Doi: 10.11729/syltlx 文章编号: 1672-9897（2014）01-0000-00

一个拼图游戏 QT 开发说明

胡振震[1]

（ 1. LEETC CHINA 100000 ）

摘 要 设计了一个拼图游戏，拼图形状由相同或不同数量的球连接而成，各连接球形状通过拼图最终可以拼成一个三角矩阵 (各形状中的球构成的三角网格)。游戏代码采用 QT 实现，主要利用 QGraphicsView 实现交互界面与行为逻辑，利用 C++ 类完成形状划分/形状唯一性/形状落位判断等核心算法。游戏操作最主要的操作是点击 (鼠标左键)，用于选择形状和空位以完成形状控制，其它控制可以由按钮或鼠标加键盘实现。

关键词 拼图游戏 QT C++ 连接球形状

中图分类号 （V211.751;TP391.9） **文献标志码** A

0 引言

最初想开发一个拼图游戏是因为朋友送了一套实体的拼图游戏玩具，给孩子玩玩有益智力发展。当时就涌起一个想法是不是可以做一个类似的小游戏，可以跨平台使用，比如在电脑，在手机上使用，这样随时随地都可以玩，也可以用 APP 的方式还给朋友的孩子，毕竟这套玩具本来属于朋友的孩子，她送出去一条实体玩具，又得到一个游戏应用，多少有些安慰。加之正好想借游戏开发这个事熟悉 C++ 和 QT，笔者在写代码这块以前主要是使用 Fortran 用于科学数值计算，最多用用 VB 做个简单界面封装一下程序，因为工作变动需要使用 C++，尽管这在大学是学过的，但久不用多少忘了，这样因为切合需求所以很快就付诸实践。

这是一个简单的拼图游戏，存在由空位构成上 (下) 三角矩阵，每个空位上可以放置一个球，而这些球以某种链接形式的形状方式存在，这些形状都是不同的，在游戏时将这些形状放入空位，当把所有形状放入填完所有空位则游戏完成。游戏完成的基本形式应如图1所示。

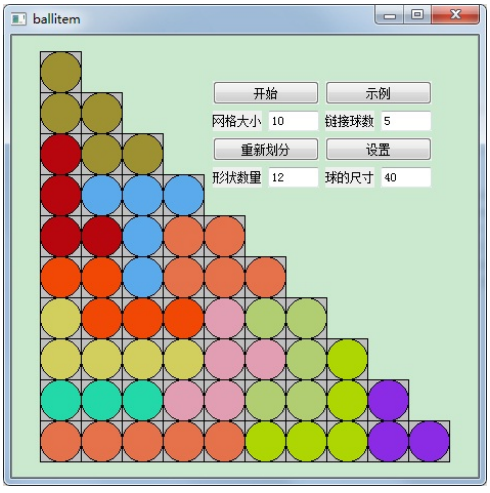

图 1 游戏基本构想

The files have version number v1.0, last revised 2017-08-01.

第一作者简介：HU ZHENZHEN，A LaTeX LOVER，Email: hzzmail@163.com。

这样一个游戏，本质是一个交互的界面，游戏者可以操作由球构成的不同形状填入空位中，直至完成全部填空。考虑用 QT 开发这个游戏，思路主要是利用 qt 的 QGraphicsView 框架来构造图形界面，操作动作的影响利用 qt 的信号槽机制实现，而底层的逻辑算法以 C++ 类及其方法来实现。总的来说是将其分为两层，qt 的图形和事件传递负责用户界面构造，而核心的算法则主要是设计 C++ 类。

因此做的第一步，其实主要是核心算法的设计，主要有: 球连接起来构成的形状的描述 (数据结构，唯一性)，形状的变换 (旋转，翻转)，形状的生成 (空位三角矩阵的随机划分)，形状放入空位时的判断，以及构成形状的球的描述等。大体上可以分为三个类，一个是球 (balltopo)，一个是球构成形状 (linkshape)，最后一个是球构成的网格 (grid，也就是下三角矩阵)。在构建界面之前，主要是初步设计和实现这三个类，并进行测试，主要包括: 网格构建与显示，形状的描述与显示，球的描述 (自身以及邻居)，球在网格中行列号与球序号的关系，形状变换和唯一性判断算法，形状放入网格时 (即落位) 的判断算法，从网格中生成形状的算法等，这可以利用纯 c++ 代码实现并进行编译测试。

之后则利用 qt 构建图形界面，需要构建一个网格类 ballgrid 继承自 grid 用于总体的控制和实现，构建一个形状类 ballshape 用于形状的绘制和信号传递，还有构成形状的 ballitem 球类用于球的绘制以及信号的传递，以及构成网格的框类 boxitem 用于网格空位的绘制以及信号的传递。其中 ballitem 类与 balltopo 类没有直接关系，只是 ballgrid 利用其具有的 balltopo 类的信息，利用 ballitem 类对象进行绘制。boxitem 类的情况相同。ballshape 类的情况大体也差不多，与 linkshape 类没有直接关系，主要是根据 ballgrid 类中的 linkshape 信息进行绘制以及作为 ballitem 对象和 ballgrid 对象传递信息的中介。在 qt 的 QGraphicsView 框架中，除了 QGraphicsItem 外，还有 QGraphicsItemGroup 用于 QGraphicsItem 的编组，理论上使用编组是描述 ballshape 的不错选择，但因为构成 QGraphicsItemGroup 时，qt 总是将 QGraphicsItemGroup 的中心设置为场景 (scene) 的中心，并不是构成 group 的所有 item 的中心，这样对于形状旋转后的位置操作带来一定的困难，所以在实际中没有采用 QGraphicsItemGroup，而完全是由 QGraphicsItem 进行操作，而利用 ballshape 类对象对所属的 QGraphicsItem 进行管理。

上述的总体思路其实不是在写代码之前的预先构思，而是在初步构思的基础上在写代码过程中不断完善的，因为最开始总会有一些问题没有预想到，也总有一些算法并不完善，比如形状划分算法和形状唯一性判断算法，划分算法目前没有采用特殊的搜索方法，而仅仅是采用随机走位方法，靠的是像蒙特卡洛的大量计算，而形状唯一性判断最初利用邻接矩阵特征值的唯一性描述，但后来发现这些特征值只能用于球的连接关系是否相同的判断，而它不等于形状的唯一性，后来增加了位置矩阵的判断，并考虑多种旋转和翻转变换综合考虑形状的唯一性。而关于 C++ 和 qt 语言的问题则是在初步了解语言特性基础上，直接就进行实践了，遇到问题查书籍查帮助，主要解决了一些问题包括: 数组作为参数的函数传递，动态数组 vector/qt 中 Qlist 使用/qt 中 Qstring 的使用，指针及其在参数传递信号槽链接中用法，类的构造/构造函数重载/用方法组成构造函数，信号和槽的构造和连接，qt 部件使用，qt 布局使用，qDebug 和 qmessagebox 的使用，类的不同继承 (没有用到多态)，结构体/typedef 使用，基本流程控制 (if，switch，for，while，do while)，qt 对象的鼠标和键盘事件 (键盘事件需要焦点)，qt 中视图场景图元的坐标以及 QPointF 和 QlineF 等使用，qt 和 c++ 中的随机数，qt 中的颜色，qt 中部件和图元的绘图机制，等等。所有的这些都在实践中摸索其使用方法，最终完成整个程序。当然还有更多的机制可以使用但目前没有去使用，毕竟这是一个游戏不能花过多的时间进去，以后只能在业余时间完善了。可以进一步完善的或使用的机制包括:

1. 图元碰撞检测机制 (目前没有考虑这个，落位的判断完全由位置进行判断)
2. 拖放机制 (目前没有采用这个，而直接用点击选择的方式实现)
3. 多媒体/动画 (游戏的操作可以配上声音，游戏的提示可以使用动画特别是完成后的奖励 (比如根据时间和难度进行计分)，球和框以及界面的配色美化 3d 效果等)
4. 状态机 (目前没有考虑状态机，而完全由标志变量处理)
5. xml 机制 (配置文件可以用 xml 写，比如默认划分方案等，基本方案是网格大小 6/7，形状大小 4; 网格大小 8-12，形状大小是 5)
6. 自动落位算法没有实现
7. 划分算法还需要进一步改进 (目前采用的方法耗时较长)
8. 唯一性算法还需要证明 (目前看起来是可以了，但也许有更好的方法，而且目前的方法没有完成数学证明)
9. 代码需要进一步完善 (动态数组代替数组，引用传递代替传值，优化减小计算量等)

1 开发环境准备

1.1 C++ 环境

C++ 环境直接使用 GNU 的编译器即下载 gcc 套装。因为在 windows 下做，所以下载安装 mingw，或者直接使用以前的安装，而新的 gcc 编译套装安装则下载一套覆盖 mingw 中的 gcc 即可，下载文件比如: i686-6.3.0-release-win32-dwarf-rt_v5-rev1.7z。安装完成后，可以使用它来进行 c++ 程序编译。编译可以写一个最简单的编译脚本，这样就不用每次都去输命令，运行一下编译的 bat 文件即可，比如:

示例 1 简单编译脚本

代码

```
cls

del /q a.exe

gcp a.cpp 1

pause
```

这是用来编译测试核心算法的 c++ 程序的，其中 gcp 也是一个 bat 文件如下:

示例 2 gcp 编译脚本

代码

```
cls
@echo off
title Gcc::g++ WD:%cd% %date% %time%

rem :如果没有输入则提示错误
   SET srcfile=%1
   SET laststr=%2
   if "%laststr%" =="" (
      echo 错误:没有输入参数，请重新输入!!!
      goto endcmd
   ) else (
      goto dealfile
   )
```

```
14
15 rem :处理输入参数
16 :dealfile
17     rem :当源文件为cpp结尾时直接去1~倒数第4个字符即去掉.cpp
18     set exefile=%srcfile:~0,-4%.exe
19
20     rem :多文件输入情况处理
21     rem :多文件输入时cpp编译出来的exe以第一个文件的文件名为exe文件名
22     rem :因此exe文件名无需继续处理只要处理源文件和进程数就足够
23     :loopread
24         shift
25         set judge=%2
26         if "%judge%" =="" (
27             goto compile
28         ) else (
29             set srcfile=%srcfile% %1
30             SET laststr=%2
31             goto loopread
32         )
33
34 rem :对源文件进行编译
35 :compile
36     echo ============================================================
37     echo compile the cpp file with g++
38     echo ============================================================
39     echo g++ -o %exefile% %srcfile%
40     g++ -o %exefile% %srcfile% -std=gnu++11
41
42
43 ::-Ofast
44 rem :删除编译过程文件
45     if exist *.mod del *.mod /Q
46     if exist *.obj del *.obj /Q
47     if exist *.o del *.o /Q
48
49 rem :执行exe文件
50     echo.
51     echo ============================================================
52     echo Run the executable file
53     echo ============================================================
54     echo %exefile% %laststr%
55     echo.
56     %exefile% %laststr%
57
58 rem :暂停敲任意键结束
59 :endcmd
60     pause
```

这个脚本原先是用来编译 fortran 程序的，对于 c++ 程序原理是一样的，只是把编译命令 gfortran 换成了 g++，其中看到删除的.mod，.obj 文件其实是 fortran 程序编译的过程文件。c++ 的过程文件是.o 文件。其中输入的最后一个参数本来是用来输入 fortran 程序并行计算时的进程数的，对于 c++ 程序来说，则变成为了 main 程序中 argc[] 参数输入。

1.2 qt 环境

qt 的安装使用也非常简单,用的是 qt5,下载的版本 5.51(qt-opensource-windows-x86-mingw492-5.5.1.exe),5.80(qt-opensource-windows-x86-mingw530-5.8.0.exe)。实际装的是 5.51 的，用 5.80 也是一样的。使用的 mingw 版，利用的是 gnu 的 c++ 编译器，而不是微软的 visual studio 编译器，这样可以少装点软件。

顺便说一点的是，因为 qt 的跨平台特性，所以在不同的平台环境下测试了不少 qt 安装，实际使用中有几点需要注意:

1. Windows 7 64/32 位下安装，使用正常。
2. Windows 7 下的虚拟机中的 xp 中安装正常，但是使用存在问题，一是 pixmap 失效 (这是在某个 xp 虚拟机中 creator 没有问题的情况下)，二是 creator 失效，无论是安装 vcredist，还是.netframework 都无法使其正常。
3. Windows 7 下的虚拟机中的 win7 中安装正常，使用也基本正常，但 pixmap 失效，在虚拟机 win7 中即便是加上绝对路径也无效。
4. Linux (银河麒麟社区 16 版)，qt 安装正常，使用也正常，但是要注意 pixmap 中的路径需要有点，比如”./a.png”，而不是”a.png”。注意 linux 安装的版本是不同的:qt-opensource-linux-x64-5.8.0.run。

qt 的开发方法分命令行方法和集成环境 (IDE) 方法。命令行方法就是利用文本编辑器写代码，用命令行命令编译。而集成环境方法就是利用 qt 提供的 creator 进行代码开发，界面设计，编译运行。qt 提供了三个主要工具是 creator，designer，assistant，分别是集成开发环境，界面设计师和帮助。creator 中也可以调用帮助，但因为帮助的使用频率很高，可以单独打开便于查找。

1.2.1 命令行方法

命令行方法步骤如下:

1. 安装 qt-mingw，比如:
   qt-opensource-windows-x86-mingw492-5.5.1.exe
   配置环境:
   set PATH=C:\Qt\Qt5.5.1\5.5\mingw492_32\bin;C:\Qt\Qt5.5.1\Tools\mingw492_32\bin;%PATH%
2. 建目录并进入:
   mkdir helloqt
   cd helloqt
   在目录下构建一个与目录同名的主 cpp 文件:
   echo>helloqt.cpp
   利用文本编辑 helloqt.cpp，构建基本 qt 程序。比如:

示例 3 简单 qt 程序

代码

```
1 #include <QApplication>
2 #include <QPushButton>
3
4 int main(int argc, char *argv[])
5 {
6         QApplication app(argc,argv);
```

```
7         QPushButton pushButton(QObject::tr("hello qt !"));
8         pushButton.show();
9         //QObject::connect(&pushButton,SIGNAL(clicked()),&app,SLOT(quit()));
10        QObject::connect(&pushButton,QPushButton::clicked,&app,QApplication::quit);
11        return app.exec();
12    }
```

其中信号槽链接给出了两种语法都是可以用的，注释掉的是 qt4 中的语法，下面一条则是 qt5 的语法。

3. 建立起工程，在该目录下的命令行运行:

   qmake −project

   编辑工程 (因为 qt5 与 qt4 的模块分类不同) 所以加入下面语句确保运行正确:

   greaterThan(QT_MAJOR_VERSION,4):QT += widgets *#trans from qt4 to qt5*

   如果要给程序加一个图标，则在工程文件中加入如下语句:

   RC_ICONS= ele.ICO *#add icon*

4. 编译构建:

   qmake helloqt.pro

   mingw32−make.exe

5. 运行

   release\helloqt

1.2.2 集成开发方法

利用 creator 集成开发环境的步骤如下:

1. 安装 qt-mingw，比如

   qt-opensource-windows-x86-mingw492-5.5.1.exe

2. 直接打开 qt creator

   新建工程，选择 qt widget application，命名工程，选择主界面基类，勾选 (或不勾选)ui，勾选表示利用 designer 构建一个 ui 界面，如图2，3，4，5所示:

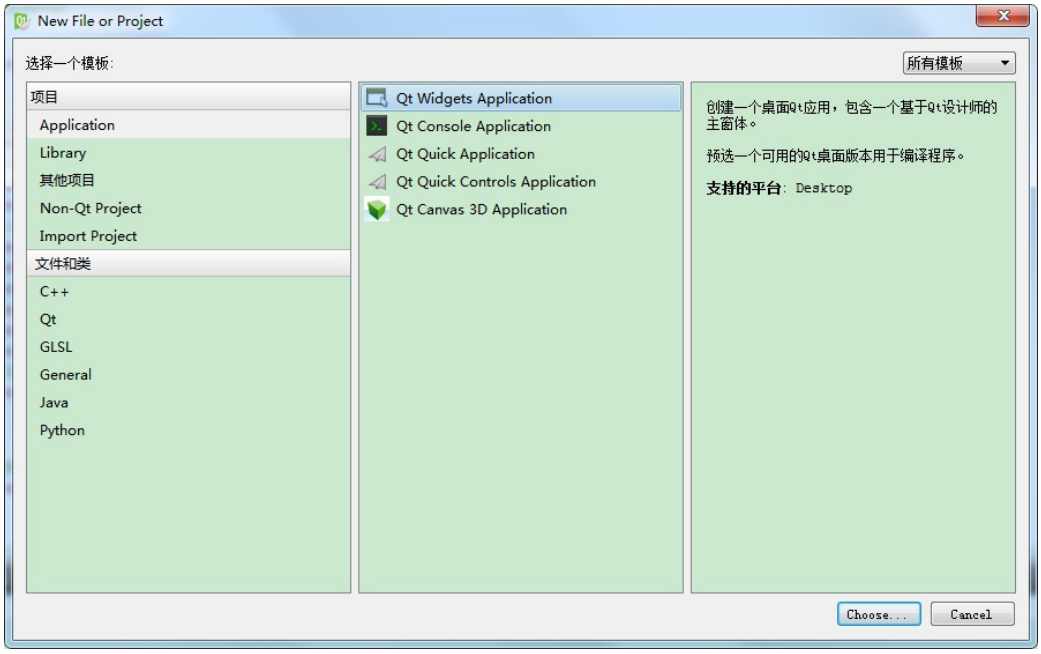

图 2 选择 qt widget application

图 3 命名工程选择位置

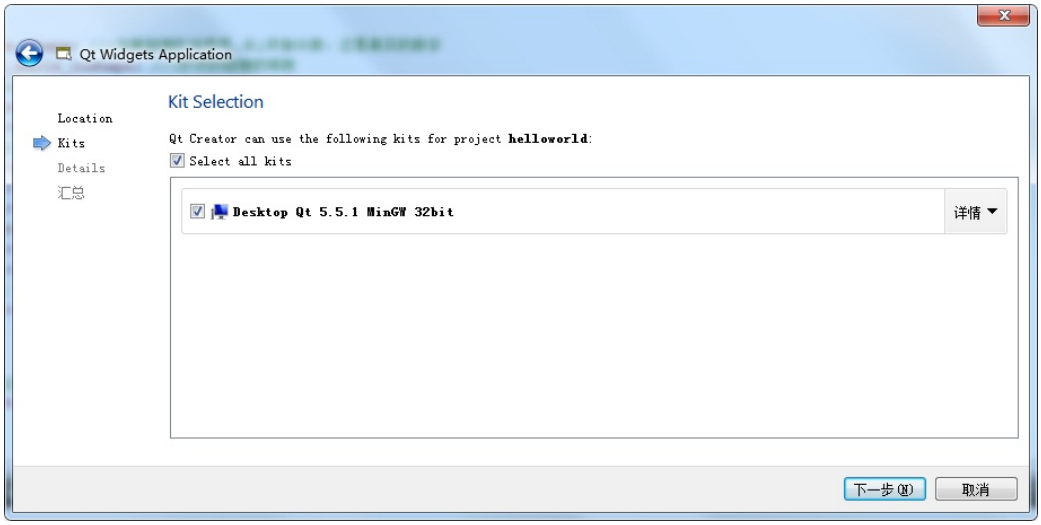

图 4 选择编译工具

图 5 选择主界面基类

双击界面文件 mainwindow.ui，用 designer 编辑工程中的 ui，其中添加一个 pushbutton，修改

其文本为 hello!，添一个加 designer 中自带的信号和槽，pushbutton 的 clicked，到 mainwindows 的 close()，完成如图6所示。

