Практикум на ЭВМ
Qt. Введение

Баев А.Ж.

Казахстанский филиал МГУ

28 ноября 2018

Установка

Делайте просто:

```
sudo apt install build-essential

sudo apt install qt5-default
```

Когда-нибудь потом поставите:

```
sudo apt install qtcreator
```

Читайте официальный сайт (там есть важные дополнения):
`https://wiki.qt.io/Install_Qt_5_on_Ubuntu`

Использование

Создаем директорию и файл проекта

```
mkdir Qgame
cd Qgame
vim Qgame.pro
```

В файле проекта `Qgame.pro` описываем:

```
SOURCES = main.cpp
CONFIG += console
```

Читать про файлы проекта здесь:
`http://doc.qt.io/qt-5/qmake-project-files.html`

Минимальная программа

```
#include <iostream>
int main() {
    std::cout << "IN␣QT" << std::endl;
    return 0;
}
```

Жмем

```
qmake
```

В первый раз получим сообщение

```
Info: creating stash file /home/alen/project/.qmake.stash
```

В директории появится Makefile. На практике понадобится только 2 правила:

```
make
make clean
```

Запуск

```
./Qgame
```

Минимальная программа с Graphic User Interface

В файле проекта Qgame.pro описываем:

```
SOURCES = main.cpp
CONFIG += console
QT += widgets
```

```
#include <QApplication>
#include <QLabel>

int main(int argc, char **argv) {
    QApplication app(argc, argv);
    QLabel label("Hello␣Nikita!");
    label.resize(200, 200);
    label.show();
    return app.exec();
}
```

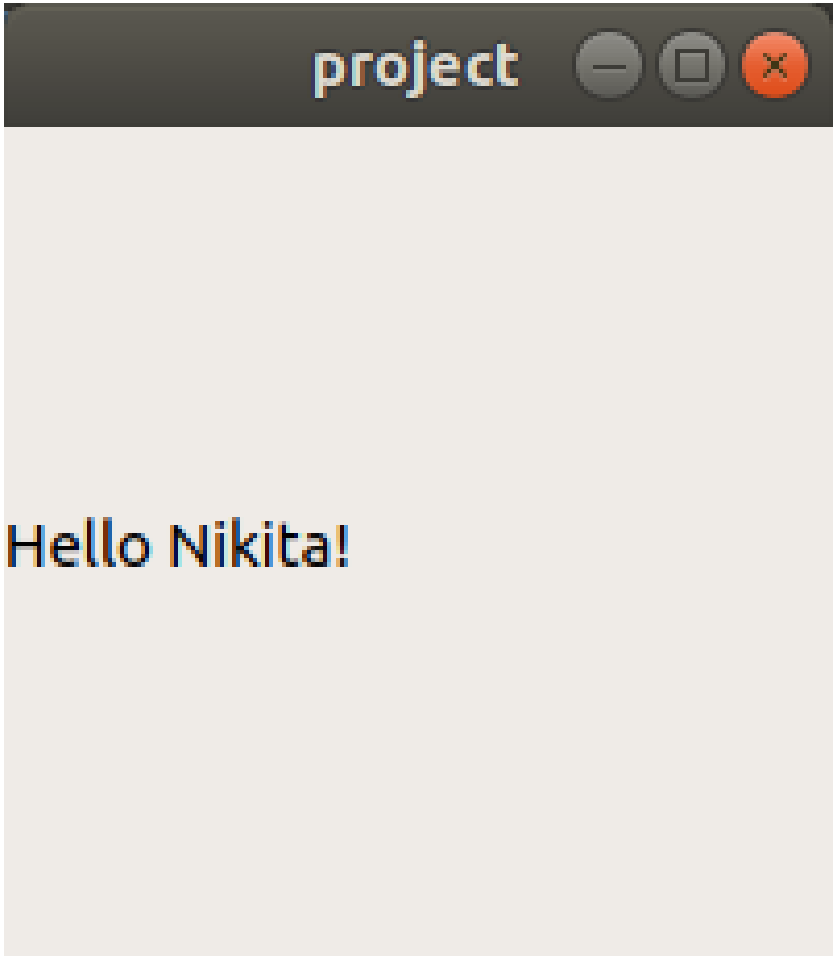
Жмем (каждый раз при изменении pro файла)

```
qmake
```

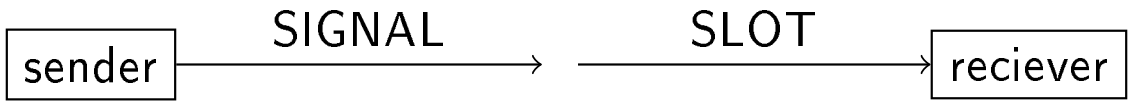
Сборка и запуск

```
make && ./Qgame
```

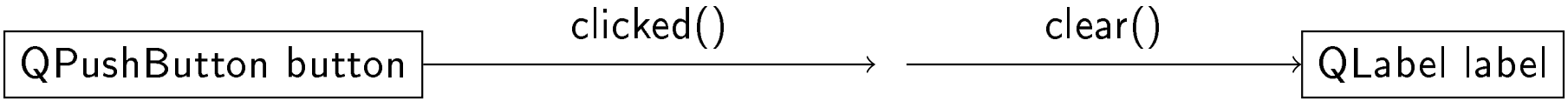
Минимальная программа с Graphic User Interface



Обработка событий



Например, по нажатию кнопки (сигнал clicked()), убрать текст на лейбле (слот clear()).



