.h"

Window::Window(): QMainWindow()
{
   QWidget *widget = new QWidget;
   QVBoxLayout *layout = new QVBoxLayout;

   QLabel *label = new QLabel(QString::fromUtf8("Silt"));

   QLineEdit *lineEdit = new QLineEdit(QString::fromUtf8("Üherealine tekst"));
```

```
QTextEdit *textEdit = new QTextEdit;
textEdit->setFont(QFont("DejaVu Sans", 10, QFont::Normal, true));
textEdit->setTextColor(QColor(Qt::red));
textEdit->setText(QString::fromUtf8("Mitmerealine\ntekst"));

QPushButton *button = new QPushButton(QString::fromUtf8("Vajuta"));

QComboBox *comboBox = new QComboBox;
comboBox->addItem("", "");
comboBox->addItem(QString::fromUtf8("Audi"), "1");
comboBox->addItem(QString::fromUtf8("BMW"), "2");

QSpinBox *spinBox = new QSpinBox;
spinBox->setMinimum(1);
spinBox->setMaximum(5);

QHBoxLayout *checkBoxLayout = new QHBoxLayout;
QCheckBox *checkBox1 = new QCheckBox(QString::fromUtf8("Õun"));
QCheckBox *checkBox2 = new QCheckBox(QString::fromUtf8("Apelsin"));
checkBoxLayout->addWidget(checkBox1);
checkBoxLayout->addWidget(checkBox2);

QHBoxLayout *radioButtonLayout = new QHBoxLayout;
QButtonGroup *radioButtonGroup = new QButtonGroup;
QRadioButton *radioButton1 = new QRadioButton(QString::fromUtf8("Jah"));
QRadioButton *radioButton2 = new QRadioButton(QString::fromUtf8("Ei"));
radioButtonGroup->addButton(radioButton1);
radioButtonGroup->addButton(radioButton2);
radioButtonLayout->addWidget(radioButton1);
radioButtonLayout->addWidget(radioButton2);

QGroupBox *groupBox = new QGroupBox(QString::fromUtf8("Kuupäev ja aeg"));
QVBoxLayout *groupBoxLayout = new QVBoxLayout;

QDateEdit *dateEdit = new QDateEdit;
dateEdit->setDisplayFormat("yyyy.MM.dd");
dateEdit->setDate(QDate::currentDate());

QTimeEdit *timeEdit = new QTimeEdit;
timeEdit->setDisplayFormat("hh:mm");
timeEdit->setTime(QTime::currentTime());

groupBoxLayout->addWidget(dateEdit);
groupBoxLayout->addWidget(timeEdit);
groupBox->setLayout(groupBoxLayout);

QSlider *slider = new QSlider(Qt::Horizontal);
slider->setRange(0, 100);
slider->setSingleStep(0.1);
slider->setPageStep(10);
slider->setTickInterval(10);
slider->setTickPosition(QSlider::TicksRight);

QDial *dial = new QDial;
dial->setRange(0, 10);
dial->setSingleStep(1);
dial->setPageStep(10);
dial->setMaximumHeight(48);

QStandardItemModel *model = new QStandardItemModel;
model->setHorizontalHeaderLabels(QStringList() << QString::fromUtf8("Nimi") << QString::fromUtf8("Vanus"));
QList<QStringList> rows = QList<QStringList>()
    << (QStringList() << QString::fromUtf8("Jaan Jalgratas") << QString("42"))
    << (QStringList() << QString::fromUtf8("Mari Maasikas") << QString("25"))
    << (QStringList() << QString::fromUtf8("Villu Varrukas") << QString("33"));
foreach(QStringList row, rows) {
    QList<QStandardItem *> items;
    foreach(QString text, row) {
        items.append(new QStandardItem(text));
```

```
        }
        model->appendRow(items);
    }
    QTableView *tableView = new QTableView;
    tableView->setModel(model);
    tableView->setMaximumHeight(120);

    layout->addWidget(label);
    layout->addWidget(lineEdit);
    layout->addWidget(textEdit);
    layout->addWidget(button);
    layout->addWidget(comboBox);
    layout->addWidget(spinBox);
    layout->addLayout(checkBoxLayout);
    layout->addLayout(radioButtonLayout);
    layout->addWidget(groupBox);
    layout->addWidget(slider);
    layout->addWidget(dial);
    layout->addWidget(tableView);
    widget->setLayout(layout);
    setCentralWidget(widget);
    setWindowTitle(QString::fromUtf8("Vidinad"));
}
```

Ülal olev kood on käesoleva õppematerjali üks pikemaid, kuid kood ise on algusest lõpuni väga lihtne. Kasutatud on selles rakenduses enamusi vidinatest, mida ühe lihtsama rakenduse loomisel võib tarvis minna.

Kõik vidinad pärinevad baasklassist nimega *QWidget*. Näites luuakse kõigepealt tsentraalne vidin ning paigutushaldur, milledest sai juba eelnevalt kirjutatud, ning seejärel luuakse erinevaid uusi vidinad. Üleval pool primitiivsemad, allpool pisut rohkem koodi kirjutamist nõudvad. *QLabel* on vidin mittemuudetava teksti ja *QLineEdit* muudetava üherealise teksti kuvamiseks. Kõikidele tekstividinatele antakse tekst ette läbi kontruktori või meetodit *setText* kasutades. *QTextEdit* on vidin mitmerealise muudetava teksti jaoks, mille juures on näidatud ka paari teksti muutmise meetodi kasutamist. *setFont* on teksti stiili ja suuruse, *setTextColor* teksti värvi määramiseks.

*QPushButton* nupp seotakse mingisuguse kasutaja poolt antava korraldusega. Hetkel ei juhtu nupule vajutades midagi, kuid järgmises peatükis vaatame rakendusele interaktiivsust lisavaid signaale ja pesasid lähemalt. Ripploend *QComboBox* lubab valida erinevate valikväärtuste hulgast välja vaid ühe. Valikuid lisatakse läbi *addItem* meetodi, kus esimeseks parameetriks kasutajale nähtav tekst ning teiseks valiku väärtus. Järgmine uus vidin on *QSpinBox*, mis võimaldab valida ühe täisarvulise väärtuse mingist kindlast vahemikust. *setMinimum* ja *setMaximum* meetodid on väärtuste vahemiku ja *setSingleStep* väärtuste vahele jääva sammu määramiseks. Komakohtadega väärtuste jaoks on olemas *QDoubleSpinBox*.

*QCheckBox* märkeruute ja *QRadioButton* raadionuppe luuakse sarnaselt, vidina kõrval kuvatavat teksti kontruktorile ette andes. Nende horisontaalselt paigutamiseks on rakenduses kasutatud *QHBoxLayout* paigutushaldureid. Raadionupud grupeeritakse tavaliselt *QButtonGroup* tüüpi vidinasse. Grupis oleva raadionupu märkimisel välistatakse kõik ülejäänud selles grupis olevad raadionupud. Märkeruutude ja raadionuppude väärtust saab kontrollida *isChecked* meetodi abil, mis tagastab Boole'i tüüpi väärtuse (*true*, kui on märgitud ja *false*, kui on märkimata).

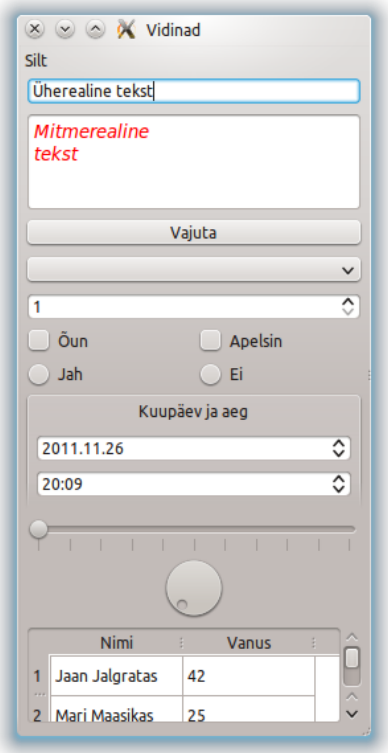
Vidinate üksteisest visuaalselt eraldamiseks võib kasutada *QGroupBox* grupeerimisvidinat. Näites on *QGroupBox* konteineriks paigutushaldurile, mille sees on kuupäeva vidin *QDateEdit* ja kellaaja vidin *QTimeEdit*. Pealkiri on grupile lisatud konstruktori kaudu. On olemas ka kuupäeva ja kellaaega ühendav vidin *QDateTimeEdit*. Kuupäeva ja kellaaja vidina kuvamisformaat on muudetav *setDisplayFormat* meetodiga ning väärtust saab anda vastavalt *setDate* ja *setTime* meetoditega. Näites on mõlemale vidinale määratud käesolev ajahetk. Mingi muu ajahetke määramiseks on abiks järgmine koodinäide:

```
dateEdit->setDate(QDate(1999, 12, 31)); // QDateEdit
timeEdit->setTime(QTime(23, 59)); // QTimeEdit
```

Järgmisena vaatleme kahte liigutatavat vidinat, mis näevad välja erinevad, kuid on oma omadusetelt sarnased, sest mõlemad pärinevad ühest samast *QAbstractSlider* abstraktsest klassist. *QSlider* on vertikaalne või horisontaalne liugur, sõltuvalt sellest, kas tema konstruktorile anda ette *Qt::Horizontal* või *Qt::Vertical*. *QDial* on aga ümmargune pööratav nupp. Nagu *QSpinBox*, on ka *QSlider* ja *QDial* mõeldud ühe täisarvulise väärtuse valimiseks mingist kindlast väärtuste vahemikust. *setRange* meetod määrab minimaalse ja maksimaalse võimaliku valitava väärtuse, *setSingleStep* määrab sammu, mis jääb valikuvariantide vahele. Meetod *setPageStep* määrab *PgUp* ja *PgDown* nuppudega liikumise sammu. *QSlider* vidina puhul *setTickInterval* määrab skaala tähiste vahemiku.

Viimane vidin selles näidisrakenduses on andmetega tabel. Tabel koosneb kahest komponendist: tabeli vidinast *QTableView* ja andmemudelist *QStandardItemModel*. *setMaximumHeight* ja *setMaximumWidth* määravad ära tabeli maksimaalsed mõõtmed. Kuna need kaks meetodit pärinevad *QWidget* klassist, siis saab neid kasutada ka kõikide teiste vidinate puhul. Andmemudel seotakse tabeliga *setModel* meetodi läbi. Andmemudeli klass *QStandardItemModel* suudab endas hoida erinevaid andmestruktuure, mida saab kasutada

tabelites, listides või puuvaadetes. Andmemudeli veeru nimetused määratakse *setHorizontalHeaderLabels* meetodiga, mille parameetriks on *QStringList* objekt ja mis koosneb *QString* andmetüüpidest. Andmed, mis mudeli ridu ja veerge täidavad, on samuti *QStringList* tüüpi. See tähendab, et iga andmemudeli rida on *QStringList* objekt ja see koosneb *QString* andmetüüpidest. Andmemudeli read kokku moodustavad ühe suure *QStringList* objekti. Koodinäites sisemise *foreach* tsükli sees moodustatakse *QStandardItem* tüüpi list, ja välimise *foreach* tsükli sees lisatakse need listid *appendRow* meetodiga andmemudelisse.




*Joonis 13. Vidinate näiterakendus.*

**Ülesanded**

1. Lisa rakendusse, olemasolevate kõrvale, juurde märkeruut "Banaan" ning raadionupp "Võibolla".
2. Lisa rakendusse kalendri vidin. Määra nädala esimeseks päevaks esmaspäev, kuva päeva lahtrite vahelised jooned ning peida vertikaalne päis (jäta alles ainult 7 veergu, üks iga nädalapäeva jaoks).

## 7. Signaalid ja pesad

Oleme jõudnud oma õppimisega nii kaugele, et hakkame vaatama, kuidas rakenduse vidinad panna reageerima mingitele tegevustele. Käesolevas peatükis loome rakenduse, kus vidinad suhtlevad omavahel. Objektide vaheliseks suhtluseks kasutatakse signaale ja pesasid. Signaalide ja pesade mehhanism on Qt raamistiku üheks põhiliseks tunnuseks, sest selle poolest erineb ta kõige rohkem teistest analoogsetest raamistikest. Selles peatükis loome rakenduse, mille peaaknasse saab trükkida oma nime ning valida liuguri abil ühest kümneni mingi suvalise arvu. Pärast nupule "OK" vajutamist avaneb dialoogi aken, milles tervitatakse nimepidi ning kuvatakse eelnevalt valitud number. Dialoogil on olemas nupp, mis selle sulgeb. Pärast näiterakenduse koodinäidet vaatleme signaale ja pesasid tehnilise poole pealt lähemalt.

*mainWindow.h*:

```
#include <QtGui>
#include "dialog.h"

class Window: public QMainWindow
{
   private:
   Q_OBJECT
   Dialog *dialog;
   QLineEdit *nameField;
   QSlider *nrSlider;
   private slots:
   void openDialog();
   public:
   Window();
};
```

*mainWindow.cpp*:

```
#include "mainWindow.h"

Window::Window(): QMainWindow()
{
   dialog = new Dialog;
   QWidget *widget = new QWidget;
   QFormLayout *layout = new QFormLayout;
   QLabel *nameLabel = new QLabel(QString::fromUtf8("Nimi:"));
   nameField = new QLineEdit;
   QLabel *nrLabel = new QLabel(QString::fromUtf8("Number:"));
   nrSlider = new QSlider(Qt::Horizontal);
   nrSlider->setRange(0, 10);
   nrSlider->setSingleStep(1);
   nrSlider->setPageStep(10);
   nrSlider->setTickInterval(1);
   nrSlider->setTickPosition(QSlider::TicksRight);
   QLCDNumber *nrLCD = new QLCDNumber();
   QHBoxLayout *nrLayout = new QHBoxLayout;
   nrLayout->addWidget(nrSlider);
   nrLayout->addWidget(nrLCD);
```

```
    QPushButton *dialogButton = new QPushButton(QString::fromUtf8("OK"));

    layout->addRow(nameLabel, nameField);
    layout->addRow(nrLabel, nrLayout);
    layout->addRow(dialogButton);
    widget->setLayout(layout);
    setCentralWidget(widget);
    setWindowTitle(QString::fromUtf8("Signaalid"));

    connect(nrSlider, SIGNAL(valueChanged(int)), nrLCD, SLOT(display(int)));
    connect(dialogButton, SIGNAL(clicked()), this, SLOT(openDialog()));
}
void Window::openDialog()
{
    dialog->setLabel(nameField->text(), QString().setNum(nrSlider->value()));
    dialog->show();
}
```

*dialog.h*:

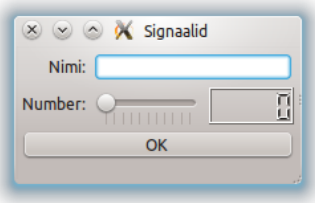
```
#pragma once