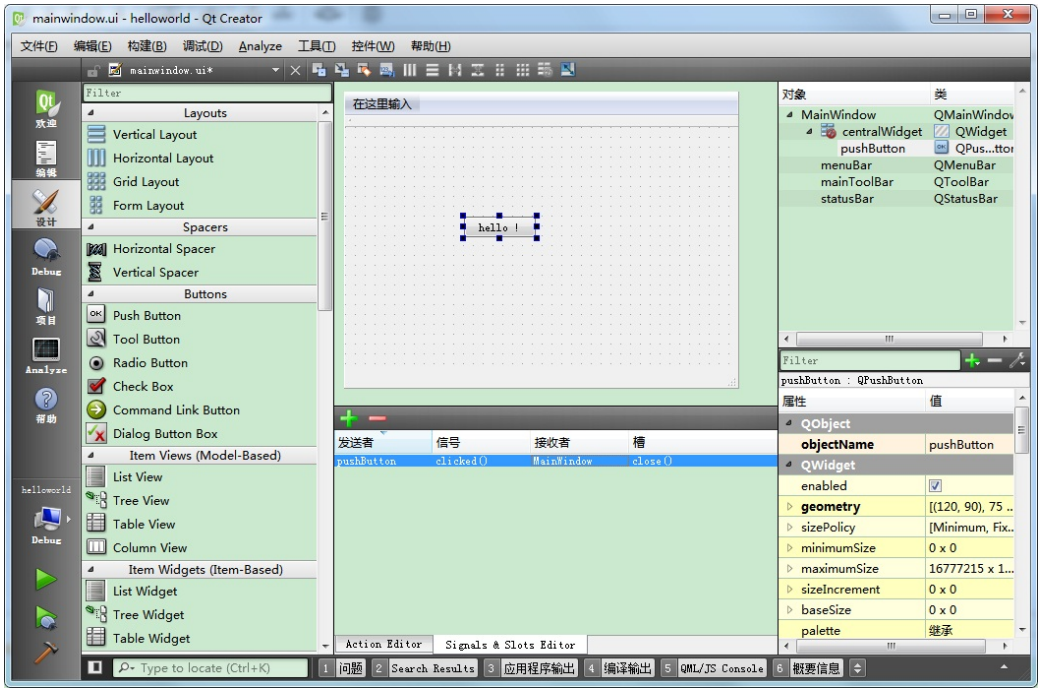

图 6 编辑 ui 界面添加信号槽

3. 菜单: 保存工程，快捷键:

   ctrl ＋ S

4. 菜单: 编译构建工程，如图7所示，快捷键:

   ctrl ＋ B

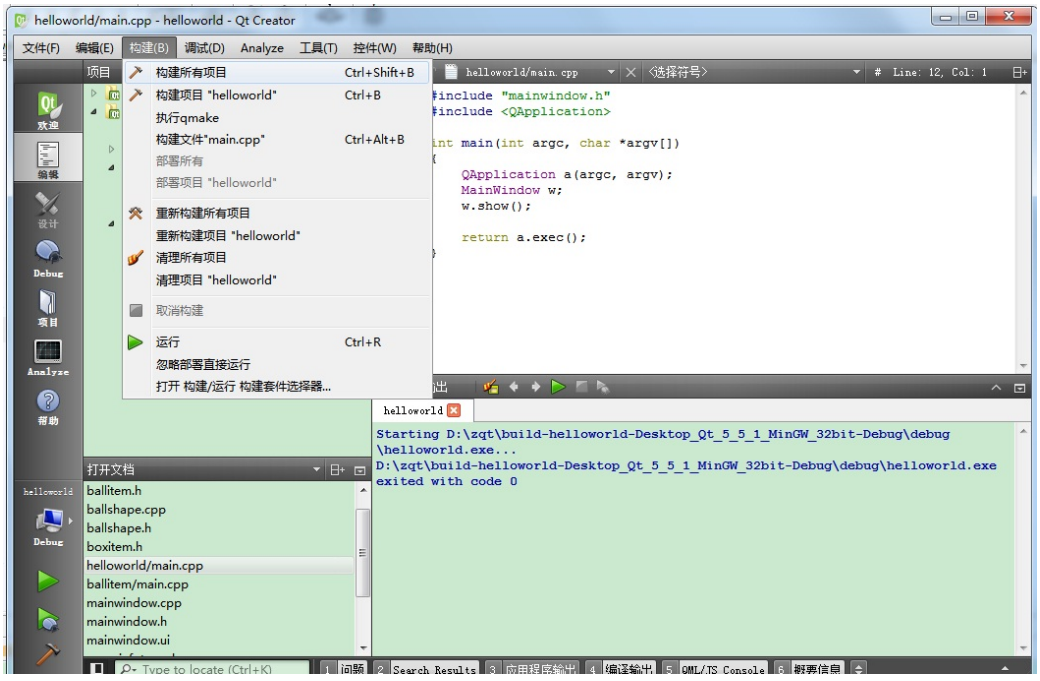

图 7 编译构建

5. 菜单: 运行，结果如图8 所示，当点击其中的按钮则界面窗口关闭，快捷键:

   ctrl ＋ R

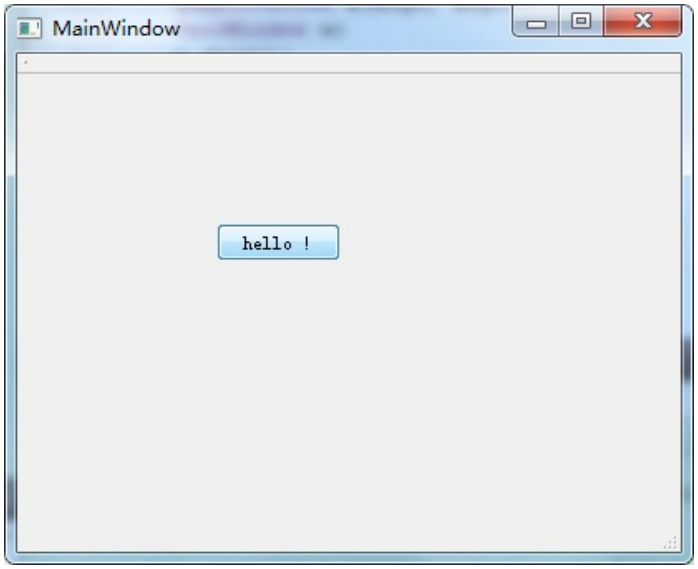

图 8 运行结果

1.3 qt/c++ 程序开发的理解要点

面向对象编程 (不仅仅是 qt/c++ 程序开发) 的一个关键问题是要理解记住: 任何事物都表示成一个对象，无论这个对象是实际事物，还是想象中。在编程过程中实际上是对这样的一个个对象在进行操作。面向对象编程技术和传统结构化编程的主要差别就是面向对象过程中对象的使用贯穿始终。一个对象用类描述，因此一种对象就需要设计一种类，在编程过程中各种对象就是要设计各种类，有的基础类 qt 已经提供了，但不能完全满足需求，因此还需要在其基础上设计新的类，要大胆的继承，利用现有类的功能，然后增加需要的属性和方法来满足最后的需求。

界面的窗体和部件 (常见如按钮，编辑框，列表框等) 是对象，只是它们具有图形化表示。前面说的 ball，shape，grid 类创建的也是对象，只是没有图形化表示，需要用 ballgrid，ballitem 等类的对象来进行对应的图形显示。对象的调用有三种方式: 对象本身，对象指针，对象引用，在使用过程中要注意使用的场合。

qt 中使用集成开发环境的话，使用的信号槽，如果是默认的信号和槽，那么通常只需要定义转到的槽的函数即可。如果需要自定义槽，则需要在类的头文件和源文件中声明和实现。

qt 界面对象具有指针 ui，界面上的对象用 ui-> 对象名即可调用。c++ 生成对象具有指针 this，可以用这个指针做很指代对象自身，这在信号槽链接时很有用。

qt 中简单的调试可以用 qDebug 进行输出，当存在莫名其妙的程序错误时，将编译文件夹删除，重新构建，特别是在类原先不继承 QObject 但修改继承 QObject 后往往会出现错误。命令行下，可以将除工程和源代码外的文件删除，重新 qmake *.PRO，然后 mingw32-make.exe，一般能解决问题。

2 程序设计

下面按类来说明程序的实现，主要从类的使用功能需求以及满足需求的实现两个方面来进行说明，同时夹杂相关的 qt 和 c++ 语言说明。

2.1 boxitem 类

2.1.1 功能分析

boxitem 类主要用途包括:

- 绘制网格的空位，空位用方框表示，用于放置各形状中球
- 可以被选中并发送信号，用于放置球过程中的判断和操作

为了可以绘制方框，并且可以被选中，考虑继承 QGraphicsRectItem。为了可以向其它对象发送信号，考虑继承 QObject。

因此类的声明为:

示例 4 boxitem 类声明

代码

```
#ifndef BOXITEM_H
#define BOXITEM_H

#include <QObject>
#include <QGraphicsRectItem>

class boxitem : public QObject, public QGraphicsRectItem
{

    Q_OBJECT

private:
    int boxid;//盒子(方框，空位)序号，从1开始计数

public:
    boxitem();
    void setboxid(int id);//用于设置盒子(框)序号

protected:
    void mousePressEvent(QGraphicsSceneMouseEvent * event);
    void mouseReleaseEvent(QGraphicsSceneMouseEvent * event);

signals:
    void tojudgesitdown(int boxid);
};

#endif // BOXITEM_H
```

2.1.2 代码实现

boxitem 类的实现也比较简单，如例5所示:

示例 5 boxitem 类定义

代码

```
#include "boxitem.h"
#include <QCursor>
#include <QKeyEvent>
#include <qDebug>

boxitem::boxitem()
{
    //setFlag(QGraphicsItem::ItemIsMovable);
    setFlag(QGraphicsItem::ItemIsSelectable);
    setFlag(QGraphicsItem::ItemIsFocusable);
}

void boxitem::setboxid(int id)
```

```
{
    boxid=id;
}

void boxitem::mousePressEvent(QGraphicsSceneMouseEvent * event)
{
    Q_UNUSED(event);
    setFocus();
    setCursor(Qt::ClosedHandCursor);
    emit tojudgesitdown(boxid);
    //qDebug()<<"focus:"<<hasFocus();
    //qDebug()<<"focus:"<<focusItem();
    //qDebug()<<"emit:"<<"box:tojudgesitdown";
}

void boxitem::mouseReleaseEvent(QGraphicsSceneMouseEvent * event)
{
    Q_UNUSED(event);
    setCursor(Qt::OpenHandCursor);
}
```

其中，最重要的是在构造函数中设置可以被选中可以作为焦点，第二是被鼠标点击时发送被点击需要判断形状落位的信号。

2.2 ballitem 类

2.2.1 功能分析

ballitem 类主要用途包括:

- 绘制形状中的球，用圆形表示
- 可以用鼠标左键点击，表示选中当前形状，用于后面的形状落位判断，因此需要发送鼠标左键被点击信号
- 可以用鼠标右键点击，表示选中当前形状，用于从落位状态回到非落位状态，因此需要发送鼠标右键被点击信号
- 可以用键盘操作，用于形状的控制，比如旋转和翻转，因此需要发送鼠标右键被点击信号

为了可以绘制圆形，并且可以被选中，考虑继承 QGraphicsEllipseItem。为了可以向其它对象发送信号，考虑继承 QObject。

因此类的声明为:

示例 6 ballitem 类声明

代码

```
#ifndef BALLITEM_H
#define BALLITEM_H

#include <QObject>
#include <QGraphicsEllipseItem>
//#include <QPainter> //该类也需要加入，否则出现不完整qpainter的错误提示

class ballitem : public QObject, public QGraphicsEllipseItem
{

    Q_OBJECT
```

```

public:
    ballitem(QObject *parent =0);
    ~ballitem();
    //void setColor(const QColor &color);
    //QRectF boundingRect() const;
    //void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget);
    void setshapeid(int id);
    void setballid(int id);

protected:
    void mousePressEvent(QGraphicsSceneMouseEvent * event);
    void mouseReleaseEvent(QGraphicsSceneMouseEvent * event);
    //void mouseMoveEvent(QGraphicsSceneMouseEvent * event);
    void keyPressEvent(QKeyEvent * event);

signals:
    void rotateormirror(int transid);
    void ballpressed(int ballid,int shapeid);
    void ballpressedright(int ballid,int shapeid);


private:
    int ballid; //标记当前在形状中的序号，是第几个球,从1开始计数
    int shapeid;//标注当前球属于哪个形状，从1开始计数
    // QColor brushcolor;

};

#endif // BALLITEM_H
```

其中因为直接利用 QGraphicsEllipseItem 画圆，而不用重新画，因此不重新实现 paint 函数，直接使用 QGraphicsEllipseItem 的一些方法。

2.2.2 代码实现

ballitem 类的实现，如例7所示:

示例 7 ballitem 类定义

代码

```
#include "ballitem.h"
#include <QCursor>
#include <QKeyEvent>
#include <QGraphicsSceneMouseEvent>
#include <qDebug>

ballitem::ballitem(QObject *parent)
{
    //setFlag(QGraphicsItem::ItemIsMovable);
    Q_UNUSED(parent);
    setFlag(QGraphicsItem::ItemIsSelectable);
    setFlag(QGraphicsItem::ItemIsFocusable);
}

ballitem::~ballitem()
{
}
```

```
18
19  void ballitem::setshapeid(int id)
20  {
21      shapeid=id;
22  }
23
24  void ballitem::setballid(int id)
25  {
26      ballid=id;
27  }
28
29
30  void ballitem::mousePressEvent(QGraphicsSceneMouseEvent * event)
31  {
32      if(event->button()==Qt::LeftButton) { //event->button()返回当前点击的键
33          setFocus();
34          setCursor(Qt::ClosedHandCursor);
35          emit ballpressed(ballid,shapeid);
36          //qDebug()<<"focus:"<<hasFocus();
37          //qDebug()<<"focus:"<<focusItem();
38          //qDebug()<<"emit:"<<"ballpressed";
39      }
40      if(event->button()==Qt::RightButton) { //event->button()返回当前点击的键
41          setCursor(Qt::ClosedHandCursor);
42          emit ballpressedright(ballid,shapeid);
43          //qDebug()<<"focus:"<<focusItem();
44          //qDebug()<<"emit:"<<"ballpressedright";
45      }
46
47
48  }
49
50  void ballitem::mouseReleaseEvent(QGraphicsSceneMouseEvent * event)
51  {
52      Q_UNUSED(event);
53      setCursor(Qt::OpenHandCursor);
54  }
55
56  void ballitem::keyPressEvent(QKeyEvent * event)
57  {
58      switch(event->key())
59      {
60          case Qt::Key_Right:
61              emit rotateormirror(1);
62              break;
63
64          case Qt::Key_Left:
65              emit rotateormirror(2);
66              break;
67
68          case Qt::Key_Up:
69              emit rotateormirror(3);
70              break;
71
72          case Qt::Key_Down:
```

```
            emit rotateormirror(4);
            break;

        default:
            break;
    }
}


//QRectF ballitem::boundingRect() const
//{
// qreal penWidth=1;
// return QRectF(0-penWidth/2,0-penWidth/2,50+penWidth,50+penWidth);
//}


//void ballitem::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
//{
// painter->setBrush(brushcolor);
// painter->drawEllipse(0,0,50,50);
//}

//void ballitem::setColor(const QColor &color)
//{
// brushcolor=color;
//}
```

在构造函数中，利用 setflag，设置 ballitem 可以被选中，可以设置焦点，使用 Q_UNUSED 函数取消因为未使用 parent 带来的警告。

setshapeid，setballid 两个函数用于设置当前 ballitem 所属形状的序号，及其在当前形状中球的排列序号 (即形状中的第几个球)。

鼠标点击事件处理函数中，根据点击的左键和右键分别发送不同的信号。

键盘敲击事件处理函数中，根据键入的键是上下左右箭头键，发送信号表示对当前球所在的形状做不同的旋转和翻转方式。

2.3 ballshape 类

2.3.1 功能分析

ballshape 类主要功能是管理本形状内部的球 (ballitem)。主要用途包括:

- 生成当前形状图形和管理其中的球 (数量，图形位置大小颜色)
- 将形状中的球加入或移出场景
- 设置当前形状内球的位置，记录在网格外的位置，重设该位置
- 获取当前形状内选择的球的位置，用于是否在网格内，以及所在网格的空位序号判断
- 需要传递当前形状被选中的信号，用于落位判断，旋转等操作

为了可以向其它对象发送信号，考虑继承 QObject。为了绘制形状，ballshape 属性中包含 ballitem 类对象用于表示形状内的球。为便于管理还包含球的大小属性，球在网格外的位置信息，当前形状序号，当前选择的球的序号。

因为 ballshape 的属性中 ballitem 类对象，因此 ballitem 类传递出的信号，最方便的是与 ballshape 中的槽进行连接，然后由此中转到其它对象比如 ballgrid 中。因此需要接受来自 ballitem 的三

个信号，左键点击，右键点击，键盘键入三个事件发出的信号。同时需要将这三个信号中转到 ballgrid 中，因此也设置并发送三个信号。

因此类的声明为:

示例 8 ballshape 类声明

代码

```
#ifndef BALLSHAPE_H
#define BALLSHAPE_H

#include <QObject>
#include "shape.h"
#include "ballitem.h"
#include <QGraphicsScene>
#include <QList>
#include "sceneinfotrans.h"


class ballshape : public QObject
{
    Q_OBJECT

public:
    ballshape(QObject *parent =0);
    ~ballshape();
    void setballshape(int id,int n_ballsinshape,int ballpos[],QColor colora);//构成当前形状
    void addtoscene(sceneinfotrans &scene);//将形状加入场景
    void removeballs(sceneinfotrans &scene);//将形状移出场景
    void setballpos(int i,int posx,int posy);//设置索引为i的球位置，i从0开始计数
    void recordpos();//记录一下游戏开始前球放在网格外的位置
    void resetpos();//恢复一下游戏开始前球放在网格外的位置
    QPointF getballpos(int sn);//得到球真实序号为sn的位置,sn从1开始计数
    void setballdiameter(int diameter);//设置球的直径

private:
    int shapeid;
    int n_balls;
    QList<ballitem *> balls;
    QList<QPointF> ballposinit;//游戏开始前放在框外的位置
    int balldiameter;

signals:
    void shapeselected(int ballid,int shapeida);//当前形状被选中，则传递给ballgird
    void shapetoreset(int ballid,int shapeida);//当前形状被选中,需要还原为原来的位置，则传递给ballgird
    void shapetotrans(int shapeida,int transid);//当前形状被选中，需要做旋转，翻转变换

public slots://链接的槽与信号的参数应是一致的。
    void transexec(int transid);//根据键盘信息旋转调整形状时的处理，主要是向ballgird传递消息
    void ballselected(int ballid,int shapeida);//当前形状和当前球被选中时的处理，主要是向ballgird传递消息
    void ballstoresetpos(int ballid,int shapeida);//当前形状被选中,需要还原为原来的位置的处理，主要是向ballgird传递消息

};

#endif // BALLSHAPE_H
```

可以看到，ballshape 与核心算法中的形状表示类 (linkshape) 没有任何关系，因为它更多的只

是管理属于当前形状中的球，以及中转信号。只是在生成当前形状图形时，输入的参数是形状表示类 (linkshape) 的对应序号形状对象的信息。

2.3.2 代码实现

ballshape 类的实现，如例9所示:

示例 9 ballshape 类定义

代码

```
#include "ballshape.h"
#include <qDebug>
//#include <cstdlib>

ballshape::ballshape(QObject *parent)
{
    Q_UNUSED(parent);
}

ballshape::~ballshape()
{}

void ballshape::setballdiameter(int diameter)//设置球的直径
{
    balldiameter=diameter;
}

void ballshape::setballshape(int id,int n_ballsinshape,int ballpos[],QColor colora)//注意输入的是实际的位置信息
{
    shapeid=id;
    n_balls=n_ballsinshape;

    //QColor colora(qrand() % 256, qrand() % 256, qrand() % 256);

    for(int i=0;i<n_balls;i++)
    {
        ballitem *balla=new ballitem();
        balla->setshapeid(id);
        balla->setballid(i+1);
        balla->setRect(0, 0,balldiameter, balldiameter);
        balla->setBrush(colora);
        balla->setPos(ballpos[2*i],ballpos[2*i+1]);
        balls.append(balla);
        QObject::connect(balla,&ballitem::rotateormirror,this,&ballshape::transexec);
        QObject::connect(balla,&ballitem::ballpressed,this,&ballshape::ballselected);
        QObject::connect(balla,&ballitem::ballpressedright,this,&ballshape::ballstoresetpos);
    }
}

void ballshape::setballpos(int i,int posx,int posy)
{
    balls[i]->setPos(posx,posy);
}

void ballshape::recordpos()//
{
    for(int i=0;i<n_balls;i++)
    {
```

```
49          ballposinit.append(balls[i]->pos());
50      }
51  }
52
53  void ballshape::resetpos()//
54  {
55      for(int i=0;i<n_balls;i++)
56      {
57          balls[i]->setPos(ballposinit[i]);
58      }
59  }
60
61  QPointF ballshape::getballpos(int i) //i是球的真实序号
62  {
63      return balls[i-1]->pos();
64  }
65
66
67  void ballshape::addtoscene(sceneinfotrans &scene)
68  {
69      for(int i=0;i<n_balls;i++)
70      {
71          scene.addItem(balls[i]);//balls[0]本身是一个指针数组，所以直接用其值即可
72      }
73  }
74
75  void ballshape::removeballs(sceneinfotrans &scene)
76  {
77      for(int i=0;i<n_balls;i++)
78      {
79          scene.removeItem(balls[i]);//balls[0]本身是一个指针数组，所以直接用其值即可
80      }
81  }
82
83  void ballshape::transexec(int transid)
84  {
85       //qDebug()<<"transid="<<transid;
86       //qDebug()<<"recieve transid from ball";
87       emit shapetotrans(shapeid,transid);
88  }
89
90  void ballshape::ballselected(int ballid,int shapeida)//
91  {
92      //qDebug()<<"ballid="<<ballid<<" shapeid"<<shapeida;
93      emit shapeselected(ballid,shapeida);
94      //qDebug()<<"shape emit info to grid";
95
96  }
97
98  void ballshape::ballstoresetpos(int ballid,int shapeida)//
99  {
100     //qDebug()<<"ballid="<<ballid<<" shapeid"<<shapeida;
101     emit shapetoreset(ballid,shapeida);
102     //qDebug()<<"shape emit info to grid";
103 }
```

在生成当前形状图形函数 (setballshape) 中，利用同序号的真实形状表示类 (linkshape) 对象的信息，设置当前形状图形的序号，其中的球数，以及各个球 (ballitem) 的属性，并将各个 ballitem 的信号与 ballshape 的槽进行连接。

setballpos，recordpos，resetpos，getballpos 函数用于设置/记录/重设/获取当前形状图形中球的位置。

addtoscene，removeballs 函数用于将形状图形中的球放入/移出场景。

三个槽函数用于中转 ballitem 发出的信号道 ballgrid 中。

2.4 ballgrid 类

2.4.1 功能分析

ballgrid 类是本程序的主类，很多运算都要在其中完成。主要用途包括:

- 生成划分形状 (默认保存/算法划分) 和管理
- 划分形状加入或移出场景
- 将方框 (空位) 网格加入或移出场景
- 游戏管理: 包括将各形状移动到形状外或恢复
- 游戏管理: 设置网格大小，形状链接球数，球的直径
- 需要接收来自形状图形的信号，接收来自方框的信号，接收来自场景中按钮的信号

为了可以接收其它对象的发送信号，考虑继承 QObject。同时需要进行形状描述，形状划分，落位判断，形状唯一性判断等核心计算，继承 grid 类，而 grid 类包含属性 linkshape 对象，balltopo 对象，便于各种计算。

类的声明为:

示例 10 ballgrid 类声明

代码

```
#ifndef BALLGRID_H
#define BALLGRID_H

#include "shape.h"
#include "ballshape.h"
#include <QList>
#include <QGraphicsScene>
#include "boxitem.h"
#include "sceneinfotrans.h"
#include <QObject>

class ballgrid : public QObject, public grid
{

        Q_OBJECT

public:
    ballgrid();

    void setshapesdefault(int arraysize);//利用预存的默认划分形状，利用继承的n_scale，n_n_balls_ofshape信息选择不同情况下的默认划分
    void setshapes(int arraysize);//利用划分算法生成划分形状
    void addtoscenedefault();//将各划分形状加入到场景中
```

```
23      void setshaperandposdefault();//将形状移动到框外面，以便开始游戏
24      void removeshapes();//将各划分形状移出场景
25
26      void setboxes();//生成空位(方框)网格
27      void addboxestoscene();//将方框网格加入场景
28      void removeboxes();//将方框网格移出场景
29      int getboxidat(QPointF posa);//根据场景中的位置获取当前位置上的方框的索引，从0开始计数
30
31      int getgridnum();//获取网格大小信息gridnum
32      int getlinknum();//获取划分形状的预设球数信息linknum
33      int getflag();//获取网格大小和划分形状的预设球数改变的标志信息flaggridchanged
34      void setworkscene(sceneinfotrans &scene);//保存当前所在的场景信息
35
36  public slots:
37      void judgesitdown(int boxid);//根据接收到盒子信息去判断能否落位
38      void dealshapetosit(int ballid,int shapeida); //记录一下接收到的要落位的形状的信息
39      void shapereset(int ballid,int shapeida); //重新恢复形状在网格外的原位置
40      void shapetrans(int shapeida,int transid);//对当前形状进行旋转等操作
41      void shapeturnright(bool a);//按钮控制向右旋转
42      void shapeturnleft(bool a);//按钮控制向左旋转
43      void shapemirrorab(bool a);//按钮控制上下翻转
44      void shaperesetpos(bool a);//按钮控制移出网格
45      void gamestart(bool a);//游戏开始，将所有形状移动到网格外
46      void gameexample(bool a);//游戏示例，将所有形状恢复填入到网格中
47      void gameset(bool a);//当网格大小和划分形状的预设球数改变时，重设游戏
48      void gamerepart(bool a);//对当前网格大小和划分形状的预设球数下利用算法重新进行划分
49      void dealgridnumtext(QString text1);//根据界面输入信息记录网格大小
50      void deallinknumtext(QString text1);//根据界面输入信息记录划分形状的预设球数
51      void setballdiameter(QString text1);//根据界面输入信息记录球和方框的直径
52
53  signals:
54      void shapecompleted(QString a);//当形状重新划分时，划分完成的信号
55
56  private:
57      QList<ballshape *> ballshapesingrid;//算法划分的形状图形的信息
58      QList<ballshape *> ballshapesingriddefault;//默认保存的形状图形的信息
59      QList<boxitem *> boxesingrid;//用于绘制网格中的框,注意只能用指针列表，而不能是实际列表，因为boxitem没有深复制
60      //的确也没有必要拷贝，用指针可以指向它们
61
62      int shapeidtosit,ballidtosit;//记录当前需要落位的形状序号及其中的焦点球(即选中的球)序号
63      int gridnum,linknum;//记录输入的网格大小和链接形状的球数
64      int flaggridchanged;//记录是否更改了网格大小和链接球数
65      int balldiameter;//设置球的直径
66      sceneinfotrans *sceneonwork;//当前网格所要加入的场景，注意用指针是避免传值
67  };
68
69  #endif // BALLGRID_H
```

其中: ballgrid 仅发送一个信号 shapecompleted，表示当形状重新划分完成，便于在游戏界面上显示当前划分总的形状数。而需要接受 11 个信号，即响应的 11 个槽函数。其中三个来自 ballitem 的左键点击，右键点击，键盘键入三个事件发出并经 ballshape 中转的信号，用于记录一下接收到的要落位的形状的信息，重新恢复形状在网格外的原位置，对当前形状进行旋转等操作。一个来自 boxitem 的左键点击事件信号，用于根据接收到方框 (盒子) 信息判断能否落位。另外 7 个信号来自

场景中的按钮，文本框等部件。4 个按钮控制产生的信号用于游戏开始将所有形状移动到网格外，用于游戏示例将所有形状恢复填入到网格中，用于当网格大小和划分形状的预设球数改变时重设游戏，用于对当前网格大小和划分形状的预设球数下利用算法重新进行划分。3 个文本框内容改变产生的信号，用于设置记录网格大小，链接形状中的球数，以及球和方框的直径。

ballgrid 的属性中主要包含图形显示和控制相关的信息，记录当前需要落位的形状序号 shapeidtosit 及其中的焦点球 (即选中的球) 序号 ballidtosit 用于落位判断，记录输入的网格大小 gridnum 和链接形状的球数 linknum 用于控制游戏难度等，设置球的直径 balldiameter 用于显示控制，当前网格所要加入的场景指针 sceneonwork 用于将当前网格放入场景中，而 ballshapesingriddefault 保存的默认形状图形的信息，ballshapesingrid 保持算法划分的形状图形的信息，boxesingrid 保持用于网格中的空位 (方框) 的信息。

上述的属性主要是图形显示，对应的是 grid 类中的属性。将 grid 类中的属性设置为 protected，这样 ballgrid 继承后可以直接使用 grid 类中的属性信息，包括真实形状的描述等等，用于 ballgrid 中的这些图形属性操作。

2.4.2 代码实现

ballgrid 类的实现，如例11所示:

示例 11 ballgrid 类定义

代码

```
#include "ballgrid.h"
#include <qDebug>
#include <QMessageBox>

ballgrid::ballgrid()
{
    shapeidtosit=0; //
    ballidtosit=0;
    gridnum=10;
    linknum=5;
    flaggridchanged=0;
    balldiameter=40;
}

void ballgrid::setworkscene(sceneinfotrans &scene)
{
    sceneonwork=&scene;
}

int ballgrid::getgridnum()
{
   return gridnum;
}
int ballgrid::getlinknum()
{
    return linknum;
}
int ballgrid::getflag()
{
   return flaggridchanged;
}

```

```
33 void ballgrid::setballdiameter(QString text1)
34 {
35     balldiameter=text1.toInt();
36 }
37
38 void ballgrid::dealgridnumtext(QString text1)
39 {
40     gridnum=text1.toInt();
41     flaggridchanged=1;
42     //qDebug()<<"gridnum:"<<gridnum;
43 }
44
45 void ballgrid::deallinknumtext(QString text1)
46 {
47     linknum=text1.toInt();
48     flaggridchanged=1;
49     //qDebug()<<"linknum:"<<linknum;
50 }
51
52 void ballgrid::gamestart(bool a)
53 {
54     Q_UNUSED(a);
55     setshaperandposdefault();
56 }
57
58 void ballgrid::gameexample(bool a)
59 {
60
61     Q_UNUSED(a);
62     for(int i=0;i<n_ofshapes;i++)
63     {
64         //int id=veclinkshape[i].getshapesn();
65         int n_ballsinshape=veclinkshape[i].getshapenballs();
66         //qDebug()<<"id:"<<id;
67         //qDebug()<<"n_ball_inshape:"<<n_ballsinshape;
68
69         for(int j=0;j<n_ballsinshape;j++)
70         {
71             int row=veclinkshape[i].getptroworig(j);
72             int col=veclinkshape[i].getptcolorig(j);
73             ballshapesingriddefault[i]->setballpos(j,col*balldiameter,row*balldiameter);
74         }
75     }
76
77 }
78
79 void ballgrid::gameset(bool a)
80 {
81     Q_UNUSED(a);
82     if(flaggridchanged==1)
83     {
84         qDebug()<<"do new partition";
85         removeboxes();
86         removeshapes();
87         setgrid(gridnum,linknum);
```

```
88          setboxes();
89          addboxestoscene();
90          setshapesdefault(n_total);
91          addtoscenedefault();
92      }else
93      {
94          QMessageBox msgBox;
95          msgBox.setWindowTitle("WARNING");
96          msgBox.setText("Grid not changed,need not to partition");
97          msgBox.exec();
98      }
99      flaggridchanged=0;
100 }
101
102 void ballgrid::gamerepart(bool a)
103 {
104     Q_UNUSED(a);
105     flaggridchanged=1;
106     qDebug()<<"redo new partition";
107     removeboxes();
108     removeshapes();
109     setgrid(gridnum,linknum);
110     setboxes();
111     addboxestoscene();
112     setshapesdefault(n_total);
113     qDebug()<<"n_ofshapes="<<n_ofshapes;
114     qDebug()<<"n_total="<<n_total;
115     addtoscenedefault();
116     flaggridchanged=0;
117 }
118
119
120 void ballgrid::setboxes()//根据setgrid生成的框生成对应的boxitem，并绘制所有的框
121 {
122     //boxitem boxb[n_total];
123     boxesingrid.clear();
124     for(int i=0;i<n_total;++i)
125     {
126         boxitem *boxa=new boxitem();
127         boxa->setRect(0, 0,balldiameter,balldiameter);
128         boxa->setBrush(Qt::lightGray);//cyan
129         coord boxpos=getrowcolformsn(i+1);
130         boxa->setPos(boxpos.col*balldiameter,boxpos.row*balldiameter);
131         boxa->setboxid(i+1);
132         boxesingrid.append(boxa);
133         QObject::connect(boxa,&boxitem::tojudgesitdown,this,&ballgrid::judgesitdown);
134     }
135     //drawboxes();
136 }
137
138 void ballgrid::addboxestoscene()//将绘制的框显示到场景中
139 {
140     int n_boxes=boxesingrid.size();
141     for(int i=0;i<n_boxes;++i)
142     {
```

```
143         sceneonwork->addItem(boxesingrid[i]);
144     }
145 }
146
147 int ballgrid::getboxidat(QPointF posa) //根据当前位置确定box的索引，从0开始计数
148 {
149     int flagingrid=0;
150     int i=0;
151     for(i=0;i<n_total;++i)
152     {
153         QPointF posb=boxesingrid[i]->pos();
154         if(QLineF(posa,posb).length() <0.00001)
155         {
156             flagingrid=1;
157             break;
158         }
159     }
160     if(flagingrid==0) //当不在网格内时返回网格中总的框数加1
161     {
162         return n_total+1;
163     }else
164     {
165         return i;
166     }
167 }
168
169 void ballgrid::removeboxes()//将绘制的框从场景中移除
170 {
171     int n_boxes=boxesingrid.size();
172     for(int i=0;i<n_boxes;++i)
173     {
174         sceneonwork->removeItem(boxesingrid[i]);
175     }
176 }
177
178 void ballgrid::judgesitdown(int boxid)//点击选中box时的形状落位判断
179 {
180     qDebug()<<"recieved info from box:"<<boxid;
181     //是否有shape要放的信息，有的话，根据shape信息调用
182     int flagyestosit=0;
183     int n_balls=0;
184     if(shapeidtosit>0) {//判断是否存在shape需要落位
185         QPointF posa=ballshapesingriddefault[shapeidtosit-1]->getballpos(ballidtosit);
186         int boxida=getboxidat(posa)+1;//当前球所在的位置
187         if(boxida <= n_total)//当在网格内时
188         {
189             clearshapeoccp(boxida,ballidtosit,shapeidtosit);//参数是框的序号，形状中当前球的序号，形状的序号，都从1开始计数
190         }
191         qDebug()<<"ball in box id:"<<boxida;
192         qDebug()<<"ball id:"<<ballidtosit;
193         qDebug()<<"shape id:"<<shapeidtosit;
194
195         setfocus(boxid);
196         n_balls=veclinkshape[shapeidtosit-1].getshapenballs();
197         std::vector<coord> ballscoords=veclinkshape[shapeidtosit-1].getvec_coord();
```

```
198         flagyestosit=canbesetdown(n_balls,ballidtosit, ballscoords);//根据焦点球序号以及形状中焦点和坐标确定形状能否放入网格中
199     }
200
201     //如果能放入则发送信息给shape，接着shape调整位置，//即绘图
202     int rowfocusinshape=veclinkshape[shapeidtosit-1].getptrow(ballidtosit-1);
203     int colfocusinshape=veclinkshape[shapeidtosit-1].getptcol(ballidtosit-1);
204     int rowfocusingrid=getrowcolformsn(boxid).row;
205     int colfocusingrid=getrowcolformsn(boxid).col;
206     if(flagyestosit>0) {
207         setshapeoccp(boxid,ballidtosit,shapeidtosit);//设置一下占用情况
208         for(int j=0;j<n_balls;j++)
209         {
210             int row=(veclinkshape[shapeidtosit-1].getptrow(j))-rowfocusinshape+rowfocusingrid;
211             int col=(veclinkshape[shapeidtosit-1].getptcol(j))-colfocusinshape+colfocusingrid;
212
213             ballshapesingriddefault[shapeidtosit-1]->setballpos(j,col*balldiameter,row*balldiameter);
214         }
215     }
216
217     if(gridocupied())
218     {
219         QMessageBox msgBox;
220         msgBox.setWindowTitle("OK");
221         msgBox.setText("Goodjob! congratulations!! ");
222         msgBox.exec();
223     }
224 }
225
226 void ballgrid::dealshapetosit(int ballid,int shapeida)
227 {
228     qDebug()<<"recieved info from shape:"<<shapeida;
229     qDebug()<<"shape:"<<shapeida<<" wants to be sit down!";
230     shapeidtosit=shapeida;
231     ballidtosit=ballid;
232     for(int i=0;i<veclinkshape[shapeida-1].getshapenballs();++i)
233     {
234         qDebug()<<"coord:"<<veclinkshape[shapeida-1].getptrow(i)<<veclinkshape[shapeida-1].getptcol(i)<<" coord outer
                "<<veclinkshape[shapeida-1].getptrowouter(i)<<veclinkshape[shapeida-1].getptcolouter(i);
235     }
236 }
237
238 void ballgrid::shapeturnright(bool a)//按钮控制向右旋转
239 {
240     Q_UNUSED(a);
241     shapetrans(shapeidtosit,1);
242 }
243
244 void ballgrid::shapeturnleft(bool a)//按钮控制向左旋转
245 {
246     Q_UNUSED(a);
247     shapetrans(shapeidtosit,2);
248 }
249
250 void ballgrid::shapemirrorab(bool a)//按钮控制上下翻转
251 {
```

```
252     Q_UNUSED(a);
253     shapetrans(shapeidtosit,3);
254 }
255
256 void ballgrid::shaperesetpos(bool a)//按钮控制移出网格
257 {
258     Q_UNUSED(a);
259     shapereset(ballidtosit,shapeidtosit);
260 }
261
262
263 void ballgrid::shapereset(int ballid,int shapeida)
264 {
265     //qDebug()<<"recieved info from shape:"<<shapeida;
266     //qDebug()<<"shape:"<<shapeida<<" wants to be sit down!";
267     shapeidtosit=shapeida;
268     ballidtosit=ballid;
269
270     QPointF posa=ballshapesingriddefault[shapeida-1]->getballpos(ballid);
271     int boxid=getboxidat(posa)+1;//当前球所在的位置
272     if(boxid < n_total)
273     {
274         clearshapeoccp(boxid,ballid,shapeida);//参数是框的序号，形状中当前球的序号，形状的序号，都从1开始计数
275     }
276     //qDebug()<<"boxid:"<<boxid;
277
278     ballshapesingriddefault[shapeida-1]->resetpos();
279     veclinkshape[shapeida-1].resetoutercoord();
280 }
281
282
283 void ballgrid::setshapesdefault(int arraysize)//利用原先存有的默认的划分，利用继承的n_scale，n_n_balls_ofshape信息选择不同
        情况下的默认划分
284 {
285     QColor colora[256];
286     for(int i=0;i<256;i++)
287     {
288         colora[i]=QColor(qrand()%256,qrand()%256,i);
289     }
290
291     ballshapesingriddefault.clear();//初始化一下
292     std::vector<linkshape> veclinka;
293     veclinkshape=veclinka;
294
295     if(flaggridchanged==0 && gridnum ==10)
296     {
297
298         n_ofshapes=12;
299         coord vectmp1[5]={{6,0},{7,0},{7,1},{7,2},{7,3}};
300         int vecnumber1[5]={22, 29, 30, 31, 32};
301         linkshape shapea=linkshape(1,5,vectmp1,vecnumber1);
302         veclinkshape.push_back(shapea);
303
304         coord vectmp2[5]={{5,5},{5,4},{5,3},{4,3},{4,4}};
305         int vecnumber2[5]={21, 20, 19, 14, 15};
```

```
306         shapea=linkshape(2,5,vectmp2,vecnumber2);
307         veclinkshape.push_back(shapea);
308
309         coord vectmp3[5]={{3,3},{3,2},{4,2},{5,2},{3,1}};
310         int vecnumber3[5]={10, 9, 13, 18, 8};
311         shapea=linkshape(3,5,vectmp3,vecnumber3);
312         veclinkshape.push_back(shapea);
313
314         coord vectmp4[5]={{8,7},{7,7},{9,7},{9,6},{9,5}};
315         int vecnumber4[5]={44, 36, 53, 52, 51};
316         shapea=linkshape(4,5,vectmp4,vecnumber4);
317         veclinkshape.push_back(shapea);
318
319         coord vectmp5[5]={{9,4},{9,3},{9,2},{9,1},{9,0}};
320         int vecnumber5[5]={50, 49, 48, 47, 46};
321         shapea=linkshape(5,5,vectmp5,vecnumber5);
322         veclinkshape.push_back(shapea);
323
324         coord vectmp6[5]={{8,5},{8,6},{7,6},{6,6},{6,5}};
325         int vecnumber6[5]={42, 43, 35, 28, 27};
326         shapea=linkshape(6,5,vectmp6,vecnumber6);
327         veclinkshape.push_back(shapea);
328
329
330         coord vectmp7[3]={{9,9},{9,8},{8,8}};
331         int vecnumber7[3]={55, 54, 45};
332         shapea=linkshape(7,3,vectmp7,vecnumber7);
333         veclinkshape.push_back(shapea);
334
335         coord vectmp8[5]={{6,3},{6,2},{6,1},{5,1},{5,0}};
336         int vecnumber8[5]={25, 24, 23, 17, 16};
337         shapea=linkshape(8,5,vectmp8,vecnumber8);
338         veclinkshape.push_back(shapea);
339
340         coord vectmp9[5]={{8,3},{8,4},{7,4},{7,5},{6,4}};
341         int vecnumber9[5]={40, 41, 33, 34, 26};
342         shapea=linkshape(9,5,vectmp9,vecnumber9);
343         veclinkshape.push_back(shapea);
344
345         coord vectmp10[5]={{2,1},{2,2},{1,1},{1,0},{0,0}};
346         int vecnumber10[5]={5, 6, 3, 2, 1};
347         shapea=linkshape(10,5,vectmp10,vecnumber10);
348         veclinkshape.push_back(shapea);
349
350         coord vectmp11[4]={{4,0},{4,1},{3,0},{2,0}};
351         int vecnumber11[4]={11, 12, 7, 4};
352         shapea=linkshape(11,4,vectmp11,vecnumber11);
353         veclinkshape.push_back(shapea);
354
355         coord vectmp12[3]={{8,1},{8,2},{8,0}};
356         int vecnumber12[3]={38, 39, 37};
357         shapea=linkshape(12,3,vectmp12,vecnumber12);
358         veclinkshape.push_back(shapea);
359
360         int ballpos[arraysize];
```

```
361
362         for(int i=0;i<n_ofshapes;i++)
363         {
364             int id=veclinkshape[i].getshapesn();
365             int n_ballsinshape=veclinkshape[i].getshapenballs();
366             //qDebug()<<"id:"<<id;
367             //qDebug()<<"n_ball_inshape:"<<n_ballsinshape;
368
369             for(int j=0;j<n_ballsinshape;j++)
370             {
371                 int row=(veclinkshape[i].getptrow(j)+veclinkshape[i].getrowofzero());
372                 int col=(veclinkshape[i].getptcol(j)+veclinkshape[i].getcolofzero());
373                 //qDebug()<<"ball sn="<<veclinkshape[i].getballsn(j);
374                 //qDebug()<<"row="<<row<<" col="<<col;
375                 ballpos[2*j]=(col*balldiameter);
376                 ballpos[2*j+1]=(row*balldiameter);
377             }
378
379             ballshape *shapea=new ballshape();
380             shapea->setballdiameter(balldiameter);
381             shapea->setballshape(id,n_ballsinshape,ballpos,colora[qrand()%256]);
382             ballshapesingriddefault.append(shapea);
383             QObject::connect(shapea,&ballshape::shapeselected,this,&ballgrid::dealshapetosit);
384             QObject::connect(shapea,&ballshape::shapetoreset,this,&ballgrid::shapereset);
385             QObject::connect(shapea,&ballshape::shapetotrans,this,&ballgrid::shapetrans);
386         }
387     } else
388     {
389         partition(n_total);
390
391         qDebug()<<"n_ofshapes="<<n_ofshapes;
392         qDebug()<<"n_total="<<n_total;
393         if(n_ofshapes<n_total) {
394             setshapes(n_total);
395             ballshapesingriddefault=ballshapesingrid;
396             QString stra=QString("%1").arg(n_ofshapes);
397             emit shapecompleted(stra);
398         }else
399         {
400             QMessageBox msgBox;
401             msgBox.setWindowTitle("WARNING");
402             msgBox.setText(" partition is not successful! \n please reset the grid scale and link ball number, \n
                    and redo partition!");
403             msgBox.exec();
404             qDebug()<<"partition is not successful! please redo!"<<n_ofshapes;
405         }
406     }
407 }
408
409 void ballgrid::addtoscenedefault()//将默认的划分的形状加入到场景中
410 {
411     if(n_ofshapes<n_total) {//只有当形状划分正确的时候做，而划分不正确时n_ofshapes设为1+n_total
412     for(int i=0;i<n_ofshapes;i++)
413     {
414         ballshapesingriddefault[i]->addtoscene(*sceneonwork);
```

```
    }
    }
}

void ballgrid::removeshapes()//将划分形状从场景移除
{
    if(n_ofshapes<n_total) {//只有当形状划分正确的时候做，而划分不正确时n_ofshapes设为1+n_total
    for(int i=0;i<n_ofshapes;i++)
    {
        ballshapesingriddefault[i]->removeballs(*sceneonwork);
    }
    }
}


void ballgrid::setshaperandposdefault()//将划分形状放到网格外
{
    int colnow=0;
    int rownow=-5;
    for(int i=0;i<n_ofshapes;i++)
    {
        if(colnow < gridnum+2 ) {//一排排完前 //一半竖着放i < n_ofshapes/2
            int rows=veclinkshape[i].getrows();
            int cols=veclinkshape[i].getcols();
            if(cols > rows )
            {
                veclinkshape[i].rotate_right();
                rows=veclinkshape[i].getrows();
                cols=veclinkshape[i].getcols();
            }

            for(int j=0;j<veclinkshape[i].getshapenballs();j++)
            {
                int row=(veclinkshape[i].getptrow(j))-rows-1;
                int col=colnow+(veclinkshape[i].getptcol(j));

               ballshapesingriddefault[i]->setballpos(j,col*balldiameter,row*balldiameter);
            }
            colnow=colnow+cols+1;
        }
        else//另一半横着放
        {
            int rows=veclinkshape[i].getrows();
            int cols=veclinkshape[i].getcols();
            if(cols < rows )
            {
                veclinkshape[i].rotate_right();
                rows=veclinkshape[i].getrows();
                cols=veclinkshape[i].getcols();
            }
            for(int j=0;j<veclinkshape[i].getshapenballs();j++)
            {
                int row=rownow+(veclinkshape[i].getptrow(j));
                int col=colnow+(veclinkshape[i].getptcol(j));

