

前言

这一节主要学习在 Qt 中怎样使用图形视图框架,实验完成的是一个简易的俄罗斯方块游戏,有了图形视图框架的支持,该游戏的设计变得非常简单,不需要考虑很多复杂的算法,比如说方块的碰撞检测,旋转,视图的设计等。从本实验中可以学到 2D 图形的绘制,游戏的逻辑设计,图形视图的应用,动画设置,背景音乐的添加,Phonon 框架的应用等知识。实验的参考资料为 <http://www.yafeilinux.com/> 网站上 yafei 作者提供的代码,本人只是看懂其源码然后自己敲了一遍,代码做了稍微的改变,其设计方法和技巧全是原创作者 yafei 的功劳。

实验说明

注意本实验没有使用 QtDesigner 来设计界面,其界面而是直接采用 c++ 代码来写的。下面分一下几个方面来介绍本实验的实现过程中应该注意的知识点:

设计方块组:

首先是设计出一个小方块,长和宽各位 20 个像素,画小方块时对其进行贴图,然后采用方块透明颜色用画刷重新刷一遍,方块外用画笔画,这样看起来比较有质感,且容易区分出边界。

俄罗斯方块有 7 种形状,另外加一个随机形状,共 8 种。其创建方法是根据指定的位置和给出的形状信息创建一个由 4 个小方块组成的方块组。

按键分配为:空格键表示变换方块组的形状,向左和右表示左右移动方块,向下方向键表示方块组加速下降。

正常情况下,利用定时器设置了时间间隔,每个一段时间,屏幕中出现了的俄罗斯方块组就会自动下降一格,如果你按了向下的方向键,则方块下降的速度非常快,直到到达下面碰撞为止。

在设计该游戏时,碰撞检测非常重要。其主要在下面几个地方用到:在对屏幕中出现的方块组进行左右或者向下移动时,程序时先试探性的向左右或向下移动,如果发现有碰撞表示移动不成功,这时候就会自动按照刚刚的逆方向移动回去。另外在游戏结束判断时,如果方块组一开始出现的时候就发生了碰撞功能,说明游戏已经结束了。碰撞函数功能的实现是采用 item 这个类自带的 collidingItems()碰撞检测函数来实现的,直接调用即可。

当需要将方块组从视图中移除的时候,其本质是将方块组中的所有的小方块都移除掉,调用的是 QGraphicsItemGroup 中的 removeFromGroup 函数。

画出俄罗斯方块的几种类型,每一个俄罗斯方块由 4 个小方块组成,因此俄罗斯方块出现的位置肯定在一个 2*2 的方块矩形内,其中中心的坐标设置为 (0, 0)。

当对方块组进行控制时,移动方法 moveBy(), 旋转方法 rotate(),碰撞检测方法 collidingItems().count(), item 移除场景方法 removeFromGroup(), 销毁 item 方法 deleteLater()等都是 Qt 库中的图形视图功能自带的,无需自己去实现,很方便。

游戏场景设计:

游戏背景包括背景色设计,方块游戏区域,提示方块出现区域,等。游戏开始时,新建提示框中的方块,且允许方块设置为当前焦点。设置好方块下降的速度,这个当然是利用定时器效果到达的。

判断是否到达满行的方法是，在给定的游戏区域每一个方块行进行检测，如果该行的方块数目达到了 10 个，则消行，消行直接调用 Qt 库函数，且将该行上面所有的方块都往下移一格。然后出现下一个方块组进行游戏。

游戏动态效果设置：

当一行的小方块达到了满行时，直接删掉看不到动态效果，如果使需要删掉的小方块先放大，后缩小，且给一定的模糊效果，这 2 者的结合就可以达到改行爆炸的动态效果了。这里的 animation 动态效果和 android 中的很类似。

游戏分数等级设置：

游戏中每消掉一行得分 1 千万，依次累加，这样看起来比较霸气，然后当得分为 1 亿时，自动进入第二等级，这时候游戏的速度变快，且场景图片也改变了。

程序中按钮等逻辑设置：

程序中有暂停，重新开始，返回主菜单，主菜单中又有开始，选项，帮助，退出等按钮，程序中还有分数，游戏等级和其它类型的提示文本等。这些逻辑功能都可以按照实际玩游戏的过程中去设置它们。

设置音效：

音效的设置采用 Qt 中支持的 Phonon 多媒体框架，可以直接采用 Phonon 类的 MediaObject 方法来创建声音对象，然后使用该类的 setCurrentSource 方法来设置对应的音乐文件，成功后就可以直接使用 play 来播放了。如果需要在界面中放置一条设置音量大小，这需要先使用这个类的 AudiOutput，然后用 createPath 来绑定音源和音频输出控件。

如果一首音乐播放完毕后它会自动进入暂停状态，直接调用 play() 方法也将无法进行播放，需要在播放完毕后使其进入停止状态，然后再调用 play() 方法。

如果要使音乐播放完后重复循环播放，则需要在音乐快结束的时候发送 aboutToFinish() 信号，连接好槽函数，在槽函数中设置音乐队列，队列的音乐源为需重复播放的乐曲。

程序启动画面的设计：

程序启动画面的设计主要是使用了 Qt 中的一个类 QSplashScreen 这个类，可以为这个类创建对象的时候传入一副图片，并且设置该对象的大小尺寸，接着调用 show() 方法即可。如果需要结束启动画面，则调用 finish() 方法，该方法的用来指定窗口初始化完成后结束启动画面。

知识点积累

c/c++ 知识点：

一个头文件中可以包含多个类，然后其对应的 cpp 文件中可以分别实现这些类的成员方法。

控制台程序是为了兼容 DOS 系统而设置的，这种程序的运行是没有自己的界面的，就像是在 DOS 窗口中执行一样。

在函数后用 const 表示不能改变类的成员。

枚举类型其实可以看做是整型的宏定义。

如果有时候的 Qt 库中自动强制类型转换出错，可能是其中某个数据类型的头文件没有被包含进去。

Qt 知识点:

QRectF 为一个 float 精度的矩形框。

QPainter 为一个绘图装置,我们可以在上面进行绘图,绘图的对象为 widget 或者 printer。

画笔是用来绘制线型的,主要样式有 3 个方面:笔帽,结合点和线型。

画刷是用来画填充封闭的几何图形的,主要有 2 个参数设置:颜色和样式。

如果编译时出现 Qt 自带的源代码处的错误提示,比如说: qtextoption.h 语法错误:缺少";"(在标识符"QtGuiModule"的前面)

定位到错误的地方时,是 Qt 系统的源码,按照道理系统源码时不会出错的,这时有能是自己写的一个类继承了系统提供的类,而这个类在头文件的括号中没有使用分号所致。

在 Qt 中使用定时器功能有 2 中方法,第一个是使用 QTimer 类。第二个是使用 QObject 子类的重载函数 timerEvent()。在使用第 1 种方法时需要将设置定时器的信号与槽连接,然后使用 start 函数来启动该定时器,使用 stop 方法来停止定时器。在使用第 2 种方法时,只需要调用系统的 startTimer()函数就可以了。

QTimer::singleShot()方法可以完成在时间间隔完成后执行后面的槽函数,相当于个直接的延时函数,使用起来应该还是蛮方便的。

颜色的第 4 个通道值可以理解为不透明度,即如果为 255 的话,就指的是完全不透明。

实验结果

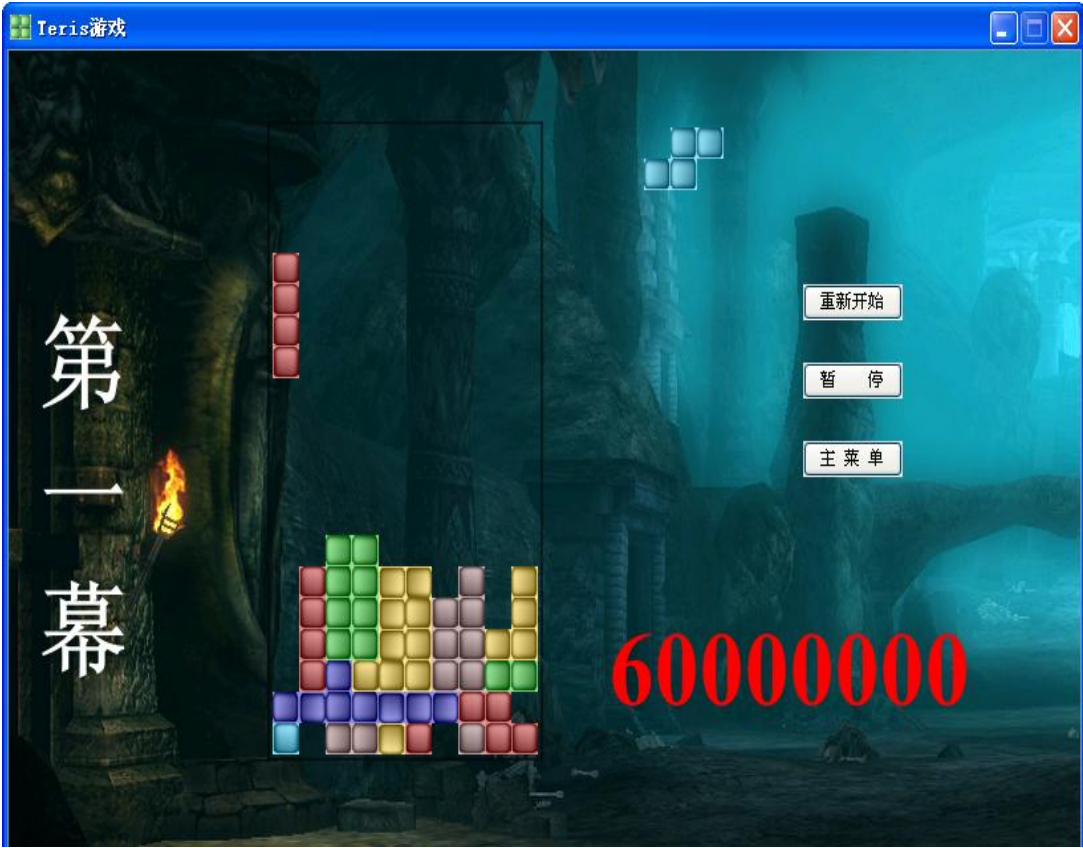
界面图:



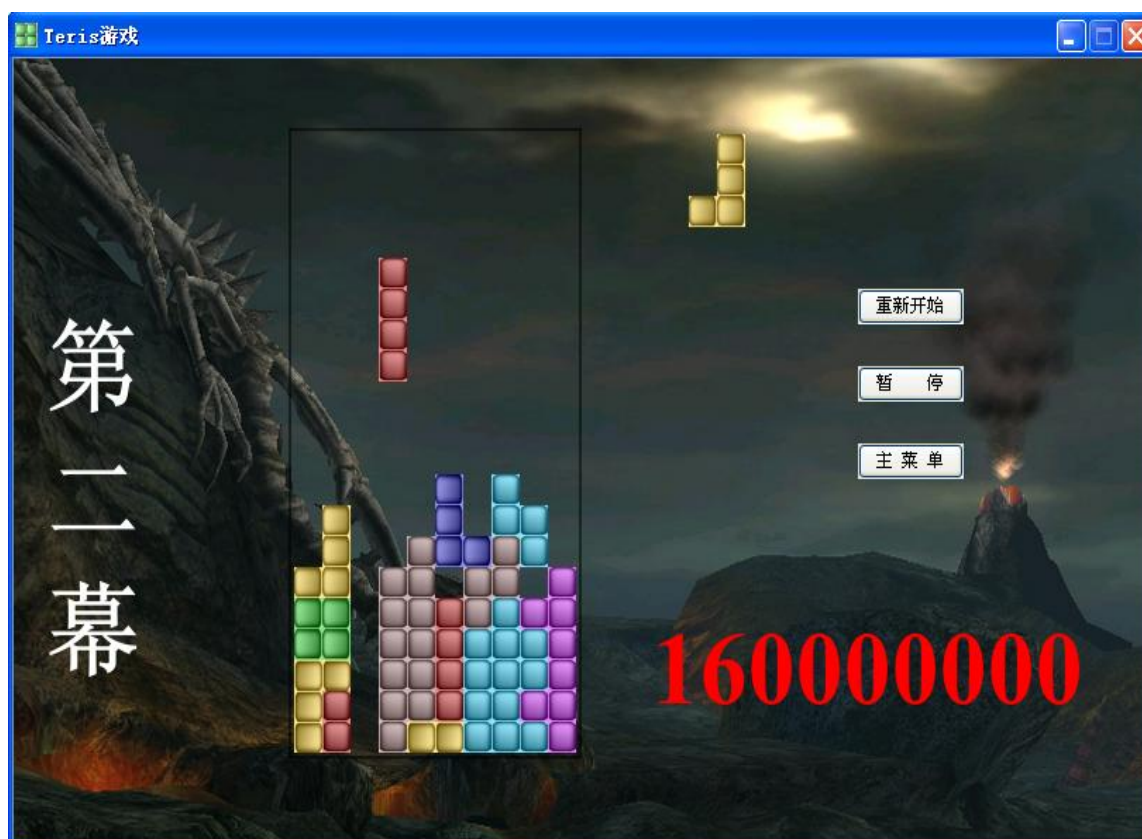
音效设置图：



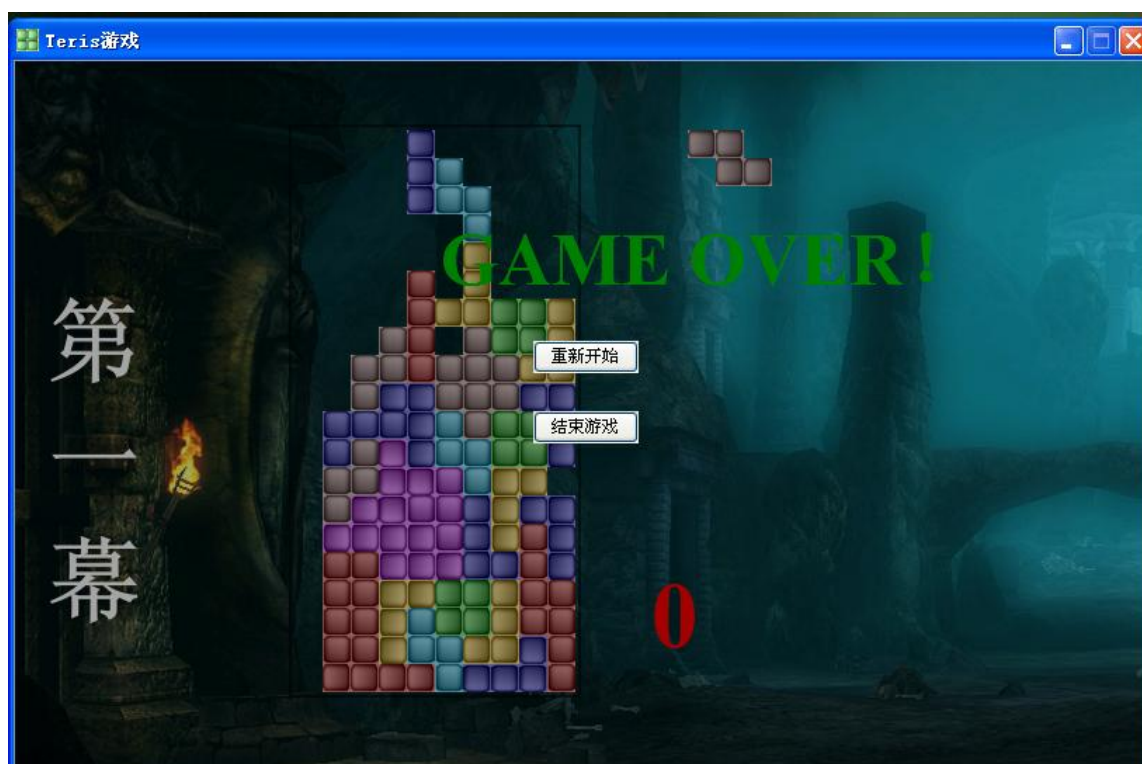
游戏第一幕截图：



游戏第二幕截图：



游戏结束界面:



实验主要部分代码及注释(附录有工程 **code** 下载地址):

main.cpp:

```

#include <QApplication>
#include <QTextCodec>
#include <QTime>
#include <QSplashScreen>

#include "myview.h"

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

    QApplication app(argc, argv);
    QTextCodec::setCodecForTr(QTextCodec::codecForLocale());
    //QTime 提供了闹钟功能
    qsrand(QTime(0, 0, 0).secsTo(QTime::currentTime())); //secsTo() 为返回当前的秒数
    QPixmap pix(":/images/logo.png");
    QSplashScreen splash(pix);
    splash.resize(pix.size());
    splash.show();
    app.processEvents(); //调用该函数的目的是为了使程序在启动画面的同时仍然能够响应鼠标事件


    MyView view; //主函数是直接调用的视图类
    view.show();

    splash.finish(&view); //当窗口 view 完成初始化工作后就结束启动画面

    return app.exec();
}

```

box.h:

```

#ifndef BOX_H
#define BOX_H

#include <QGraphicsObject>
#include <QGraphicsItemGroup>
```

```

//#include <QTimer>

class OneBox : public QGraphicsObject
{
public:
    OneBox(const QColor &color = Qt::red);
    QRectF boundingRect() const;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget);
    QPainterPath shape() const;

private:
    QColor brushColor;
};

class BoxGroup : public QObject, public QGraphicsItemGroup
{
    Q_OBJECT
public:
    enum BoxShape { IShape, JShape, LShape, OShape, SShape, TShape, ZShape,
RandomShape}; //8 中俄罗斯方框形状
    BoxGroup();
    QRectF boundingRect() const; //在函数后用 const 表示不能改变类的成员
    void clear_box_group(bool destroy_box = false);
    void create_box(const QPointF &point, BoxShape shape = RandomShape); //在函
数的申明处可以将参数设定为默认值，定义处不需要
    bool isColliding();
    BoxShape getCurrentShape() {return current_shape;} //获得当前俄罗斯方块的形状

protected:
    void keyPressEvent(QKeyEvent *event);

signals:
    void need_new_box();
    void game_finished();

public slots:
    void move_one_step();
    void startTimer(int interval);
    void stop_timer();

