Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1-2

По дисциплине: «ОСиСП»

Выполнил:

Студент 3 курса

Группы ПО-4(1)

Грибовский Д. С.

Проверил:

Дряпко А.В.

Брест 2021

Лабораторная работа № 1-2

Вариант 6:

Цель: приобрести практические навыки проектирования и разработки приложений с графическим пользовательским интерфейсом в ОС Windows средствами Qt

Задание: игра «Тетрис». Ограниченный набор фигурок (не более 3). Параметры колодца: ширина – 15 клеток, глубина – 20 клеток. Очки начисляются за полностью заполненные горизонтальные уровни клеток, при этом такие клетки исчезают.

Код:

**main.cpp**

#include "tetrixwindow.h"

#include <QApplication>

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

{

QApplication app(argc, argv);

TetrixWindow window;

window.show();

return app.exec();

}

**Tetrixboard.h**

#ifndef TETRIXBOARD\_H

#define TETRIXBOARD\_H

#include <QBasicTimer>

#include <QFrame>

#include <QPointer>

#include "tetrixpiece.h"

QT\_BEGIN\_NAMESPACE

class QLabel;

QT\_END\_NAMESPACE

class TetrixBoard : public QFrame

{

Q\_OBJECT

public:

TetrixBoard(QWidget \*parent = nullptr);

void setNextPieceLabel(QLabel \*label);

QSize sizeHint() const override;

QSize minimumSizeHint() const override;

public slots:

void start();

void pause();

signals:

void scoreChanged(int score);

void linesRemovedChanged(int numLines);

protected:

void paintEvent(QPaintEvent \*event) override;

void keyPressEvent(QKeyEvent \*event) override;

void timerEvent(QTimerEvent \*event) override;

private:

enum { BoardWidth = 10, BoardHeight = 22 };

TetrixShape &shapeAt(int x, int y) { return board[(y \* BoardWidth) + x]; }

int timeoutTime() { return 1000 / (1 + 1); }

int squareWidth() { return contentsRect().width() / BoardWidth; }

int squareHeight() { return contentsRect().height() / BoardHeight; }

void clearBoard();

void dropDown();

void oneLineDown();

void pieceDropped(int dropHeight);

void removeFullLines();

void newPiece();

bool tryMove(const TetrixPiece &newPiece, int newX, int newY);

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

QBasicTimer timer;

bool isStarted;

bool isPaused;

bool isWaitingAfterLine;

TetrixPiece curPiece;

TetrixPiece nextPiece;

int curX;

int curY;

int numLinesRemoved;

int score;

TetrixShape board[BoardWidth \* BoardHeight];

};

#endif

**tetrixboard.cpp**

#include "tetrixboard.h"

#include <QKeyEvent>

#include <QLabel>

#include <QPainter>

TetrixBoard::TetrixBoard(QWidget \*parent)

: QFrame(parent), isStarted(false), isPaused(false)

{

setFrameStyle(QFrame::Panel | QFrame::Sunken);

setFocusPolicy(Qt::StrongFocus);

clearBoard();

nextPiece.setRandomShape();

}

QSize TetrixBoard::sizeHint() const

{

return QSize(BoardWidth \* 15 + frameWidth() \* 2,

BoardHeight \* 15 + frameWidth() \* 2);

}

QSize TetrixBoard::minimumSizeHint() const

{

return QSize(BoardWidth \* 5 + frameWidth() \* 2,

BoardHeight \* 5 + frameWidth() \* 2);

}

void TetrixBoard::start()

{

if (isPaused)

return;

isStarted = true;

isWaitingAfterLine = false;

numLinesRemoved = 0;

score = 0;

clearBoard();

emit linesRemovedChanged(numLinesRemoved);

emit scoreChanged(score);

newPiece();

timer.start(timeoutTime(), this);

}

void TetrixBoard::pause()

{

if (!isStarted)

return;

isPaused = !isPaused;

if (isPaused) {

timer.stop();

} else {

timer.start(timeoutTime(), this);

}

update();

