

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 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满所有空位则游戏完成。游戏完成的基本形式应如图1所示。

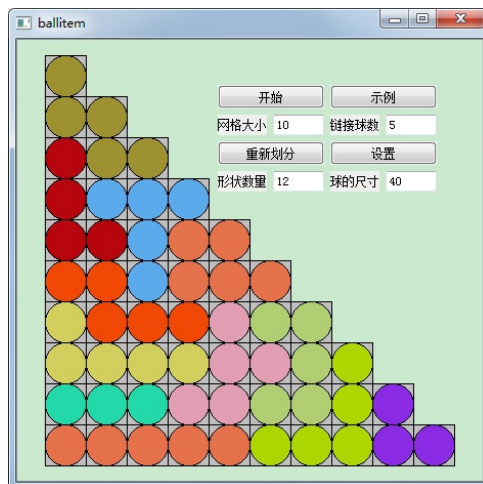


图 1 游戏基本构想

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),最后一个是由球构成的网格(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计算法等,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
1  cls
2
3  del /q a.exe
4
5  gcp a.cpp 1
6
7  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
1  cls
2  @echo off
3  title Gcc::g++ WD:%cd% %date% %time%
4
5  rem :如果没有输入则提示错误
6  SET srcfile=%1
7  SET laststr=%2
8  if "%laststr%" == "" (
9      echo 错误:没有输入参数, 请重新输入!!!
10     goto endcmd
11 ) else (
12     goto dealfile
13 )
```