```

```
470                 ballshapesingriddefault[i]->setballpos(j,col*balldiameter,row*balldiameter);
471             }
472             rownow=rownow+rows+1;
473         }
474         ballshapesingriddefault[i]->recordpos();
475         veclinkshape[i].recordoutercoord();
476     }
477 }
478
479
480 void ballgrid::shapetrans(int shapeida,int transid)//形状变换:旋转和翻转
481 {
482     if(shapeidtosit!=shapeida){
483         qDebug()<<"error:"<<shapeidtosit<<shapeida;
484     }
485
486     QPointF posa=ballshapesingriddefault[shapeida-1]->getballpos(ballidtosit);
487     int boxid=getboxidat(posa)+1;//当前球所在的位置
488     qDebug()<<"pos :"<<posa.x()<<posa.y();
489
490     switch(transid)
491     {
492         case 1://Qt::Key_Right:
493             veclinkshape[shapeida-1].saveoldcoord();//保存原始坐标信息
494             if(boxid <= n_total)//当在网格内时
495             {
496                 clearshapeoccp(boxid,ballidtosit,shapeida);//参数是框的序号，形状中当前球的序号，形状的序号，都从1开始计数
497                 veclinkshape[shapeida-1].rotate_right();
498                 int flagcanbe=canbesetdown(boxid,ballidtosit,shapeida);//参数是框的序号，形状中当前球的序号，形状的序号，都
                        从1开始计数
499                 if(flagcanbe==1)
500                 {
501                     setshapeoccp(boxid,ballidtosit,shapeida);
502                 }else
503                 {
504                     QMessageBox msgBox;
505                     msgBox.setWindowTitle("WARNING");
506                     msgBox.setText("can not rotate right");
507                     msgBox.exec();
508                     veclinkshape[shapeida-1].resetoldcoord();//恢复原始坐标信息
509                     setshapeoccp(boxid,ballidtosit,shapeida);
510                 }
511             }
512             else
513             {
514                 veclinkshape[shapeida-1].rotate_right();
515             }
516             break;
517
518         case 2://Qt::Key_Left:
519             veclinkshape[shapeida-1].saveoldcoord();//保存原始坐标信息
520             if(boxid <= n_total)//当在网格内时
521             {
522                 clearshapeoccp(boxid,ballidtosit,shapeida);//参数是框的序号，形状中当前球的序号，形状的序号，都从1开始计数
523                 veclinkshape[shapeida-1].rotate_left();
```

```
524             int flagcanbe=canbesetdown(boxid,ballidtosit,shapeida);//参数是框的序号，形状中当前球的序号，形状的序号，都
                    从1开始计数
525             if(flagcanbe==1)
526             {
527                 setshapeoccp(boxid,ballidtosit,shapeida);
528             }else
529             {
530                 QMessageBox msgBox;
531                 msgBox.setWindowTitle("WARNING");
532                 msgBox.setText("can not rotate left");
533                 msgBox.exec();
534                 veclinkshape[shapeida-1].resetoldcoord();//恢复原始坐标信息
535                 setshapeoccp(boxid,ballidtosit,shapeida);
536             }
537         }
538         else
539         {
540             veclinkshape[shapeida-1].rotate_left();
541         }
542
543         break;
544
545     case 3://Qt::Key_Up:
546         //qDebug()<<"mirror up and down";
547         veclinkshape[shapeida-1].saveoldcoord();//保存原始坐标信息
548         if(boxid <= n_total)//当在网格内时
549         {
550             clearshapeoccp(boxid,ballidtosit,shapeida);//参数是框的序号，形状中当前球的序号，形状的序号，都从1开始计数
551             veclinkshape[shapeida-1].mirror_ab();
552             int flagcanbe=canbesetdown(boxid,ballidtosit,shapeida);//参数是框的序号，形状中当前球的序号，形状的序号，都
                    从1开始计数
553             if(flagcanbe==1)
554             {
555                 setshapeoccp(boxid,ballidtosit,shapeida);
556             }else
557             {
558                 QMessageBox msgBox;
559                 msgBox.setWindowTitle("WARNING");
560                 msgBox.setText("can not mirror up and down");
561                 msgBox.exec();
562                 //qDebug()<<"can not mirror up and down";
563                 veclinkshape[shapeida-1].resetoldcoord();//恢复原始坐标信息
564                 setshapeoccp(boxid,ballidtosit,shapeida);
565             }
566         }
567         else
568         {
569             veclinkshape[shapeida-1].mirror_ab();
570         }
571         break;
572
573     //case 4://Qt::Key_Down:
574     // veclinkshape[shapeida-1].mirror_lr();
575     // break;
576
```

```
577         default:
578             break;
579     }
580
581     int n_balls=veclinkshape[shapeida-1].getshapenballs();
582     int rowfocusinshape=veclinkshape[shapeidtosit-1].getptrow(ballidtosit-1);
583     int colfocusinshape=veclinkshape[shapeidtosit-1].getptcol(ballidtosit-1);
584
585     for(int j=0;j<n_balls;j++)
586         {
587             int row=(veclinkshape[shapeidtosit-1].getptrow(j))-rowfocusinshape;
588             int col=(veclinkshape[shapeidtosit-1].getptcol(j))-colfocusinshape;
589
590             ballshapesingriddefault[shapeidtosit-1]->setballpos(j,col*balldiameter+posa.x(),row*balldiameter+posa.y());
591         }
592 }
593
594 void ballgrid::setshapes(int arraysize)//根据grid函数partion后得到的std::vector<linkshape> veclinkshape;
595 {
596     int ballpos[arraysize];//形状中球的位置的临时数组设置的大一些，因为各个形状中球数是不同的
597     //int ballpos[arraysize];
598
599     QColor colora[256];
600     for(int i=0;i<256;i++)
601     {
602         colora[i]=QColor(qrand()%256,qrand()%256,i);
603     }
604
605     ballshapesingrid.clear();
606     for(int i=0;i<n_ofshapes;i++)
607     {
608         int id=veclinkshape[i].getshapesn();
609         int n_ballsinshape=veclinkshape[i].getshapenballs();
610         //qDebug()<<"id:"<<id;
611         //qDebug()<<"n_ball_inshape:"<<n_ballsinshape;
612
613         for(int j=0;j<n_ballsinshape;j++)
614         {
615             int row=(veclinkshape[i].getptrow(j)+veclinkshape[i].getrowofzero());
616             int col=(veclinkshape[i].getptcol(j)+veclinkshape[i].getcolofzero());
617             //qDebug()<<"ball sn="<<veclinkshape[i].getballsn(j);
618             //qDebug()<<"row="<<row<<" col="<<col;
619             ballpos[2*j]=(col*balldiameter);
620             ballpos[2*j+1]=(row*balldiameter);
621         }
622
623         ballshape *shapea=new ballshape();
624         shapea->setballdiameter(balldiameter);
625         shapea->setballshape(id,n_ballsinshape,ballpos,colora[qrand()%256]);
626         ballshapesingrid.append(shapea);//形状图形列表ballshapesingrid，对应的生成的形状veclinkshape
627
628         QObject::connect(shapea,&ballshape::shapeselected,this,&ballgrid::dealshapetosit);
629         QObject::connect(shapea,&ballshape::shapetoreset,this,&ballgrid::shapereset);
630         QObject::connect(shapea,&ballshape::shapetotrans,this,&ballgrid::shapetrans);
631     }
```

```
}
```

2.5 grid 类

2.5.1 功能分析

在游戏中实际上有两个下三角矩阵网格，一个是由空位 (方框，盒子) 构成的下三角矩阵，一个时由球构成的下三角矩阵，球这个三角矩阵需要进行形状划分等算法，而空位三角矩阵需要进行落位判断等。但这两个三角矩阵只是两种表现形式，真正实际运算中需要的只是一套抽象的三角矩阵，这一套三角矩阵用方框表示构成空位网格，用球表示构成球网格，球网格再划分成不同的球连接形状。

这一个抽象的三角矩阵由 grid 类来实现。grid 类是支持网格划分，形状唯一性判断，落位判断等。

grid 类的保护属性包括 (设置保护是为了方便子类使用):

示例 12 grid 类的保护属性

代码

```
    int n_scale;//网格的大小，行列数均为n_scale
        int n_balls_ofshape;//链接形状中的球数
        int n_ofshapes;//划分成的形状数
    int n_total;
        int n_focus;//<当前网格中的焦点,用于在其位置及邻居位置判断占用情况,从1开始计数，为实际球序号
    std::vector<balltopo> vecgrid;//球列表
        std::vector<linkshape> veclinkshape;//形状列表
```

主要用途包括:

- 生成网格
- 生成/管理网格中的球
- 生成/管理中的形状
- 形状在网格中落位以及移出

setgrid，trigridinit 函数完成网格的生成，以及网格中球的拓扑信息初始化。

partition，checksameshape 函数完成网格形状划分，以及形状的唯一性判断。

setfocus，重载的两个 canbesetdown 函数完成网格落位焦点设置 (即网格空位焦点)，以及根据该焦点和两种不同形式的形状信息进行能否落位判断。而 setshapeoccp，clearshapeoccp 函数用于形状在网格中落位或移出时的标记设置。gridocupied 函数用于所有形状都落位完成，即空位均被占据的判断。

其它 get 开头的函数用于给出各种属性信息，以及网格中球 (或空位) 的序号和行列位置。

2.5.2 代码实现

ballgrid 类的声明和实现，在核心算法设计和测试的时候就放在了一起，所以只有一个文件，如例13所示:

示例 13 grid 类定义

代码

```
#ifndef SHAPE_H
#define SHAPE_H

#include <stdio.h>
#include <vector>
```

```
6  #include <math.h>
7  #include <time.h>
8  #include <shapelink.h>
9  #include <shapeball.h>
10 #include <iostream>
11 #include <cstdlib>
12 #include <qDebug>
13 #include <QMessageBox>
14
15
16 /*
17 球构成的下三角矩阵
18 */
19 class grid
20 {
21 private:
22     int gridid;
23
24 protected:
25     int n_scale;//网格的大小，行列数均为n_scale
26         int n_balls_ofshape;//链接形状中的球数
27         int n_ofshapes;//划分成的形状数
28     int n_total;
29         int n_focus;//<当前网格中的焦点,用于在其位置及邻居位置判断占用情况,从1开始计数，为实际球序号
30     std::vector<balltopo> vecgrid;//球列表
31         std::vector<linkshape> veclinkshape;//形状列表
32
33 public:
34
35 grid()//构造函数,无参数构造函数可以用来声明数组
36 {}
37
38 grid(int n,int m)//重载构造函数
39 {
40     setgrid(n,m);
41 }
42
43 void setgrid(int n,int m)
44 {
45     n_scale=n;
46     n_balls_ofshape=m;
47     std::vector<balltopo> vecgrida;//每次改变都清空
48     vecgrid=vecgrida;
49     balltopo trigrid[n_scale][n_scale];
50     //下三角网格，只是一个下三角矩阵即可
51     int nij=0;
52     //printf("construct:\n");
53     for (int i=0;i<n_scale;i++){//存储结构是先行后列.
54         for(int j=0;j<=i;j++){
55             vecgrid.push_back(trigrid[i][j]);
56             nij++;
57             //printf("%d ",nij);
58         }
59     }
60     //printf("\n");
```

```
61      trigridinit();
62  }
63
64  void trigridinit(){//下三角网格信息初始化
65      balltopo trigrid[n_scale][n_scale];
66      int n=n_scale;
67      int sumlinelessi=0;
68      int sn=0;
69      for(int i=1; i<=n; i++)//行
70      {
71          sumlinelessi=sumlinelessi+(i-1);//i行前的球数,每行的球数都等于该行的行号
72          for(int j=1; j<=i; j++)//列
73          {
74              //序号
75              sn=sumlinelessi+j; //sn_self=sum_{k=1}^{i-1}(i)+j,注意:序号从1开始
76                                                  //但数组是从0开始的
77              //左侧邻居
78              int left=0;
79              if(j==1){
80                  left=-1;//j=1时,左侧无邻居,所以设置为-1
81              }
82              else{
83                  left=sn-1;
84              }
85
86              //右侧邻居
87              int right=0;
88              if(j==i){
89                  right=-1;//j=i时,右侧无邻居,所以设置为-1
90              }
91              else{
92                  right=sn+1;
93              }
94
95              //上方邻居
96              int above=0;
97              if(i==j){
98                  above=-1;//i=j时,上方无邻居,所以设置为-1
99              }
100             else{
101                 above=sn-i+1;
102             }
103
104             //下方邻居
105             int below=0;
106             if(i==n){
107                 below=-1;//i=n时,下方无邻居,所以设置为-1
108             }
109             else{
110                 below=sn+i;
111             }
112
113             //printf("sn:%d,left:%d,right:%d,above:%d,below:%d\n",sn,left,right,above,below);
114             int toposet[5]={sn,left,right,above,below};
115             trigrid[i-1][j-1].settopo(toposet);
```

```
        }
    }
    n_total=sn;
    int nij=0;
    for (int i=0;i<n_scale;i++)
    {
        for(int j=0;j<=i;j++)//[i][j]与矢量中序号是一一对应的。
        {
            trigrid[i][j].setoccp(0);
            trigrid[i][j].setshpd(0);
            vecgrid[nij]=trigrid[i][j];
            nij++;
        }
    }
}

//利用形状中的球数和特征值列表判断形状是否相同，//0表示不同，1表示相同
int checksameshape(linkshape a,linkshape b)
{
    if(a.getshapenballs() != b.getshapenballs()) {
        return 0;
    }else
    {
        for(int i=0;i<a.getshapenballs();i++)
        {
            if(fabs(a.getshapeeigen(i)-b.getshapeeigen(i))>0.000001) return 0;
        }
    }

    //printf("eigen is the same!\n");

    int flagsame=0;
    //printf("ball a rows=%d , cols=%d\n",a.getrows(),a.getcols());
    //printf("ball b rows=%d , cols=%d\n",b.getrows(),b.getcols());
    if(a.getrows() != a.getcols()) {//行列不相等时，只要做翻转就可以比较了
        if(a.getrows()==b.getrows() && a.getcols()==b.getcols() )//当行数列数完全相同
        {
            int sumrowsa=0;
            int sumcolsa=0;
            int sumrowsb=0;
            int sumcolsb=0;
            for(int i=0;i<a.getshapenballs();i++)//正常比较坐标位置行和列加和后的总数
            {
                sumrowsa=sumrowsa+a.getptrow(i);
                sumcolsa=sumcolsa+a.getptcol(i);
                sumrowsb=sumrowsb+b.getptrow(i);
                sumcolsb=sumcolsb+b.getptcol(i);
            }
            //printf("ball a pos sum of rows=%d , of cols=%d\n",sumrowsa,sumcolsa);
            //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
            if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;

            b.mirror_lr();//左右翻转一下
            sumrowsb=0;
            sumcolsb=0;
```

```
171             for(int i=0;i<a.getshapenballs();i++)
172             {
173                 sumrowsb=sumrowsb+b.getptrow(i);
174                 sumcolsb=sumcolsb+b.getptcol(i);
175             }
176             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
177             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
178
179             b.mirror_ab();//接着上下翻转一下
180             sumrowsb=0;
181             sumcolsb=0;
182             for(int i=0;i<a.getshapenballs();i++)
183             {
184                 sumrowsb=sumrowsb+b.getptrow(i);
185                 sumcolsb=sumcolsb+b.getptcol(i);
186             }
187             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
188             if (sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
189
190             b.mirror_lr();//接着再左右翻转一下，4中情况都覆盖到了
191             sumrowsb=0;
192             sumcolsb=0;
193             for(int i=0;i<a.getshapenballs();i++)
194             {
195                 sumrowsb=sumrowsb+b.getptrow(i);
196                 sumcolsb=sumcolsb+b.getptcol(i);
197             }
198             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
199             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
200         }
201         else if(a.getrows()==b.getcols() && a.getcols()==b.getrows() )//当a行数=b列数，a列数=b行数
202         {
203             b.rotate_right();//转过来使得行，列数相同
204             int sumrowsa=0;
205             int sumcolsa=0;
206             int sumrowsb=0;
207             int sumcolsb=0;
208             for(int i=0;i<a.getshapenballs();i++)//正常比较坐标位置行和列加和后的总数
209             {
210                 sumrowsa=sumrowsa+a.getptrow(i);
211                 sumcolsa=sumcolsa+a.getptcol(i);
212                 sumrowsb=sumrowsb+b.getptrow(i);
213                 sumcolsb=sumcolsb+b.getptcol(i);
214             }
215             //printf("ball a pos sum of rows=%d , of cols=%d\n",sumrowsa,sumcolsa);
216             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
217             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
218
219             b.mirror_lr();//左右翻转一下
220             sumrowsb=0;
221             sumcolsb=0;
222             for(int i=0;i<a.getshapenballs();i++)
223             {
224                 sumrowsb=sumrowsb+b.getptrow(i);
225                 sumcolsb=sumcolsb+b.getptcol(i);
```

```
226             }
227             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
228             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
229
230             b.mirror_ab();//接着上下翻转一下
231             sumrowsb=0;
232             sumcolsb=0;
233             for(int i=0;i<a.getshapenballs();i++)
234             {
235                 sumrowsb=sumrowsb+b.getptrow(i);
236                 sumcolsb=sumcolsb+b.getptcol(i);
237             }
238             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
239             if (sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
240
241             b.mirror_lr();//接着再左右翻转一下，4中情况都覆盖到了
242             sumrowsb=0;
243             sumcolsb=0;
244             for(int i=0;i<a.getshapenballs();i++)
245             {
246                 sumrowsb=sumrowsb+b.getptrow(i);
247                 sumcolsb=sumcolsb+b.getptcol(i);
248             }
249             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
250             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
251         }
252         else
253         {
254             return 0;
255         }
256     }
257     else//行列相等时，需要做旋转和翻转
258     {
259         if(b.getrows() != b.getcols())
260         {
261             return 0;
262         }else
263         {
264             int sumrowsa=0;
265             int sumcolsa=0;
266             int sumrowsb=0;
267             int sumcolsb=0;
268             for(int i=0;i<a.getshapenballs();i++)//正常比较坐标位置行和列加和后的总数
269             {
270                 sumrowsa=sumrowsa+a.getptrow(i);
271                 sumcolsa=sumcolsa+a.getptcol(i);
272                 sumrowsb=sumrowsb+b.getptrow(i);
273                 sumcolsb=sumcolsb+b.getptcol(i);
274             }
275             //a.pos_show();
276             //b.pos_show();
277             //printf("ball a pos sum of rows=%d , of cols=%d\n",sumrowsa,sumcolsa);
278             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
279             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
280
```

```
281             b.rotate_right();//顺时针旋转90度
282             sumrowsb=0;
283             sumcolsb=0;
284             for(int i=0;i<a.getshapenballs();i++)
285             {
286                 sumrowsb=sumrowsb+b.getptrow(i);
287                 sumcolsb=sumcolsb+b.getptcol(i);
288             }
289             //b.pos_show();
290             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
291             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
292
293             b.rotate_right();//顺时针旋转180度
294             sumrowsb=0;
295             sumcolsb=0;
296             for(int i=0;i<a.getshapenballs();i++)
297             {
298                 sumrowsb=sumrowsb+b.getptrow(i);
299                 sumcolsb=sumcolsb+b.getptcol(i);
300             }
301             //b.pos_show();
302             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
303             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
304
305             b.rotate_right();//顺时针旋转270度
306             sumrowsb=0;
307             sumcolsb=0;
308             for(int i=0;i<a.getshapenballs();i++)
309             {
310                 sumrowsb=sumrowsb+b.getptrow(i);
311                 sumcolsb=sumcolsb+b.getptcol(i);
312             }
313             //b.pos_show();
314             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
315             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
316
317             b.rotate_right();//顺时针旋转360度，还原
318             b.mirror_ab();//接着上下翻转一下
319             sumrowsb=0;
320             sumcolsb=0;
321             for(int i=0;i<a.getshapenballs();i++)
322             {
323                 sumrowsb=sumrowsb+b.getptrow(i);
324                 sumcolsb=sumcolsb+b.getptcol(i);
325             }
326             //b.pos_show();
327             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
328             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
329
330             b.rotate_right();//接着再顺时针旋转90度
331             sumrowsb=0;
332             sumcolsb=0;
333             for(int i=0;i<a.getshapenballs();i++)
334             {
335                 sumrowsb=sumrowsb+b.getptrow(i);
```

```
336                 sumcolsb=sumcolsb+b.getptcol(i);
337             }
338             //b.pos_show();
339             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
340             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
341
342             b.rotate_right();//接着再顺时针旋转180度
343             sumrowsb=0;
344             sumcolsb=0;
345             for(int i=0;i<a.getshapenballs();i++)
346             {
347                 sumrowsb=sumrowsb+b.getptrow(i);
348                 sumcolsb=sumcolsb+b.getptcol(i);
349             }
350             //b.pos_show();
351             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
352             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
353
354             b.rotate_right();//接着再顺时针旋转270度
355             sumrowsb=0;
356             sumcolsb=0;
357             for(int i=0;i<a.getshapenballs();i++)
358             {
359                 sumrowsb=sumrowsb+b.getptrow(i);
360                 sumcolsb=sumcolsb+b.getptcol(i);
361             }
362             //b.pos_show();
363             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
364             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
365
366         }
367
368     }
369
370     if(flagsame==0) return 0;
371     return 1;
372 }
373
374 void partition(int ntotal)//将球划分到形状中
375 {
376         int i,j,k;
377         int sn_now,sn_ball,posrow_now,poscol_now;
378     linkshape shapevec[ntotal];//这个地方是要特别注意的，因为数组的声明需要的是常数，无法直接用类的属性，即便某些时候不出
            错，但也不是好的方式，因此增加一个输入ntotal相当于是给出了常数。
379     linkshape shapevecrec[ntotal];
380         float randtemp;
381         srand(time(NULL));
382         int shapenumber,shapenumberrec;
383         int flagnoleft;//网格中存在球未划分的标记=0，表示还需要划分，>0否则不需要划分
384         int flaginshape;//用于表示当前球是在形状中的标记
385         int flagsameshape;
386
387
388     posrow_now=0;
389     poscol_now=0;
```

```
    shapenumberrec=ntotal+1;
        int ndopart;
    for(ndopart=0;ndopart<2000;ndopart++){
                //printf("----------------------------------------\n");
                //printf("times of partitions:%d\n",ndopart);
        //qDebug()<<"times of partitions:"<<ndopart;

        for(i=0;i<ntotal;i++)//清除划分状态，便于再次划分
                {
                        vecgrid[i].setshpd(0);
                }

                shapenumber=0;
                flagnoleft=0;
                int ndoshapes=0;//用于避免无法划分出不同形状完全覆盖网格时的死循环

        while (flagnoleft==0 && ndoshapes<200)
                {
                        ndoshapes++;
                        //printf("times of create different shape to cover all the grid :%d\n",ndoshapes);
            coord ptpos[n_balls_ofshape]={0,0};
                        int ptnum[n_balls_ofshape]={0};


                        //确定起点
//   k=0;
//   i=0;
// while(i<ntotal)
//   {
//    if(vecgrid[i].getshpd()==0){
//     sn_now=i;
//     posrow_now=0;
//     poscol_now=0;
//     vecgrid[sn_now].setshpd(1);
//     ptnum[k]=sn_now;
//     ptpos[k].row=posrow_now;
//     ptpos[k].col=poscol_now;
//     //printf("i=%d",i);
// i=ntotal;
//    } else {
//     i++;
//     //printf("i=%d",i);
// if(i >= ntotal) flagnoleft=1;
//     //printf("flagnoleft=%d",flagnoleft);
//    }
//   }

            //确定起点机制换一种,采用随机选取的方式进行
            k=0;
            do
            {
                flagnoleft=1;//首先确定是否存在没有连接的球
                for(i=0;i<n_total;i++)
                {
                    if(vecgrid[i].getshpd()==0) flagnoleft=0;
```

```
            }
            sn_now=rand()%n_total;
        }
        while(vecgrid[sn_now].getshpd()==1 && flagnoleft==0);
        if(vecgrid[sn_now].getshpd()==0 && flagnoleft==0)
        {
            vecgrid[sn_now].setshpd(1);
            coord coorda=getrowcolformsn(sn_now+1);
            posrow_now=coorda.row;
            poscol_now=coorda.col;
            ptnum[k]=sn_now;
            ptpos[k].row=posrow_now;
            ptpos[k].col=poscol_now;
        }

                if(flagnoleft==0) {

                    // printf("\ninitial ball numbers:");
                    // for(j=0;j<=k;j++)
                    // {
                        // printf("%d ",ptnum[j]+1);//球真实序号比矢量中的序号大1
                    // }
                    // printf("\ninitial ball postions:");
                    // for(j=0;j<=k;j++)
                    // {
                        // printf("(%d,%d) ",ptpos[j].row,ptpos[j].col);
                    // }
                    // printf("\n");


                    int nstep=0;
                    while(k<n_balls_ofshape-1 && nstep < 100){//当球数未达到要求，走动次数小于100次时做
                        nstep++;
                        randtemp=(rand()%100/100.0);
                        //printf("steps of moving try to create a shape:%d. ",nstep);
                        //printf("nstep=%d\n",nstep);
                        //printf("randt=%f\n",randtemp);

                        if(randtemp < 0.25)//右移
                        {
                            //printf("sn_now=%d\n",sn_now);
                            //printf("right of sn_now=%d\n",vecgrid[sn_now].getright()-1);
                            sn_ball=vecgrid[sn_now].getright();
                            if(sn_ball>0) {
                                if(vecgrid[sn_ball-1].getshpd() == 0){
                                    sn_now=sn_ball-1;//矢量中的序号比球真实序号小1
                                    poscol_now++;
                                    k++;
                                    ptnum[k]=sn_now;
                                    ptpos[k].row=posrow_now;
                                    ptpos[k].col=poscol_now;
                                    vecgrid[sn_now].setshpd(1);
                                }else{//处理往回走的情况，但不能走到其它形状中
                                    flaginshape=0;
                                    for(j=0;j<=k;j++)
```

```
                                {
                                        if(sn_ball-1==ptnum[j]) flaginshape=1;
                                }
                                if(flaginshape==1) {//当移动到的球在形状内则确定移动
                                        sn_now=sn_ball-1;
                                        poscol_now++;
                                }
                        }
                }
        }
        else if(randtemp < 0.5)//左移
        {
                //printf("sn_now=%d\n",sn_now);
                //printf("left of sn_now=%d\n",vecgrid[sn_now].getleft()-1);
                sn_ball=vecgrid[sn_now].getleft();
                if(sn_ball>0) {
                        if(vecgrid[sn_ball-1].getshpd() == 0){
                                sn_now=sn_ball-1;//矢量中的序号比球真实序号小1
                                poscol_now--;
                                k++;
                                ptnum[k]=sn_now;
                                ptpos[k].row=posrow_now;
                                ptpos[k].col=poscol_now;
                                vecgrid[sn_now].setshpd(1);
                        }else{//处理往回走的情况，但不能走到其它形状中
                                flaginshape=0;
                                for(j=0;j<=k;j++)
                                {
                                        if(sn_ball-1==ptnum[j]) flaginshape=1;
                                }
                                if(flaginshape==1) {
                                        sn_now=sn_ball-1;
                                        poscol_now--;
                                }
                        }
                }
        }
        else if(randtemp < 0.75)//上移
        {
                //printf("sn_now=%d\n",sn_now);
                //printf("above of sn_now=%d\n",vecgrid[sn_now].getabove()-1);
                sn_ball=vecgrid[sn_now].getabove();
                if(sn_ball>0) {
                        if(vecgrid[sn_ball-1].getshpd() == 0){
                                sn_now=sn_ball-1;//矢量中的序号比球真实序号小1
                                posrow_now--;
                                k++;
                                ptnum[k]=sn_now;
                                ptpos[k].row=posrow_now;
                                ptpos[k].col=poscol_now;
                                vecgrid[sn_now].setshpd(1);
                        }else{//处理往回走的情况，但不能走到其它形状中
                                flaginshape=0;
                                for(j=0;j<=k;j++)
                                {
```

```
555                                     if(sn_ball-1==ptnum[j]) flaginshape=1;
556                                 }
557                                 if(flaginshape==1) {
558                                     sn_now=sn_ball-1;
559                                     posrow_now--;
560                                 }
561                             }
562                         }
563                     }
564                     else//下移
565                     {
566                         //printf("sn_now=%d\n",sn_now);
567                         //printf("below of sn_now=%d\n",vecgrid[sn_now].getbelow()-1);
568                         sn_ball=vecgrid[sn_now].getbelow();
569                         if(sn_ball>0) {
570                             if(vecgrid[sn_ball-1].getshpd() == 0){
571                                 sn_now=sn_ball-1;//矢量中的序号比球真实序号小1
572                                 posrow_now++;
573                                 k++;
574                                 ptnum[k]=sn_now;
575                                 ptpos[k].row=posrow_now;
576                                 ptpos[k].col=poscol_now;
577                                 vecgrid[sn_now].setshpd(1);
578                             }else{//处理往回走的情况，但不能走到其它形状中
579                                 flaginshape=0;
580                                 for(j=0;j<=k;j++)
581                                 {
582                                     if(sn_ball-1==ptnum[j]) flaginshape=1;
583                                 }
584                                 if(flaginshape==1) {
585                                     sn_now=sn_ball-1;
586                                     posrow_now++;
587                                 }
588                             }
589                         }
590                     }
591                     //printf("sn_now=%d, real_sn_of_ball=%d\n",sn_now,sn_now+1);
592                     // int tempa;
593                     // std::cin>>tempa;
594                 }
595
596
597                 //显示一下当前的形状的球序号和相对坐标
598                 // trigriddisplay();
599                 // printf("\n");
600                 // printf("ball numbers:");
601                 // for(j=0;j<=k;j++)
602                 // {
603                     // printf("%d ",ptnum[j]+1);
604                 // }
605                 // printf("\n");
606                 // printf("ball postions:");
607                 // for(j=0;j<=k;j++)
608                 // {
609                     // printf("(%d,%d) ",ptpos[j].row,ptpos[j].col);
```

```
                        // }
                        // printf("\n");
                        // printf("number of balls in this shape:%d\n",k+1);//从0开始的当前形状中的球数+1。
                        //int tempa;
                        //std::cin>>tempa;


                        //将当前形状放到形状数组中
                        coord vectmp[k+1];
                        int vecnumber[k+1];
                        for(j=0;j<=k;j++)
                        {
                                vectmp[j].row=ptpos[j].row;
                                vectmp[j].col=ptpos[j].col;
                                vecnumber[j]=ptnum[j]+1;//输出真实的球序号，而不是从0开始的序号。
                        }
                        linkshape shapea=linkshape(shapenumber+1,k+1,vectmp,vecnumber);
                        //先判断形状是否与原先构成的相同
                        if(shapenumber==0) {
                                shapevec[shapenumber]=shapea;
                                shapenumber++;
                        }else{
                                flagsameshape=0;
                                for(j=0;j<shapenumber;j++)
                                {
                                        if(checksameshape(shapea,shapevec[j]))
                                        {
                                                flagsameshape=1;
                                                //printf("flagsameshape=%d\n",flagsameshape);
                                        }
                                }
                                if(flagsameshape==0){//和之前的形状不相同
                                        shapevec[shapenumber]=shapea;
                                        shapenumber++;
                                }else{
                                        for(j=0;j<=k;j++)
                                        {
                                                vecgrid[ptnum[j]].setshpd(0);
                                        }
                                }
                        }
                        //int tempa;
                        //std::cin>>tempa;
                }
        }

        //输出当前次，划分的形状
        // printf("\n");
        // printf("do the %d th time's partition\n",ndopart);
        // printf("number of shapes in grid is %d\n",shapenumber);
    //qDebug()<<"total number of shapes in grid is "<<shapenumberrec;
        // for(j=0;j<shapenumber;j++)
        // {
                // printf("j=%d ,shape sn: %d\n",j,shapevec[j].getshapesn());
                // printf("j=%d ,total number of balls in shape: %d\n",j,shapevec[j].getshapenballs());
```

```
665                         // printf("j=%d ,serial number of balls in shape:",j);
666                         // for(i=0;i<shapevec[j].getshapenballs();i++){
667                         // printf("%d ",shapevec[j].getballnumber(i));}
668                         // printf("\n");
669                 // }
670                 // if(flagnoleft==0) {
671                         // printf("partition is not completed,grid is not covered\n");
672                 // }
673
674                 //用形状数来判断是否是最佳划分，形状数最小的肯定是最佳的，
675                 //并记录到shapevecrec数组中
676         if(shapenumber<shapenumberrec and flagnoleft==1 ) {
677                         for(j=0;j<shapenumber;j++)
678                         {
679                                 shapevecrec[j]=shapevec[j];
680                         }
681                         shapenumberrec=shapenumber;
682             break;//为减小计算时间，不做最佳分配了，因为游戏也不需要最佳，不是做算法。
683                 }
684
685                 //int tempa;
686                 //std::cin>>tempa;
687         }
688
689                 //输出并记录最佳划分，划分的形状
690         if(shapenumberrec<n_total){//如果没有划分则不做输出
691             //printf("\n");
692             //trigriddisplay();
693             //printf("\n");
694             //printf("best partition for %d times' attempt:\n",ndopart);
695             //printf("total number of shapes in grid is %d\n",shapenumberrec);
696             //qDebug()<<"best partition for"<<ndopart <<" times' attempt:";
697             qDebug()<<"total number of shapes in grid is "<<shapenumberrec;
698             for(j=0;j<shapenumberrec;j++)
699             {
700                 //printf("j=%d ,shape sn: %d\n",j,shapevecrec[j].getshapesn());
701                 //printf("j=%d ,total number of balls in shape: %d\n",j,shapevecrec[j].getshapenballs());
702                 //printf("j=%d ,serial number of balls in shape:",j);
703                 //for(i=0;i<shapevecrec[j].getshapenballs();i++){
704                 //printf("%d ",shapevecrec[j].getballnumber(i));}
705                 //printf("\n");
706                 //把链接形状放到veclinkshape形状中
707                 veclinkshape.push_back(shapevecrec[j]);
708             }
709         }
710         n_ofshapes=shapenumberrec;//记录形状数，当形状数大于球数时表示形状划分失败，用于后面的判断
711
712 }
713
714 void setfocus(const int i){//设置焦点框序号,序号为框的真实序号，从1开始计数
715     n_focus=i;
716 }
717
718 int canbesetdown(int n_balls,int sn_focus_inshape,std::vector<coord> ballscoords)//根据焦点球序号以及形状中焦点和坐标确定
        形状能否放入网格中
```

```
719 //n_balls形状中球数，sn_focus_inshape为形状中焦点球序号从1开始计数，ballscoords形状中各点行列坐标
720 {
721         int row=ballscoords[sn_focus_inshape-1].row;
722         int col=ballscoords[sn_focus_inshape-1].col;
723         int flagcanbe=1;
724         for(int i=0;i<n_balls;i++)
725         {
726                 int drow=ballscoords[i].row -row;
727                 int dcol=ballscoords[i].col -col;
728                 int sn_ball=getnumdrowdcola(drow,dcol);
729         //qDebug()<<"drow:"<<drow;
730         //qDebug()<<"dcol:"<<dcol;
731         //qDebug()<<"sn_ball now:"<<sn_ball;
732                 if(sn_ball<0 || vecgrid[sn_ball-1].getoccp()==1){//当球不在范围内或者球所在框已被占据则无法放入
733                         flagcanbe=0;
734                 }
735         }
736
737         return flagcanbe;
738 }
739
740 int canbesetdown(int boxid,int ballid,int shapeid)//参数是框的序号，形状中当前球的序号，形状的序号，都从1开始计数
741 {
742     setfocus(boxid);
743     int n_balls=veclinkshape[shapeid-1].getshapenballs();
744     int row=veclinkshape[shapeid-1].getptrow(ballid-1);
745     int col=veclinkshape[shapeid-1].getptcol(ballid-1);
746     int flagcanbe=1;
747     for(int i=0;i<n_balls;i++)
748     {
749         int drow=veclinkshape[shapeid-1].getptrow(i)-row;
750         int dcol=veclinkshape[shapeid-1].getptcol(i)-col;
751         int sn_ball=getnumdrowdcola(drow,dcol);
752         //qDebug()<<"drow:"<<drow;
753         //qDebug()<<"dcol:"<<dcol;
754         //qDebug()<<"sn_ball now:"<<sn_ball;
755         if(sn_ball<0 || vecgrid[sn_ball-1].getoccp()==1)//当球不在范围内或者球所在框已被占据则无法放入
756         {
757             flagcanbe=0;
758         }
759     }
760     return flagcanbe;
761 }
762
763 void clearshapeoccp(int boxid,int ballid,int shapeid)//参数是框的序号，形状中当前球的序号，形状的序号，都从1开始计数
764 {
765
766     setfocus(boxid);
767     int n_balls=veclinkshape[shapeid-1].getshapenballs();
768     int row=veclinkshape[shapeid-1].getptrow(ballid-1);
769     int col=veclinkshape[shapeid-1].getptcol(ballid-1);
770     for(int i=0;i<n_balls;i++)
771     {
772         int drow=veclinkshape[shapeid-1].getptrow(i)-row;
773         int dcol=veclinkshape[shapeid-1].getptcol(i)-col;
```

```
774         int sn_ball=getnumdrowdcola(drow,dcol);
775         vecgrid[sn_ball-1].setoccp(0);
776         //qDebug()<<"sn_ball clear occp:"<<sn_ball;
777     }
778 }
779
780 void setshapeoccp(int boxid,int ballid,int shapeid)//参数是框的序号，形状中当前球的序号，形状的序号，都从1开始计数
781 {
782
783     setfocus(boxid);
784     int n_balls=veclinkshape[shapeid-1].getshapenballs();
785     int row=veclinkshape[shapeid-1].getptrow(ballid-1);
786     int col=veclinkshape[shapeid-1].getptcol(ballid-1);
787     for(int i=0;i<n_balls;i++)
788     {
789         int drow=veclinkshape[shapeid-1].getptrow(i)-row;
790         int dcol=veclinkshape[shapeid-1].getptcol(i)-col;
791         int sn_ball=getnumdrowdcola(drow,dcol);
792         vecgrid[sn_ball-1].setoccp(1);
793     }
794 }
795
796 int gridocupied() //判断网格是否全被占据
797 {
798     int flaggridoccp=1;
799     for(int i=0;i<n_total;i++)
800     {
801         if(vecgrid[i].getoccp()==0) flaggridoccp=0;
802     }
803     return flaggridoccp;
804 }
805
806 int getntotal()
807 {
808     return n_total;
809 }
810
811 coord getrowcolformsn(int sn_ball){//输入是球的真实序号
812         int nij=0;
813     coord a={0,0};
814         for(int i=0;i<n_scale;i++)
815         {
816                 for(int j=0;j<=i;j++)
817                 {
818                         nij++;
819                         if(nij==sn_ball)
820                         {
821                 a.row=i;
822                 a.col=j;
823                 break;
824                         }
825                 }
826         }
827     return a;
828 }
```

```
829
830 int getsnfromrowcol(int row,int col){//输入的行和列是以0为起点的
831         if(row<0) return -1;
832         if(col<0) return -1;
833         if(row>n_scale) return -1;
834         if(col>row) return -1;
835
836         int nij=0;
837         for(int i=0;i<row;i++)
838         {
839                 for(int j=0;j<=i;j++)
840                 {
841                         nij++;
842                 }
843         }
844         return nij+col+1;//sn_ball
845 }
846
847 int getnumdrowdcol(int drow,int dcol){//根据焦点球序号以及与焦点球的行距和列距确定当前球序号，利用球的位置和序号关系
848
849          coord a=getrowcolformsn(n_focus);//获取焦点球的行列
850          int row=a.row+drow;
851          int col=a.col+dcol;
852          return getsnfromrowcol(row,col);
853
854 }
855
856 int getnumdrowdcola(int drow,int dcol){//根据焦点球序号以及与焦点球的行距和列距确定当前球序号，利用球的拓扑关系
857
858     //这种思路要注意的是因为列数总是小于等于当前行的行号，
859     //当drow>0则所以先走行，再走列，不会出现先越界的情况
860     //当drow<=0则先走列，再走行，则不会出现先越界的情况
861         int flagoutrange=0;
862         int sn_now=n_focus-1;//sn_now是在矢量中的序号，等于球真实序号-1
863     if(drow >0) {
864                 for(int j=0;j<fabs(drow);j++)
865                 {
866             if(sn_now>=0) {//注意行增加是往下走，即below
867                 sn_now=vecgrid[sn_now].getbelow()-1;//sn_now是在矢量中的序号，等于球真实序号-1
868                         }else
869                         {
870                                 flagoutrange=1;
871                                 j=fabs(drow);
872                         }
873                 }
874     }
875         if(dcol <=0) {
876                 for(int j=0;j<fabs(dcol);j++)
877                 {
878                         if(sn_now>=0) {
879                                 sn_now=vecgrid[sn_now].getleft()-1;//sn_now是在矢量中的序号，等于球真实序号-1
880                         }else
881                         {
882                                 flagoutrange=1;
883                                 j=fabs(dcol);
```

```
884                         }
885                 }
886         }else{
887                 for(int j=0;j<fabs(dcol);j++)
888                 {
889                         if(sn_now>=0) {
890                                 sn_now=vecgrid[sn_now].getright()-1;//sn_now是在矢量中的序号，等于球真实序号-1
891                         }else
892                         {
893                                 flagoutrange=1;
894                                 j=fabs(dcol);
895                         }
896                 }
897         }
898     if(drow <=0) {
899         for(int j=0;j<fabs(drow);j++)
900         {
901             if(sn_now>=0) {
902                 sn_now=vecgrid[sn_now].getabove()-1;//sn_now是在矢量中的序号，等于球真实序号-1
903             }else
904             {
905                 flagoutrange=1;
906                 j=fabs(drow);
907             }
908         }
909     }
910         if(flagoutrange==1){
911                 return -1;
912         }else{
913         return sn_now+1;
914         }
915
916 }
917
918 void trigriddisplay(){//显示下三角网格信息函数
919     balltopo trigrid[n_scale][n_scale];
920         int nij=0;
921     for (int i=0;i<n_scale;i++){
922             for(int j=0;j<=i;j++)
923                         {
924                                 //矩阵和矢量的关系是
925                 trigrid[i][j]=vecgrid[nij];
926                                 nij++;
927                         }
928         }
929
930     for(int i=1; i<=n_scale; i++)//行
931         {
932             for(int j=1; j<=i; j++)//列
933             {
934                 if(j==i){printf(" %5d\n",trigrid[i-1][j-1].getsn());}
935                 else{printf(" %5d",trigrid[i-1][j-1].getsn());}
936             }
937         }
938
```

```
939         // for(int i=0;i<n_total;i++)
940         // {
941                 // printf("%5d ",vecgrid[i].getsn());
942         // }
943         // printf("\n");
944
945 }
946
947 void trigridtest(){//查看当前球及其邻居序号测试
948     balltopo trigrid[n_scale][n_scale];
949         int nij=0;
950     for (int i=0;i<n_scale;i++)
951         {
952         for(int j=0;j<=i;j++)
953                 {
954             trigrid[i][j]=vecgrid[nij];
955                         //printf("%d sn=%d\n",nij,vecgrid[nij].getsn());
956                         nij++;
957                 }
958     }
959     int n=n_scale;
960
961     for(int i=1; i<=n; i++)//行
962         {
963             for(int j=1; j<=i; j++)//列
964             {
965                 trigriddisplay();
966                 int topoget[5]={0};
967                 trigrid[i-1][j-1].gettopo(topoget);
968                 printf("%5c%5d%5c\n",' ',topoget[3],' ');
969                 printf("%5d%5d%5d\n",topoget[1],topoget[0],topoget[2]);
970                 printf("%5c%5d%5c\n",' ',topoget[4],' ');
971
972                 //char ch;
973                 //printf("please put in a key to continue:");
974                 //ch = getchar();
975             }
976         }
977 }
978
979 };
980
981
982  #endif // SHAPE_H
```