```



```
private:
    BoxShape current_shape;
    QTransform old_transform;
    QTimer *timer;

};

#endif // BOX_H
```

box.cpp:

```
#include "box.h"
#include <QPainter>
#include <QKeyEvent>
#include <QTimer>

//OneBox 是从 QGraphicsObject 继承而来的
OneBox::OneBox(const QColor &color) : brushColor(color) {

}

//该函数为指定后面的绘图区域的外边框
QRectF OneBox::boundingRect() const {

    qreal pen_width = 1;
    //小方块的边长为 20.5 像素
    return QRectF(-10-pen_width/2, -10-pen_width/2, 20+pen_width,
20+pen_width);

}

void OneBox::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget){

    //贴图，看起来有质感，否则单独用颜色去看，会感觉那些方块颜色很单一
    painter->drawPixmap(-10, -10, 20, 20, QPixmap(":/images/box.gif"));
    painter->setBrush(brushColor); //设置画刷颜色
    QColor penColor = brushColor;
    penColor.setAlpha(20); //将颜色的透明度减小，使方框边界和填充色直接能区分开
    painter->setPen(penColor); //色绘制画笔
    //这里画矩形框，框内填充部分用画刷画，框外线条用画笔画
```

```

    painter->drawRect(-10, -10, 20, 20); //画矩形框
}

//在局部坐标点上返回 item 的 shape, 但是好像没有其它地方调用了该函数
QPainterPath OneBox::shape() const{

    //QPainterPath 是一个绘图操作的容器
    QPainterPath path;
    path.addRect(-9.5, -9.5, 19, 19);
    return path;
}

//BoxGroup 是从 QGraphicsItemGroup, QObject 继承而来的
BoxGroup::BoxGroup() {

    setFlags(QGraphicsItem::ItemIsFocusable); //允许设置输入焦点
    old_transform = transform(); //返回当前 item 的变换矩阵, 当 BoxGroup 进行旋转后, 可以使用它来进行恢复
    timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(move_one_step()));
    current_shape = RandomShape;
}

QRectF BoxGroup::boundingRect() const {

    qreal pen_width = 1;
    return QRectF(-40-pen_width/2, -40-pen_width/2, 80+pen_width,
80+pen_width); //2*2 个小方块组成一个小方块组
}

void BoxGroup::keyPressEvent(QKeyEvent *event) {

    switch(event->key())
    {
        //向下键位坠落键
        case Qt::Key_Down:
            moveBy(0, 20); //moveBy 是系统自带的函数, 不需要我们自己去实现
            while(!isColliding()) {
                moveBy(0, 20);
            }
    }
}

```

```

        moveBy(0, -20); // 往回跳
        clear_box_group(); // 到达底部后就将当前方块组的 4 个 item 移除, 不销毁方块组
        emit need_new_box(); // 发射信号, 在 MyView 中接收
        break;
    case Qt::Key_Left:
        moveBy(-20, 0);
        if(isColliding()) {
            moveBy(20, 0);
        }
        break;
    case Qt::Key_Right:
        moveBy(20, 0);
        if(isColliding()) {
            moveBy(-20, 0);
        }
        break;
    // 实现小方块组变形
    case Qt::Key_Space:
        rotate(90); // 方块组的旋转也不需要自己实现, Qt 库中自带该方法
        if(isColliding())
            rotate(-90); // 变形后碰撞了, 就逆向变形回去
        break;
    }
}

// 检测是否有碰撞
bool BoxGroup::isColliding() {

    QList<QGraphicsItem *> item_list = childItems(); // 返回子 item 列表
    QGraphicsItem *item;
    foreach(item, item_list) {
        if(item->collidingItems().count() > 1) // collidingItems 返回与当前 item 碰撞的
        子 item 列表
            return true; // 代表至少有一个 item 发生了碰撞
    }
    return false;
}

// 将方块组从视图中移除掉, 如果有需要 (即参数为 true 的情况下) 则销毁掉
// 其本质是将所有的小方块从方块组中移除掉, 达到从视图中将方块组移除的目的
void BoxGroup::clear_box_group(bool destroy_box) {

```

```

QList<QGraphicsItem *> item_list = childItems();
QGraphicsItem *item;
foreach(item, item_list) {
    removeFromGroup(item); //将 item 从方块组中移除掉
    if(destroy_box) {
        OneBox *box = (OneBox*)item;
        box->deleteLater(); //当控制返回到事件循环时, 该目标被删除, 即销毁
    }
}
}

```

//创建俄罗斯方块组, 根据形状参数选择方块组的颜色和形状

```

void BoxGroup::create_box(const QPointF &point, BoxShape shape) {

    static const QColor color_table[7] = {
        QColor(200, 0, 0, 100), QColor(255, 200, 0, 100), QColor(0, 0, 200, 100),
        QColor(0, 200, 0, 100), QColor(0, 200, 255, 100), QColor(200, 0, 255, 100),
        QColor(150, 100, 100, 100)
    };

    int shape_id = shape; //Box_Shape 是枚举型, 其实也是整型, 因为它相当于整型的宏定义
    if(shape == RandomShape) {
        shape_id = qrand()%7; //随机取一个颜色
    }

    QColor color = color_table[shape_id]; //根据 id 选颜色
    QList<OneBox *> list;
    setTransform(old_transform); //恢复方块组前的变换矩阵
    for(int i = 0; i < 4; ++i) { //4 个小方块组成一个方块组
        OneBox *temp = new OneBox(color);
        list << temp; //将小方块加入 list 列表
        addToGroup(temp);
    }

    switch(shape_id) {
        case IShape:
            current_shape = IShape; //横着的一杆
            list.at(0)->setPos(-30, -10);
            list.at(1)->setPos(-10, -10);
            list.at(2)->setPos(10, -10);
            list.at(3)->setPos(30, -10);
            break;
        case JShape:
            current_shape = JShape; //J 型
            list.at(0)->setPos(10, -10);
            list.at(1)->setPos(10, 10);

