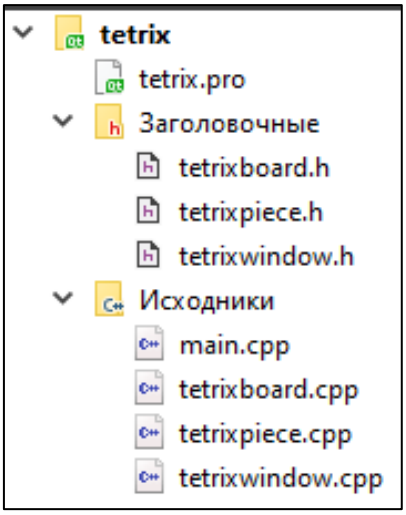
Создание игры Тетрис в Qt

1. Распакуйте архив tetrix.zip в каталог D:/For Students.
2. Запустите Qt Creator и откройте через него файл tetrix.pro, находящийся в распакованной папке. Структура проекта должна выглядеть примерно так



3. Рассмотрим предназначение этих файлов:
   - main.cpp – главный файл проекта. В нем, как и в других рассмотренных ранее проектах, создается приложение, вызывается окно графического интерфейса и запускается приложение. Помимо этого, в этом файле инициализируется зерно генератора псевдослучайных чисел. Это нужно для того, чтобы последовательность появляющихся фигур не повторялась при повторных запусках программы. Этот файл мы далее редактировать не будем.
   - файлы tetrixwindow.h и tetrixwindow.cpp содержат описание и определение класса окна приложения. В отличие от предыдущих проектов, здесь мы не используем графический редактор интерфейса. Все визуальные компоненты создаются и настраиваются программно.
   - файлы tetrixboard.h и tetrixboard.cpp предназначены для описания игрового поля. Здесь описывается вся игровая механика.
   - файлы tetrixpiece.h и tetrixpiece.cpp предназначены для описания одной фигуры, состоящей из четырех квадратов. Такая фигура называется тетрамино. Также в этих файлах содержатся алгоритмы работы с тетрамино.
4. Перейдем к рассмотрению классов и их реализаций. Начнем с класса **TetrixPiece**. Состав заголовочного файла:

```
// Описание внешнего вида фигур
enum TetrixShape { NoShape, ZShape, SShape, LineShape, TShape, SquareShape,
                   LShape, MirroredLShape };
class TetrixPiece
{
public:
    TetrixPiece() { setShape(NoShape); }
    // Выбор случайной фигуры
    void setRandomShape();
    // Создание фигуры
    void setShape(TetrixShape shape);
    // Получение формы фигуры
    TetrixShape shape() const { return pieceShape; }
    // Получение координат
    int x(int index) const { return coords[index][0]; }
    int y(int index) const { return coords[index][1]; }
    // Получение минимально возможных и максимально возможных
    // значений для координат фигуры (чтобы не выходить за границу поля)
    int minX() const;
    int maxX() const;
    int minY() const;
    int maxY() const;
    // Повороты
    TetrixPiece rotatedLeft() const;
    TetrixPiece rotatedRight() const;
private:
    // Изменение координат фигуры
    void setX(int index, int x) { coords[index][0] = x; }
    void setY(int index, int y) { coords[index][1] = y; }
    // Форма фигуры
    TetrixShape pieceShape;
    // Координаты фигуры
    int coords[4][2];
};
```

5. Добавим в файл **tetrixpiece.cpp** реализацию методов. Начнем с метода TetrixPiece::setRandomShape, внутри которого следует добавить команду для генерации случайной фигуры, результат которой передается в метод установления формы фигуры `setShape(TetrixShape(qrand() % 7 + 1));`
   Метод установления формы фигуры TetrixPiece::setShape содержит трехмерный массив констант, задающий координаты фигуры. Переданный параметр shape отвечает за выбор правильного подмножества координат из массива.
   Метод TetrixPiece::minX() предназначен для получения минимальной (т.е. крайней левой) координаты фигуры по горизонтальной оси. Также заданы три аналогичных метода для задания крайней правой, самой низкой и самой высокой координат.
   Метод TetrixPiece::rotatedLeft() предназначен для поворота фигуры влево. Для простоты решения квадрат мы не переворачиваем. У всех остальных фигур меняются координаты. Добавьте в метод его реализацию:

```
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;
```

Реализация метода TetrixPiece::rotatedRight() работает аналогичным образом (нужно ее добавить):

```
    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;
```