2.6 balltopo 类

2.6.1 功能分析

balltopo 类用于表示网格 grid 中的球的拓扑信息。

balltopo 类的属性包括:

示例 14 balltopo 类的属性

代码

```
1     int sn_self;//< topo[0] //是真实序号，从1开始计数
2     int sn_left;//< topo[1] //存在邻居则设置邻居序号，不存在则设置为-1
3     int sn_right;
```

```
4     int sn_above;
5     int sn_below;
6         int occupied; //用于判断网格中的球的位置是否已经占用,1表示占用,0表示不占用
7         int shaped;//用于表示已经划分到形状中
```

其中自身和邻居的序号信息用于表示拓扑关系，occupied 用于落位判断，shaped 用于形状划分。

balltopo 类的方法主要是用来获取或设置属性。

2.6.2 代码实现

balltopo 类的声明和实现，在核心算法设计和测试的时候就放在了一起，所以只有一个文件，如例15所示:

示例 15 balltopo 类定义

代码

```
1  #ifndef SHAPEBALL_H
2  #define SHAPEBALL_H
3
4  #include <stdio.h>
5  #include <vector>
6  #include <math.h>
7  #include <iostream>
8  #include <cstdlib>
9  #include <qDebug>
10
11 /**球的拓扑类
12 主要是设置球二维拓扑中左右上下的邻居信息
13 存在邻居则设置为邻居的编号
14 否则设置为负
15 */
16 class balltopo
17 {
18 private:
19     int sn_self;//< topo[0] //是真实序号，从1开始计数
20     int sn_left;//< topo[1] //存在邻居则设置邻居序号，不存在则设置为-1
21     int sn_right;
22     int sn_above;
23     int sn_below;
24         int occupied; //用于判断网格中的球的位置是否已经占用,1表示占用,0表示不占用
25         int shaped;//用于表示已经划分到形状中
26
27 public:
28     void settopo(int topo[5])//根据设置球二维拓扑中左右上下的邻居信息
29     {
30         sn_self=topo[0];
31         sn_left=topo[1];
32         sn_right=topo[2];
33         sn_above=topo[3];
34         sn_below=topo[4];
35     }
36
37         void setoccp(int occp)
38     {
39                 occupied=occp;
40     }
41
```

```
42          void setshpd(int shpd)
43      {
44                  shaped=shpd;
45      }
46
47          int getoccp()
48      {
49                  return occupied;
50      }
51
52          int getshpd()
53      {
54                  return shaped;
55      }
56
57      int getsn()//得到当前球的编号/序号
58      {
59          return sn_self;
60      }
61
62      void gettopo(int topo[5])//得到当前球在二维拓扑中左右上下的邻居信息
63      {
64          topo[0]=sn_self;
65          topo[1]=sn_left;
66          topo[2]=sn_right;
67          topo[3]=sn_above;
68          topo[4]=sn_below;
69      }
70
71          int getright()
72      {
73          return sn_right;
74      }
75
76          int getleft()
77      {
78          return sn_left;
79      }
80
81          int getabove()
82      {
83          return sn_above;
84      }
85          int getbelow()
86      {
87          return sn_below;
88      }
89
90  };
91
92
93  typedef struct axiscood{//坐标结构体,注意xy表示在形状矩阵中的行号和列号.行号从上到下增加,列号从左到右增加
94  int row;//x表示row
95  int col;//y表示col
96  }coord;
```

```
97
98
99 #endif // SHAPEBALL_H
```

在该文件内还定义了一个结构体 axiscood 用于表示球的行列坐标信息，并定义了一个表示坐标的新数据类型 coord。

2.7 linkshape 类

2.7.1 功能分析

linkshape 类用于表示网格 grid 中划分出的形状的信息，实现形状的描述。

linkshape 类的属性包括: 形状序号，形状中的球数，形状中球的序号和行列坐标信息以及邻接矩阵及其特征值，还有一些存储的备用的坐标信息。

示例 16 linkshape 类的属性

代码

```
1       int sn_shape; //<当前链接形状序号,从1开始计数，这是真实的数字
2       int n_balls_inshape; //<形状的链接的球数
3
4   std::vector<int> vec_balls_number;//球的序号
5       int rows,cols; //<链接形状当前的总行数和总列数
6   int rowofzero,colofzero; //因为要做规范化话，则把规范化后(0,0)的对应行号记录下来
7       std::vector<coord> vec_coord_shape;//位置坐标矢量与形状矩阵相关
8       std::vector<int> vec_matrix_adjacent;//邻接矩阵矢量
9       std::vector<float> vec_eigen_matrix;//邻接矩阵特征值列表
10
11  std::vector<coord> vec_coord_origin;// 位置坐标矢量－在网格中的初始坐标记录下来
12  std::vector<coord> vec_coord_outer;//位置坐标矢量－在网格外的初始坐标记录下来
13  std::vector<coord> vec_coord_save;//位置坐标矢量－把坐标变换前的信息记录下来便于恢复
```

linkshape 类的核心算法主要是在构造函数中，根据输入的信息要完成形状的数据结构表示，包括利用矩阵特征值算法求解邻接矩阵特征值，并用冒泡算法排序以便比较。重点函数包括: 向左旋转 90 度，向右旋转 90 度，上下翻转，左右翻转四个函数用于形状变换后的坐标计算。

而其他方法主要是用来获取或设置属性。

2.7.2 代码实现

linkshape 类的声明和实现，在核心算法设计和测试的时候就放在了一起，所以只有一个文件，如例17所示:

示例 17 linkshape 类定义

代码

```
1  #ifndef SHAPELINK_H
2  #define SHAPELINK_H
3
4  #include <shapeball.h>
5  #include <stdio.h>
6  #include <vector>
7  #include <math.h>
8  #include <time.h>
9  #include <iostream>
10 #include <cstdlib>
11 #include <qDebug>
12
13
14 class linkshape //链接形状是由几个球连接起来构成的形状
```

```
{
private:
        int sn_shape; //<当前链接形状序号,从1开始计数，这是真实的数字
        int n_balls_inshape; //<形状的链接的球数

    std::vector<int> vec_balls_number;//球的序号
        int rows,cols; //<链接形状当前的总行数和总列数
    int rowofzero,colofzero; //因为要做规范化话，则把规范化后(0,0)的对应行号记录下来
        std::vector<coord> vec_coord_shape;//位置坐标矢量与形状矩阵相关
        std::vector<int> vec_matrix_adjacent;//邻接矩阵矢量
        std::vector<float> vec_eigen_matrix;//邻接矩阵特征值列表

    std::vector<coord> vec_coord_origin;//位置坐标矢量－在网格中的初始坐标记录下来
    std::vector<coord> vec_coord_outer;//位置坐标矢量－在网格外的初始坐标记录下来
    std::vector<coord> vec_coord_save;//位置坐标矢量－把坐标变换前的信息记录下来便于恢复

public:

        //默认的构造函数，如果要形成一个数组，是必须的。
        linkshape()
    {}

    //重载的构造函数，//sn当前形状序号，n_balls 为当前形状中的球数，coordmatrix[] 坐标矩阵
        linkshape(int sn, int n_balls ,coord coordmatrix[])//输入当前形状序号,链接的球数,位置坐标矢量
        {
                printf("\n");
                printf("construct a new shape \n");
                sn_shape=sn;
                n_balls_inshape=n_balls;
                int i=0;
                int j=0;

                //对位置坐标矢量做从0开始的规范化
                int rowmin=100;
                int rowmax=-100;
                int colmin=100;
                int colmax=-100;
                for(i=0;i<n_balls;i++){
                        if(coordmatrix[i].row < rowmin) rowmin=coordmatrix[i].row;
                        if(coordmatrix[i].row > rowmax) rowmax=coordmatrix[i].row;
                        if(coordmatrix[i].col < colmin) colmin=coordmatrix[i].col;
                        if(coordmatrix[i].col > colmax) colmax=coordmatrix[i].col;
                }
                const int jm=colmax-colmin;//使用了const后才使得matrix_shape定义正确
                const int im=rowmax-rowmin;//表示从0开始的最后一行行序号，总的行数等于im+1
                rows=im;//为方便起见总行数也从0开始计数
                cols=jm;
        rowofzero=rowmin;
        colofzero=colmin;
                for(i=0;i<n_balls;i++){
                        coordmatrix[i].row=coordmatrix[i].row-rowmin;
                        coordmatrix[i].col=coordmatrix[i].col-colmin;
                }
                printf("postition of balls: \n");
                for(i=0;i<n_balls;i++){
```

```
70                  vec_coord_shape.push_back(coordmatrix[i]);
71                  printf("row=%5d%5c col=%5d\n",coordmatrix[i].row,' ',coordmatrix[i].col);//row表示行号,col表示列号
72              }
73              printf("\n");
74
75
76              //由位置坐标矢量构建邻接矩阵
77              int matrix_adjacent[n_balls][n_balls];
78              for(i=0;i<n_balls;i++){
79                  for(j=0;j<n_balls;j++){
80                      if(i==j){
81                          matrix_adjacent[i][j]=0;
82                      }else{//当两球的某一个坐标相同另一个坐标相差1时表示两个球相邻
83                          if((coordmatrix[i].row==coordmatrix[j].row && abs(coordmatrix[i].col-coordmatrix[j].col)
                                ==1) || (coordmatrix[i].col==coordmatrix[j].col && abs(coordmatrix[i].row-
                                coordmatrix[j].row)==1) ) {
84                              matrix_adjacent[i][j]=1;
85                          }else{
86                              matrix_adjacent[i][j]=0;
87                          }
88                      }
89                  }
90              }
91
92              printf("adjacent matrix of shape: \n");
93              for(i=0;i<n_balls;i++){
94                  for(j=0;j<n_balls;j++){
95                      printf("%5d%5c",matrix_adjacent[i][j],' ');
96                  }
97                  printf("\n");
98              }
99
100             //由邻接矩阵得到其特征值列表
101             double eigen_matrix[n_balls];
102             double eps=0.000001;
103             double mtemp[n_balls][n_balls];
104             double vtemp[n_balls][n_balls];
105             for(i=0;i<n_balls;i++){
106             for(j=0;j<n_balls;j++){
107                 mtemp[i][j]=matrix_adjacent[i][j];
108                 }}
109             double *vpt=&vtemp[0][0];
110             double *mpt=&mtemp[0][0];
111
112             int flag=cjcbi(mpt,n_balls,vpt,eps,100);//如果用多维数组则必须要用指向数组的指针来传递，指针即地址
113             if(flag>0){
114                 printf("eigen elements=");
115             for (i=0; i<n_balls; i++) {
116                 eigen_matrix[i]=mtemp[i][i];
117                 printf("%13.7e ",mtemp[i][i]);
118             }
119             }
120
121             printf("\nin order=");
122             maopao(eigen_matrix, n_balls);
```

```
123             for (i=0; i<n_balls; i++) {
124                     printf("%13.7e ",eigen_matrix[i]);
125                     vec_eigen_matrix.push_back(eigen_matrix[i]);
126             }
127             printf("\n");
128
129     }
130
131
132 //重载的构造函数，//sn当前形状序号，n_balls 为当前形状中的球数，coordmatrix[] 坐标矩阵，ballnumbers球序号
133     linkshape(int sn, int n_balls ,coord coordmatrix[],int ballnumbers[])//输入当前形状序号,链接的球数,位置坐标矢量,球的序
            号
134     {
135             //printf("\n");
136             //printf("construct a new shape \n");
137             sn_shape=sn;
138             n_balls_inshape=n_balls;
139
140             int i=0;
141             int j=0;
142
143             for(i=0;i<n_balls;i++){
144                     vec_balls_number.push_back(ballnumbers[i]);
145         vec_coord_origin.push_back(coordmatrix[i]);
146             }
147
148             //对位置坐标矢量做从0开始的规范化
149             int rowmin=100;
150             int rowmax=-100;
151             int colmin=100;
152             int colmax=-100;
153             for(i=0;i<n_balls;i++){
154                     if(coordmatrix[i].row < rowmin) rowmin=coordmatrix[i].row;
155                     if(coordmatrix[i].row > rowmax) rowmax=coordmatrix[i].row;
156                     if(coordmatrix[i].col < colmin) colmin=coordmatrix[i].col;
157                     if(coordmatrix[i].col > colmax) colmax=coordmatrix[i].col;
158             }
159             const int jm=colmax-colmin;//使用了const后才使得matrix_shape定义正确
160             const int im=rowmax-rowmin;//表示从0开始的最后一行行序号，总的行数等于im+1
161             rows=im;//为方便起见总行数也从0开始计数
162             cols=jm;
163     rowofzero=rowmin;
164     colofzero=colmin;
165             for(i=0;i<n_balls;i++){
166                     coordmatrix[i].row=coordmatrix[i].row-rowmin;
167                     coordmatrix[i].col=coordmatrix[i].col-colmin;
168             }
169             //printf("postition of balls: \n");
170             for(i=0;i<n_balls;i++){
171                     vec_coord_shape.push_back(coordmatrix[i]);
172                     //printf("row=%5d%5c col=%5d\n",coordmatrix[i].row,' ',coordmatrix[i].col);//row表示行号,col表示列号
173             }
174             //printf("\n");
175
176
```

```
177             //由位置坐标矢量构建邻接矩阵
178             int matrix_adjacent[n_balls][n_balls];
179             for(i=0;i<n_balls;i++){
180                     for(j=0;j<n_balls;j++){
181                             if(i==j){
182                                     matrix_adjacent[i][j]=0;
183                             }else{//当两球的某一个坐标相同另一个坐标相差1时表示两个球相邻
184                                     if((coordmatrix[i].row==coordmatrix[j].row && abs(coordmatrix[i].col-coordmatrix[j].col)
                                                ==1) || (coordmatrix[i].col==coordmatrix[j].col && abs(coordmatrix[i].row-
                                                coordmatrix[j].row)==1) ) {
185                                       matrix_adjacent[i][j]=1;
186                                     }else{
187                                       matrix_adjacent[i][j]=0;
188                                     }
189                             }
190                     }
191             }
192
193             // printf("adjacent matrix of shape: \n");
194             // for(i=0;i<n_balls;i++){
195                     // for(j=0;j<n_balls;j++){
196                             // printf("%5d%5c",matrix_adjacent[i][j],' ');
197                     // }
198                     // printf("\n");
199             // }
200
201             //由邻接矩阵得到其特征值列表
202             double eigen_matrix[n_balls];
203             double eps=0.000001;
204             double mtemp[n_balls][n_balls];
205             double vtemp[n_balls][n_balls];
206             for(i=0;i<n_balls;i++){
207             for(j=0;j<n_balls;j++){
208                     mtemp[i][j]=matrix_adjacent[i][j];
209                     }}
210             double *vpt=&vtemp[0][0];
211             double *mpt=&mtemp[0][0];
212
213             int flag=cjcbi(mpt,n_balls,vpt,eps,100);//如果用多维数组则必须要用指向数组的指针来传递，指针即地址
214             if(flag>0){
215                     //printf("eigen elements=");
216                     for (i=0; i<n_balls; i++) {
217                             eigen_matrix[i]=mtemp[i][i];
218                             //printf("%13.7e ",mtemp[i][i]);
219                     }
220             }
221
222             //printf("\neigen in order=");
223             maopao(eigen_matrix, n_balls);
224             for (i=0; i<n_balls; i++) {
225                     //printf("%13.7e ",eigen_matrix[i]);
226                     vec_eigen_matrix.push_back(eigen_matrix[i]);
227             }
228             //printf("\n");
229
```

```
230         }
231
232     void saveoldcoord()//记录放置在网格外时的坐标
233     {
234         vec_coord_save=vec_coord_shape;
235     }
236
237     void resetoldcoord()//记录放置在网格外时的坐标
238     {
239         vec_coord_shape=vec_coord_save;
240     }
241
242     void recordoutercoord()//记录放置在网格外时的坐标
243     {
244         vec_coord_outer=vec_coord_shape;
245     }
246
247     void resetoutercoord()//恢复放置在网格外时的坐标
248     {
249         vec_coord_shape=vec_coord_outer;
250     }
251
252     std::vector<coord> getvec_coord()
253     {
254         return vec_coord_shape;
255     }
256
257         int getballnumber(const int i)//返回形状中球的序号
258         {
259                 return vec_balls_number[i];
260         }
261
262         int getshapesn()
263         {
264                 return sn_shape;
265         }
266
267     int getshapenballs()
268     {
269         return n_balls_inshape;
270     }
271
272         int getrows()
273         {
274                 return rows;
275         }
276
277     int getcols()
278     {
279         return cols;
280     }
281
282     int getrowofzero()
283         {
284         return rowofzero;
```

```
285         }
286
287     int getcolofzero()
288     {
289         return colofzero;
290     }
291
292     int getballsn(const int i)
293     {
294         return vec_balls_number[i];
295     }
296
297         float getshapeeigen(const int i)
298         {
299                 return vec_eigen_matrix[i];
300         }
301
302     int getptrow(const int i)//给出球在形状中的行坐标,i从0开始计数
303         {
304                 return vec_coord_shape[i].row;
305         }
306
307     int getptcol(const int i)//给出球在形状中的列坐标
308     {
309         return vec_coord_shape[i].col;
310     }
311
312     int getptroworig(const int i)//给出球在形状中的行坐标,i从0开始计数
313     {
314         return vec_coord_origin[i].row;
315     }
316
317     int getptcolorig(const int i)//给出球在形状中的列坐标
318     {
319         return vec_coord_origin[i].col;
320     }
321
322     int getptcolouter(const int i)//给出球在形状中的列坐标
323         {
324         return vec_coord_outer[i].col;
325         }
326
327     int getptrowouter(const int i)//给出球在形状中的行坐标
328     {
329         return vec_coord_outer[i].row;
330     }
331
332     coord getptcoord(const int i)//给出球在形状中的坐标
333     {
334         return vec_coord_shape[i];
335     }
336
337     coord getptcoordouter(const int i)//给出球在形状中的坐标
338     {
339         return vec_coord_outer[i];
```

```
340 }
341
342     void pos_show()//输出形状矩阵
343     {
344         //各个点的坐标
345         int i,j,k;
346         printf("\nshape balls' coords:{");
347         for(i=0;i<n_balls_inshape-1;i++){//vector的下标与数组的下标是一样的也是从0开始。
348             printf("(%2d,%2d),",vec_coord_shape[i].row,vec_coord_shape[i].col);
349         }
350         printf("(%2d,%2d)",vec_coord_shape[n_balls_inshape-1].row,vec_coord_shape[n_balls_inshape-1].col);
351         printf("}\n");
352
353
354         //形状矩阵
355         int matrix_shape[rows+1][cols+1]={0};
356         std::cout<<"max rows from row=0 is "<<rows<<", max cols from col=0 is "<<cols<<std::endl;
357         printf("\n");
358         for(i=0;i<rows+1;i++)//初始化一下
359             for(j=0;j<cols+1;j++) matrix_shape[i][j]=0;
360         for(k=0;k<n_balls_inshape;k++){
361             i=vec_coord_shape[k].row;
362             j=vec_coord_shape[k].col;
363             matrix_shape[i][j]=k+1;//形状矩阵位置上用球的序号表示
364         }
365         for(i=0;i<=rows;i++){
366             for(j=0;j<=cols;j++){
367                 printf("%5d%5c",matrix_shape[i][j],' ');
368             }
369             printf("\n");
370         }
371         printf("\n");
372     }
373
374     void rotate_right()//向右旋转90度以后的坐标位置矢量
375     {
376         for(int i=0;i<n_balls_inshape;i++)
377         {
378             int row=vec_coord_shape[i].row;
379             int col=vec_coord_shape[i].col;
380             vec_coord_shape[i].row=col;
381             vec_coord_shape[i].col=rows-row;//+1,起点从0开始不需要加1
382         }
383         int temp=rows;
384         rows=cols;
385         cols=temp;
386     }
387
388     void rotate_left()//向左旋转90度以后的坐标位置矢量
389     {
390         for(int i=0;i<n_balls_inshape;i++)
391         {
392             int row=vec_coord_shape[i].row;
393             int col=vec_coord_shape[i].col;
394             vec_coord_shape[i].row=cols-col;
```

```
395                     vec_coord_shape[i].col=row;
396             }
397             int temp=rows;
398             rows=cols;
399             cols=temp;
400     }
401
402     void mirror_lr()//左右翻转以后的坐标位置矢量
403     {
404             for(int i=0;i<n_balls_inshape;i++)
405             {
406                     int row=vec_coord_shape[i].row;
407                     int col=vec_coord_shape[i].col;
408                     vec_coord_shape[i].row=row;
409                     vec_coord_shape[i].col=cols-col;
410             }
411     }
412
413     void mirror_ab()//上下翻转以后的坐标位置矢量
414     {
415             for(int i=0;i<n_balls_inshape;i++)
416             {
417                     int row=vec_coord_shape[i].row;
418                     int col=vec_coord_shape[i].col;
419                     vec_coord_shape[i].row=rows-row;
420                     vec_coord_shape[i].col=col;
421             }
422     }
423
424     /*实对称矩阵的特征值和特征向量求解，参考徐士良-Fortran常用算法程序集-第二版.pdf
425     输入数组a
426     数组的秩n
427     数组的特征向量v
428     判断残差eps
429     迭代次数jt
430     输出:特征值在a的对角线上，特征向量在v中
431     */
432 int cjcbi(double *a,int n,double* v,double eps,int jt)
433   { int i,j,p,q,u,w,t,s,l;
434     p=0;
435     q=0;
436     u=0;
437     double fm,cn,sn,omega,x,y,d;
438     l=1;
439     for (i=0; i<=n-1; i++)
440       { v[i*n+i]=1.0;
441         for (j=0; j<=n-1; j++)
442           if (i!=j) v[i*n+j]=0.0;
443       }
444     while (1==1)
445       { fm=0.0;
446         for (i=1; i<=n-1; i++)
447         for (j=0; j<=i-1; j++)
448           { d=fabs(a[i*n+j]);
449             if ((i!=j)&&(d>fm))
```

```
            { fm=d; p=i; q=j;}
          }
        if (fm<eps) return(1);
        if (l>jt) return(-1);
        l=l+1;
        u=p*n+q; w=p*n+p; t=q*n+p; s=q*n+q;
        x=-a[u]; y=(a[s]-a[w])/2.0;
        omega=x/sqrt(x*x+y*y);
        if (y<0.0) omega=-omega;
        sn=1.0+sqrt(1.0-omega*omega);
        sn=omega/sqrt(2.0*sn);
        cn=sqrt(1.0-sn*sn);
        fm=a[w];
        a[w]=fm*cn*cn+a[s]*sn*sn+a[u]*omega;
        a[s]=fm*sn*sn+a[s]*cn*cn-a[u]*omega;
        a[u]=0.0; a[t]=0.0;
        for (j=0; j<=n-1; j++)
        if ((j!=p)&&(j!=q))
          { u=p*n+j; w=q*n+j;
            fm=a[u];
            a[u]=fm*cn+a[w]*sn;
            a[w]=-fm*sn+a[w]*cn;
          }
        for (i=0; i<=n-1; i++)
          if ((i!=p)&&(i!=q))
            { u=i*n+p; w=i*n+q;
              fm=a[u];
              a[u]=fm*cn+a[w]*sn;
              a[w]=-fm*sn+a[w]*cn;
            }
        for (i=0; i<=n-1; i++)
          { u=i*n+p; w=i*n+q;
            fm=v[u];
            v[u]=fm*cn+v[w]*sn;
            v[w]=-fm*sn+v[w]*cn;
          }
      }
    return(1);
  }

        /*冒泡算法，对数组进行排序，从大到小排序
        输入:数组stu
        数组长度n
        */
void maopao(double stu[], int n)
{
        int i,j,k,s;
        double temp1,temp2;
        for(j=1;j<=n-1;j++){//遍历次数是n-1次
                if(fmod(j,2) != 0) {//j为奇数次，从前向后遍历
                        k=(j+1)/2;
                        for(i=k+1;i<=n+1-k;i++)//通过遍历终止值，减少大小比较次数,因为一次遍历能得出一个方向的极值
                        {
                                s=i-1;
                                if( stu[s] > stu[s-1]){//从大到小排序，若要从小到大则应改为<号
```

```
                                temp1=stu[s];
                                stu[s]=stu[s−1];
                                stu[s−1]=temp1;
                                }
                        }}
                else
                        {//j为偶数次，从后向前遍历
                        k=j/2;
                        for(i=n−k;i>=k+1;i−−)//注意这里i>=k+1，因为i是递减的
                        {
                                s=i−1;
                                if(stu[s] > stu[s−1]){//从大到小排序，若要从小到大则应改为<号
                                temp2=stu[s];
                                stu[s]=stu[s−1];
                                stu[s−1]=temp2;
                                }
                        }}
                //printf("\n");
                //for(i=0;i<n;i++) printf("%13.7e ",stu[i]);
        }
}
};



#endif // SHAPELINK_H
```

2.8 sceneinfotrans 类

sceneinfotrans 类是继承 QGraphicsScene 的场景类，这个类本来并不需要，是在信号传递过程设计中引入的，但其实没有必要。因为开始在游戏网格大小变化时，对场景中按钮和 ballgrid 对象的信号传递考虑不周全希望引入一个中介，即场景来中转。但在 ballgrid 类设计了一个简单函数用于传递场景后就不再需要。