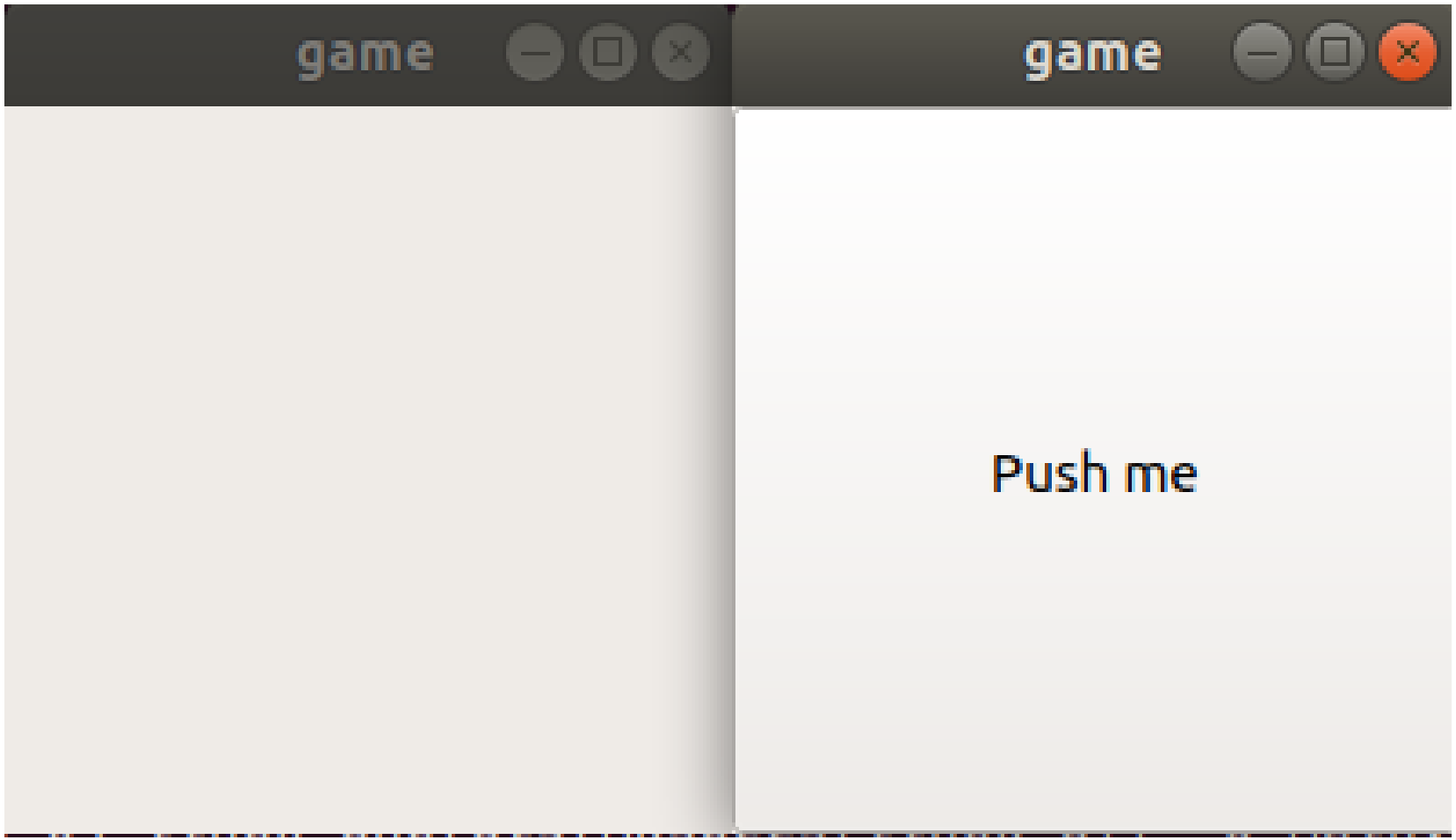
Сигнал — прототип метода класса (реализации нет, цель — зафиксировать типы и количество аргументов).
Слоты — это метод класса (прототип должен соответствовать сигналу).

Обработка событий

```
#include <QApplication>
#include <QLabel>
#include <QPushButton>

int main(int argc, char **argv) {
    QApplication app(argc, argv);
    QLabel label("Hello␣Nikita!");
    label.resize(200, 200);
    label.show();

    QPushButton button("Push␣me");
    button.resize(200, 200);
    button.show();

    QObject::connect(&button, SIGNAL(clicked()),
                     &label, SLOT(clear()));
    return app.exec();
}
```

Обработка событий

При нажатии на кнопку в одном окне, во втором пропадает текст.



