# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет “Львівська політехніка”**



**Інститут післядипломної освіти**

**ЗВІТ**

**Про виконання лабораторної роботи №3**

**«Проектування та реалізація ПЗ для мобільних пристроїв на основі кросплатформної бібліотеки QT»**

**з дисципліни «Інженерія кросплатформного програмного забезпечення»**

Виконав:

слухач групи ПЗС-21

Гринчук Тарас

Прийняв:

ст. викл.Сенів М.М.

« »\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2015 р.

∑ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЛЬВІВ – 2015

**Тема роботи**: Проектування та реалізація ПЗ для мобільних пристроїв на основі кросплатформної бібліотеки QT.

**Мета роботи:** Навчитися використовувати засоби бібліотеки QT при створенні додатків для мобільних пристроїв.

## 1. Теоретичні відомості

Qt має модульну структуру, кожен модуль є окремою бібліотекою. У файлі проекту необхідно вказувати, які модулі використовує додаток. За замовчуванням до кожного проекту підключається два основних модулі: Core і Gui. Приклад задання додаткових модулів в pro-файлі:

QT + = xml sql

Перелік модулів Qt для платформи Symbian:

*Core* – Містить набір основних класів не пов'язаних з GUI  
*Gui* – Класи для створення графічного інтерфейсу користувача  
*SQL* – Класи для взаємодії з базами даних  
*XML* – Підтримка XML (SAX, DOM)  
*XMLPatterns* – Підтримка XPath та XQuery  
*Network* – Необхідні класи для створення мережевих додатків  
*SVG* – Класи для роботи з масштабована векторна графіка  
*OpenGL* – Підтримка OpenGL  
*Webkit* – Використання "движка" WebKit для відображення веб-сторінок  
*Script* – Підтримка мови сценаріїв Qt Scripts  
*Phonon* – Підтримка мультимедійного контенту  
  
Перетворення даних (int-string)

QString::number(int)  
  
Робота з файлами  
  
Для роботи з файлами потрібно включити відповідний заголовний файл:

#Include <QFile>  
  
Процедура запису рядка у файл:

QFile file ("c :/ / test.txt");  
file.open (QIODevice :: Append | QIODevice :: Text);  
QTextStream out (& file);  
out << temp;  
out << "\ n";  
file.close ();

## 2. Хід роботи

Розробимо проект гри «Тетрісь» за допомогою бібліотеки QT Jambi, яке э призначена для розробки кросплатформенного ПЗ мовою Java.

Об'єкт гри полягає в тому, щоб скласти шматки виключений з верхньої частини ігрового поля таким чином, що вони заповнюють цілі рядки в нижній частині ігрового поля.

Коли ряд заповнюється, всі блоки на цьому рядку видаляються, гравець заробляє кількість очок, і частини вище переміщаються вниз, щоб зайняти цей рядок. Якщо більш ніж один рядок заповнюється, блоки на кожному рядку будуть видалені, а гравець заробляє додаткові очки.

Ліва клавіша курсор переміщається поточний шматок один простір ліворуч, праворуч курсор переміщається ключові він один пробіл вправо, вгору ключ курсор обертається шматок проти годинникової стрілки на 90 градусів, а кнопку курсора вниз повертає шматок за годинниковою стрілкою 90 градусів.

Цей приклад показує, як проста гра може бути створена з використанням тільки двох класів:

**Клас Tetrix** використовується для відображення рахунок гравця, кількість життів, і інформацію про наступній частині, щоб з'явитися.

**Клас TetrixBoard** містить ігрову логіку, обробляє введення з клавіатури, і відображає штук на ігровому полі. Це також показує рахунок гравця, кількість життів, і інформацію про наступній частині, щоб з'явитися.

Клас **TetrixPiece** містить інформацію про кожної частини.

При такому підході, клас **TetrixBoard** є найбільш складним клас, так як він обробляє ігрову логіку і візуалізацію. Однією з переваг є те, що класи **Tetrix** і **TetrixPiece** дуже прості і містять тільки мінімум коду.

**Клас Tetrix**

Клас **Tetrix** використовується для відображення інформації про грі і містить ігрову зону:

class TetrixBoard extends QFrame

{

Ми використовуємо особисті змінні-члени (показано нижче), щоб користувачеві почати нову гру, призупинити поточну гру, і для виходу.