}

void TetrixBoard::paintEvent(QPaintEvent \*event)

{

QFrame::paintEvent(event);

QPainter painter(this);

QRect rect = contentsRect();

if (isPaused) {

painter.drawText(rect, Qt::AlignCenter, tr("Pause"));

return;

}

int boardTop = rect.bottom() - BoardHeight\*squareHeight();

for (int i = 0; i < BoardHeight; ++i) {

for (int j = 0; j < BoardWidth; ++j) {

TetrixShape shape = shapeAt(j, BoardHeight - i - 1);

if (shape != NoShape)

drawSquare(painter, rect.left() + j \* squareWidth(),

boardTop + i \* squareHeight(), shape);

}

}

if (curPiece.shape() != NoShape) {

for (int i = 0; i < 4; ++i) {

int x = curX + curPiece.x(i);

int y = curY - curPiece.y(i);

drawSquare(painter, rect.left() + x \* squareWidth(),

boardTop + (BoardHeight - y - 1) \* squareHeight(),

curPiece.shape());

}

}

}

void TetrixBoard::keyPressEvent(QKeyEvent \*event)

{

if (!isStarted || isPaused || curPiece.shape() == NoShape) {

QFrame::keyPressEvent(event);

return;

}

switch (event->key()) {

case Qt::Key\_Left:

tryMove(curPiece, curX - 1, curY);

break;

case Qt::Key\_Right:

tryMove(curPiece, curX + 1, curY);

break;

case Qt::Key\_Down:

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

break;

case Qt::Key\_Up:

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

break;

case Qt::Key\_Space:

dropDown();

break;

case Qt::Key\_D:

oneLineDown();

break;

default:

QFrame::keyPressEvent(event);

}

}

void TetrixBoard::timerEvent(QTimerEvent \*event)

{

if (event->timerId() == timer.timerId()) {

if (isWaitingAfterLine) {

isWaitingAfterLine = false;

newPiece();

timer.start(timeoutTime(), this);

} else {

oneLineDown();

}

} else {

QFrame::timerEvent(event);

}

}

void TetrixBoard::clearBoard()

{

for (int i = 0; i < BoardHeight \* BoardWidth; ++i)

board[i] = NoShape;

}

void TetrixBoard::dropDown()

{

int dropHeight = 0;

int newY = curY;

while (newY > 0) {

if (!tryMove(curPiece, curX, newY - 1))

break;

--newY;

++dropHeight;

}

pieceDropped(dropHeight);

}

void TetrixBoard::oneLineDown()

{

if (!tryMove(curPiece, curX, curY - 1))

pieceDropped(0);

}

void TetrixBoard::pieceDropped(int dropHeight)

{

for (int i = 0; i < 4; ++i) {

int x = curX + curPiece.x(i);

int y = curY - curPiece.y(i);

shapeAt(x, y) = curPiece.shape();

}

removeFullLines();

if (!isWaitingAfterLine)

newPiece();

}

void TetrixBoard::removeFullLines()

{

int numFullLines = 0;

for (int i = BoardHeight - 1; i >= 0; --i) {

bool lineIsFull = true;

for (int j = 0; j < BoardWidth; ++j) {

if (shapeAt(j, i) == NoShape) {

lineIsFull = false;

break;

}

}

if (lineIsFull) {

++numFullLines;

for (int k = i; k < BoardHeight - 1; ++k) {

for (int j = 0; j < BoardWidth; ++j)

shapeAt(j, k) = shapeAt(j, k + 1);

}

for (int j = 0; j < BoardWidth; ++j)

shapeAt(j, BoardHeight - 1) = NoShape;

}

}

if (numFullLines > 0) {

numLinesRemoved += numFullLines;

score += 10 \* numFullLines;

emit linesRemovedChanged(numLinesRemoved);

emit scoreChanged(score);

timer.start(500, this);

isWaitingAfterLine = true;

curPiece.setShape(NoShape);

update();

}

}

void TetrixBoard::newPiece()

