实验六 俄罗斯方块游戏

1.程序介绍

该程序为名称为俄罗斯方块游戏,设计的思路为构建一个具有可玩性的俄罗斯方块游戏。俄罗斯方块是由七种四格骨牌构成，全部都由四个方块组成。开始时，一个随机的方块会从区域上方开始缓慢继续落下。落下期间，玩家可以以 90 度为单位旋转方块，以格子为单位左右移动方块，或让方块加速落下。当方块下落到区域最下方或着落到其他方块上无法再向下移动时，就会固定在该处，然后一个新的随机的方块会出现在区域上方开始落下。当区域中某一横行的格子全部由方块填满时，则该列会被消除并成为玩家的得分。同时消除的行数越多，得分指数级上升。在游戏中的目的就是尽量得分。当固定的方块堆到区域最顶端而无法消除层数时，游戏就会结束。

2.操作说明

2.1 基本游戏规则

一个用于摆放小型正方形的平面虚拟场地，其标准大小：行宽为 10，列高为 20，以每个小正方形为单位。2、一组由 4 个小型正方形组成的规则图形，英文称为 Tetromino，中文通称为方块共有 7 种，分别以 S、Z、L、J、I、O、T 这 7 个字母的形状来命名。

玩家可以做的操作有：以 90 度为单位旋转方块，以格子为单位左右移动方块，让方块加速落下。

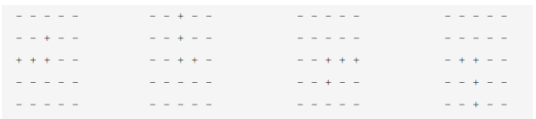
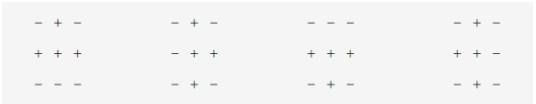
方块移到区域最下方或是着地到其他方块上无法移动时，就会固定在该处，而新的方块出现在区域上方开始落下。

当区域中某一列横向格子全部由方块填满,则该列会消失并成为玩家的得分。同时删除的列数越多，得分指数上升。

当固定的方块堆到区域最上方而无法消除层数时，则游戏结束。

俄罗斯方块类型如下:

表格 1 俄罗斯方块类型及表示

类型	功能	描述
I	一次最多消除四层	
J	最多消除三层，或消除二层	
L	最多消除三层，或消除二层	
O	消除一至二层	
S	最多二层，容易造成孔洞	
Z	最多二层，容易造成孔洞	
T	最多三层	