```

```

list.at(2)->setPos(10, 30);
list.at(3)->setPos(-10, 30);
break;
case LShape:
current_shape = LShape; //L 型的方块组
list.at(0)->setPos(-10, -10);
list.at(1)->setPos(-10, 10);
list.at(2)->setPos(-10, 30);
list.at(3)->setPos(10, 30);
break;
case OShape: //田字型
current_shape = OShape;
list.at(0)->setPos(-10, -10);
list.at(1)->setPos(10, -10);
list.at(2)->setPos(-10, 10);
list.at(3)->setPos(10, 10);
break;
case SShape: //S 型
current_shape = SShape;
list.at(0)->setPos(10, -10);
list.at(1)->setPos(30, -10);
list.at(2)->setPos(-10, 10);
list.at(3)->setPos(10, 10);
break;
case TShape: //土字型
current_shape = TShape;
list.at(0)->setPos(-10, -10);
list.at(1)->setPos(10, -10);
list.at(2)->setPos(30, -10);
list.at(3)->setPos(10, 10);
break;
case ZShape: //Z 字型
current_shape = ZShape;
list.at(0)->setPos(-10, -10);
list.at(1)->setPos(10, -10);
list.at(2)->setPos(10, 10);
list.at(3)->setPos(30, 10);
break;
default: break;
}

```

setPos(point); //将准备好的俄罗斯方块放入指定的位置，然后进行碰撞检测

```
if(isColliding()) {
```

 //如果俄罗斯方块一出现后就发生了碰撞，因为它是从中间出来的，所以一开始不可能与左右两边发生碰撞，

//只能是与下面碰撞，因此如果发生了碰撞，说明游戏已经结束，就可以发送游戏结束信号了，且定时器停止。

```
        stop_timer();  
        emit game_finished();  
    }  
}
```

//这个是系统里的函数，本程序中是在主函数中启动的

//其实是该子类中的 timeEvent() 函数调用的

```
void BoxGroup::startTimer(int interval) {  
    timer->start(interval); //启动定时器并且设置定时器间隔，然后在 BoxGroup() 的构造函数  
    中设置了该定时器的信号与槽函数  
}
```

//每当定时器到时间了，小方块组就向下移一步

```
void BoxGroup::move_one_step() {  
    moveBy(0, 20); //该函数是父类的函数，这里指向下移动一个单位，因为向下为正坐标  
    if(isColliding()) { //发生碰撞的情况下  
        moveBy(0, -20);  
        clear_box_group(); //将方块组移除视图  
        emit need_new_box(); //发生信号通知程序需要新的方块组出现  
    }  
}
```

```
void BoxGroup::stop_timer() {  
  
    timer->stop(); //定时器停止  
}
```



myview.h:



```
#ifndef MYVIEW_H  
#define MYVIEW_H  
  
#include <QGraphicsView>  
#include <phonon>  
  
class BoxGroup;
```

```

class MyView : public QGraphicsView
{
    Q_OBJECT
public:
    explicit MyView(QWidget *parent = 0); //关键字 explicit 是为了防止隐式类型转换

public slots:
    void start_game();
    void clear_full_rows();
    void move_box();
    void game_over();
    void restart_game();
    void finish_game();
    void pause_game();
    void return_game();
    void about_to_finish();

protected:
    void keyPressEvent(QKeyEvent *event);

private:
    QGraphicsLineItem *top_line;
    QGraphicsLineItem *bottom_line;
    QGraphicsLineItem *left_line;
    QGraphicsLineItem *right_line;
    BoxGroup *box_group;
    BoxGroup *next_box_group;
    qreal game_speed;
    QList<int> rows;
    QGraphicsTextItem *game_score; //分数文本图形对象
    QGraphicsTextItem *game_level; //游戏等级文本图形对象
    QGraphicsWidget *mask_widget;
    //首页和游戏中需要用到的各种按钮
    QGraphicsWidget *start_button;
    QGraphicsWidget *finish_button;
    QGraphicsWidget *restart_button;
    QGraphicsWidget *pause_button;
    QGraphicsWidget *option_button;
    QGraphicsWidget *return_button;
    QGraphicsWidget *help_button;
    QGraphicsWidget *exit_button;
    QGraphicsWidget *show_menu_button;

    //显示人机交互的文本信息

```

```

    QGraphicsTextItem *game_welcome_text;
    QGraphicsTextItem *game_pause_text;
    QGraphicsTextItem *game_over_text;

    //添加背景音乐和消行声音
    Phonon::MediaObject *background_music;
    Phonon::MediaObject *clearrow_sound;

    void init_view();
    void init_game();
    void update_score(const int full_row_num = 0);

signals:

};

#endif // MYVIEW_H

```

myview.cpp:

```

#include "myview.h"
#include "box.h"
#include <QIcon>
#include <QPropertyAnimation>
#include <QGraphicsBlurEffect>
#include <QTimer>
#include <QPushButton>
#include <QGraphicsProxyWidget>
#include <QApplication>
#include <QLabel>
#include <QFileInfo>

static const qreal INITSSPEED = 500; //游戏的初始化速度
static const QString SOUNDPATH = "./sounds/";

MyView::MyView(QWidget *parent) :
    QGraphicsView(parent)
{
    init_view();
}