6. Перейдем к классу **TetrixBoard**. Заголовочный файл выглядит так (обратите внимание, здесь мы задаем не только свои слоты, но и свои сигналы):

```
class TetrixBoard : public QFrame
{
    Q_OBJECT
public:
    TetrixBoard(QWidget *parent = 0);
    // Отображение следующей фигуры
    void setNextPieceLabel(QLabel *label);
    // Задание размеров игровой области
    QSize sizeHint() const override;
    QSize minimumSizeHint() const override;
// Слоты начала игры и паузы
public slots:
    void start();
    void pause();
// Сигналы
signals:
    // Изменение количества набранных очков
    void scoreChanged(int score);
    // Переход на новый уровень
    void levelChanged(int level);
    // Удаление собранных линий
    void linesRemovedChanged(int numLines);
protected:
    // Перерисовка экрана
    void paintEvent(QPaintEvent *event) override;
    // Обработка нажатий клавиш
    void keyPressEvent(QKeyEvent *event) override;
    // Срабатывание таймера
    void timerEvent(QTimerEvent *event) override;
```

```
private:
    // Размеры игрового поля
    enum { BoardWidth = 10, BoardHeight = 22 };
    // Набор методов реализации игровой механики
    TetrixShape &shapeAt(int x, int y) { return board[(y * BoardWidth) + x]; }
    int timeoutTime() { return 1000 / (1 + level); }
    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();
    void showNextPiece();
    bool tryMove(const TetrixPiece &newPiece, int newX, int newY);
    void drawSquare(QPainter &painter, int x, int y, TetrixShape shape);
    // Поля класса для хранения нужных значений
    QBasicTimer timer;
    QPointer<QLabel> nextPieceLabel;
    bool isStarted;
    bool isPaused;
    bool isWaitingAfterLine;
    TetrixPiece curPiece;
    TetrixPiece nextPiece;
    int curX;
    int curY;
    int numLinesRemoved;
    int numPiecesDropped;
    int score;
    int level;
    TetrixShape board[BoardWidth * BoardHeight];
};
```

7. Рассмотрим файл **tetrixboard.cpp** подробнее и добавим реализацию некоторых методов.
   Конструктор TetrixBoard::TetrixBoard создает игровое поле с нужными настройками. В частности, устанавливаются характеристики фрейма (в нем будет игровое поле), задаются параметры начала игры (не начата) и паузы (не нажата), игровое поле очищается, создается следующая фигура.
   Метод TetrixBoard::setNextPieceLabel содержит присвоение метке на экране типа следующей фигуры.
   Методы TetrixBoard::sizeHint и TetrixBoard::minimumSizeHint предназначены для указания минимальных размеров игрового поля в зависимости от размера блока.
   Слот TetrixBoard::start вызывается при нажатии на кнопку Start. Если нажата пауза, то осуществляется досрочный выход из метода, чтобы не потерять текущее состояние игры. Далее выполняются все нужные для начала игры установки, включая обнуление счетчиков убранных линий и очков, установлению первого уровня, очистке игрового поля и вызову ряда функций и сигналов. Добавьте в код реализацию этого метода-слота

```
if (isPaused)
    return;
isStarted = true;
isWaitingAfterLine = false;
numLinesRemoved = 0;
numPiecesDropped = 0;
score = 0;
level = 1;
```

```
clearBoard();
emit linesRemovedChanged(numLinesRemoved);
emit scoreChanged(score);
emit levelChanged(level);
newPiece();
timer.start(timeoutTime(), this);
```

При нажатии на кнопку Pause игровое поле будет скрываться, а игровой процесс приостанавливаться. Это реализовано в слоте TetrixBoard::pause.

Метод TetrixBoard::paintEvent отвечает за прорисовку игрового процесса. При нажатии паузы он скрывает игровое поле. Если же игровой процесс идет, метод прорисовывает нужную фигуру.

Метод TetrixBoard::keyPressEvent отвечает за обработку нажатий клавиш. Попробуйте самостоятельно понять по коду, какие клавиши отвечают за выполнение различных игровых действий.

Метод TetrixBoard::timerEvent обрабатывает срабатывание таймера, а метод TetrixBoard::clearBoard очищает игровое поле.

Метод TetrixBoard::dropDown двигает фигуру вниз до конца и вызывает обработку касания фигуры низа. Добавьте реализацию этого метода

```
int dropHeight = 0;
int newY = curY;
while (newY > 0) {
    if (!tryMove(curPiece, curX, newY - 1))
        break;
    --newY;
    ++dropHeight;
}
pieceDropped(dropHeight);
```

Метод TetrixBoard::oneLineDown пытается сдвинуть фигуру вниз на один ряд. Если не вышло, значит, фигура достигла допустимого края и нужно вызвать соответствующий метод.