#include <QtGui>

class Dialog: public QDialog
{
    private:
    QLabel *label;
    public:
    Dialog();
    void setLabel(QString, QString);
};
```

*dialog.cpp*:

```
#include "dialog.h"

Dialog::Dialog(): QDialog()
{
    QVBoxLayout *layout = new QVBoxLayout;
    label = new QLabel;
    QPushButton *closeButton = new QPushButton(QString::fromUtf8("Sulge"));

    layout->addWidget(label);
    layout->addWidget(closeButton);
    setLayout(layout);
    resize(200, 100);
    setModal(true);
    setWindowTitle(QString::fromUtf8("Tere!"));

    connect(closeButton, SIGNAL(clicked()), this, SLOT(hide()));
}
void Dialog::setLabel(QString name, QString nr)
{
    label->setText(QString::fromUtf8("Tere, ") + name + QString::fromUtf8("! Valisid numbri ") + nr + QString::fromUtf8("."));
}
```

Kuigi koodi tundub selles näidisrakenduses olema üksjägu, siis suur enamus sellest on juba eelmistes peatükkides läbi vaadatud. Uuteks vidinateks on siin *QDialog* ja *QLCDNumber*. Selle peatüki tähtsaimaks koodi osaks on hoopiski meetod nimega *connect*. *Connect* loob ühenduse saatja objektist tuleva signaali ja vastuvõtja objekti mingi meetodi (pesa) vahel.

Kui funktsioonil õnnestub signaal saata vastuvõtja pesasse, siis tagastatakse *true*, vastasel juhul *false*. Funktsiooni *connect* parameetriteks on saatja objekt, saadetav signaal, vastuvõtja objekt ning pesa. Viiendaks parameetriks on ühenduse tüüp ja selle vaikimisi väärtuseks on *Qt::AutoConnection*. Üks signaal võib olla ühendatud paljudesse pesadesse ja teistesse signaalidesse. Mitu signaali võib olla ühendatud ühte ja samasse pessa (selleks tuleb appi võtta *QSignalMapper*). Kui signaal on ühendatud mitmesse pessa, siis pesad aktiveeritakse samas järjekorras nagu ühendused loodi. Et mingis klassis saaks üldse deklareerida signaale ja pesasid, tuleb selle klassi definitsiooni privaat sektsioonis kasutada *Q_OBJECT* makrot.




*Joonis 14. Signaalide ja pesade näiterakendus.*

**Ülesanded**

1. Näita *QLCDNumber* vidinas arve sisse tuleva signaali väärtusest kümme korda suuremana.
2. Kui nime lahtrisse pole midagi kirjutatud, blokeeri "OK" nupp.

## 8. Tõlkimine

Mõnikord võib tulla vajadus luua rakendusi erinevaid keeli kõnelevale kasutajaskonnale. Selliste nõuetega tuleb Qt raamistik mängleva kergusega toime. Mitme keele toe lisamine rakendusse ei nõua suurt koodi ümberkirjutamist ning tõlkimisega saab hakkama ka inimene, kes pole kunagi programmeerimisega tegelenud. Qt rakenduste tõlgetete muutmine ei nõua isegi rakenduse uuesti kompileerimist. Mugavaks tõlkimiseks on olemas Qt Linguist programm, mis kuulub ka Qt SDK komplekti. Selles peatükis vaatamegi, kuidas luua mitmekeelseid Qt rakendusi. Teeme läbi järgnevad sammud: loome võõrkeelse tõlgitava rakenduse, genereerime tõlkefaili, tõlgime stringid ning lõpuks kompileerime tõlke, et meie rakendus saaks seda kasutada. Järgnevas rakenduses kinnistame ka signaalide ja pesade kasutamist.

*main.cpp*:

```
#include <QtGui>
#include "mainWindow.h"

int main(int argc, char *argv[])
{
   QApplication app(argc, argv);
   QTranslator translator;
   //translator.load("languages/" + QLocale::system().name());
   translator.load("translations/et");
   app.installTranslator(&translator);
   Window *mainWindow = new Window;
   mainWindow->show();
   return app.exec();
}
```

*mainWindow.h*:

```
#include <QtGui>

class Window: public QMainWindow
{
   private:
   Q_OBJECT
   QPushButton *button;
   public:
   Window();
};
```

*mainWindow.cpp*:

```
#include "mainWindow.h"

Window::Window(): QMainWindow()
{
   QWidget *widget = new QWidget;
   QVBoxLayout *layout = new QVBoxLayout;
```

```
    button = new QPushButton(tr("Push me!"));
    layout->addWidget(button);
    widget->setLayout(layout);
    setCentralWidget(widget);
    setWindowTitle(tr("Translating"));

    connect(button, SIGNAL(clicked()), qApp, SLOT(quit()));
}
```

Tõlgitavate Qt rakenduste kirjutamiseks on vaja teada kolme põhilist asja: tõlgete laadimist *QTranslator* mooduli abil, tõlkemooduli paigaldamist rakendusse ning rakenduses leiduvate stringide tõlgitavaks muutmist *tr*-meetodi abil. Kuna *QTranslator* objekt tuleb tekitada enne vidinaid, mis sisaldavad tõlgitavaid tekste, siis kirjutame tõlkemooduliga seonduva koodi päris rakenduse algusesse, *main.cpp* faili.

Antud ingliskeelsete stringidega näiterakenduses laetakse alati sisse eestikeelne tõlkefail. Enne vastavat koodirida on kommentaarina toodud näide tõlkefaili dünaamiliseks laadimiseks, sõltuvalt operatsioonisüsteemi lokaadiseadetest. Teisisõnu, kui operatsioonisüsteemi keeleks on määratud näiteks vene keel, siis laetakse vene keele lokaadinimega tõlkefail *(ru)*. Pane tähele, et tõlkefailide laadimisel ei kasutata faililaiendit.

Kompileeri ja käivita rakendus. Kuna me pole veel tõlkefaili loonud, siis on kõik stringid veel inglisekeelsed. Tõlkefaili loomiseks loo uus kataloog nimega *translations* ning lisa projektifaili (*.pro) järgmine rida:

```
TRANSLATIONS = translations/et.ts
```

Antud näiteks tekitame eestikeelse tõlkefaili *et.ts* eraldi kataloogi nimega *translations*. Genereerimine antud asukohta tühja tõlkefaili, kasutades abimeest nimega *lupdate*, mis on leitav menüüst nõnda: *Tools -> External -> Linguist -> Update Translations (lupdate)*

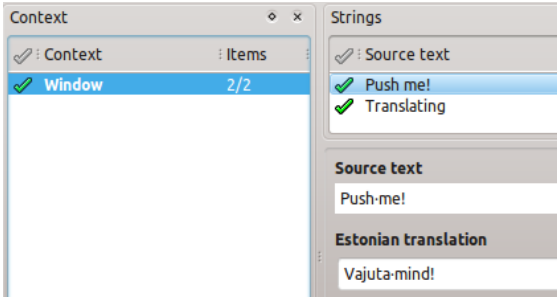
Ilma Qt Creator programmita oleks vastav käsurida selline:

```
lupdate projekt.pro -ts translations/et.ts
```

Esimeseks argumendiks anname ette projektifaili ning teiseks (võtme *ts* järel) genereeritava keelefaili asukoha.

TS-failid on XML-kujul inimesele arusaadavad tõlgete lähtefailid. Tõlkefailiga toimetamiseks tuleb see kõigepealt lisada projekti failipuusse. Selleks ava failipuus projekti nimel paremakliki menüü ning vali *Add Existing Files...*. Seejärel ava paremakliki menüü juba

lisatud tõlkefailil ning vali *Open With -> Qt Linguist*. Tõlkefail avaneb Qt Linguist tõlkimisprogrammis. Nagu näha joonisel 14, siis vasakul ääres on loetletud kontekstid ning keskel on loetletud kõik stringid, mis valitud konteksti alla kuuluvad. Selekteeri esimene string ning sisesta lahtrisse *Estonian translation* eestikeelne tõlge. Tee nõnda kõikide stringidega, kuni iga stringi ees on roheline linnuke. Salvesta tõlkefail (*Ctrl+S* või *File -> Save*).




*Joonis 15. Stringide tõlkimine rakenduses Qt Linguist.*

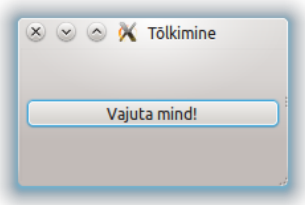
Seejärel kompileerime tõlkefaili rakendusele sobivale kompaktsele binaarkujule, kasutades abimeest *lrelease*, mille leiab menüüst järgnevalt: *Tools -> External -> Linguist -> Release Translations (lrelease)*

Vastav käsurida on:

```
lrelease translations/*.ts
```

Argumendiks anname siin ette XML-kujul keelefaili asukoha.

Kopeeri tõlkefailide kataloog samasse kataloogi rakenduse käivitusfailiga. Qt Creator tekitab käivitusfaili jaoks eraldi projektikataloogi, mille nimene lisab *build-desktop* ja selle leiab "õige" projektikataloogi kõrvalt. Kui siiamaani on läinud kõik hästi, siis proovi uuesti käivitada näiterakendus. Rakendus peaks olema nüüd tõlgitud eestikeelseks ja seda illustreerib joonis 15. Juhul kui rakendus ei ole muutunud eesti keelseks, tuleks üle vaadata projekti seaded, et käivituskataloog oleks sama kataloog, kus asub rakenduse käivitusfail. Vali Qt Creator vasakust servast *Projects*, edasi *Run Settings* ning määra õige käivituskataloog.




*Joonis 16. Tõlgete näiterakendus.*

**Ülesanded**

1. Muuda "Vajuta mind!" nupu eesti keelne tõlge tekstiks "Ära vajuta mind!".
2. Lisa juurde tõlge vabalt valitud keeles (võib olla ka väljamõeldud keel) ning pane rakendus seda tõlget kasutama.

## 9. Laadid

Vidinate peatükis sai mainitud, et Qt vidinad näevad erinevates operatsioonisüsteemides vaikimisi pisut erinevad välja, kuid mõõdud ja paigutus on samad. Käesolevas peatükis räägime kuidas Qt vidinatele anda omapärast ning kõikides operatsioonisüsteemides identset väljanägemist. Qt vidinate välimust on võimalik määrata, sarnaselt HTML *(HyperText Markup Language)* elementidele, võimalusterohkete CSS laadilehtede kaudu. Samuti nagu tõlgete puhul, ei nõua ka laadilehtede muutmine rakenduse uuesti kompileerimist. Allpool on toodud CSS laadilehtede kasutamist tutvustava näiterakenduse kood.

*main.cpp*:

```
#include <QtGui>
#include "mainWindow.h"

void loadCSS()
{
   QFile file("style.css");
   if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
      return;
   QTextStream in(&file);
   QString style;
   while(!in.atEnd()) {
      style += in.readLine();
   }
   qApp->setStyleSheet(style);
}
int main(int argc, char *argv[])
{
```

```
    QApplication app(argc, argv);
    loadCSS();
    Window *mainWindow = new Window;
    mainWindow->show();
    return app.exec();
}
```

*mainWindow.h*:

```
#include <QtGui>

class Window: public QMainWindow
{
    private:
    Q_OBJECT
    QPushButton *button;
    public:
    Window();
};
```

*mainWindow.cpp*:

```
#include "mainWindow.h"

Window::Window(): QMainWindow()
{
    QWidget *widget = new QWidget;
    QVBoxLayout *layout = new QVBoxLayout;
    button = new QPushButton(QString::fromUtf8("Stiilne Nupp"));
    button->setObjectName("closeButton");
    layout->addWidget(button);
    widget->setLayout(layout);
    setCentralWidget(widget);
    setWindowTitle(QString::fromUtf8("Laadid"));

    connect(button, SIGNAL(clicked()), qApp, SLOT(quit()));
}
```

*style.css*:

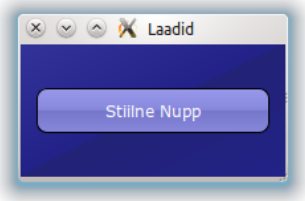
```
QMainWindow {
    background-color: QLinearGradient(x1: 0, y1: 0, x2: 1, y2: 1,
        stop: 0 #339, stop: 0.49 #228, stop: 0.5 #227, stop: 1 #228);
}
QPushButton#closeButton {
    background-color: QLinearGradient(x1: 0, y1: 0, x2: 0, y2: 1,
        stop: 0 #88d, stop: 0.49 #77c, stop: 0.5 #66b, stop: 1 #77c);
    font-family: 'Verdana', sans-serif;
    color: white;
    font-size: 12px;
    border-width: 1px;
    border-color: black;
    border-style: solid;
    border-radius: 8px;
    padding: 8px;
    margin: 8px;
}
QPushButton#closeButton:hover {
    background-color: QLinearGradient(x1: 0, y1: 0, x2: 0, y2: 1,
        stop: 0 #99e, stop: 0.49 #88d, stop: 0.5 #77c, stop: 1 #88d);
}
```