Единое окно

Чтобы расположить несколько виджетов в едином окне, необходимо указывать родителя виджета (не путать с родителем C++ класса!!!). Дочерний виджет наследует доступные свойства родительского виджета и отображается как часть родительского виджета.

В QT предусмотрен класс по умолчанию для общей подложки — QMainWindow.

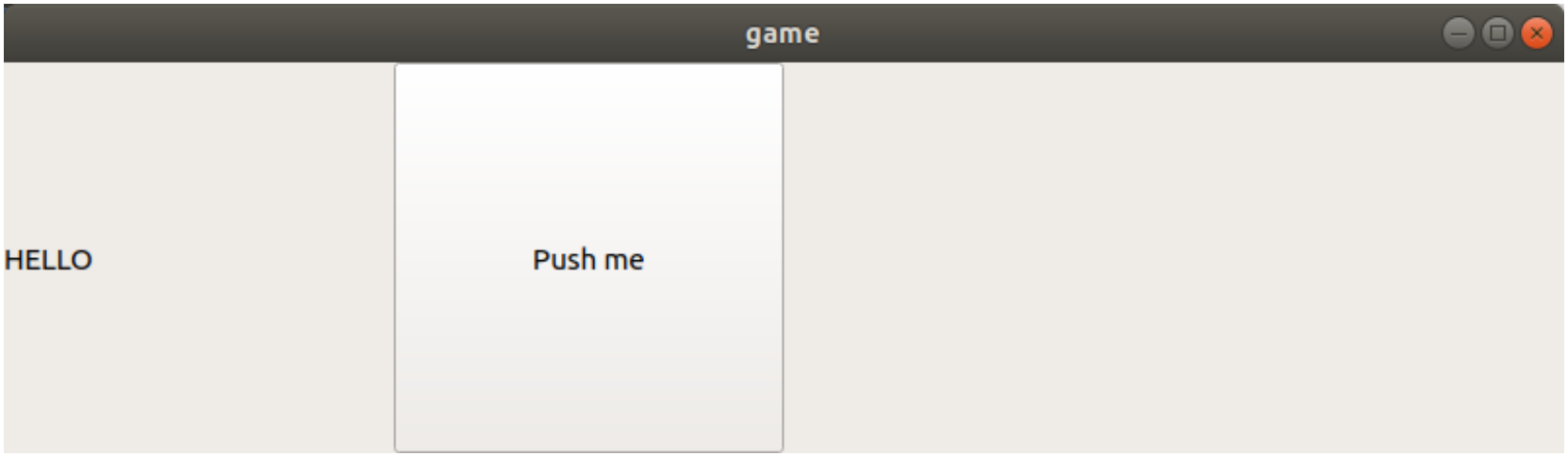
Читать — классы компоновки
`http://doc.crossplatform.ru/qt/4.6.x/layout.html`

Единое окно

```
#include <QLabel>
#include <QPushButton>
#include <QMainWindow>
int main(int argc, char **argv) {
  QApplication app(argc, argv);
  QMainWindow window;
  window.resize(800, 200);
  window.show();

  QLabel label("HELLO", &window);
  label.setGeometry(0, 0, 200, 200);

  QPushButton button("Push␣me", &window);
  button.setGeometry(200, 0, 200, 200);

  QObject::connect(&button, SIGNAL(clicked()),
                   &label, SLOT(clear()));
  return app.exec();
}
```

Единое окно

При нажатии на кнопку пропадает текст.



Нужен дополнительный функционал? Наследуемся!

Допустим, мы хотим добавить более сложную логику. Например, по нажатию на кнопку менять текст HELLO и WOLRD. Для этого необходимо обучить либо метке (QLabel), либо кнопку (QPushButton) делать более сложное действие, но при этом оставить уже существующий функционал. Что это напоминает?

Наследование!

Будем наследовать QLabel.

Нужен дополнительный функционал? Наследуемся!

Файл проекта:

```
SOURCES += main.cpp mylabel.cpp
HEADERS += mylabel.h
QT += widgets
```

Нужен дополнительный функционал? Наследуемся!

Класс усовершенствованной метки (mylabel.h):

```
#ifndef MYLABEL_H
#define MYLABEL_H

#include <QLabel>
#include <QWidget>
#include <QString>

class MyLabel : public QLabel {
  Q_OBJECT
  int counter;

public:
  MyLabel(QString text, QWidget *parent = NULL);

public slots:
  void changeText();
};

#endif
```