2.2 游戏界面

页面布局为右边提示窗口，左边游戏界面，致敬经典 PSP 游戏风格。在打开游戏会有欢快的游戏背景音乐。

图 1 游戏页面

2.3 游戏开始

整体游戏可以上下左右来控制，左右移动，上变换，下加速，空格暂停。实践体验如图所示：

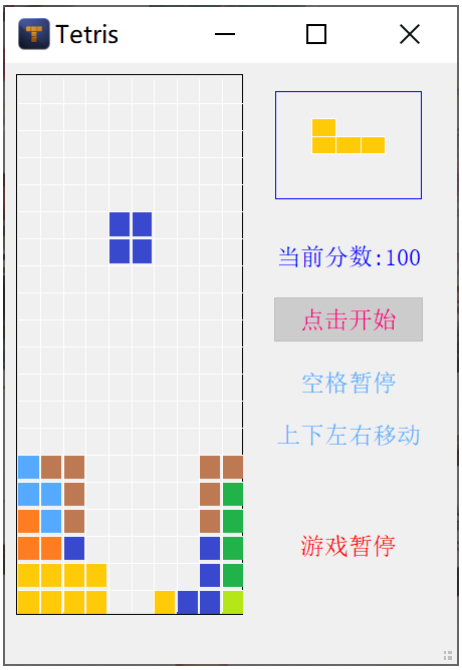


图 2 游戏暂停状态

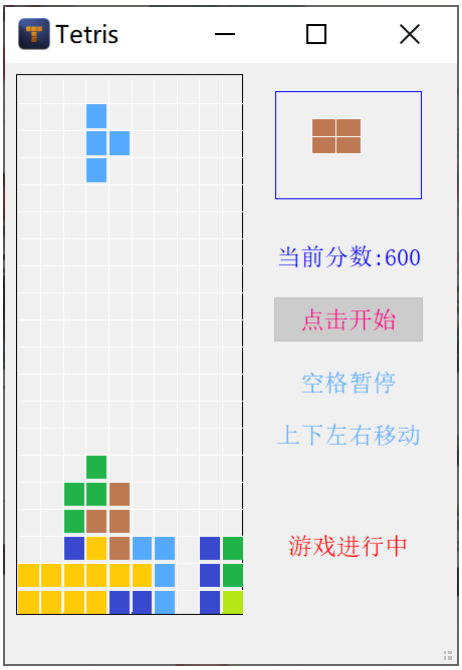


图 3 游戏进行时

2.4 游戏结束

游戏结束提示分数，可以选择继续游戏重新开始。

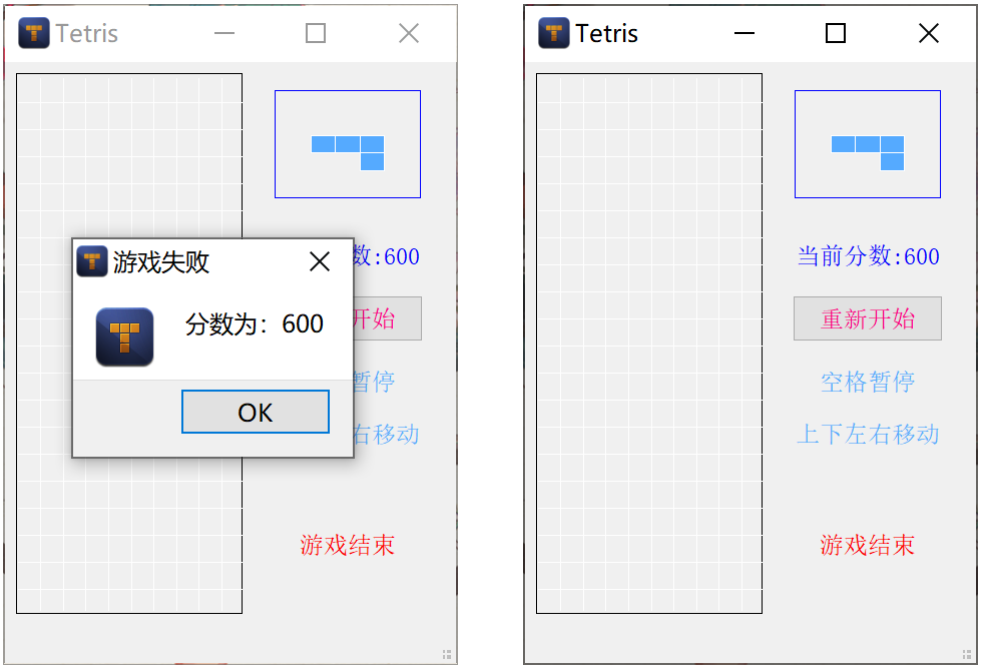


图 4 游戏结束展示

3.设计理念

3.1 设计目标

俄罗斯方块其实就是一副能够实现人机交互的动画，可以让人来控制四格拼版个的图形样式和位置的呈现。基本功能包括：游戏主界面显示模块、方块及数据显示模块、方块移动控制模块、游戏界面颜色控制模块、游戏进度、等级控制模块等。本项目结构如下：其运行界面如图所示。

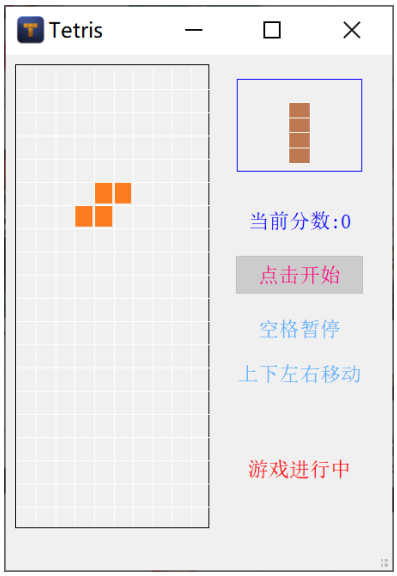


图 5 俄罗斯方块页面

3.2 设计分析和算法分析

3.2.1 页面搭建

这里采用 Qt 框架自带的 UI 界面进行美化排布，效果如下图所示，利用 UI 自动生成按钮等类，在页面美化中，添加样本样式，利用 Qt 自带的 QSS 进行美化（操作如前端 CSS 类似）。

