Министерсво образоваия Республики Беларусь

Учреждение образования

«Бресткий государтвенный технический университет»

Кафедра ИИТ

Лабораторная работа №1

за 5 семестр

По дисциплине «ОСиСП»

Выполнил:

Студент 3 курса

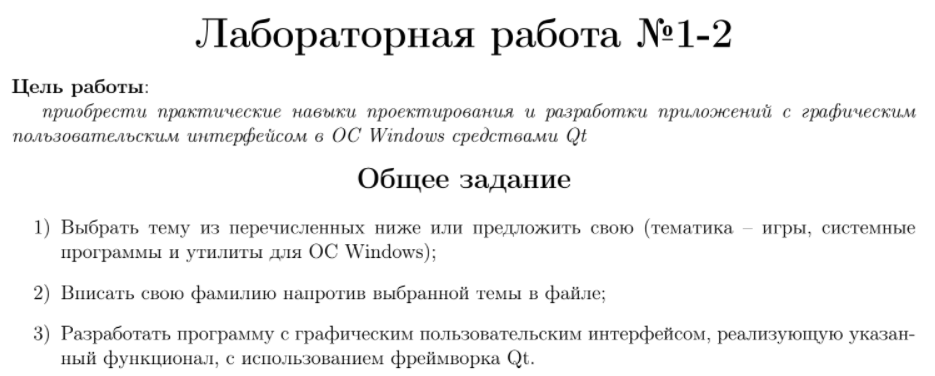
Группы ПО-4(2)

Кречко К.А.

Проверил:

Булей Е.В.

Брест 2021



**Вариант – Игра “пятнашки”**

**Код программы:**

**Заголовочные:**

**gameboard.h:**

#ifndef GAMEBOARD\_H

#define GAMEBOARD\_H

#include <QAbstractListModel>

#include <vector>

*class* **GameBoard** : *public* QAbstractListModel

{

Q\_OBJECT

Q\_PROPERTY(int dimension READ dimension CONSTANT)

Q\_PROPERTY(int hiddenElementValue READ boardSize CONSTANT)

*public*:

*using* Position = std::pair<size\_t, size\_t>;

*static* *constexpr* size\_t defaultPuzzleDimension {4};

**GameBoard**(*const* size\_t boardDimension = defaultPuzzleDimension,

QObject\* parent = *nullptr*);

*struct* **Tile** {

size\_t value {};

Tile& *operator* = (*const* size\_t newValue){

value = newValue;

*return* \**this*;

}

bool *operator* == (*const* size\_t other){

*return* other == value;

}

};

Q\_INVOKABLE bool **move** (int index);

size\_t **dimension**() *const*;

size\_t **boardSize**() *const*;

int ***rowCount***(*const* QModelIndex& parent = QModelIndex{}) *const* *override*;

QVariant ***data***(*const* QModelIndex& index, int role = Qt::*DisplayRole*) *const* *override*;

*private*:

void **shuffle**();

bool **isPositionValid**(*const* size\_t position) *const*;

bool **isBoardValid**() *const*;

Position **getRowCol** (size\_t index) *const*;

std::vector<Tile> m\_rawBoard;

*const* size\_t m\_dimension;

*const* size\_t m\_boardSize;

};

#endif *//* *GAMEBOARD\_H*

**Исходники:**

**gameboard.cpp:**

#include "gameboard.h"

#include <algorithm>

#include <random>

*namespace*

{

bool **isAbjacent** (*const* GameBoard::Position f, *const* GameBoard::Position s){

*if* (f == s)

*return* *false*;

*const* *auto* calcDistance = [](*const* size\_t pos1, size\_t pos2){

int distance = *static\_cast*<int>(pos1);

distance -= *static\_cast*<int>(pos2);

distance = std::abs(distance);

*return* distance;

};

bool result {*false*};

*if*(f.first == s.first){

int distance = calcDistance(f.second, s.second);

*if* (distance == 1)

result = *true*;

} *else* *if* (f.second == s.second){

int distance = calcDistance(f.first, s.first);

*if* (distance == 1)

result = *true*;

}

*return* result;

}

}

GameBoard::**GameBoard**(*const* size\_t boardDimension, QObject \*parent)

: QAbstractListModel {*parent*},

m\_dimension {boardDimension},

m\_boardSize {m\_dimension \* m\_dimension}

{

m\_rawBoard.resize(m\_boardSize);

std::iota(m\_rawBoard.begin(), m\_rawBoard.end(), 1); *//генерация* *последовательности*

shuffle();