```

```

void MyView::init_view() {

    setRenderHint(QPainter::Antialiasing); //使用抗锯齿的方式渲染
    setCacheMode(CacheBackground); //设置缓存背景，这样可以加快渲染速度
    setWindowTitle(tr("Teris 游戏"));
    setWindowIcon(QIcon(":/images/icon.png")); //设置标题处的图标
    setMinimumSize(810, 510); //2 者设置成一样说明视图尺寸不能再更改
    setMaximumSize(810, 510);

    QGraphicsScene *scene = new QGraphicsScene; //新建场景指针
    scene->setSceneRect(5, 5, 800, 500); //场景大小
    scene->setBackgroundBrush(QPixmap(":/images/background.png"));
    setScene(scene); //设置场景

    //俄罗斯方块可移动区域外界的 4 条线, 与外界预留 3 个像素是为了方便进行碰撞检测
    top_line = scene->addLine(197, 47, 403, 47);
    bottom_line = scene->addLine(197, 453, 403, 453);
    left_line = scene->addLine(197, 47, 197, 453);
    right_line = scene->addLine(403, 47, 403, 453);

    //添加当前方块组
    box_group = new BoxGroup; //通过新建 BoxGroup 对象间接达到调用 box 的 2 个类
    connect(box_group, SIGNAL(need_new_box()), this, SLOT(clear_full_rows()));
    connect(box_group, SIGNAL(game_finished()), this, SLOT(game_over()));
    scene->addItem(box_group);

    //添加提示方块组
    next_box_group = new BoxGroup;
    scene->addItem(next_box_group);

    game_score = new QGraphicsTextItem(0, scene); //文本的父 item 为对应的场景
    game_score->setFont(QFont("Times", 50, QFont::Bold)); //为文本设置字体
    game_score->setPos(450, 350); //分数在场景中出现的位置

    game_level = new QGraphicsTextItem(0, scene);
    game_level->setFont(QFont("Times", 50, QFont::Bold));
    game_level->setPos(20, 150);

    //开始游戏
    // start_game();

    //初始状态时不显示游戏区域, 不显示游戏分数和游戏等级
    top_line->hide();
    bottom_line->hide();
    left_line->hide();
    right_line->hide();
}

```

```

game_score->hide();
game_level->hide();

//添加黑色遮罩
QWidget *mask = new QWidget;
mask->setAutoFillBackground(true);
mask->setPalette(QPalette(QColor(0, 0, 0, 50))); //alpha 为不透明度
mask->resize(900, 600);

//addWidget() 函数的返回值是 QGraphicsProxyWidget，如果不添加相应的头文件，则此处会
报错
mask_widget = scene->addWidget(mask);
mask_widget->setPos(-50, -50);
mask_widget->setZValue(1); //该层薄纱放在原图的上面，这里有点类似于 opengl 中的 3 维
绘图

//选项面板
QWidget *option = new QWidget;
//将关闭按钮放在 option 上
QPushButton *option_close_button = new QPushButton(tr("关 闭"), option); //
第 2 个参数为按钮所在的 widget
//设置按钮的字体颜色是白色
QPalette palette;
palette.setColor(QPalette::ButtonText, Qt::black); //第一个参数调色版的 role，
这里指的是按钮字体颜色
option_close_button->setPalette(palette);
//设置关闭按钮的位置，和单击后的响应
option_close_button->move(120, 300);
connect(option_close_button, SIGNAL(clicked()), option, SLOT(hide())); //单
击后消失

option->setAutoFillBackground(true);
option->setPalette(QPalette(QColor(0, 0, 0, 180)));
option->resize(300, 400);
QGraphicsWidget *option_widget = scene->addWidget(option);
option_widget->setPos(250, 50);
option_widget->setZValue(3);
option_widget->hide();

//帮助面板
QWidget *help = new QWidget;
QPushButton *help_close_button = new QPushButton(tr("帮 助"), help);
help_close_button->setPalette(palette);
help_close_button->move(120, 300);

```



```

connect(help_close_button, SIGNAL(clicked()), help, SLOT(hide()));

help->setAutoFillBackground(true);
help->setPalette(QPalette(QColor(0, 0, 0, 180)));
help->resize(300, 400);
QGraphicsWidget *help_widget = scene->addWidget(help);
help_widget->setPos(250, 50);
help_widget->setZValue(3);
help_widget->hide();

//游戏欢迎文本
game_welcome_text = new QGraphicsTextItem(0, scene); //第一个参数为文本内容, 第二个参数为父 item
game_welcome_text->setHtml(tr("<font color=green>Tetris 游戏</font>"));
game_welcome_text->setFont(QFont("Times", 40, QFont::Bold));
game_welcome_text->setPos(300, 100);
game_welcome_text->setZValue(2); //放在第 2 层

//游戏暂停文本
game_pause_text = new QGraphicsTextItem(0, scene); //第一个参数为文本内容, 第二个参数为父 item
game_pause_text->setHtml(tr("<font color=green>游戏暂停中! </font>"));
game_pause_text->setFont(QFont("Times", 40, QFont::Bold));
game_pause_text->setPos(300, 100);
game_pause_text->setZValue(2); //放在第 2 层
game_pause_text->hide();

//游戏结束文本
game_over_text = new QGraphicsTextItem(0, scene); //第一个参数为文本内容, 第二个参数为父 item
game_over_text->setHtml(tr("<font color=green>GAME OVER! </font>"));
game_over_text->setFont(QFont("Times", 40, QFont::Bold));
game_over_text->setPos(300, 100);
game_over_text->setZValue(2); //放在第 2 层
game_over_text->hide();

// 游戏中使用的按钮

QPushButton *button1 = new QPushButton(tr("开 始"));
QPushButton *button2 = new QPushButton(tr("选 项"));
QPushButton *button3 = new QPushButton(tr("帮 助"));
QPushButton *button4 = new QPushButton(tr("退 出"));
QPushButton *button5 = new QPushButton(tr("重新开始"));
QPushButton *button6 = new QPushButton(tr("暂 停"));