Хоча вікно успадковує QWidget, конструктор не має аргумента, що дозволяє батьківський віджет визначити. Це тому, що вікно завжди буде використовуватися в якості віджету верхнього рівня.

public Tetrix() {

this(null);

}

public Tetrix(QWidget parent)

{

super(parent);

board = new TetrixBoard(null);

Почнемо з побудови примірника **TetrixBoard** для ігрової зони та етикетки, що показує наступний шматок, щоб бути скинуті в ігровій зоні; Етикетка спочатку порожня.

scoreLcd = new QLCDNumber(5);

scoreLcd.setSegmentStyle(QLCDNumber.SegmentStyle.Filled);

Три кнопки з ярликами побудовані так, що користувач може почати нову гру, призупинити поточну гру, і вийти з програми:

startButton = new QPushButton("&Start");

startButton.setFocusPolicy(Qt.FocusPolicy.NoFocus);

quitButton = new QPushButton("&Quit");

quitButton.setFocusPolicy(Qt.FocusPolicy.NoFocus);

pauseButton = new QPushButton("&Pause");

pauseButton.setFocusPolicy(Qt.FocusPolicy.NoFocus);

Ці кнопки налаштовані так, що вони ніколи не отримують фокус введення; Тим не менш, кнопки будуть як і раніше відповідати на Alt комбінації клавіш.

startButton.clicked.connect(board, "start()");

quitButton.clicked.connect(this, "close()");

pauseButton.clicked.connect(board, "pause()");

board.scoreChanged.connect(scoreLcd, "display(int)");

board.levelChanged.connect(levelLcd, "display(int)");

board.linesRemovedChanged.connect(linesLcd, "display(int)");

Сигнали від плати також підключені до РК-віджетів з метою оновлення рахунка, кількість життів, і лінії видаляються з ігрової зони.

Ми розміщуємо етикетку, віджети, і дошка в QGridLayout поряд з деякими мітками, які ми створюємо з createLabel:

layout = new QGridLayout();

layout.addWidget(createLabel("NEXT"), 0, 0);

layout.addWidget(nextPieceLabel, 1, 0);

layout.addWidget(createLabel("LEVEL"), 2, 0);

layout.addWidget(levelLcd, 3, 0);

layout.addWidget(startButton, 4, 0);

layout.addWidget(board, 0, 1, 6, 1);

layout.addWidget(createLabel("SCORE"), 0, 2);

layout.addWidget(scoreLcd, 1, 2);

layout.addWidget(createLabel("LINES REMOVED"), 2, 2);

layout.addWidget(linesLcd, 3, 2);

layout.addWidget(quitButton, 4, 2);

layout.addWidget(pauseButton, 5, 2);

setLayout(layout);

setWindowTitle("Tetrix");

setWindowIcon(new QIcon("classpath:com/trolltech/classpath:com/trolltech/images/qt-logo.png"));

resize(550, 370);

}

Нарешті, ми встановлюємо макет сітки на віджеті, дають вікном назву, і змінити його розмір до потрібного розміру.

Зручний метод createLabel просто створює новий ярлик на, дає йому відповідну орієнтацію, і повертає його:

private QLabel createLabel(String text)

{

QLabel lbl = new QLabel(text);

lbl.setAlignment(new Qt.Alignment(Qt.AlignmentFlag.AlignBottom));

return lbl;

}

Оскільки кожна мітка буде використовуватися в макеті віджета, він стане наслідниеом віджета **TetrixWindow** і, як результат, він буде вилучений, коли вікно буде видалене.

private TetrixBoard board = null;

private QPushButton startButton = null;

private QPushButton quitButton = null;

private QPushButton pauseButton = null;

private QLabel nextPieceLabel = null;

private QLCDNumber scoreLcd = null;

private QLCDNumber levelLcd = null;

private QLCDNumber linesLcd = null;

private QGridLayout layout = null;

Метод Main () виглядає наступним чином:

public static void main(String args[])

{

QApplication.initialize(args);

Tetrix window = new Tetrix();

window.show();

QApplication.exec();

}

}

## Клас TetrixBoard

Клас успадковує від TetrixBoard QFrame і містить логічні ігри і відображення функцій:

class TetrixBoard extends QFrame

{

static final int redTable[] = new int[8];

static final int greenTable[] = new int[8];

static final int blueTable[] = new int[8];

static {

redTable[0] = 0x00;

redTable[1] = 0xCC;

redTable[2] = 0x66;

redTable[3] = 0x66;

redTable[4] = 0xCC;

redTable[5] = 0xCC;

redTable[6] = 0x66;

redTable[7] = 0xDA;

greenTable[0] = 0x00;

greenTable[1] = 0x66;

greenTable[2] = 0xCC;

greenTable[3] = 0x66;

greenTable[4] = 0xCC;

greenTable[5] = 0x66;

greenTable[6] = 0xCC;

greenTable[7] = 0xAA;

blueTable[0] = 0x00;

blueTable[1] = 0x66;

blueTable[2] = 0x66;

blueTable[3] = 0xCC;

blueTable[4] = 0x66;

blueTable[5] = 0xCC;

blueTable[6] = 0xCC;

blueTable[7] = 0x00;