{

curPiece = nextPiece;

nextPiece.setRandomShape();

curX = BoardWidth / 2 + 1;

curY = BoardHeight - 1 + curPiece.minY();

if (!tryMove(curPiece, curX, curY)) {

curPiece.setShape(NoShape);

timer.stop();

isStarted = false;

}

}

bool TetrixBoard::tryMove(const TetrixPiece &newPiece, int newX, int newY)

{

for (int i = 0; i < 4; ++i) {

int x = newX + newPiece.x(i);

int y = newY - newPiece.y(i);

if (x < 0 || x >= BoardWidth || y < 0 || y >= BoardHeight)

return false;

if (shapeAt(x, y) != NoShape)

return false;

}

curPiece = newPiece;

curX = newX;

curY = newY;

update();

return true;

}

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

{

static constexpr QRgb colorTable[8] = {

0x000000, 0xCC6666, 0x66CC66, 0x6666CC,

0xCCCC66, 0xCC66CC, 0x66CCCC, 0xDAAA00

};

QColor color = colorTable[int(shape)];

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

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);

}

**tetrixpiece.h**

#ifndef TETRIXPIECE\_H

#define TETRIXPIECE\_H

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

LShape, MirroredLShape };

class TetrixPiece

{

public:

TetrixPiece() { setShape(NoShape); }

void setRandomShape();

void setShape(TetrixShape shape);

TetrixShape shape() const { return pieceShape; }

int x(int index) const { return coords[index][0]; }

int y(int index) const { return coords[index][1]; }

int minX() const;

int maxX() const;

int minY() const;

int maxY() const;

TetrixPiece rotatedLeft() const;

TetrixPiece rotatedRight() const;

private:

void setX(int index, int x) { coords[index][0] = x; }

void setY(int index, int y) { coords[index][1] = y; }

TetrixShape pieceShape;

int coords[4][2];

};

#endif

**tetrixpiece.cpp**

#include "tetrixpiece.h"

#include <QtCore>

void TetrixPiece::setRandomShape()

{

setShape(TetrixShape(QRandomGenerator::global()->bounded(7) + 1));

}

void TetrixPiece::setShape(TetrixShape shape)

{

static constexpr int coordsTable[8][4][2] = {

{ { 0, 0 }, { 0, 0 }, { 0, 0 }, { 0, 0 } },

{ { 0, -1 }, { 0, 0 }, { -1, 0 }, { -1, 1 } },

{ { 0, -1 }, { 0, 0 }, { 1, 0 }, { 1, 1 } },

{ { 0, -1 }, { 0, 0 }, { 0, 1 }, { 0, 2 } },

{ { -1, 0 }, { 0, 0 }, { 1, 0 }, { 0, 1 } },

{ { 0, 0 }, { 1, 0 }, { 0, 1 }, { 1, 1 } },

{ { -1, -1 }, { 0, -1 }, { 0, 0 }, { 0, 1 } },

{ { 1, -1 }, { 0, -1 }, { 0, 0 }, { 0, 1 } }

};

for (int i = 0; i < 4 ; i++) {

for (int j = 0; j < 2; ++j)

coords[i][j] = coordsTable[shape][i][j];

}

pieceShape = shape;

}

int TetrixPiece::minX() const

{

int min = coords[0][0];

for (int i = 1; i < 4; ++i)

min = qMin(min, coords[i][0]);

return min;

}

int TetrixPiece::maxX() const

{

int max = coords[0][0];

for (int i = 1; i < 4; ++i)

max = qMax(max, coords[i][0]);

return max;

}

int TetrixPiece::minY() const

{

int min = coords[0][1];

for (int i = 1; i < 4; ++i)

min = qMin(min, coords[i][1]);

return min;

}

int TetrixPiece::maxY() const

{

int max = coords[0][1];

for (int i = 1; i < 4; ++i)

max = qMax(max, coords[i][1]);

return max;

}

TetrixPiece TetrixPiece::rotatedLeft() const

{

if (pieceShape == SquareShape)

return \*this;