```
QPushButton#closeButton:pressed {
  border-width: 2px;
}
```

(Smith, 2009)

CSS faili saab luua *Add new... -> General -> Text File* kaudu. Kui eelnevas peatükis kirjutasime tõlkemooduliga seonduva koodi *main.cpp* faili, siis nüüd teeme sama CSS laadilehe lugemisega seonduva koodiga. CSS laadilehe sisu omistatakse QString objektile ning see määratakse tervele rakendusele *setStyleSheet* meetodiga qApp viida kaudu. Kuna *setStyleSheet* meetod pärineb *QWidget* klassist, siis on samuti võimalik mingit laadilehte määrata ainult mingile kindlale vidinale (ja tema alamvidinatele). Laade saab vidinatele määrata vidina klassi nime või objekti nime järgi. Näiterakenduses on *QMainWindow* vidinale määratud laad klassi nime järgi. Kui rakenduses oleks veel mõni *QMainWindow* vidin, siis sellele määrataks täpselt sama laad. *QPushButton* nupuvidinale, tekstiga "Stiilne Nupp", on omistatud objekti nimi "*closeButton*" ja selle järgi määratud ka laad. Nupule määratud laadil on kasutatud ka pseudoklasse, mis määravad ära laadi vastavalt hiirekursori asendile nupu suhtes. HTML-i juurde kuuluva CSS-iga on siin üks erinevus. Nimelt, kui tahta määrata laadi, mil hiirekursor on vajutatud alla, olles samal ajal nupu kohal, siis tuleb kasutada *:active* asemel *:pressed* pseudoklassi.

Teiseks erinevuseks nn "harilike" CSS laadilehtedega, on see, et mõned CSS funktsioonid on asendatud Qt raamistiku klassidega. Näiterakenduses on kasutatud laadilehes *linear-gradient* asemel *QLinearGradient* klassi, mis võimaldab luua täpselt samasuguse tulemuse, kui originaal funktsioon. *QLinearGradient* klassi kasutatakse laadilehtedes pisut teistmoodi, kui rakenduse koodis. Esimesel juhul, x1 ja y1 tähistavad gradiendi alguspunkti ning x2 ja y2 lõpp-punkti. Mõlemad paarid saavad omada ujukoma väärtusi vahemikus 0 kuni 1. Näiteks *x1=0, y1=0* tähistab vidina vasakut ülemist nurka ja *x2=1, y2=1* tähistab vidina paremat alumist nurka. Algus ja lõpp-punkti vahele või kohale saab määrata kindlale positsioonile ja kindla värviga n arv peatuspunkte. Punktide vahele jääva ala värvus genereeritakse üleminekuna esimese punkti toonist teise punkti tooniks. *QLinearGradient* klassiga rakenduse koodis puutume kokku järgmises peatükis. Enne rakenduse käivitamist kopeeri *style.css* fail ka käivitusfaili kataloogi. Antud CSS laadilehega näiterakendust on kujutatud joonisel 17.




*Joonis 17. Laadide näiterakendus.*

**Ülesanded**

1. Muuda akna ja nupu taustad punakaks.
2. Lisa nupu alla tekstisisestuskastike *QTextEdit*. Muuda CSS laadilehega selle kirjastiili, tausta ning tee nurgad nupuga sama ümmarguseks.

## 10. Graafika

Käesolevas õppematerjalis on viimaseks juhendiks peatükk, mis õpetab, kuidas näidata Qt rakendustes pilte ning joonistada värvilisi kujundeid. Qt raamistikus on joonistamiseks loodud *QPainter* klass, mis pakub vahendeid erinevate kujundite joonistamiseks, lihtsatest joontest kuni keeruliste hulknurkadeni. Joonistada võib nii kahe-, kui ka kolmemõõtmelises koordinaatsüsteemis. Vidinate koordinaatsüsteemi nullpunkt asub üleval vasakus nurgas. Järgnevas näidisrakenduses tegeleme kahemõõtmelise süsteemiga, mis on alustamiseks lihtsam. Enne, kui asud rakenduse kirjutamise juurde, otsi välja mõni pildifail. Pildi soovituslik suurus selles rakenduses on 250x250 pikselit.

*mainWindow.h*:

```
#include <QtGui>

class Window: public QMainWindow
{
   public:
   Window();
};
```

*mainWindow.cpp*:

```
#include "mainWindow.h"
#include "painting.h"

Window::Window(): QMainWindow()
```

```
{
    QWidget *widget = new QWidget;
    QHBoxLayout *layout = new QHBoxLayout;
    Painting *painting = new Painting;
    painting->setFixedSize(250, 250);
    QLabel *imageLabel = new QLabel;
    imageLabel->setPixmap(QPixmap("images/NyanCat.png"));
    layout->addWidget(imageLabel);
    layout->addWidget(painting);
    widget->setLayout(layout);
    setCentralWidget(widget);
    setWindowTitle(QString::fromUtf8("Graafika"));
}
```

*painting.h*:

```
#include <QtGui>
#include <iostream>

class Painting: public QWidget
{
    protected:
    void paintEvent(QPaintEvent *);
};
```

*painting.cpp*:

```
#include "painting.h"

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

    painter.setPen(Qt::NoPen);
    painter.setBrush(Qt::Dense7Pattern);
    painter.drawRect(0, 0, 250, 250);

    painter.setPen(QPen(Qt::blue, 8, Qt::SolidLine));
    painter.setBrush(Qt::green);
    painter.drawEllipse(35, 35, 180, 180);

    painter.setPen(QPen(Qt::black, 2, Qt::DashLine));
    painter.drawLine(50, 130, 200, 130);

    painter.setPen(QPen(Qt::black, 1, Qt::SolidLine));
    painter.setBrush(Qt::white);
    QPointF baseline(80, 120);
    QFont font("Georgia", 64);
    QPainterPath path;
    path.addText(baseline, font, "Qt");
    painter.drawPath(path);