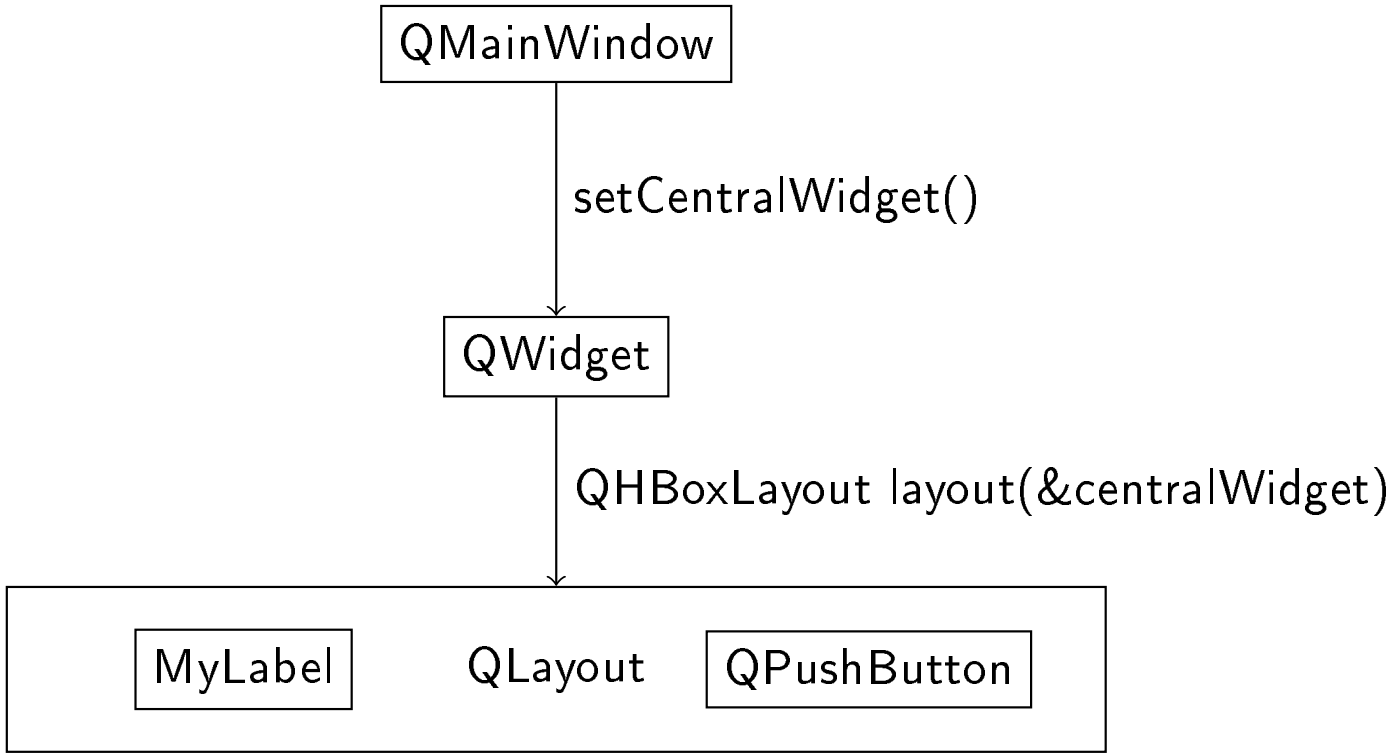
Нужен дополнительный функционал? Наследуемся!

Класс усовершенствованной метки (mylabel.cpp):

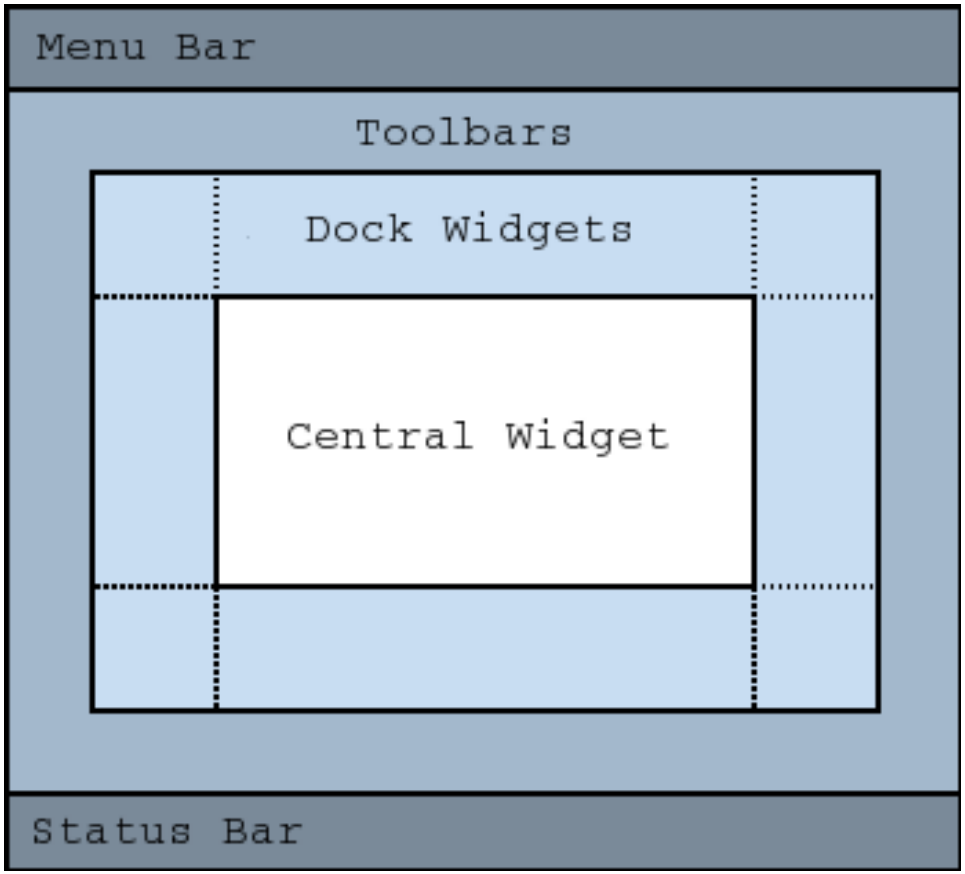
```
#include "mylabel.h"

MyLabel::MyLabel(QString text, QWidget *parent)
        :QLabel(text, parent) {
  counter = 0;
}

void MyLabel::changeText() {
  if (counter % 2 == 0) {
    setText("Tom");
  } else {
    setText("Jerry");
  }
  counter++;
}
```