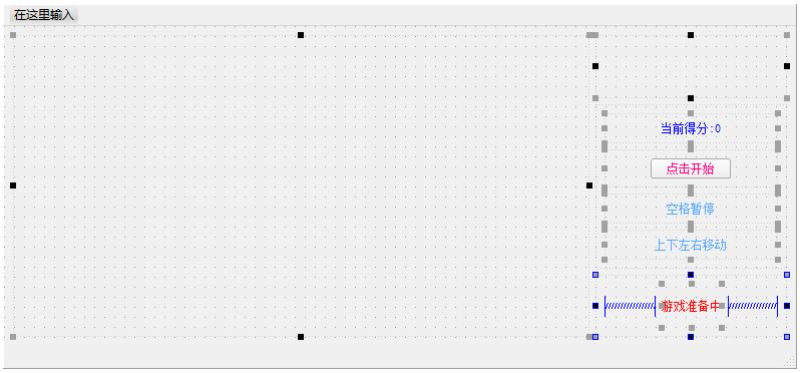


图 6UI 界面美化

3.2.2 页面生成

利用 UI 生成的类，进行类的升级关联绑定，生成类如图 13 所示，在 mainwindow.cpp 析构函数中进行绑定操作。代码如下：

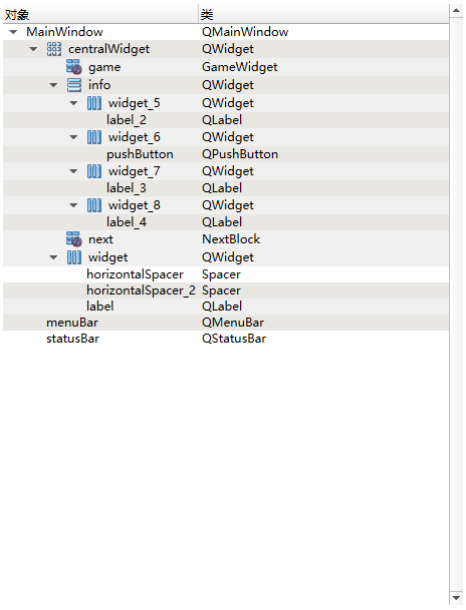
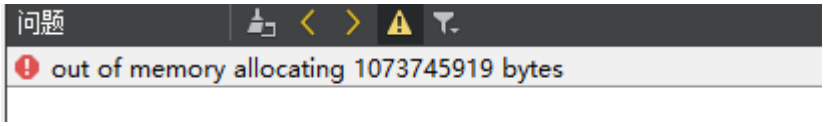


图 7UI 生成类

2.2.3 资源的导入

这里因为涉及音乐处理，资源文件过大，采用二进制注册的方法，先导入资

源生成 res.qrc 文件。如果直接运行会报错，如图所示。



由于资源文件 qrc 过大，超出分配的内存范围，因此我们需要利用二进制资源，而生成二进制资源就需要我们刚刚的 qrc 文件利用 cmd 打开终端，定位到 res.qrc 的目录下，输入命令 rcc -binary .\res.qrc -o res.rcc 即可。

```
#include "mainwindow.h"
#include <QApplication>
#define GAME_RES_PATH  "./res.rcc" //rcc 文件路径


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


    //注册外部的二进制资源文件
    QResource::registerResource(GAME_RES_PATH);


    QApplication a(argc, argv);
    MainWindow w;
    w.show();


    return a.exec();
}
```

代码块 1 主函数调用注册二进制

2.2.4 控制类构建

这里实现俄罗斯方块的基本游戏规则，生成每个方块的数组和他的变换，生成方块函数采用随机数访问数组下标，控制边界检查，始终在访问范围内，每次下移动操作的实现和游戏消除以及结束检查。为了方便预测值传给主游戏界面，

这里重载了等于=操作符。Control 类头文件及实现代码如下：

```
#ifndef CONTROL_H
#define CONTROL_H
#include<QVector>
#include<QPoint>
#include<QDebug>

class Control
{
public:
    Control();
    //当前处于的 x 坐标
    int x;
    //当前处于的 y 坐标
    int y;
    //方块的类型
    int type;
    //方块类型变成了那种了
    int process;
    //颜色类型
    int color;



    static QVector<QVector<int>> map; //map 类  地图信息
    static QVector<QVector<QVector<QPoint>>> block; //留存砖块形状 配合 control 类生成方块
    static QVector<QVector<int>> colorMap;

    //生成新的砖块
```

```
    void newBlock();

    //砖块发生移动的时候 检查下标
    bool indexCheck(int moveX,int moveY,int newProcess);

    //移动操作
    void move(int type);

    //垃圾回收 即发生移动过后 清楚上一步的痕迹
    void gc();

    //重新绘制
    void rePaint();

    //判断清除一行
    void clearRow();

    //重置等于操作
    void operator=(Control&);

    void gameOver();
};

#endif // CONTROL_H
```