}

bool GameBoard::**move**(int index)

{

*if* (!isPositionValid(*static\_cast*<size\_t>(index)))

*return* *false*;

*const* Position elementPosition {getRowCol(index)};

*auto* hiddenElementIterator = std::find(m\_rawBoard.begin(), m\_rawBoard.end(), boardSize()); *//hidden* *element*

Q\_ASSERT(hiddenElementIterator != m\_rawBoard.end());

Position hiddenElementPosition {getRowCol(std::distance(m\_rawBoard.begin(), hiddenElementIterator))};

*if* (!isAbjacent(elementPosition, hiddenElementPosition))

*return* *false*;

*//std::iter\_swap(hiddenElementIterator,* *elementPosition);*

std::swap(*hiddenElementIterator->value*, *m\_rawBoard[index].value*);

emit dataChanged(createIndex(0,0), createIndex(m\_boardSize, 0));

*return* *true*;

}

size\_t GameBoard::**boardSize**() *const*

{

*return* m\_boardSize;

}

size\_t GameBoard::**dimension**() *const*

{

*return* m\_dimension;

}

int GameBoard::***rowCount***(*const* QModelIndex &parent) *const*

{

Q\_UNUSED(parent)

*return* m\_rawBoard.size(); *//возвращаем* *кол-во* *элементов* *на* *доске*

}

QVariant GameBoard::***data***(*const* QModelIndex &index, int role) *const*

{

*if*(!index.isValid() || role != Qt::*DisplayRole*)

*return* {};

*const* int rowIndex {index.row()};

*if*(!isPositionValid(rowIndex))

*return* {};

*return* QVariant::fromValue(m\_rawBoard[rowIndex].value);

}

GameBoard::Position GameBoard::**getRowCol**(*const* size\_t index) *const*

{

Q\_ASSERT(m\_dimension > 0);

size\_t row = index / m\_dimension;

size\_t column = index % m\_dimension;

*return* std::make\_pair(*row*, *column*);

}

void GameBoard::**shuffle**()

{

*static* *auto* seed = std::chrono::system\_clock::now().time\_since\_epoch().count();

*static* std::mt19937 generator(*seed*);

*do* {

std::shuffle(m\_rawBoard.begin(), m\_rawBoard.end(), *generator*); *//перемешивание* *последовательности*

} *while*(!isBoardValid());

}

bool GameBoard::**isPositionValid**(*const* size\_t position) *const*

{

*return* position < m\_boardSize;

}

bool GameBoard::**isBoardValid**() *const*

{

int inv {0};

*for*(size\_t i{0}; i < m\_boardSize; ++i){

*for* (size\_t j = 0; j < i; ++j){

*if*(m\_rawBoard[j].value > m\_rawBoard[i].value)

++inv;

}

}

*const* size\_t start\_point = 1;

*for*(size\_t i = 0; i<m\_boardSize; ++i)

*if*(m\_rawBoard[i].value == m\_boardSize)

inv += start\_point + i / m\_dimension;

*return* (inv % 2) == 0;

}

**main.cpp**

#include <QGuiApplication>

#include <QQmlApplicationEngine>

#include "gameboard.h"

int main(int argc, char \*argv[])

{

#if QT\_VERSION < QT\_VERSION\_CHECK(6, 0, 0)

QCoreApplication::setAttribute(Qt::AA\_EnableHighDpiScaling);

#endif

QGuiApplication app(*argc*, *argv*);

GameBoard model;

qmlRegisterType<GameBoard> ("Game", 1, 0, "GameBoardModel");

QQmlApplicationEngine engine; *//для* *запуска* *QML*

*const* QUrl url(QStringLiteral("qrc:/main.qml"));

QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,

&app, [url](QObject \*obj, *const* QUrl &objUrl) {

*if* (!obj && url == objUrl)

QCoreApplication::exit(-1);

}, Qt::*QueuedConnection*);

engine.load(url);

*return* app.exec();

}

**Ресурсы:**

**GameBoard.qml:**

*import* QtQuick 2.0

*import* Game 1.0;

GridView {

id: *root*

model: GameBoardModel{

}

cellHeight: *height* / root.model.dimension

cellWidth: *width* / root.model.dimension

delegate: Item {

id: *\_backgroundDelegate*

width: *root*.cellWidth

height: *root*.cellHeight

visible: display !== root.model.hiddenElementValue

Tile {

displayText: display *//для* *нумерации* *(display* *это* *роль)*

anchors.fill: *\_backgroundDelegate*

anchors.margins: 5

MouseArea{

anchors.fill: parent

onClicked: {

root.model.move(index);