Наводим frontend марафет

Как расположить все виджеты равномерно по главному виджету? С помощью центрального виджета и сеток для выравнивания.



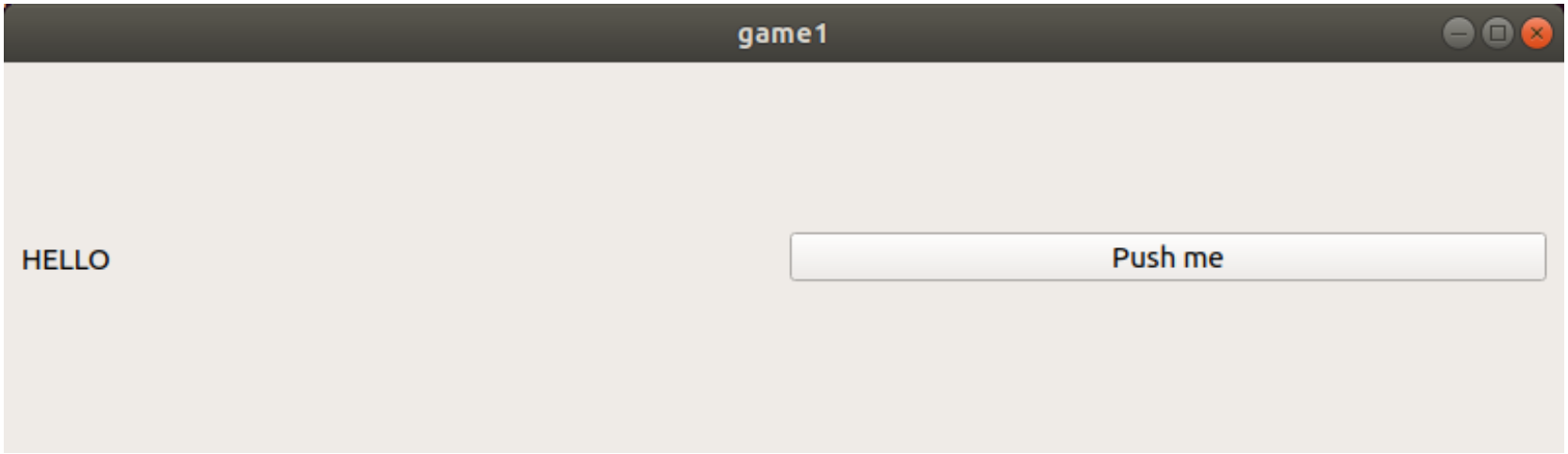
Наводим frontend марафет



Наводим frontend марафет

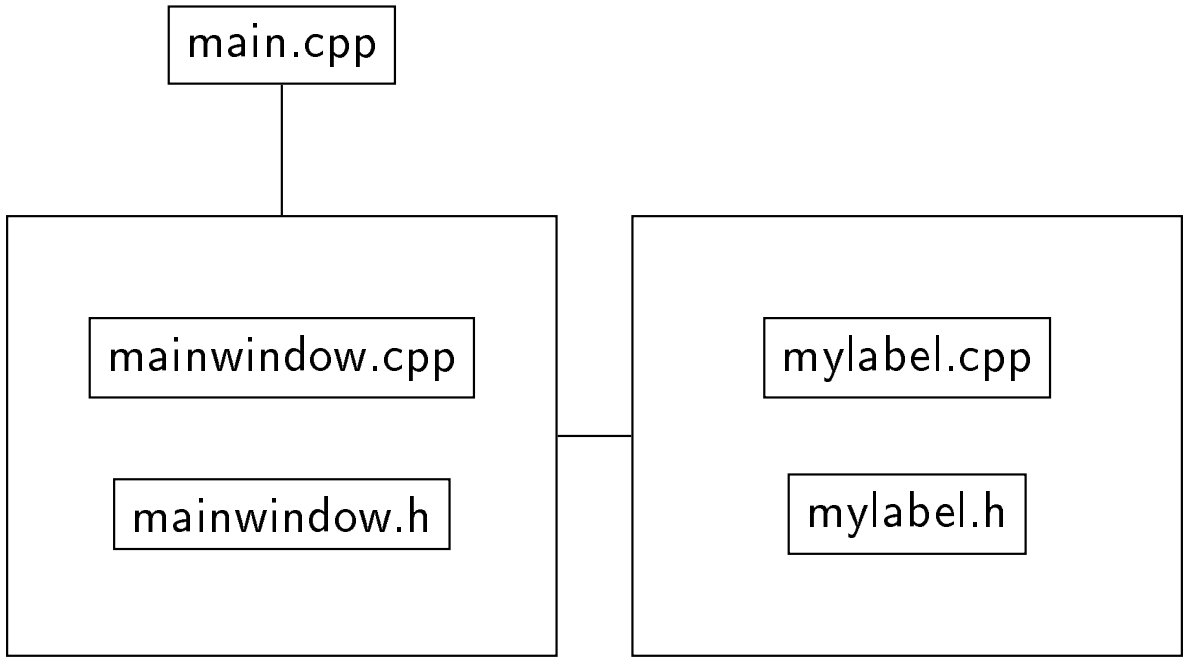
```
QMainWindow window;
window.resize(800, 200);
window.show();

QWidget centralWidget(&window);
window.setCentralWidget(&centralWidget);

MyLabel label("HELLO", &window);

QPushButton button("Push␣me", &window);

QHBoxLayout layout(&centralWidget);
layout.addWidget(&label);
layout.addWidget(&button);
```