代码块 2 Control 类头文件

```
#include "control.h"
#include<QRandomGenerator>
#include"gamewidget.h"
#include<QTimer>
```

```
#include"mainwindow.h"

//方块类型
QVector<QVector<QVector<QPoint>>> Control::block=
{
  {
      //四种变换
      {{0, 0}, {1, 0}, {1, 1}, {1, 2}},  // *
      {{2, 0}, {0, 1}, {1, 1}, {2, 1}},  // *
      {{1, 0}, {1, 1}, {1, 2}, {2, 2}},  // ** L 类型方块
      {{0, 1}, {1, 1}, {2, 1}, {0, 2}}
  },//这个是 L
  {
      {{1, 0}, {1, 1}, {1, 2}, {1, 3}}, // *
      {{0, 1}, {1, 1}, {2, 1}, {3, 1}}, // *
      {{1, 0}, {1, 1}, {1, 2}, {1, 3}}, // *
      {{0, 1}, {1, 1}, {2, 1}, {3, 1}}  // *  直方块
  },//这个是直线
  {
      {{1, 0}, {2, 0}, {1, 1}, {1, 2}}, //  *
      {{0, 1}, {1, 1}, {2, 1}, {2, 2}}, //  *
      {{1, 0}, {1, 1}, {0, 2}, {1, 2}}, // **  反 L
      {{0, 0}, {0, 1}, {1, 1}, {2, 1}}
  },//这个是反 L
  {
      {{0, 0}, {1, 0}, {2, 0}, {1, 1}},
      {{0, 0}, {0, 1}, {0, 2}, {1, 1}}, //  *
      {{0, 2}, {1, 2}, {2, 2}, {1, 1}},//  ***  就这个类似三角形的方块
      {{2, 0}, {2, 1}, {2, 2}, {1, 1}}
  },//这个是 土
  {
      {{0, 0}, {1, 0}, {0, 1}, {1, 1}},
      {{0, 0}, {1, 0}, {0, 1}, {1, 1}}, // **
      {{0, 0}, {1, 0}, {0, 1}, {1, 1}}, // **  正方形方块
      {{0, 0}, {1, 0}, {0, 1}, {1, 1}}
  },//这个是田
  {
      {{1, 1}, {2, 1}, {2, 0}, {1, 2}},
      {{1, 1}, {0, 1}, {1, 2}, {2, 2}},  //   *
      {{1, 1}, {2, 1}, {2, 0}, {1, 2}}, //   **
      {{1, 1}, {0, 1}, {1, 2}, {2, 2}}  //   *   旋转是 Z 的方块
  },//这个是 Z
```

```
    {
        {{0, 0}, {0, 1}, {1, 1}, {1, 2}},//   *
        {{0, 1}, {1, 1}, {1, 0}, {2, 0}},//   **
        {{0, 0}, {0, 1}, {1, 1}, {1, 2}},//    * 旋转式反 Z 的方块
        {{0, 1}, {1, 1}, {1, 0}, {2, 0}}
    },//这个是反 Z
};



QVector<QVector<int>> Control::map;
QVector<QVector<int>> Control::colorMap;

Control::Control()
{
}



//生成砖块
void Control::newBlock()
{
        x = 3;
        y = 0;

        //Qt5 新随机数方法 返回[0,7)
        type=QRandomGenerator::global()->bounded(0,7);
        process=QRandomGenerator::global()->bounded(0,4);

        qDebug()<<type<<process;
        color=QRandomGenerator::global()->bounded(0,8);
}



// moveX 代表 x 坐标位移大小 moveY 代表 moveY 坐标大小 newProcess 代表用户想要进行砖块转换
bool Control::indexCheck(int moveX,int moveY,int newProcess)
{

    for(int i=0;i<4;i++)
    {
        //i<4 因为每种砖块都是由 4 个构成
        int tempx=this->x+block[this->type][newProcess][i].x()+moveX;
        int tempy=this->y+block[this->type][newProcess][i].y()+moveY;
```

```
        //条件说明 x 超过边界 y 超过边界 想要移动的地方已经具有砖块 都返回 false 不允许当前移动操作
        if(tempx<0||tempx>9||tempy<0||tempy>19||map[tempx][tempy])
            return false;
    }
    return true;
}



void Control::move(int type)
{
    //标志是否停止
    bool stopFlag=false;
    //移除刚才的痕迹 进行下一步绘制
    gc();

    //1 代表想上 即方块进行变化
    if(type==1){
        int newProcess=(this->process+1)%4;
        if(indexCheck(0,0,newProcess))
        {
            this->process=newProcess;
        }
    }
    //2 代表向下移动 如果落地 新建状况 并且检测是否有行可以消除
    else if(type==2){
        stopFlag=indexCheck(0,1,this->process);
        if(stopFlag)
        {
            y++;
        }
        else
        {
            //回溯一次 下面的 paint 准备绘制新方块
            rePaint();
            //有方块落地 检测是否可以消行
            clearRow();
            *this=GameWidget::anotherCon;
        }
    }
```

```
    //3 代表向左移动
    else if(type==3)
    {
        if(indexCheck(-1,0,this->process))
        {
            x--;
        }
    }
    //4 代表向右移动
    else if(type==4)
    {
        if(indexCheck(1,0,this->process))
        {
            x++;
        }
    }
    rePaint();
}

//垃圾回收 即发生移动过后 清楚上一步的痕迹
void Control::gc(){
    for(int i=0;i<4;i++)
    {

map[this->x+block[this->type][this->process][i].x()][this->y+block[this->type][this->proc
ess][i].y()]=0;
    }
}



void Control::rePaint()
{
    for(int i=0;i<4;i++)
    {
        int tempx=this->x+block[this->type][this->process][i].x();
        int tempy=this->y+block[this->type][this->process][i].y();

        //新砖块刷新 可地图已经有了内容 即死亡
        if(map[tempx][tempy])
        {
            GameWidget::timer->stop();
            GameWidget::gameOver=true;
```

```
            MainWindow::running=false;
        }
        else
        {
            colorMap[tempx][tempy]=this->color;
            map[tempx][tempy] = 1;
        }
    }
}


void Control::clearRow(){
    int i,j;
    //因为是按 x y 格式进行标记的 所以删除方式比较特殊
    for(j=0;j<map[0].size();j++)
    {
        int flag1=true;
        for(i=0;i < map.size();i++)
        {
            if(map[i][j]==0)
                flag1=false;
        }
        //清除整行向下移动
        if(flag1)
        {
            for(int k=0;k < map.size();k++)
            {
                map[k].remove(j);
                map[k].insert(0,0); //插入到最前！！达到自动更新坐标
            }
            MainWindow::score+=100;
        }
    }
}

void Control::operator=(Control &another)
{
    this->x=another.x;
    this->y=another.y;
    this->type=another.type;
    this->process=another.process;
    this->color=another.color;
```

```
    another.newBlock();
}
```