```

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：

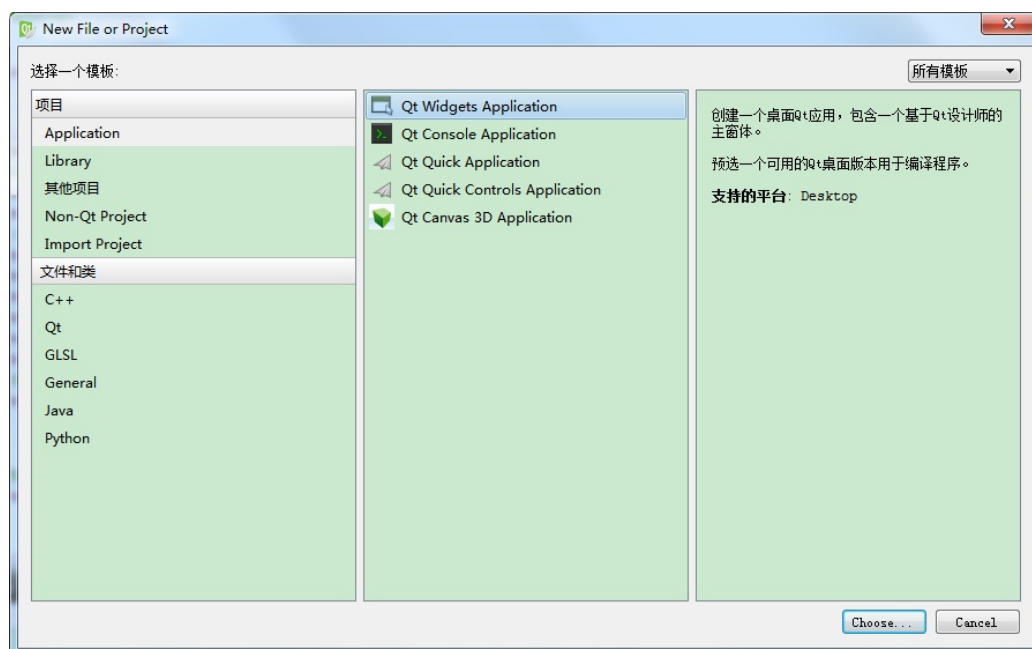


图 2 选择 qt widget application



图 3 命名工程选择位置

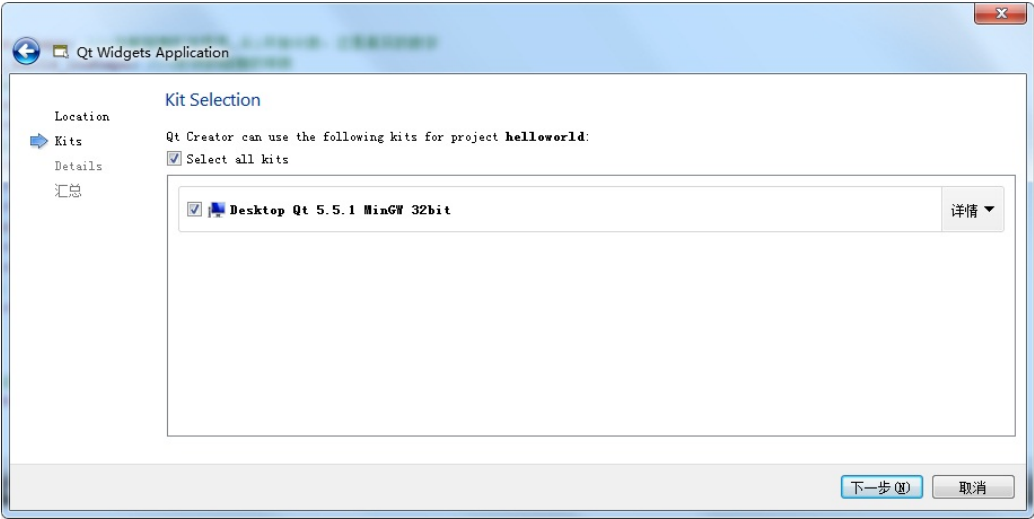


图 4 选择编译工具

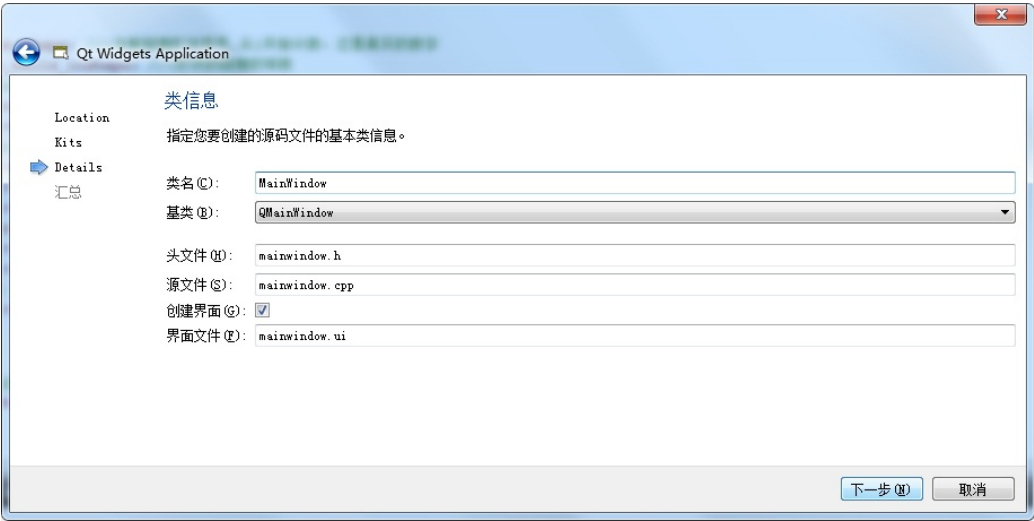


图 5 选择主界面基类

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。

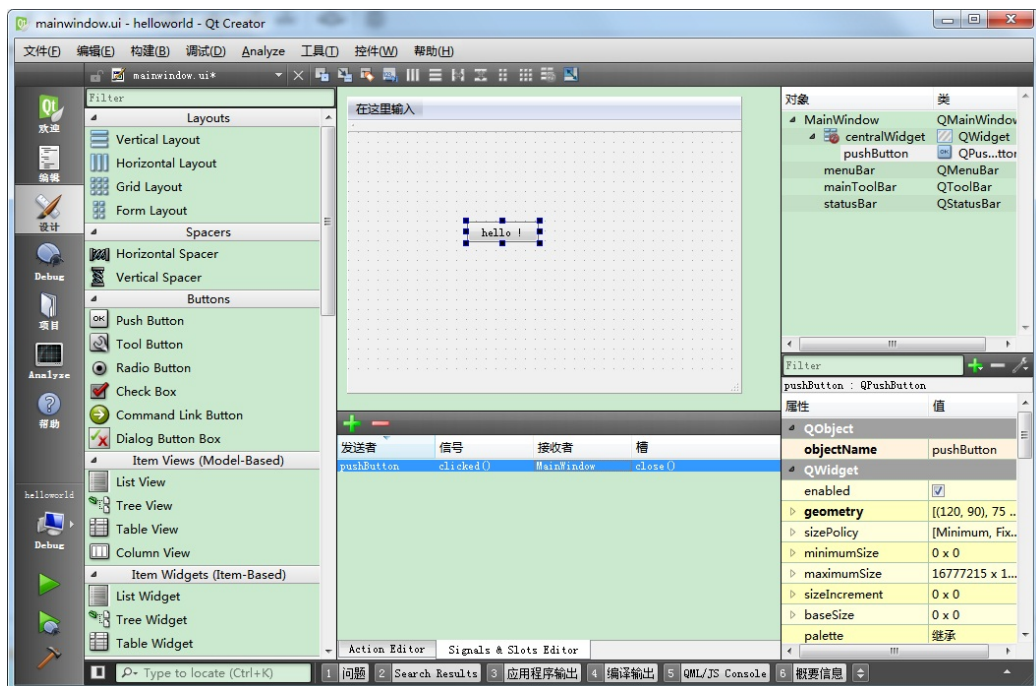


图 6 编辑 ui 界面添加信号槽

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

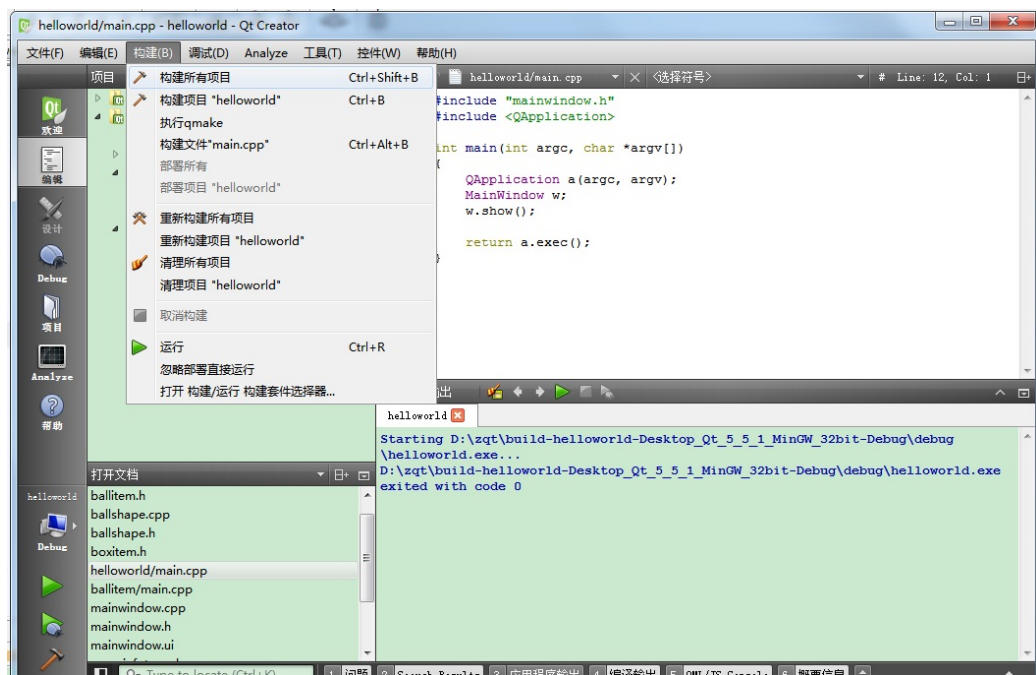


图 7 编译构建

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

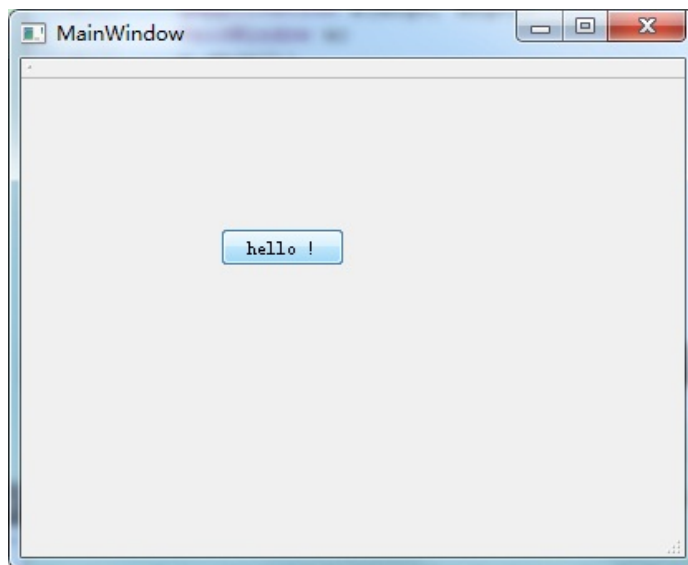


图 8 运行结果

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

```

1  #ifndef BOXITEM_H
2  #define BOXITEM_H
3
4  #include <QObject>
5  #include <QGraphicsRectItem>
6
7  class boxitem : public QObject, public QGraphicsRectItem
8  {
9
10     Q_OBJECT
11
12 private:
13     int boxid; // 盒子(方框, 空位)序号, 从1开始计数
14
15 public:
16     boxitem();
17     void setboxid(int id); // 用于设置盒子(框)序号
18
19 protected:
20     void mousePressEvent(QGraphicsSceneMouseEvent * event);
21     void mouseReleaseEvent(QGraphicsSceneMouseEvent * event);
22
23 signals:
24     void tojudgesitdown(int boxid); // 要进行落位判断的信号, 参数是box的id号
25 };
26
27 #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

```

1  #include "boxitem.h"
2  #include <QCursor>
3  #include <QKeyEvent>
4  #include <qDebug>
5