Наводим frontend марафет



Наводим backend марафет

Как расположить код так, чтобы main.cpp был минималистичным? Отнаследуемся!



Заодно решим проблему размещения логики для взаимодействия элементов — MainWindow : public QMainWindow.

Наводим backend марафет

main.cpp

```
#include <QApplication>

#include "mainwindow.h"

int main(int argc, char **argv) {
  QApplication app(argc, argv);

  MainWindow window;
  window.show();

  return app.exec();
}
```

Наводим backend марафет

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QWidget>
#include <QHBoxLayout>
#include <QPushButton>
#include "mylabel.h"

class MainWindow : public QMainWindow {
  Q_OBJECT
  QWidget *centralWidget;
  QHBoxLayout *layout;
  MyLabel *label;
  QPushButton *button;
public:
  MainWindow(QWidget *parent = NULL);
  ~MainWindow();
};
#endif
```

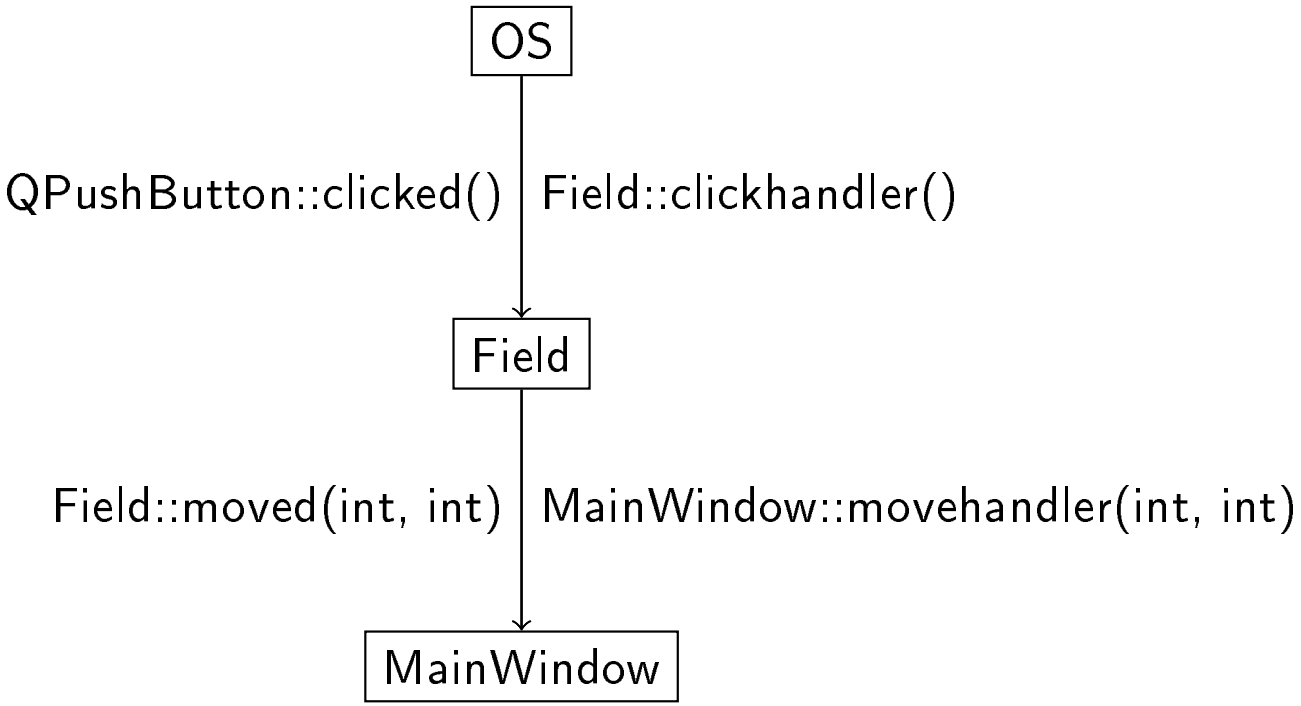
Наводим backend марафет

mainwindow.cpp

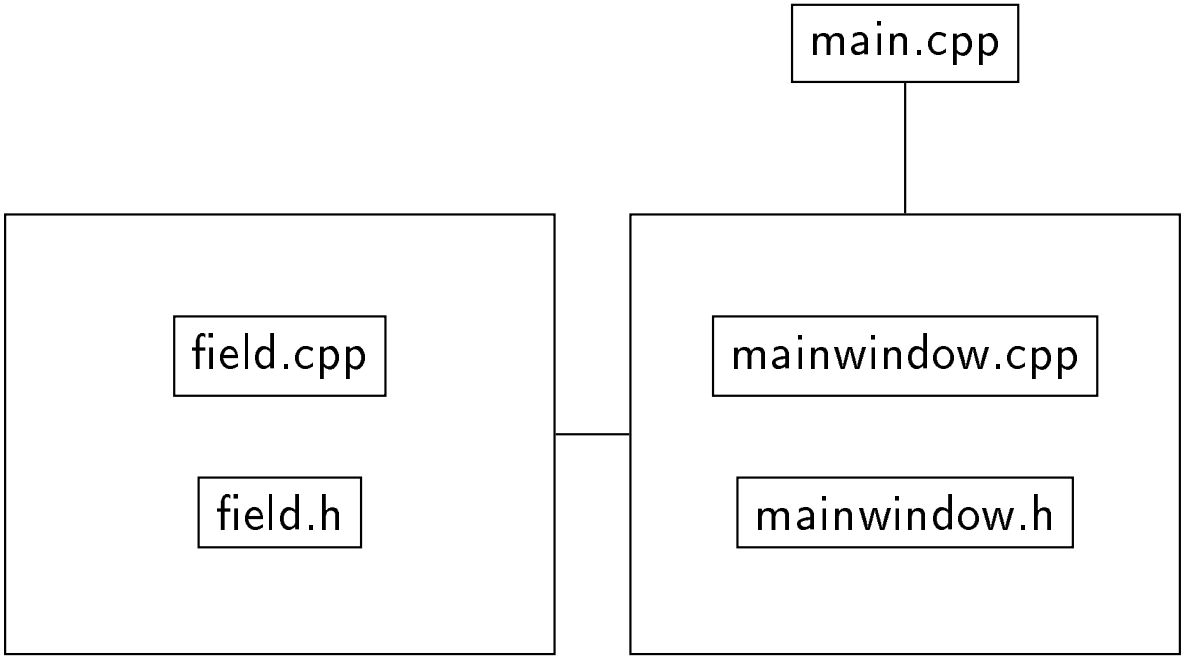
```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent): QMainWindow(parent) {
  resize(800, 200);
  centralWidget = new QWidget(this);
  setCentralWidget(centralWidget);