代码块 3 Control 类实现代码

2.2.5 游戏主界面

游戏主界面，主要用来刷新控制页面显示，这里实现尺寸和每一个小方块的大小，颜色设置，调用 paintEvent 来实现控制，这里设置 Control 类才表示下落的俄罗斯方块类型。GameWidget 继承于 QWidget，头文件及实现代码如下：

```
#ifndef GAMEWIDGET_H
#define GAMEWIDGET_H

#include <QWidget>
#include "control.h"
#include<QPainter>
#include<QPen>
#include<QKeyEvent>
#include<QTimer>
#include<mainwindow.h>

class GameWidget : public QWidget
{
    Q_OBJECT
public:
    //这个是游戏主窗口
    //这里主要绘制图像和捕捉按键事件 而具体的逻辑 交给 control 类实现
    explicit GameWidget(QWidget *parent = nullptr);
```

```
    static Control con;
    static Control anotherCon;

    static QTimer *timer;

    static QVector<QColor> color;

    static bool gameOver;

signals:
    void GameOver();

public slots:
protected:
    //virtual void keyPressEvent (QKeyEvent *);
    virtual void paintEvent (QPaintEvent *);
};

#endif // GAMEWIDGET_H
```

代码块 4 GameWidget 头文件

```
#include "gamewidget.h"

Control  GameWidget::con;
Control  GameWidget::anotherCon;
QTimer* GameWidget::timer;

bool GameWidget::gameOver=false;
```

```
QVector<QColor> GameWidget::color=
{
    QColor::fromRgb(85,170,255),QColor::fromRgb(255,125,33),
    QColor::fromRgb(33,178,74),QColor::fromRgb(189,121,82),
    QColor::fromRgb(140,0,16),QColor::fromRgb(57,73,206),
    QColor::fromRgb(255,203,8),QColor::fromRgb(181,231,24)
};

GameWidget::GameWidget(QWidget *parent) : QWidget(parent)
{
    //设置尺寸
    this->resize(350,700);
    //调整尺寸
    for(int i=0;i<10;i++)
    {
        Control::map.push_back(QVector<int>(20,0));
    }
    for(int i=0;i<10;i++)
    {
        Control::colorMap.push_back(QVector<int>(20,0));
    }
    //con.rePaint();
}

void GameWidget::paintEvent (QPaintEvent *)
```

```
{
    QPainter p(this);
    p.setViewport(this->rect());
    p.setWindow(-1,-1,101,201);
    QPen pen;
    pen.setWidth(0);
    pen.setColor("black");
    p.setPen(pen);
    p.drawRect(-1,-1,100,200);
    //根据 1 的标志 来绘画砖块
    for(int i=0;i<Control::map.size();i++)
    {
        for(int j=0;j<Control::map[i].size();j++)
        {
            if(Control::map[i][j])
            {
                p.fillRect(i*10, j*10,9,9,color[Control::colorMap[i][j]]);
            }
        }
    }
    pen.setWidth(0);
    pen.setColor("white");
    p.setPen(pen);

    //画线
    for(int i=1;i<20;i++)
    {
        p.drawLine(0,i*10,99,i*10);
```

```
    }
    for(int i=1;i<10;i++)
    {
        p.drawLine(i*10,1,i*10,198);
    }
    if(gameOver)
    {
        delete timer;
        timer = nullptr;
        emit GameOver();
    }
}
```

代码块 5 GameWidget 实现代码

2.2.6 预测窗口代码

其实现逻辑于主页面类似，不需要移动，只需要在游戏改变状态时随机生成一个方块并显示。NextBlock 头文件及实现代码如下：

```
#ifndef NEXTBLOCK_H
#define NEXTBLOCK_H
#include <QWidget>
#include<QPen>
#include<QPainter>
#include"gamewidget.h"
#include<QColor>
#include "mainwindow.h"

class NextBlock : public QWidget
{
```

```
    Q_OBJECT
public:
    explicit NextBlock(QWidget *parent = nullptr);

    void paintEvent (QPaintEvent *e);
signals:

public slots:
};

#endif // NEXTBLOCK_H
```