TetrixPiece result;

result.pieceShape = pieceShape;

for (int i = 0; i < 4; ++i) {

result.setX(i, y(i));

result.setY(i, -x(i));

}

return result;

}

TetrixPiece TetrixPiece::rotatedRight() const

{

if (pieceShape == SquareShape)

return \*this;

TetrixPiece result;

result.pieceShape = pieceShape;

for (int i = 0; i < 4; ++i) {

result.setX(i, -y(i));

result.setY(i, x(i));

}

return result;

}

**tetrixwindow.h**

#ifndef TETRIXWINDOW\_H

#define TETRIXWINDOW\_H

#include <QWidget>

QT\_BEGIN\_NAMESPACE

class QLCDNumber;

class QLabel;

class QPushButton;

QT\_END\_NAMESPACE

class TetrixBoard;

class TetrixWindow : public QWidget

{

Q\_OBJECT

public:

TetrixWindow(QWidget \*parent = nullptr);

private:

QLabel \*createLabel(const QString &text);

TetrixBoard \*board;

QLCDNumber \*scoreLcd;

QLCDNumber \*linesLcd;

QPushButton \*startButton;

QPushButton \*quitButton;

QPushButton \*pauseButton;

};

#endif

**tetrixwindow.cpp**

#include "tetrixboard.h"

#include "tetrixwindow.h"

#include <QCoreApplication>

#include <QGridLayout>

#include <QLabel>

#include <QLCDNumber>

#include <QPushButton>

TetrixWindow::TetrixWindow(QWidget \*parent)

: QWidget(parent), board(new TetrixBoard)

{

scoreLcd = new QLCDNumber(5);

scoreLcd->setSegmentStyle(QLCDNumber::Filled);

linesLcd = new QLCDNumber(5);

linesLcd->setSegmentStyle(QLCDNumber::Filled);

startButton = new QPushButton(tr("&Start"));

startButton->setFocusPolicy(Qt::NoFocus);

quitButton = new QPushButton(tr("&Quit"));

quitButton->setFocusPolicy(Qt::NoFocus);

pauseButton = new QPushButton(tr("&Pause"));

pauseButton->setFocusPolicy(Qt::NoFocus);

connect(startButton, &QPushButton::clicked, board, &TetrixBoard::start);

connect(quitButton , &QPushButton::clicked, qApp, &QCoreApplication::quit);

connect(pauseButton, &QPushButton::clicked, board, &TetrixBoard::pause);

#if \_\_cplusplus >= 201402L

connect(board, &TetrixBoard::scoreChanged,

scoreLcd, qOverload<int>(&QLCDNumber::display));

connect(board, &TetrixBoard::linesRemovedChanged,

linesLcd, qOverload<int>(&QLCDNumber::display));

#else

connect(board, &TetrixBoard::scoreChanged,

scoreLcd, QOverload<int>::of(&QLCDNumber::display));

connect(board, &TetrixBoard::linesRemovedChanged,

linesLcd, QOverload<int>::of(&QLCDNumber::display));

#endif

QGridLayout \*layout = new QGridLayout;

layout->addWidget(createLabel(tr("")), 0, 0);

layout->addWidget(createLabel(tr("")), 2, 0);

layout->addWidget(startButton, 4, 0);

layout->addWidget(board, 0, 1, 6, 1);

layout->addWidget(createLabel(tr("SCORE")), 0, 2);

layout->addWidget(scoreLcd, 1, 2);

layout->addWidget(createLabel(tr("LINES REMOVED")), 2, 2);

layout->addWidget(linesLcd, 3, 2);

layout->addWidget(quitButton, 4, 2);

layout->addWidget(pauseButton, 5, 2);

setLayout(layout);

setWindowTitle(tr("Tetrix"));

resize(550, 370);

}

QLabel \*TetrixWindow::createLabel(const QString &text)

{

QLabel \*label = new QLabel(text);

label->setAlignment(Qt::AlignHCenter | Qt::AlignBottom);

return label;

}

Результат:  