  layout = new QHBoxLayout(centralWidget);
  label = new MyLabel("HELLO", this);
  button = new QPushButton("Push␣me", this);

  layout->addWidget(label);
  layout->addWidget(button);
  QObject::connect(button, SIGNAL(clicked()),
                   label, SLOT(changeText()));
}

MainWindow::~MainWindow() {
  delete button;
  delete label;
```

Задача морской бой

Много кнопок. При нажатии на кнопку, необходимо определять какая именно нажата. Плохой вариант — для каждой кнопки написать свой слот основного виджета. Хороший вариант — добавить идентификатор и параметризованный сигнал в кнопку, который будет отправляться в параметризованный слот основного виджета.



Задача морской бой



Задача морской бой

mainwindow.h

```
SOURCES += main.cpp mainwindow.cpp field.cpp
HEADERS += mainwindow.h field.h
QT += widgets
```

Задача морской бой

mainwindow.h

```
#include <QMainWindow>
#include <QWidget>
#include <QGridLayout>
#include <QPushButton>
#include <QVector>
#include "field.h"

class MainWindow : public QMainWindow {
  Q_OBJECT
  QWidget *centralWidget;
  QGridLayout *layout;
  QVector< QVector<Field *> > fields;
public:
  MainWindow(QWidget *parent = NULL);
  ~MainWindow();
public slots:
  void movehandler(int row, int col);
};
```

Задача морской бой

mainwindow.cpp

```
#include "mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
            QMainWindow(parent) {
  resize(500, 500);
  centralWidget = new QWidget(this);
  setCentralWidget(centralWidget);
  layout = new QGridLayout(centralWidget);
  int size = 5;
  fields.resize(size);
  for (int i = 0; i < size; ++i) {
    fields[i].resize(size);
    for (int j = 0; j < size; ++j) {
      Field *newfield = new Field(i, j, this);
      newfield->setText("*");
      fields[i][j] = newfield;
      layout->addWidget(newfield, i, j);
      connect(newfield, SIGNAL(moved(int, int)),
              this, SLOT(movehandler(int, int)));
    }
  }
```

Задача морской бой

mainwindow.cpp