代码块 6 NextBlock 头文件

```
#include "nextblock.h"

NextBlock::NextBlock(QWidget *parent) : QWidget(parent)
{
}



//{0, 0}, {1, 0}, {0, 1}, {1, 1}
void NextBlock::paintEvent (QPaintEvent *e)
{
    Q_UNUSED(e);
    QPainter p(this);
    p.setViewport(this->rect());
    p.setWindow(0,0,80,80);
```

```
    QPen pen;
    pen.setWidth(0);
    pen.setColor("blue");
    p.setPen(pen);
    p.drawRect(10,10,60,60);


GameWidget::anotherCon.block[GameWidget::anotherCon.type][GameWidget::anotherCon.process]
;


    for(int i=0;i<4;i++)
    {
        int width =
GameWidget::anotherCon.block[GameWidget::anotherCon.type][GameWidget::anotherCon.process]
[i].x();
        int height =
GameWidget::anotherCon.block[GameWidget::anotherCon.type][GameWidget::anotherCon.process]
[i].y();

p.fillRect(25+width*10,25+height*10,10,10,GameWidget::color[GameWidget::anotherCon.color]
);


        pen.setWidth(0);
        pen.setColor("white");
        p.setPen(pen);
        p.drawRect(25+width*10,25+height*10,10,10);
    }


}
```

代码块 7 NextBlock 实现代码

2.2.7 背景音乐及初始化

这里使用 Qt 框架下的 QSound 类，来实现多线程播放背景音乐。用 QIcon 类实现图标的插入,初始化地图信息。与 GameWidget 的信号 signals::GameOver(); 进行槽连接。代码如下:

```
int MainWindow::running=false;
int MainWindow::score=0;
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    this->resize(450,600);
    ui->next->hide();

    //启动背景音乐(循环播放)
    //QSound::play(SOUND_BACKGROUND);
    QSound *sound = new QSound(":/BG1.wav",this);
    sound->setLoops(-1);
    sound->play();


    connect(ui->game,&GameWidget::GameOver,[=](){
       ui->pushButton->setText("重新开始");
       ui->pushButton->setDisabled(false);
       //"游戏结束""最终得分为:%1"
//       QString str = "游戏结束最终得分为:%1"
       ui->label->setText(QString("游戏结束"));
       //QMessageBox::about(this,"游戏失败","分数为: " + QString::number(score)+" ");

       GameWidget::gameOver=false;

       for(int i=0;i<10;i++)
       {
           for(int j=0;j<20;j++)
           {
                Control::map[i][j]=0;
                Control::colorMap[i][j]=0;
           }
       }
```

```
        QMessageBox::about(this,"游戏失败","分数为：" + QString::number(score)+" ");

        score=0;
    });

    this->setWindowIcon(QIcon(":/Tetris.ico"));
    this->setWindowTitle("Tetris");
}
```

代码块 8 背景音乐及初始化

2.2.8 游戏开始按钮

点击按钮，匹配相应槽函数，主要实现信息的更新与显示。代码如下：

```
void MainWindow::on_pushButton_clicked()
{
    running=true;

    ui->game->timer = new QTimer(this);
    ui->game->timer->start(1000);
    update();
    ui->game->con.newBlock();
    ui->game->anotherCon.newBlock();
    ui->game->con.rePaint();

    connect(ui->game->timer,&QTimer::timeout,[=](){
        ui->game->con.move(2);
        update();
    });

    ui->next->show();
    ui->pushButton->setDisabled(true);
    ui->label->setText("游戏进行中");
}
```

代码块 9 开始按钮代码

2.2.9 键盘操作控制

主要实现 W/S/A/D 于↑/↓/←/→操作绑定，调用在控制类中的 move（）函数实现，代码如下：

```
void MainWindow::keyPressEvent(QKeyEvent *e){
    if(running)
    {
        if(e->key()==Qt::Key_Up||e->key()==Qt::Key_W)
        {
            ui->game->con.move(1);
        }
        else if(e->key()==Qt::Key_Down||e->key()==Qt::Key_S)
        {
            ui->game->con.move(2);
        }
        else if(e->key()==Qt::Key_Left||e->key()==Qt::Key_A)
        {
            ui->game->con.move(3);
        }
        else if(e->key()==Qt::Key_Right||e->key()==Qt::Key_D)
        {
            ui->game->con.move(4);
        }
        update();
        ui->label_2->setText(QString("当前分数:%1").arg(score));
    }
    if(e->key()==Qt::Key_Space)
    {
        if(ui->game->timer)
        {
            if(running){
                running=false;
                ui->game->timer->stop();
                ui->label->setText(QString("游戏暂停"));
                update();
            }
            else
            {
                running=true;
                ui->game->timer->start(1000);
                ui->label->setText("游戏进行中");
            }
        }
    }
}
```