}

}

}

}

}

**main.qml**

*import* QtQuick 2.15

*import* QtQuick.Window 2.15

Window { *//компонент* *для* *создания* *окна* *(единственный,* *корневой)*

id: *root*

width: 640

height: 480

visible: true

title: *qsTr*("Пятнашечки")

GameBoard {

anchors.fill: *parent*

}

}

**Tile.qml**

*import* QtQuick 2.0

Rectangle {

id: *root*

*property* *string* displayText: ""

color: "black"

radius: 10

border.color: "pink"

border.width: 1

Text {

id: *\_firstText*

anchors.centerIn: *root*

color: "pink"

text: *root*.displayText

font {

pointSize: *Math*.min(*root*.width, *root*.height) / 2

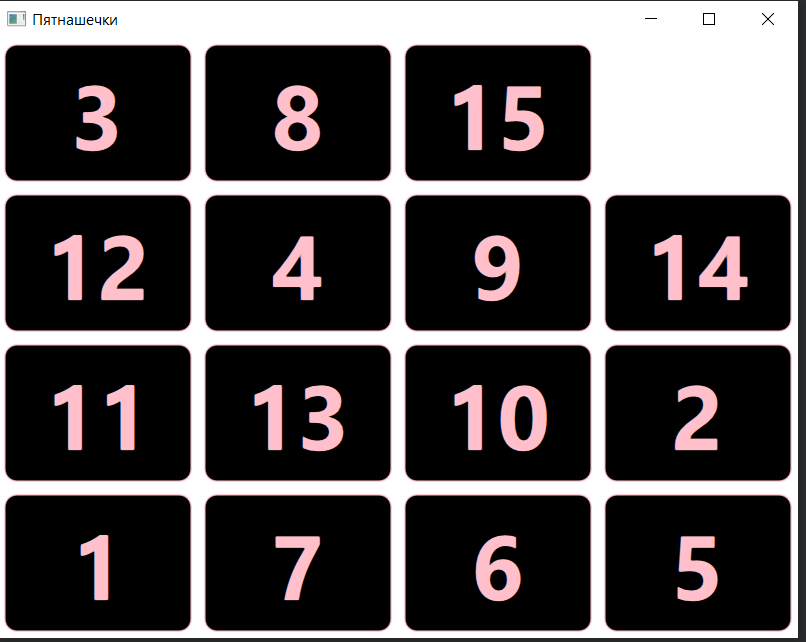
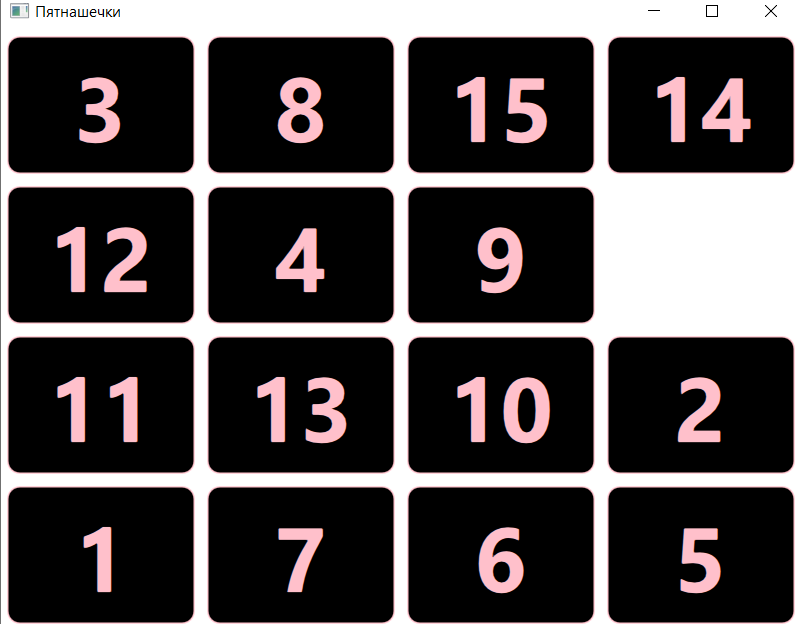
bold: true

}

}

}

**Результат:**

**Вывод:** Я приобрел практические навыки проектирования и разработки приложений с графическим пользовательским интерфейсом в ОС Windows средствами Qt.