```

```

QPushButton *button7 = new QPushButton(tr("主 菜 单"));
QPushButton *button8 = new QPushButton(tr("返回游戏"));
QPushButton *button9 = new QPushButton(tr("结束游戏"));

connect(button1, SIGNAL(clicked()), this, SLOT(start_game()));
connect(button2, SIGNAL(clicked()), option, SLOT(show()));
connect(button3, SIGNAL(clicked()), help, SLOT(show()));
connect(button4, SIGNAL(clicked()), qApp, SLOT(quit())); //此处槽函数的接收对象为应用程序本身

connect(button5, SIGNAL(clicked()), this, SLOT(restart_game()));
connect(button6, SIGNAL(clicked()), this, SLOT(pause_game()));
connect(button7, SIGNAL(clicked()), this, SLOT(finish_game()));
connect(button8, SIGNAL(clicked()), this, SLOT(return_game())); //返回主菜单
connect(button9, SIGNAL(clicked()), this, SLOT(finish_game()));

start_button = scene->addWidget(button1); //restart_button 并不是 QPushButton 类型, 而是 QGraphicsItem 类型, 后面的类似
option_button = scene->addWidget(button2);
help_button = scene->addWidget(button3);
exit_button = scene->addWidget(button4);
restart_button = scene->addWidget(button5);
pause_button = scene->addWidget(button6);
show_menu_button = scene->addWidget(button7);
return_button = scene->addWidget(button8);
finish_button = scene->addWidget(button9);

//设置位置
start_button->setPos(370, 200);
option_button->setPos(370, 250);
help_button->setPos(370, 300);
exit_button->setPos(370, 350);
restart_button->setPos(600, 150);
pause_button->setPos(600, 200);
show_menu_button->setPos(600, 250);
return_button->setPos(370, 200);
finish_button->setPos(370, 250);

//将这些按钮都放在 z 方向的第二层
start_button->setZValue(2);
option_button->setZValue(2);
help_button->setZValue(2);
exit_button->setZValue(2);
restart_button->setZValue(2);
return_button->setZValue(2);

```

```

finish_button->setZValue(2);

//一部分按钮隐藏起来
restart_button->hide();
finish_button->hide();
pause_button->hide();
show_menu_button->hide();
return_button->hide();

//设置声音
background_music = new Phonon::MediaObject(this); //背景音乐对象
cleararrow_sound = new Phonon::MediaObject(this); //消行声音对象
//AudioOutput 是一个将数据送到音频输出的设备
Phonon::AudioOutput *audio1 = new Phonon::AudioOutput(Phonon::MusicCategory,
this);
Phonon::AudioOutput *audio2 = new Phonon::AudioOutput(Phonon::MusicCategory,
this);
Phonon::createPath(background_music, audio1); //
Phonon::createPath(cleararrow_sound, audio2); //绑定音源和音频输出控件

Phonon::VolumeSlider *volume1 = new Phonon::VolumeSlider(audio1, option); //
参数 1 为音频输出设备，参数 2 为对应的父 widget
Phonon::VolumeSlider *volume2 = new Phonon::VolumeSlider(audio2, option);
QLabel *volume_label1 = new QLabel(tr("音乐声: "), option);
QLabel *volume_label2 = new QLabel(tr("音效声: "), option);
volume1->move(100, 100);
volume2->move(100, 200);
volume_label1->move(65, 105);
volume_label2->move(60, 205);

//音乐播放完后发射信号触发 about_to_finish() 槽函数
connect(background_music, SIGNAL(aboutToFinish()), this,
SLOT(about_to_finish()));

// 因为播放完会进入暂停状态，再调用 play() 将无法进行播放，需要在播放完后使其进入停
止状态
connect(cleararrow_sound, SIGNAL(finished()), cleararrow_sound, SLOT(stop()));
background_music->setCurrentSource(Phonon::MediaSource(SOUNDPATH +
"background.mp3"));
cleararrow_sound->setCurrentSource(Phonon::MediaSource(SOUNDPATH +
"cleararrow.mp3"));
background_music->play(); //初始化的时候直接播放背景音乐
}

```

//开始游戏

```
void MyView::start_game() {  
  
    game_welcome_text->hide();  
    start_button->hide();  
    option_button->hide();  
    help_button->hide();  
    exit_button->hide();  
    mask_widget->hide();  
    init_game();  
}
```

//初始化游戏，为什么要分2个函数来写？

```
void MyView::init_game() {  
  
    box_group->create_box(QPointF(300, 70)); //创建方块组,在中间位置处出现  
    box_group->setFocus(); //设置人机交互焦点，这样就可以使用键盘来控制它  
    box_group->startTimer(INITSSPEED); //启动定时器  
    game_speed = INITSSPEED; //游戏速度，暂停时需要用到  
    next_box_group->create_box(QPoint(500, 70)); //创建提示方块组  
  
    scene()->setBackgroundBrush(QPixmap(":/images/background01.png"));  
    game_score->setHtml(tr("<font color = red >0</font>"));  
    game_level->setHtml(tr("<font color = white>第<br>一<br>幕</font>")); //br 为  
    换行  
  
    restart_button->show();  
    pause_button->show();  
    show_menu_button->show();  
    game_score->show();  
    game_level->show();  
    top_line->show();  
    bottom_line->show();  
    left_line->show();  
    right_line->show();  
    // 可能以前返回主菜单时隐藏了 boxGroup  
    box_group->show();  
  
    //设置游戏开始的背景音乐  
    background_music->setCurrentSource(Phonon::MediaSource(SOUNDPATH +  
    "background01.mp3"));  
    background_music->play();  
}
```

```

}

void MyView::clear_full_rows() {

    for(int y = 429; y > 50; y -= 20) {
        //每隔 20 行取一个 item 出来，括号里面的参数不能弄错，否则没有方块消失的效果
        QList<QGraphicsItem *> list = scene()->items(199, y, 202, 22,
Qt::ContainsItemShape); //返回指定区域内所有可见的 item
        if(list.count() == 10) { //如果一行已满，则销毁该行的所有小方块
            foreach(QGraphicsItem *item, list) {
                OneBox *box = (OneBox *) item;
                // box->deleteLater();
                /*采用动态效果消失方块行*/
                QGraphicsBlurEffect *blur_effect = new QGraphicsBlurEffect; //创建
模糊效果对象
                box->setGraphicsEffect(blur_effect); //给每个小方块添加模糊设置
                QPropertyAnimation *animation = new QPropertyAnimation(box,
"scale"); //添加动态效果，尺寸变换效果
                animation->setEasingCurve(QEasingCurve::OutBounce); //为动态效果设
置缓冲取曲线，是用来插值的
                animation->setDuration(250); //持续 250 毫秒
                animation->setStartValue(4); //其实尺寸
                animation->setEndValue(0.25); //结束尺寸
                animation->start(QAbstractAnimation::DeleteWhenStopped); //动画结
束后删除该类
                connect(animation, SIGNAL(finished()), box,
SLOT(deleteLater())); //动画结束后才调用小方块销毁函数
                // cleararrow_sound->play();

            }
            rows << y; //将满行的行号保存到 rows 中
        }
    }

    //如果满行，则下移上面的方块
    if(rows.count() > 0) {
        cleararrow_sound->play();
        QTimer::singleShot(400, this, SLOT(move_box())); //只执行一次定时器，等动态
效果完后再下移上面的方块
        // cleararrow_sound->play();
    }
    else {