Метод TetrixBoard::pieceDropped обрабатывает упавшую фигуру. Он изменяет количество упавших фигур, увеличивает уровень при необходимости, увеличивает количество очков, удаляет заполненные ряды и, если это возможно, добавляет новую фигуру.

Метод TetrixBoard::removeFullLines отвечает за удаление заполненных рядов.

Метод TetrixBoard::newPiece «забирает» фигуру из поля следующей и генерирует новую фигуру, которую помещает в это поле.

Метод TetrixBoard::showNextPiece отвечает за отображение следующей фигуры.

Метод TetrixBoard::tryMove отвечает за попытку движения фигуры и, если фигура была сдвинута, обновляет поле.

Метод TetrixBoard::drawSquare рисует заполненный нужным цветом квадрат.

8. Перейдем к классу **TetrixWindow**. В заголовочном файле содержится перечисление используемых визуальных элементов, конструктор, а также добавлен метод создания метки.

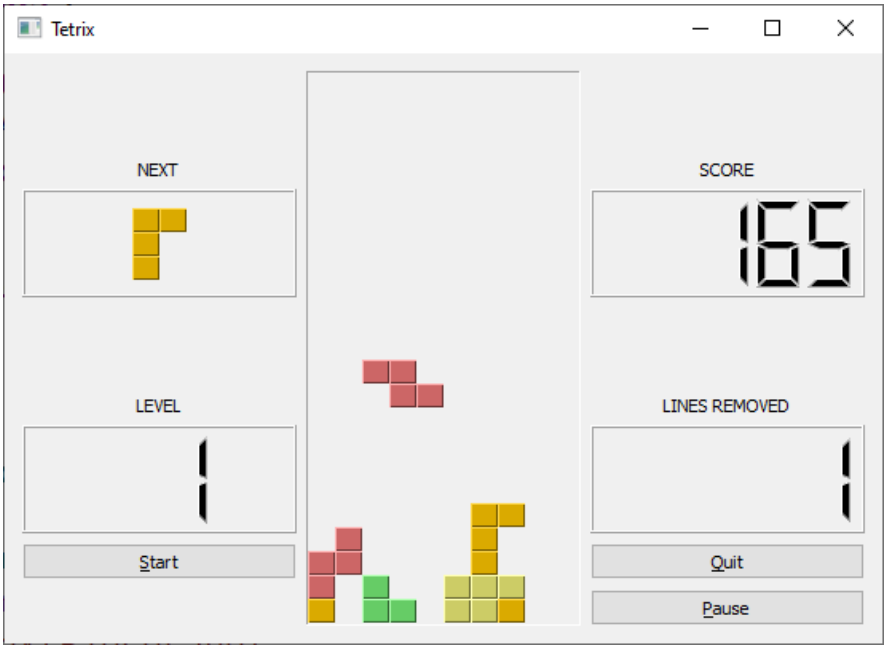
```
class TetrixWindow : public QWidget
{
    Q_OBJECT
public:
    TetrixWindow();
private:
    QLabel *createLabel(const QString &text);
    TetrixBoard *board;
    QLabel *nextPieceLabel;
    QLCDNumber *scoreLcd;
    QLCDNumber *levelLcd;
    QLCDNumber *linesLcd;
    QPushButton *startButton;
    QPushButton *quitButton;
    QPushButton *pauseButton;
};
```

9. Рассмотрим файл **tetrixwindow.cpp** подробнее.
   В конструкторе создаются нужные элементы:
   - Игровое поле board (экземпляр описанного выше класса).
   - Метка для отображения следующей фигуры.
   - Поля для отображения количества набранных очков scoreLcd, уровня levelLcd и количества убранных линий linesLcd принадлежат классу QLCDNumber, чтобы сделать интерфейс более интересным.
   - Кнопки startButton (начало), quitButton (выход), pauseButton (пауза) создаются, нажатие на эти кнопки связывается с соответствующими слотами.

   Также здесь связываются созданные сигналы и поля для отображения очков, уровня и количества линий.
   Для красивого отображения элементов используется лэйаут layout. Созданные ранее элементы выравниваются по сетке. Это позволяет сохранять их относительные размеры и расположение при изменении окна приложения.
   В конце конструктора устанавливается заголовок приложения, а также первоначальный размер окна.
   Метод TetrixWindow::createLabel описывает создание метки. Добавьте в код его реализацию

```
QLabel *lbl = new QLabel(text);
lbl->setAlignment(Qt::AlignHCenter | Qt::AlignBottom);
return lbl;
```

10. Запускаем проект.



11. Измените приложение таким образом, чтобы нажатие стрелочки вниз опускало фигуру.
12. Попробуйте самостоятельно придумать, как еще можно изменить возможности приложения.