6  boxitem::boxitem()
7  {
8      //setFlag(QGraphicsItem::ItemIsMovable);
9      setFlag(QGraphicsItem::ItemIsSelectable);
10     setFlag(QGraphicsItem::ItemIsFocusable);
11 }
12
13 void boxitem::setboxid(int id) // 用于设置盒子(框)序号

```

```

14 {
15     boxid=id;
16 }
17
18 void boxitem::mousePressEvent(QGraphicsSceneMouseEvent * event)//鼠标点击事件处理
19 {
20     Q_UNUSED(event);
21     setFocus();
22     setCursor(Qt::ClosedHandCursor);
23     emit tojudgesitdown(boxid);//发送信号，用于落位判断
24     //QDebug()<<"focus:"<<hasFocus();
25     //QDebug()<<"focus:"<<focusItem();
26     //QDebug()<<"emit:"<<"box:tojudgesitdown";
27 }
28
29 void boxitem::mouseReleaseEvent(QGraphicsSceneMouseEvent * event)
30 {
31     Q_UNUSED(event);
32     setCursor(Qt::OpenHandCursor);
33 }

```

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

```

1  #ifndef BALLITEM_H
2  #define BALLITEM_H
3
4  #include <QObject>
5  #include <QGraphicsEllipseItem>
6  // #include <QPainter> //该类也需要加入，否则出现不完整qpainter的错误提示
7
8  class ballitem : public QObject, public QGraphicsEllipseItem
9  {
10
11      Q_OBJECT

```

```

12
13 public:
14     ballitem(QObject *parent =0);
15     ~ballitem();
16     //void setColor(const QColor &color);
17     //QRectF boundingRect() const;
18     //void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget);
19     void setshapeid(int id); //设置当前球所属形状 id 号