    QLinearGradient gradient(0, 150, 0, 190);
    gradient.setColorAt(0.01, Qt::blue);
    gradient.setColorAt(0.5, Qt::black);
    gradient.setColorAt(0.99, Qt::white);
    painter.fillRect(90, 150, 70, 40, gradient);
}
```

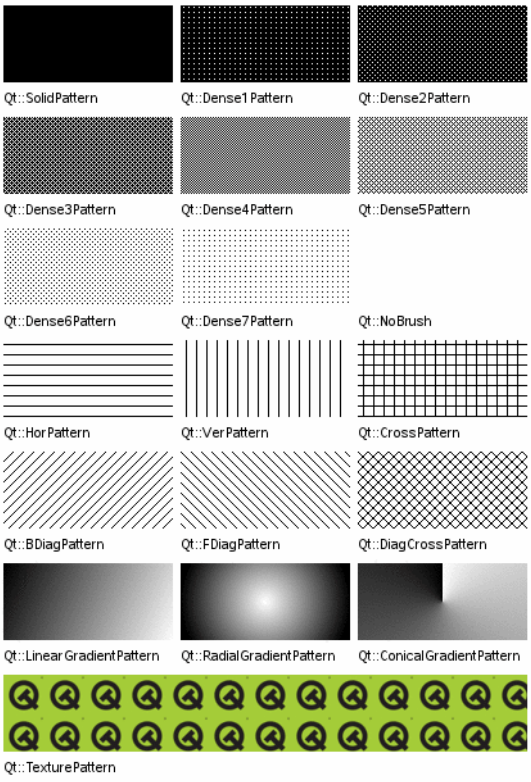
Rakenduses on eraldi failidesse jagatud klassid *Window* ja *Painting*. *QHBoxLayout* paigutushalduriga lisatakse rakenduse peaakna vasakusse serva pilt ning paremasse serva

*QPainter* klassi meetoditega tehtud joonistus. Pildi kuvamiseks Qt rakenduses, luuakse *QLabel* vidin, millele määratakse *QPixmap* objekt, kuhu on laetud pildifail. Joonistusvidin *Painter* pärineb *QWidget* klassist, kus on defineeritud meetod *paintEvent*, mis joonistab vidina uuesti ühel järgmistest juhtudest:

- *repaint()* või *update()* meetodi väljakutsumisel
- vidin muutub nähtavaks peale peidus olemist
- paljud teised juhud

Kui siiani oleme peaakna või dialoogi koodi kirjutanud klassi konstruktorisse, siis antud rakenduse puhul kirjutame kõik *paintEvent* meetodi sisse. *Painter* vidina objekti loome *mainWindow.cpp* failis, kuid objekti koodi kirjutame *painter.cpp* faili. Määrame joonistusvidina püsivaks suuruseks *setFixedSize* meetodiga 250x250 pikselit nagu pildilgi. Selles failis vaatleme ristküliku, ellipsi, joone ning teksti joonistamist. Kasutame ka erinevaid mustreid ja värvilahendusi. Kujundi värvid määrab ära pliiats ehk *QPen*, millega joonistatakse piirjooned ning pintsel *QBrush*, millega värvitakse kujundi sisu. Kujundi sisuks võib olla mingi värv, värvide üleminek, muster või tekstuur. *QPen* ja *QBrush* tuleb määrata enne kujundi joonistamist, vastavalt *setPen* ja *setBrush* meetodeid rakendades. Kui neid muuta ei soovi, pole vaja ka enne kujundi joonistamist uuesti määrata.

Loome *paintEvent* meetodi sees kõigepealt *Painter* objekti, mille konstruktoris ütleme, et vidinaks, millele joonistame, on tema ise. Kõige esimese asjana joonistame meetodiga *drawRect* poole akna suuruse ristküliku, mis jääb kõikide teiste kujundite alla taustaks. Iga järgnev kujund asetseb eelmisest kihi võrra kõrgemal. *drawRect* kaks esimest parameetrit on kujundi asukoht vidina koordinaatteljestikus. Kolmas ja neljas parameeter tähistavad kujundi laiust ja kõrgust. Selle kujundi puhul me ei kasuta piirjoonte joonistamiseks pliiatsit. Sisu värvimiseks kasutame täpimustrit *Dense7Pattern*. Kõik Qt raamistikus kasutatavad mustrid on välja toodud joonisel 18.




*Joonis 18. Qt mustrid.*

Järgmisena joonistame sinise paksu pidevjoonega ringi oma joonistusvidina keskele. Ringi ja ellipsit joonestatakse *drawEllipse* meetodiga, kus kaks esimest parameetrit on ellipsit ümbritseva kujutletava ristküliku ülemise vasaku nurga asukoht. Kolmas ja neljas parameeter on vastavalt selle sama ristküliku laius ja kõrgus. Kui määrame ümbritseva ristküliku laiuse ja kõrguse võrdseteks, saamegi ringi. Lihtsamalt öeldes, käib ellipsi joonistamine sama loogika järgi, nagu ristküliku joonistamine. Kolmandaks kujundiks on must katkendlik joon. Kuna joonel seest midagi värvida ei ole, siis pintslit me siin ei määra.

Neljandaks joonistame teksti järgi kujundeid. Kõigepealt sätime paika värvid - must pliiats ja valge pintsel. Järgmisena määrame teksti alusjoone alguspunkti ja valime sobiva kirjastiili. Teksti alusjoone alguspunkt on *QPointF*-tüüpi, kirjastiil *QFont*-tüüpi. Seejärel loome uue *QPainterPath*-tüüpi objekti, mille meetod *addText*, vastavalt etteantud alusjoonele, kirjastiilile ning tekstile, loob tekstisümbolite piirjoontest raja. Lõpuks laseme joonistusvidina meetodil *drawPath* joonistada selle raja järgi kujundi.

Viimaseks kujundiks on sini-must-valge lineaarse gradiendiga ristkülik. Pliiatsi ja pintsli asemel määrame siin vaid gradiendi *QLinearGradient*. Nagu eelmises peatükis sai mainitud,

erineb *QLinearGradient* kasutamine rakenduse koodis, selle kasutamisest CSS laadilehes. *QLinearGradient* konstruktoris määratakse gradiendi suund. Antud juhul *x1=0, y1=150* ja *x2=0, y2=190* ehk gradient kulgeb ülevalt alla, algab punktist, kus *y=150*, lõpeb punktis, kus *y=190* ning tema pikkus on seega 40 pikselit. Sellisel juhul x-väärtused ei olegi olulised, sest gradient mõjub lõpmatult kulgemisele ristuvas suunas. Gradiendi algus ja lõpp-punkti vahele lisatakse kindla värviga punkte *setColorAt* meetodiga, kus esimeseks parameetriks on punkti asukoht gradiendis ning teiseks parameetriks on gradiendi värv selles punktis. Sarnaselt CSS laadilehtedega, on ka siin gradiendi alguspunkti tähiseks 0 ning lõpp-punkti tähiseks 1. Selles näites, joonistusvidina meetod *fillRect* joonistab ristküliku määratud positsioonile ja värvib selle seest gradiendiga. Gradiendi asemel võib kasutada ka lihtsalt värve või mustreid.

Meie viimane rakendus ongi nüüd valmis saanud. Samuti nagu tõlkimise ja laadide puhul, kopeeri pildikataloog käivitusfailiga samasse kataloogi. Kui näiterakendus kompileerida ja käivitada, peaks ta välja nägema sarnane joonisel 19 kujutatuga.


*Joonis 19. Graafika näiterakendus.*

(Bodnar, 2008)

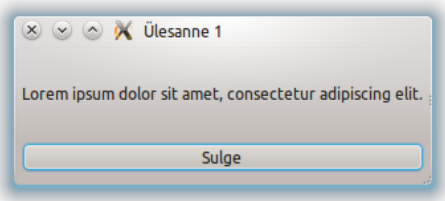
**Ülesanded**

1. Muuda sini-must-valge üleminek punane-roheline-lilla üleminekuks.
2. Lisa alla paremasse nurka üks punase tausta ja musta joonega ring. Tee selle ringi sisse oma eesnime esimese tähe järgi kollane, musta servaga kujund.

## 11. Ülesanded

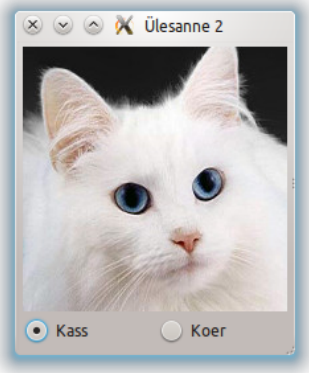
Praeguseks hetkeks peaks käesoleva õppematerjali siiamaani läbi töötanu omama algteadmisi Qt rakenduste loomisest. Neid teadmisi on nüüd võimalik kohe proovile panna. Kasutades käesolevat õppematerjali ning vajadusel materjale "Olemasoleva materjali" peatükist, on järgnevalt iseseisvaks lahendamiseks esitatud kolm ülesannet. Alustades küllaltki lihtsa ja lõpetades rohkem aega ja mõtlemist nõudva ülesandega. Lahendamise järjekord pole oluline. Ülesannete võimalikud lahendused leiab õppematerjali lõpust peatükist "Lisad".

1. Koosta rakendus, kus on aken pealkirjaga "Ülesanne nr 1" ning sisutekstiga "*Lorem ipsum dolor sit amet, consectetur adipiscing elit.*". Teksti all on nupp, millele vajutades rakendus sulgub.



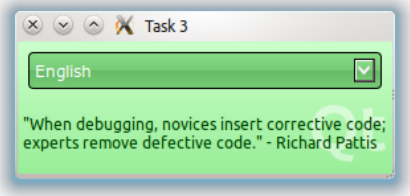

*Joonis 20. Ülesande 1 rakendus.*

2. Koosta rakendus, kus on aken pealkirjaga "Ülesanne nr 2". Akna sees asetsevad teineteise suhtes horisontaalselt kaks raadionuppu. Näiteks siltidega "kass" ja "koer". Raadionuppude kohal on vidin, kuhu laetakse kummalegi raadionupule vajutades pilt. Kui aktiivseks tehakse nupp "kass", näidatakse rakenduses kassi pilti. Kui aktiivseks tehakse nupp "koer", näidatakse koera pilti. Vaikimisi ei ole kumbagi pilt nähtav. Sobivad pildid otsib lahendaja endale ise.




*Joonis 21. Ülesande 2 rakendus.*

3. Koosta rakendus, kus on aken pealkirjaga "Task 3" ning sisutekstiga ""*When debugging, novices insert corrective code; experts remove defective code.*" – *Richard Pattis*". Sisuteksti kohal on rippmenüü, kust saab valida rakendusele keelt. Rippmenüü loetelus on inglise keel, eesti keel ja üks ülesande lahendaja poolt valitud keel. Keelt peab saama muuta ilma rakendust taaskäivitamata. Rakenduse vaikmisi keeleks on inglise keel. Muuda rippmenüü ja akna tausta välimust, kasutades CSS laadilehe võimalusi.