代码块 10 键盘控制代码

3.3 类图关系

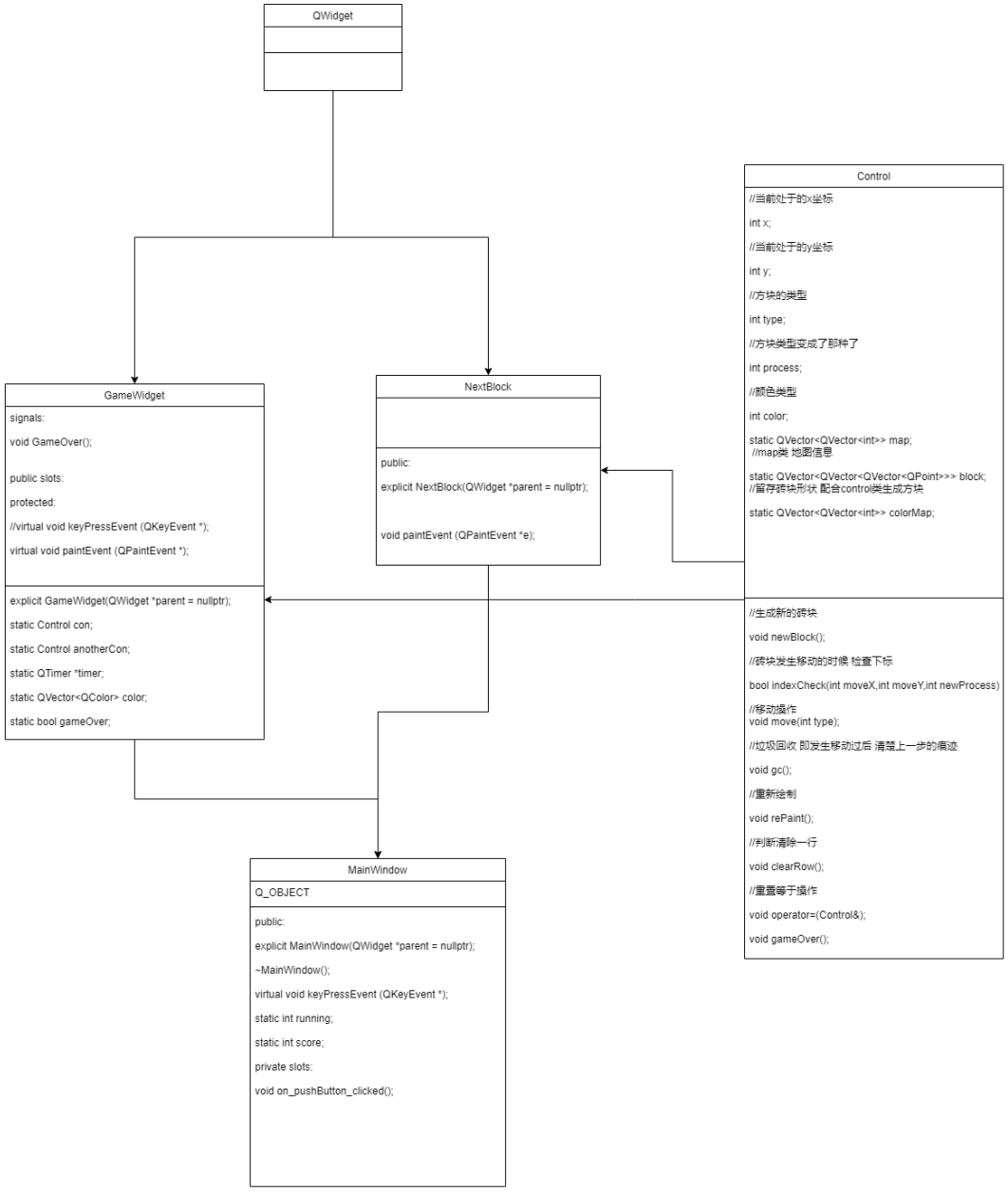


图 8 类图关系

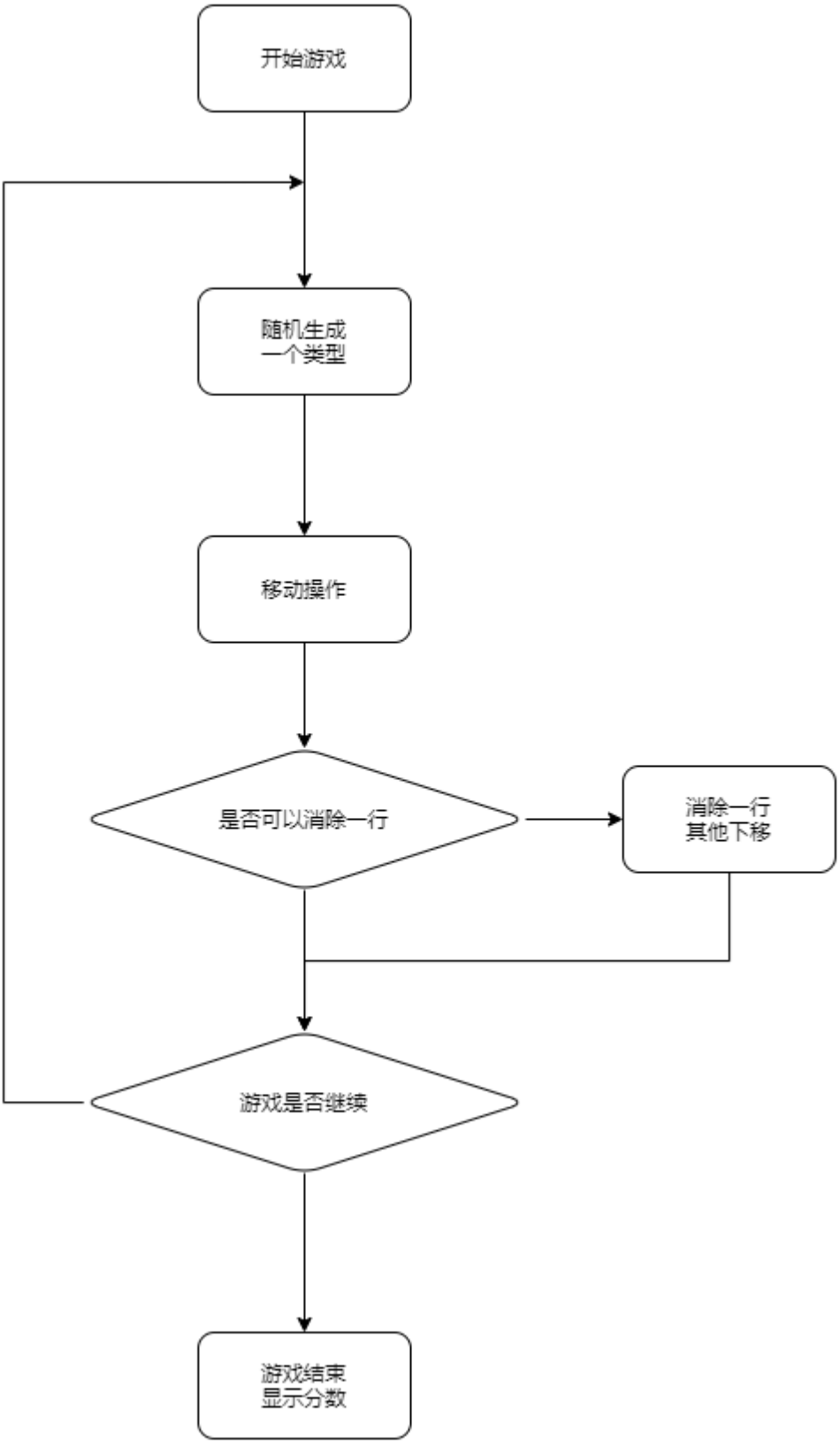


图 9 类关系流程图

4.程序展示

一些操作实际展示，展示效果如下图：