```
MainWindow::~MainWindow() {
  for (int i = 0; i < fields.size(); ++i) {
    for (int j = 0; j < fields.size(); ++j) {
      delete fields[i][j];
    }
    fields[i].clear();
  }
  fields.clear();
  delete layout;
  delete centralWidget;
}

void MainWindow::movehandler(int row, int col) {
  qDebug() << row << col;
  fields[row][col]->setText("PRESSED");
  fields[row][col]->setEnabled(false);
}
```

Задача морской бой

field.h

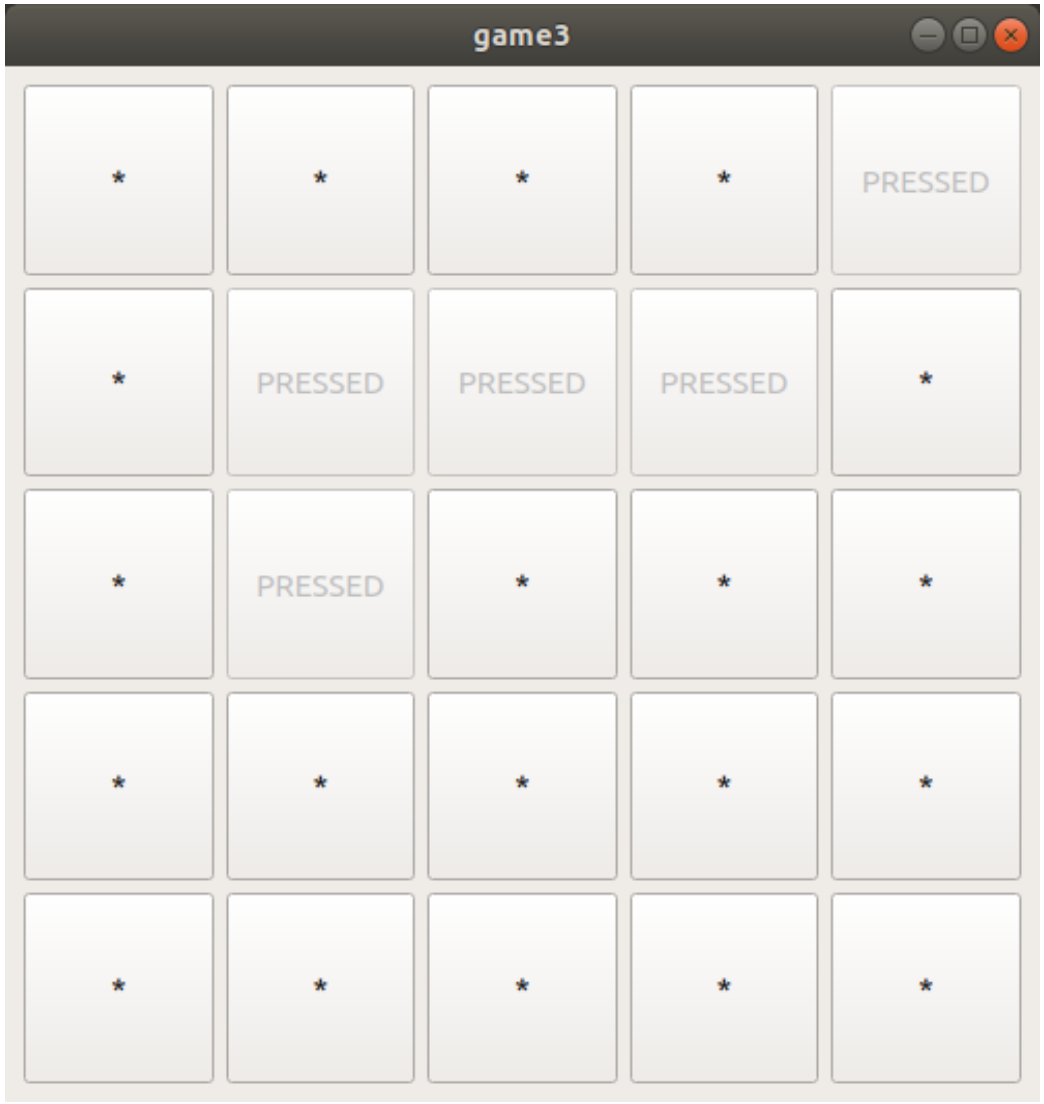
```
#ifndef FIELD_H
#define FIELD_H

#include <QPushButton>
#include <QWidget>

class Field : public QPushButton {
  Q_OBJECT
  int row, col;

public:
  Field(int row, int col, QWidget *parent = NULL);

signals:
  void moved(int row, int col);

public slots:
  void clickhandler();
};
```

Задача морской бой

field.cpp

```
#include "field.h"

Field::Field(int row, int col, QWidget *parent)
      :QPushButton(parent), row(row), col(col) {
  qDebug() << "CONSTRUCTOR";
  connect(this, SIGNAL(clicked()),
          this, SLOT(clickhandler()));
}

void Field::clickhandler() {
  emit moved(row, col);
}
```

Задача морской бой



Отладка

```
#include <QDebug>

qDebug() << row << col;
```

Литература

Самая непонятная ошибка

```
undefined reference to vtable for ...
```

Лечится пересборкой

```
make clean
qmake
make
```

Литература

Макс Шлее. Qt 5.10. Профессиональное программирование на C++. — БХВ-Петербург. — 2018.