};

enum TetrixShape { NoShape, ZShape, SShape, LineShape, TShape, SquareShape,

LShape, MirroredLShape }

private static final int BoardWidth = 10;

private static final int BoardHeight = 22;

Кольори, які використовуються для відображення кожного типу фігури визначаються в окремих таблицях для червоного, зеленого і синього компонентів. Ми також визначити набір форм, які будуть використовуватися на шматки в грі, і встановлених фіксованих розмірів для ігрової зони.

private boolean isStarted = false;

private boolean isPaused = false;

private boolean isWaitingAfterLine = false;

private int numLinesRemoved = 0;

private int numPiecesDropped = 0;

private int score = 0;

private int level = 0;

private int curX = 0;

private int curY = 0;

private QLabel nextPieceLabel = null;

private QBasicTimer timer = new QBasicTimer();

Нова гра почалася, коли викликається start(). Це скидає стан гри, рахунок гравця і рівень, і вміст борту:

public void start()

{

if (isPaused)

return ;

isStarted = true;

isWaitingAfterLine = false;

numLinesRemoved = 0;

numPiecesDropped = 0;

score = 0;

level = 1;

clearBoard();

linesRemovedChanged(numLinesRemoved);

scoreChanged(score);

levelChanged(level);

newPiece();

timer.start(timeoutTime(), this);

}

Ми також випускаємо сигнали для інформування інших компонентів цих змін, перш ніж створювати нову частину, яка готова бути скинута в ігровій зоні. Ми запускаємо таймер, який визначає, як часто шматок падає вниз на один рядок на борту.

pause()використовується для тимчасової зупинки поточної гри, зупинивши внутрішній таймер:

public void pause()

{

if (!isStarted)

return ;

isPaused = !isPaused;

if (isPaused) {

timer.stop();

} else {

timer.start(timeoutTime(), this);

}

update();

}

Ми перевіряємо, чи є подія по будь-якому з ключів, які гравець використовує, щоб контролювати поточну частину і, якщо так, то ми dbrkbrf’vj відповідний метод для обробки введення:

if (event.key() == Qt.Key.Key\_Left.value())

tryMove(curPiece, curX - 1, curY);

else if (event.key() == Qt.Key.Key\_Right.value())

tryMove(curPiece, curX + 1, curY);

else if (event.key() == Qt.Key.Key\_Down.value())

tryMove(curPiece.rotatedRight(), curX, curY);

else if (event.key() == Qt.Key.Key\_Up.value())

tryMove(curPiece.rotatedLeft(), curX, curY);

else if (event.key() == Qt.Key.Key\_Space.value())

dropDown();

else if (event.key() == Qt.Key.Key\_D.value())

oneLineDown();

else

super.keyPressEvent(event);

}

Якщо фігура може бути розміщена на борту в потрібному місці, ми оновлюємо поточний шматок і свою позицію, і повертаєvj істину в разі успіху.

Метод drawSquare() малює блоки (зазвичай квадрати), які складають кожен шматок, використовуючи різні кольори для частин з різними формами:

void drawSquare(QPainter painter, int x, int y, TetrixShape shape)

{

QColor color = new QColor(redTable[shape.ordinal()], greenTable[shape.ordinal()], blueTable[shape.ordinal()]);

painter.fillRect(x + 1, y + 1, squareWidth() - 2, squareHeight() - 2,

new QBrush(color));

painter.setPen(color.lighter());

painter.drawLine(x, y + squareHeight() - 1, x, y);

painter.drawLine(x, y, x + squareWidth() - 1, y);

painter.setPen(color.darker());

painter.drawLine(x + 1, y + squareHeight() - 1, x + squareWidth() - 1, y + squareHeight() - 1);

painter.drawLine(x + squareWidth() - 1, y + squareHeight() - 1, x + squareWidth() - 1, y + 1);

}

}

Вигляд головного вікна гри зображено на рис. 1:

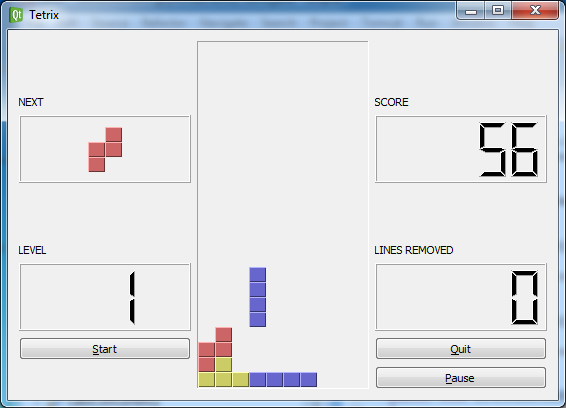


Рис 1. Вигляд головного вікна гри

## ВИСНОВКИ

На даній лабораторній роботі я навчився використовувати засоби бібліотеки QT при створенні додатків для мобільних пристроїв.