```



```

        //没有满行，则新出现提示方块，且提示方块出更新新的提示方块
        box_group->create_box(QPointF(300, 70),
next_box_group->getCurrentShape());
        next_box_group->clear_box_group(true);
        next_box_group->create_box(QPointF(500, 70)); //
    }

}

void MyView::move_box() {

    for(int i = rows.count(); i > 0; --i) {
        int row = rows.at(i-1); //取出满行的行号,从最上面的位置开始
        //取出从区域上边界到当前满行之间所形成的矩形区域
        foreach(QGraphicsItem *item, scene()->items(199, 49, 202, row-47,
Qt::ContainsItemShape)) {
            item->moveBy(0, 20);
        }
    }

    //更新分数
    update_score(rows.count());
    //出现新的方块组
    rows.clear();
    box_group->create_box(QPointF(300, 70),
next_box_group->getCurrentShape());
    next_box_group->clear_box_group(true);
    next_box_group->create_box(QPointF(500, 70));
}

void MyView::update_score(const int full_row_num) {

    long score = full_row_num*1000000; //每消一行的一千万分，比较霸气
    int current_score = game_score->toPlainText().toInt();
    current_score += score;
    game_score->setHtml(tr("<font color = red>%1</font>").arg(current_score));
    if(current_score >= 100000000) {
        game_level->setHtml(tr("<font color = white>第<br>二<br>幕</font>"));
        scene()->setBackgroundBrush(QPixmap(":/images/background02.png"));
        game_speed = 300;
        box_group->stop_timer(); //重新设置定时器参数
        box_group->startTimer(game_speed);
    }
}

```

```

        if
(QFileInfo(background_music->currentSource().fileName()).baseName() !=
"background02") {
            background_music->setCurrentSource(Phonon::MediaSource(SOUNDPATH +
"background02.mp3"));
            background_music->play();
        }
    }
}

void MyView::game_over() {

    pause_button->hide();
    show_menu_button->hide();
    mask_widget->show();
    game_over_text->show();
    restart_button->setPos(370, 200);
    finish_button->show();

}

void MyView::restart_game()
{
    mask_widget->hide();
    game_over_text->hide();
    finish_button->hide();
    restart_button->setPos(600, 150);

    //销毁当前方块组和当前方块中的所有小方块
    next_box_group->clear_box_group(true);
    box_group->clear_box_group();
    box_group->hide();
    foreach(QGraphicsItem *item, scene()->items(199, 49, 202,
402,Qt::ContainsItemBoundingRect)) {
        scene()->removeItem(item);
        OneBox *box = (OneBox*)item;
        box->deleteLater();
    }
    init_game();
}

void MyView::finish_game()

```

```

{
    game_over_text->hide();
    finish_button->hide();
    restart_button->setPos(600, 150);
    restart_button->hide();
    pause_button->hide();
    show_menu_button->hide();
    game_score->hide();
    game_level->hide();
    top_line->hide();
    bottom_line->hide();
    left_line->hide();
    right_line->hide();

    next_box_group->clear_box_group(true);
    box_group->clear_box_group();
    box_group->hide();
    foreach(QGraphicsItem *item, scene()->items(199, 49, 202,
402, Qt::ContainsItemBoundingRect)) {
        scene()->removeItem(item);
        OneBox *box = (OneBox*)item;
        box->deleteLater();
    }

    mask_widget->show();
    game_welcome_text->show();
    start_button->show();
    option_button->show();
    help_button->show();
    exit_button->show();
    scene()->setBackgroundBrush(QPixmap(":/images/background.png"));

    //游戏结束时更改背景音乐
    background_music->setCurrentSource(Phonon::MediaSource(SOUNDPATH +
"background.mp3"));
    background_music->play();
}

void MyView::pause_game()
{
    box_group->stop_timer(); //中断游戏最主要的是停止方块下移的定时器工作
    restart_button->hide();
    pause_button->hide();
}

```

```

        show_menu_button->hide();
        mask_widget->show();
        game_pause_text->show();
        return_button->show();
    }

void MyView::return_game()
{
    return_button->hide();
    game_pause_text->hide();
    mask_widget->hide();
    restart_button->show();
    pause_button->show();
    show_menu_button->show();
    box_group->startTimer(game_speed);
}

void MyView::about_to_finish()
{
    background_music->enqueue(background_music->currentSource()); //设置后续的播放序列为当前的音频文件，达到重复播放的目的
}

void MyView::keyPressEvent(QKeyEvent *event) {
    if(pause_button->isVisible()) //当屏幕中可以看到暂停按钮时，代表游戏还在运行，这时候的焦点给下降中的方块组
        box_group->setFocus();
    else
        box_group->clearFocus(); //如果游戏暂停，则方块组不能获得焦点
    QGraphicsView::keyPressEvent(event);
}

```



实验总结

通过本次实验，了解到了 Qt 中图像视图的基本使用。在本程序中并没有使用复杂的算法，因为 Qt 的图形视图框架对很多方法进行了封装，比如说碰撞检测功能，图形项组，图形旋转，移动 item，移除场景，销毁 item 等。

参考资料

<http://www.yafeilinux.com/>

附录：实验工程 code 下载。