*Joonis 22. Ülesande 3 rakendus.*

## 12. Edasiõppimiseks

Oled läbi töötanud käesoleva õppematerjali, teinud läbi iseseisvad prooviülesanded, kuid see kõik on alles Qt raamistiku jäämäe tipp. Avastamist jagub siin veel lugematuteks õhtuteks. Et Sinu edasist Qt raamistiku avastamist veidi lihtsustada, on siia peatükki koondatud mõned soovitused, milliseid Qt raamistikku kuuluvaid mooduleid ja tehnoloogiaid edasi võiks uurida.

Viimases praktilises peatükis vaatasime, kuidas Qt rakendustes joonistada ja pilte kuvada. Igati loogiline jätk sellele oleks õppida juurde HTML sisu renderdamist. Qt rakendustes on HTML sisu näitamiseks moodul *QtWebKit*, mis põhineb vabatarkvaralisel WebKit mootoril.

Nagu üks kaasaegne veebibrauser, suudab see renderdada HTML ja XHTML *(Extensible HyperText Markup Language)* formaadis dokumente koos CSS laadilehtede ja JavaScriptiga.

Kui HTML lehtede renderdamine selge, võiks edasi uurida võrgundusest Qt raamistikus. HTTP *(Hypertext Transfer Protocol)* ja FTP *(File Transfer Protocol)* ühendusi, vajadusel kombineerituna SSL *(Secure Sockets Layer)* protokolliga, pakub *QtNetwork* moodul. *QtNetwork* mooduli abil on võimalik luua nii klientrakendusi, kui ka servereid.

Viimase asjana pakuks välja Qt Quick tehnoloogiaga tutvumise. Qt Quick on mugav vahend loomaks rakendusi mobiilsetele seadmetele. Qt Quick tehnoloogial põhinevate rakenduste kasutajaliidesed on kirjutatud QML keeles, mis on lähedane JavaScriptiga. QML keeles deklareeritakse graafilised objektid, olekumuutused ja animatsioonid. QML keeles kirjutatud objekte on võimalik kerge vaevaga ühildada Qt C++ komponentidega. Qt Quick trakenduste loomisel on abiks *Qt Quick Designer* programm, millest oli lühidalt kirjutatud ka arendustööriistade peatükis.

# Õppematerjali testimine

Kuna käesolev materjal eeldab, et selle lugeja omab algteadmisi programmeerimiskeelest C++, siis testisin õppematerjali informaatika tudengite peal, C++ kursuse raames. Kokku kestis testimise periood neli akadeemilist tundi kahe nädala jooksul. Testimise perioodi alguseks olid tudengid jõudnud samuti 4 akadeemilist tundi C++ keelega tutvuda.

Jõudsin õpilaste peal läbi teha peatükid esimesest rakendusest kuni signaalide ja pesadeni. Iga peatüki lõpus olevatest ülesannetest lahendasime paar lihtsaimat. Laadidest ja graafika kasutamisest vaid rääkisin lõpus kokkuvõtlikult. Et kogu õppematerjal koos kõikide ülesannetega läbi töötada, oleks vaja läinud hinnanguliselt vähemalt kümmet akadeemilist tundi. Õppematerjal äratas mitmes õpilases huvi Qt raamistiku vastu ja nad pidased seda põnevaks. Valdavalt sujus kõik nii nagu olin eeldanud, kuid tuli ette ka paar ootamatut probleemi, mis olid tingitud sellest, et arendasin ja testisin õppematerjali erinevate platvormide peal. Arendamiseks kasutasin Kubuntu Linux 11.10 operatsioonisüsteemi ja klassis, kus ma testisin, olid Windows 7 operatsioonisüsteemiga arvutid.

Esimeses rakenduses avastasime, et täpitähed ei toimi ka UTF-8 kodeeringus olevate *QString* objektide puhul. Suurem probleem tekkis esile tõlkimise peatükis. Rakendus ei laadinud ära tõlkefaili. Täpitähtede probleemi jätsime sel hetkel lihtsalt lahendamata. Tõlkefaili laadimist valmis rakendusse ei suutnudki me selle piiratud aja jooksul välja mõelda. Nagu programmeerimisel ikka juhtub, tuli lahendada mitmel õpilasel koodiveast tingitud probleeme. Pärast õppematerjali testimist püüdsin leida lahendusi tekkinud probleemidele Windows 7 platvormil ja vastavalt parandada käesolevat materjali.

Qt Creator programmi koodiredaktoris peab kodeeringuks olema UTF-8. Vaikimisi kasutatakse operatsioonisüsteemi kodeeringut. Linuxis on süsteemi kodeering UTF-8, kuid Windowsil mitte. Kodeeringu vahetamine lahendas täpitähtede probleemi. Tõlkefail laetakse rakendusse, kui rakendus käivitada samast kataloogist, kus ta ka asub. Selleks tuleb projekti seadetest määrata õige käivituskataloog. Muutsin vastavates peatükkides materjali nõnda, et edasised õppijad eelmainitud probleemide küüsi ei satuks. Kui oleksin esmalt ise õppematerjali põhjalikumalt Windows 7 peal läbi testinud, oleks arvatavasti need vead juba enne õpilastega läbi proovimist välja tulla.

# Kokkuvõte

Käesolev õppematerjal on loodud seminaritöö tulemusena. Seminaritöö eesmärk oli luua lihtne eesti keelne õppematerjal, mis annaks algteadmisi Qt raamistikus rakenduste loomisest, ning mis võiks huvi pakkuda paljudele programmeerimishuvilistele. Kuna Qt on väga võimalusterohke, siis õppematerjal käsitleb vaid väikest osa raamistikku kuuluvatest komponentidest.

Kõigepealt on tutvustatud olemasolevaid materjale, kust lisaks antud õppematerjalile võib veel teadmisi saada. Seejärel on lühidalt kirjeldatud Qt raamistikku üldiselt ning milliseid viise on olemas raamistikus rakenduste loomiseks. Pärast tutvustavat osa järgnevad praktilised peatükid, kus iga peatükk keskendub mingile kindlale Qt raamistiku komponendile, millele tuginedes on loodud väike näiterakendus. Välja on toodud terve näiterakenduse lähtekood, mis on algusest lõpuni lahti seletatud, ning pilt sellest, milline valmis rakendus peaks välja nägema. Iga praktilise peatüki lõpus ning pärast kõiki neid peatükke on koostatud ülesanded nuputamiseks ja uute teadmiste kinnistamiseks. Neile, kelles tärkas huvi juurde õppida, on pühendatud peatükk, kust leiab soovitusi selle kohta, missuguste Qt komponentidega järgmiseks võiks tutvuda.

Kui õppematerjali sisuosa sai valmis, testisin loodud materjali C++ kursuse raames informaatika tudengite peal. Kokku jõudsime nelja akadeemilise tunni jooksul läbi teha umbes poole õppematerjali kogumahust. Liikusime õppematerjaliga enamasti edasi sujuvalt, kuid mõnes kohas tekkis ka tõrkeid. Pärast õppematerjali esmast õpilaste peal testimist, viisin sisse parandusi, et tulevikus oleks võimalik samu probleeme vältida.

Töö autorile oli õppematerjali loomise kogemus uus. Esmalt panin paika eeldused õppematerjaliga töötamiseks, et ei peaks seletama lahti asju, mis jäävad antud teemast kõrvale ja saaks keskenduda rohkem olulisele. Püüdsin võimalikult detailselt edasi anda uut teavet, seda vajadusel piltidega varustades. Töö on oma eesmärgi täitnud, kui selle läbitöötanu on võimeline looma lihtsamaid Qt rakendusi ja tal on tekkinud huvi Qt raamistikuga edasi tegeleda.

# Kasutatud kirjandus


- Bodnar, J. (2008). *Painting in Qt4.* Viimati loetud 04.01.2011 aadressilt http://zetcode.com/tutorials/qt4tutorial/painting/
- KDevelop. (2011). *Welcome to KDevelop.org.* Viimati loetud 17.12.2011 aadressilt: http://www.kdevelop.org/
- Nokia Corporation. (2011). *Getting Started Programming with Qt.* Viimati loetud 03.01.2012 aadressilt: http://doc.qt.nokia.com/4.8/
- QDevelop. (2011). *Project Home.* Viimati loetud 18.12.2011 aadressilt: http://code.google.com/p/qdevelop/
- Smith, D. (2009). *Qt Stylesheets Tutorial.* Viimati loetud 03.01.2012 aadressilt: http://thesmithfam.org/blog/2009/09/10/qt-stylesheets-tutorial/
- wysota. (2008). *Qstring vs string.* Viimati loetud 20.12.2011 aadressilt: http://www.qtcentre.org/threads/15700-Qstring-vs-string/

# Lisad

## 1. Ülesannete võimalikud lahendused

### Ülesanne 1:

*main.cpp*:

```
#include <QtGui>
#include "mainWindow.h"

int main(int argc, char *argv[])
{
   QApplication app(argc, argv);
   Window *mainWindow = new Window;
   mainWindow->show();
   return app.exec();
}
```

*mainWindow.h*:

```
#include <QtGui>

class Window: public QMainWindow
{
   private:
   Q_OBJECT
   public:
   Window();
};
```

*mainWindow.cpp*:

```
#include "mainWindow.h"

Window::Window(): QMainWindow()
{
   QWidget *widget = new QWidget;
   QVBoxLayout *layout = new QVBoxLayout;
   QLabel *label = new QLabel(QString::fromUtf8(
      "Lorem ipsum dolor sit amet, consectetur adipiscing elit."));
   QPushButton *button = new QPushButton(QString::fromUtf8("Sulge"));
   layout->addWidget(label);
   layout->addWidget(button);
   widget->setLayout(layout);
   setCentralWidget(widget);
   setWindowTitle(QString::fromUtf8("Ülesanne 1"));

   connect(button, SIGNAL(clicked()), qApp, SLOT(quit()));
}
```

### Ülesanne 2:

*main.cpp*:

```
#include <QtGui>
#include "mainWindow.h"

int main(int argc, char *argv[])
{
   QApplication app(argc, argv);
   Window *mainWindow = new Window;
   mainWindow->show();
   return app.exec();
}
```

*mainWindow.h*:

```
#include <QtGui>

class Window: public QMainWindow
{
   private:
   Q_OBJECT
   QLabel *imageLabel;
   private slots:
   void changePicture(QString picture);
   public:
   Window();
};
```

*mainWindow.cpp*:

```
#include "mainWindow.h"
#include <iostream>

Window::Window(): QMainWindow()
{
   QWidget *widget = new QWidget;
   QVBoxLayout *layout = new QVBoxLayout;

   imageLabel = new QLabel;
   imageLabel->setFixedSize(200, 200);

   QHBoxLayout *radioButtonLayout = new QHBoxLayout;
   QButtonGroup *radioButtonGroup = new QButtonGroup;
   QRadioButton *radioButtonCat = new QRadioButton(QString::fromUtf8("Kass"));
   QRadioButton *radioButtonDog = new QRadioButton(QString::fromUtf8("Koer"));
   radioButtonGroup->addButton(radioButtonCat);
   radioButtonGroup->addButton(radioButtonDog);
   radioButtonLayout->addWidget(radioButtonCat);
   radioButtonLayout->addWidget(radioButtonDog);

   layout->addWidget(imageLabel);
   layout->addLayout(radioButtonLayout);
   widget->setLayout(layout);
   setCentralWidget(widget);
   setWindowTitle(QString::fromUtf8("Ülesanne 2"));

   QSignalMapper *signalMapper = new QSignalMapper(this);
   signalMapper->setMapping(radioButtonCat, QString("cat.jpg"));
   signalMapper->setMapping(radioButtonDog, QString("dog.jpg"));

   connect(radioButtonCat, SIGNAL(clicked()), signalMapper, SLOT(map()));
   connect(radioButtonDog, SIGNAL(clicked()), signalMapper, SLOT(map()));

   connect(signalMapper, SIGNAL(mapped(QString)), this, SLOT(changePicture(QString)));
}
```

```
void Window::changePicture(QString picture)
{
   imageLabel->setPixmap(QPixmap("pictures/" + picture));
}
```

### Ülesanne 3:

*main.cpp*:

```
#include <QtGui>
#include "mainWindow.h"

void loadCSS()
{
   QFile file("style.css");
   if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
      return;
   QTextStream in(&file);
   QString style;
   while(!in.atEnd()) {
      style += in.readLine();
   }
   qApp->setStyleSheet(style);
}
int main(int argc, char *argv[])
{
   QApplication app(argc, argv);
   loadCSS();
   Window *mainWindow = new Window;
   mainWindow->show();
   return app.exec();
}
```

*mainWindow.h*:

```
#include <QtGui>

class Window: public QMainWindow
{
   private:
   Q_OBJECT
   QTranslator translator;
   QComboBox *comboBox;
   QLabel *label;
   private slots:
   void changeLanguage();
   public:
   Window();
   void translate();
};
```

*mainWindow.cpp*:

```
#include "mainWindow.h"

Window::Window(): QMainWindow()
{
   QWidget *widget = new QWidget;
   QVBoxLayout *layout = new QVBoxLayout;
```

```
    comboBox = new QComboBox;
    comboBox->addItem(QString::fromUtf8("English"), "en");
    comboBox->addItem(QString::fromUtf8("Eesti"), "et");
    comboBox->addItem(QString::fromUtf8("Suomi"), "fi");

    label = new QLabel();

    layout->addWidget(comboBox);
    layout->addWidget(label);
    widget->setLayout(layout);
    setCentralWidget(widget);
    translate();

    connect(comboBox, SIGNAL(currentIndexChanged(int)), this, SLOT(changeLanguage()));
}
void Window::changeLanguage()
{
    qApp->removeTranslator(&translator);
    if(translator.load("translations/" + comboBox->itemData(comboBox->currentIndex()).toString()))
        qApp->installTranslator(&translator);
    translate();
}
void Window::translate() {
    setWindowTitle(tr("Task 3"));
    label->setText(tr(
        "\"When debugging, novices insert corrective code;\n"
        "experts remove defective code.\" - Richard Pattis"));
}
```