这里保留是考虑未来可能对于场景需要增加一些新的功能，所以保留。其中构造函数，信号和槽目前都没有任何处理。

其声明如例18所示:

示例 18 sceneinfotrans 类声明

代码

```
#ifndef SCENEINFOTRANS_H
#define SCENEINFOTRANS_H

#include<QGraphicsScene>

class sceneinfotrans : public QGraphicsScene
{
public:
    sceneinfotrans();

public slots:
    void scenetobeset(bool a);

signals:
    void toshowgrid(sceneinfotrans &b);
```

```

};

#endif // SCENEINFOTRANS_H
```

2.9 主程序

主程序 main.CPP 主要完成 ballgrid 对象，场景对象，视图对象的构建，完成在场景中添加一些按钮，标签，文本框，并完成一些信号和槽的链接。

其实现如例**??**所示:

示例 19 主程序 main

代码

```
//#include "mainwindow.h"
#include "ballitem.h"
#include "ballshape.h"
#include "ballgrid.h"
#include "sceneinfotrans.h"
#include <QApplication>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QGraphicsWidget>
#include <QGraphicsProxyWidget>
#include <QPushButton>
#include <QLabel>
#include <QLineEdit>
#include <QGraphicsGridLayout>
#include <QGraphicsLinearLayout>
#include <qDebug>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    //MainWindow w;
    //w.show();

    sceneinfotrans scene;

    QPushButton *buttonstart=new QPushButton(QObject::tr("开始"));
    QPushButton *buttonexample=new QPushButton(QObject::tr("示例"));
    buttonstart->setMaximumHeight(20);
    buttonexample->setMaximumHeight(20);
    //buttonstart->setFixedSize(50, 30);

    QPushButton *buttonset=new QPushButton(QObject::tr("设置"));
    QPushButton *buttonpart=new QPushButton(QObject::tr("重新划分"));
    QLabel * labelgridnum=new QLabel(QObject::tr("网格大小"));
    QLabel * labellinknum=new QLabel(QObject::tr("链接球数"));
    QLineEdit * textgridnum=new QLineEdit("10");
    textgridnum->setMaximumWidth(50);//设置最大尺度长度为50像素
    QLineEdit * textlinknum=new QLineEdit("5");
    textlinknum->setMaximumWidth(50);
    //textlinknum->setMaximumHeight(30);

    QLabel * label_nshapes=new QLabel(QObject::tr("形状数量"));
    QLabel * labeldiameter=new QLabel(QObject::tr("球的尺寸"));
```

```
44  QLineEdit * text_nshapes=new QLineEdit("0");
45  text_nshapes->setMaximumWidth(50);//设置最大尺度长度为50像素
46  QLineEdit * textdiameter=new QLineEdit("40");
47  textdiameter->setMaximumWidth(50);
48
49  QPushButton *turnright=new QPushButton(QObject::tr("向右旋转"));
50  QPushButton *turnleft=new QPushButton(QObject::tr("向左旋转"));
51  QPushButton *mirrorab=new QPushButton(QObject::tr("上下翻转"));
52  QPushButton *resetpos=new QPushButton(QObject::tr("移出网格"));
53
54  QGraphicsWidget *button1=scene.addWidget(buttonstart);//用代理的方式将部件加入场景中
55  QGraphicsWidget *button2=scene.addWidget(buttonexample);
56  QGraphicsWidget *button3=scene.addWidget(buttonset);
57  QGraphicsWidget *button4=scene.addWidget(buttonpart);
58
59  QGraphicsWidget *button5=scene.addWidget(turnright);//用代理的方式将部件加入场景中
60  QGraphicsWidget *button6=scene.addWidget(turnleft);
61  QGraphicsWidget *button7=scene.addWidget(mirrorab);
62  QGraphicsWidget *button8=scene.addWidget(resetpos);
63
64  QGraphicsWidget *label1=scene.addWidget(labelgridnum);
65  QGraphicsWidget *label2=scene.addWidget(labellinknum);
66  QGraphicsWidget *text1=scene.addWidget(textgridnum);
67  QGraphicsWidget *text2=scene.addWidget(textlinknum);
68
69  QGraphicsWidget *label3=scene.addWidget(label_nshapes);
70  QGraphicsWidget *label4=scene.addWidget(labeldiameter);
71  QGraphicsWidget *text3=scene.addWidget(text_nshapes);
72  QGraphicsWidget *text4=scene.addWidget(textdiameter);
73
74  QGraphicsGridLayout *layout = new QGraphicsGridLayout;
75  layout->addItem(button1, 0, 0);
76  layout->addItem(button2, 0, 1);
77  QGraphicsLinearLayout *left=new QGraphicsLinearLayout;
78  left->addItem(label1);
79  left->addItem(text1);
80
81  QGraphicsLinearLayout *right=new QGraphicsLinearLayout;
82  right->addItem(label2);
83  right->addItem(text2);
84
85  QGraphicsLinearLayout *left1=new QGraphicsLinearLayout;
86  left1->addItem(label3);
87  left1->addItem(text3);
88
89  QGraphicsLinearLayout *right1=new QGraphicsLinearLayout;
90  right1->addItem(label4);
91  right1->addItem(text4);
92
93  layout->addItem(left, 1, 0);
94  layout->addItem(right, 1, 1);
95  layout->addItem(button3, 2, 1);
96  layout->addItem(button4, 2, 0);
97  layout->addItem(left1, 3, 0);
98  layout->addItem(right1, 3, 1);
```

```
    layout->addItem(button5, 4, 1);
    layout->addItem(button6, 5, 1);
    layout->addItem(button7, 6, 1);
    layout->addItem(button8, 7, 1);
    //layout->setMaximumWidth(200);//没有效果，估计策略是按钮自身是第一优先级
    //layout->setMaximumHeight(200);
    //layout->setColumnMaximumWidth(1,50);
    //layout->setRowMaximumHeight(1,30);

    QGraphicsWidget *form = new QGraphicsWidget;
    form->setLayout(layout);
    scene.addItem(form);
    form->setPos(textdiameter->text().toInt()*4,textdiameter->text().toInt()/2);

    ballgrid gridballs;

    QObject::connect(&gridballs,&ballgrid::shapecompleted,text_nshapes,&QLineEdit::setText);
    QObject::connect(buttonstart,&QPushButton::clicked,&gridballs,&ballgrid::gamestart);
    QObject::connect(buttonexample,&QPushButton::clicked,&gridballs,&ballgrid::gameexample);
    QObject::connect(textgridnum,&QLineEdit::textChanged,&gridballs,&ballgrid::dealgridnumtext);
    QObject::connect(textlinknum,&QLineEdit::textChanged,&gridballs,&ballgrid::deallinknumtext);
    QObject::connect(textdiameter,&QLineEdit::textChanged,&gridballs,&ballgrid::setballdiameter);
    QObject::connect(buttonset,&QPushButton::clicked,&gridballs,&ballgrid::gameset);
    QObject::connect(buttonpart,&QPushButton::clicked,&gridballs,&ballgrid::gamerepart);

    QObject::connect(turnright,&QPushButton::clicked,&gridballs,&ballgrid::shapeturnright);
    QObject::connect(turnleft,&QPushButton::clicked,&gridballs,&ballgrid::shapeturnleft);
    QObject::connect(mirrorab,&QPushButton::clicked,&gridballs,&ballgrid::shapemirrorab);
    QObject::connect(resetpos,&QPushButton::clicked,&gridballs,&ballgrid::shaperesetpos);

    if(gridballs.getflag()==0 && gridballs.getgridnum()==10)
    {
        gridballs.setgrid(gridballs.getgridnum(),gridballs.getlinknum());
        gridballs.setboxes();
        gridballs.setshapesdefault(gridballs.getntotal());
    }

    gridballs.setworkscene(scene);
    gridballs.addboxestoscene();
    gridballs.addtoscenedefault();

    QGraphicsView view(&scene);
    view.resize(750,650);
    view.show();

    return a.exec();
}
```

3 编译为 window 程序和 andriod 的 apk 软件

换了个电脑后，采用 qt5.8.0 的环境编译。

3.1 编译为 window 程序

编译 win 程序比较简单，选择打开工程后，按 crtl+R 即可完成编译和运行。

3.2　编译为 andriod 的 apk 软件

编译 andriod 程序，首先需要构建 qt 编译 andriod 的工具环境，设置 creater 的设备选项中的 andriod 选项卡中的工具链，如图9所示。其中需要安装的工具包括: jdk-8u144-windows-i586.exe，android-studio-bundle-145.3537739-windows.exe，android-ndk-r13b-windows-x86.zip，apache-ant-1.10.1-bin.zip，qt-opensource-windows-x86-android-5.8.0.exe。下载和安装方法可以百度搜索，有不少朋友进行了介绍。

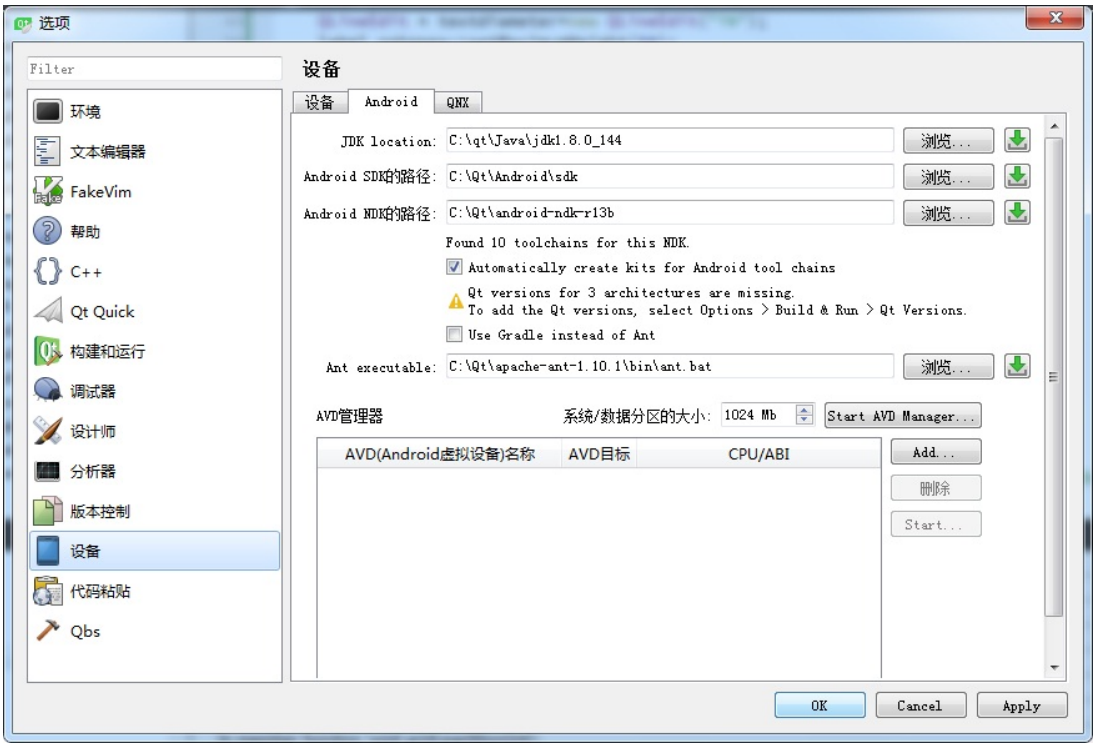

图 9 编译 andriod 程序工具链

然后设置工程选项，编译工具集，如图10所示。

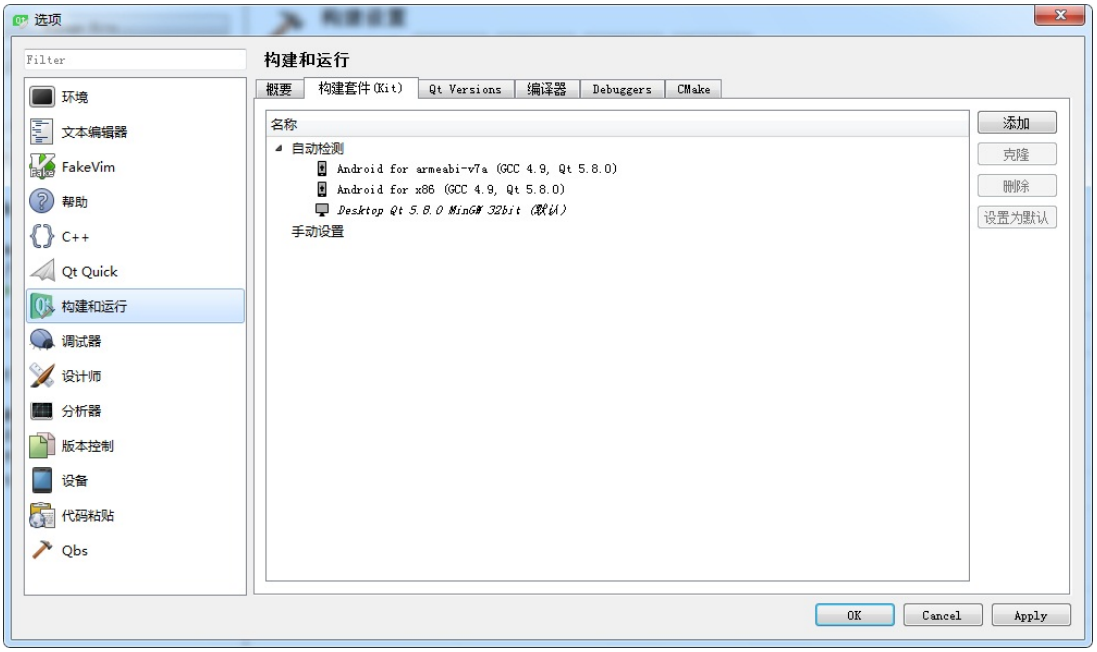

图 10 工程设置选择 android 工具链

最后，按 crtl+B 即可完成编译生成 apk 文件。

将 apk 文件安装到手机上可以进行测试和游戏。

需要注意的是，基于 widget 的 qt 程序编译成 andriod 程序后如按钮等控件的大小会发生变化，这些需要在代码中进行调整。

4 游戏操作

最终构成的游戏界面如图11所示。

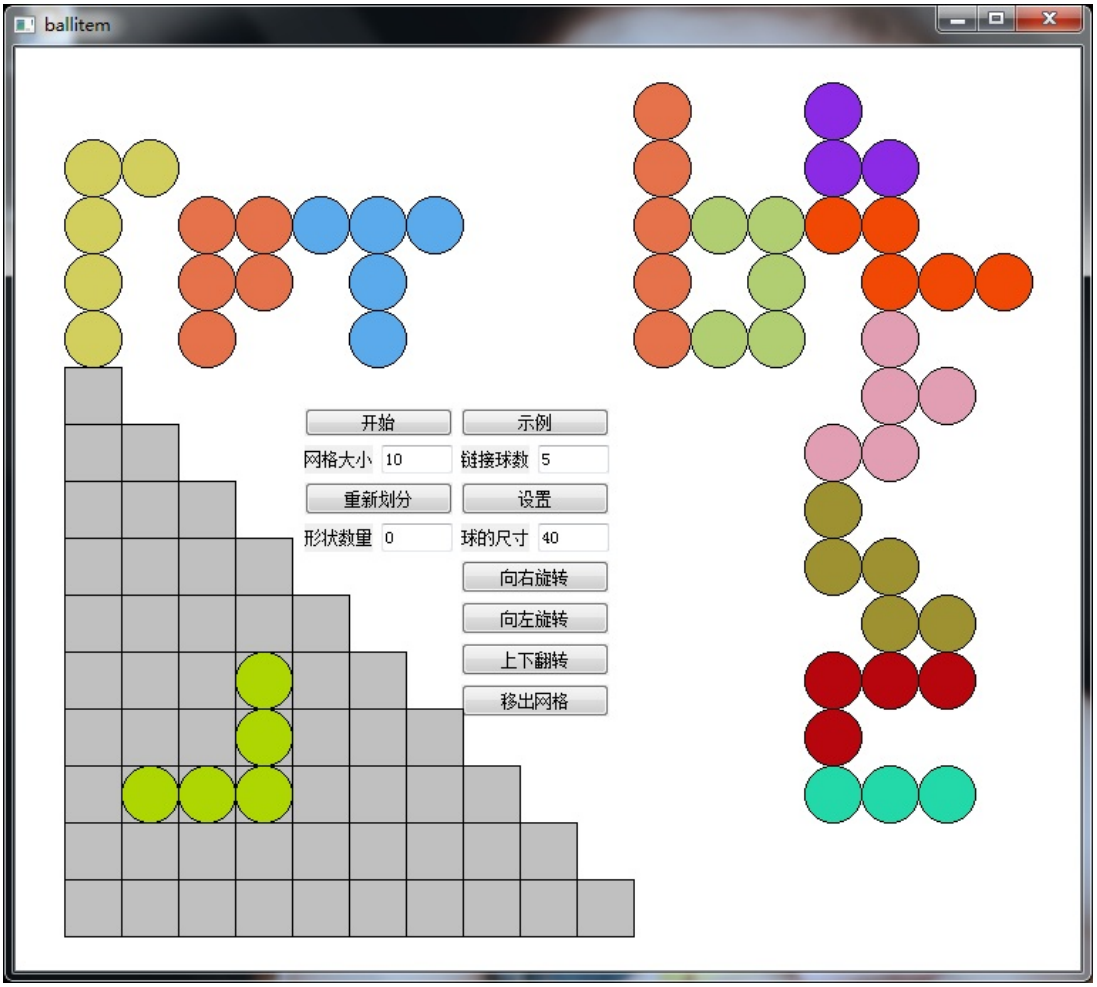

图 11 游戏界面

4.1 游戏全局控制

点击开始按钮开始进行游戏。点击示例按钮可以看到某种拼图答案，注意: 实现拼图的方案有很多种，示例只是其中一种。如果完成当前拼图，想换一套形状的方案，可以点击重新划分按钮，游戏将重新划分生成新的形状。当觉得游戏过于简单或困难，可以设置网格大小和链接球数的大小，点击设置或重新划分按钮即可根据新的网格大小和链接球数生成一套新的形状以便进行游戏。其中生成形状数量将在形状数量文本框显示。如果觉得游戏中球的大小不合适可以重新设置球的尺寸，即在球的尺寸文本框设置。

4.2 形状选择和落位

采用点击方式选择形状并落位。当点击到当前形状的某个球时，该形状及该球被选择。然后点击空位矩阵中的某个空位时，当前空位被选择。这时进行落位判断，判断的起点是选中的球落在选中的空位上，若当前形状的所有球都可以放入一个空位则形状放下，落位成功，否则落位操作无效。

4.3 形状的旋转翻转复位

点击选中某一形状后，这时该形状被点击的球设置为球的旋转和翻转中心，旋转和翻转将围绕该球进行。按键盘向右箭头键或界面上的向右旋转按钮，形状向右旋转 90 度。按键盘向左箭头键或界面上的向左旋转按钮，形状向右旋转 90 度。按键盘向上箭头键或界面上的上下翻转按钮，形状上下翻转。按界面上的移出网格按钮或者用右键点选该形状，形状将复位到原来在网格外的位置。

5 结论

设计并实现了一个拼图游戏。利用 QT 的跨平台特性，分别编译成 windows 下的程序和 android 下的程序，真正实现了跨平台应用。

当然其中还存在不少问题需要改进，特别是在 andriod 下的体验。

整个代码和文档都放在 github 上，欢迎朋友们提供宝贵意见，更欢迎做出改进，欢迎 PR!

目 录

插 图

示 例