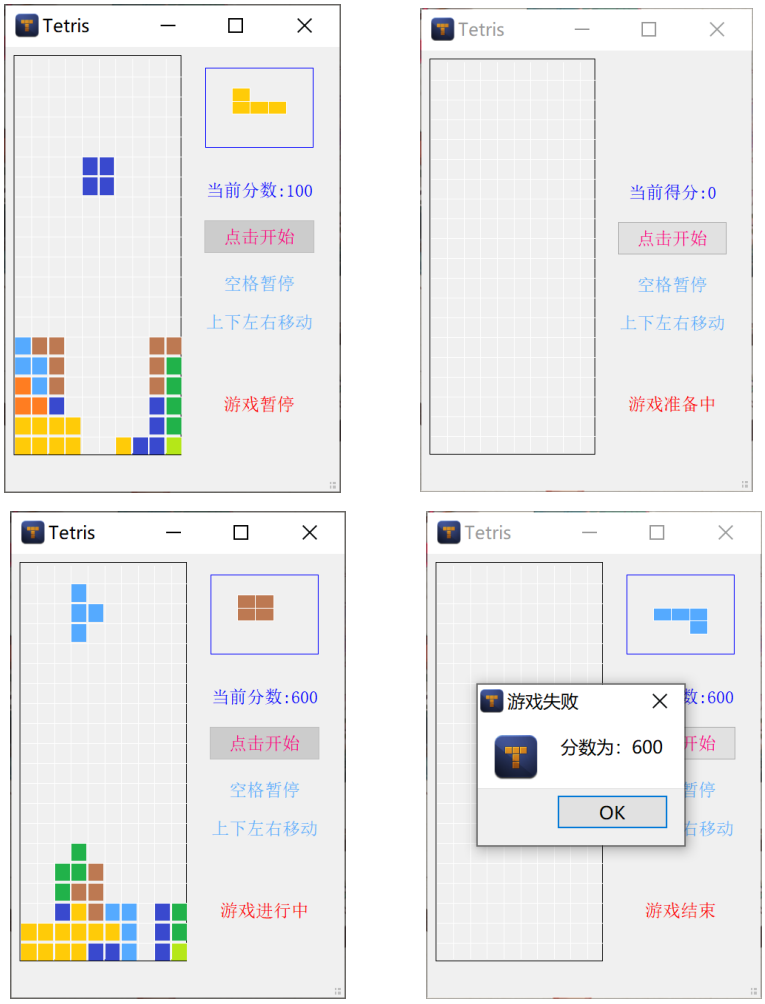


图 10 成果展示图

5.总结思考

通过使用 Qt 应用框架实现俄罗斯方块的基本游戏结构，对页面进行处理，采用继承思想，实现了游戏界面类的开发。该程序是一个集继承、图形界面、事件处理等面向对象编程知识的综合应用的实例程序。