20     void setballid(int id); //设置当前球在所属形状中的序号
21
22 protected:
23     void mousePressEvent(QGraphicsSceneMouseEvent * event);
24     void mouseReleaseEvent(QGraphicsSceneMouseEvent * event);
25     //void mouseMoveEvent(QGraphicsSceneMouseEvent * event);
26     void keyPressEvent(QKeyEvent * event);
27
28 signals:
29     void rotateormirror(int transid); //形状旋转或翻转信号, transid 表示项左, 右旋转, 上下, 左右翻转
30     void ballpressed(int ballid, int shapeid); //当前球和形状被选中信号
31     void ballpressedright(int ballid, int shapeid); //当前球所在形状移出网格的信号
32
33
34 private:
35     int ballid; //标记当前在形状中的序号, 是第几个球, 从 1 开始计数
36     int shapeid; //标记当前球属于哪个形状, 从 1 开始计数
37     // QColor brushcolor;
38
39 };
40
41 #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

```

1  #include "ballitem.h"
2  #include <QCursor>
3  #include <QKeyEvent>
4  #include <QGraphicsSceneMouseEvent>
5  #include <qDebug>
6
7  ballitem::ballitem(QObject *parent)
8  {
9      //setFlag(QGraphicsItem::ItemIsMovable);
10     Q_UNUSED(parent);
11     setFlag(QGraphicsItem::ItemIsSelectable);
12     setFlag(QGraphicsItem::ItemIsFocusable);
13 }
14
15 ballitem::~ballitem()
16 {
17 }

```

```
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:
```

```

73         emit rotateormirror(4);
74         break;
75
76     default:
77         break;
78 }
79 }
80
81
82 //QRectF ballitem::boundingRect() const
83 //{
84 // qreal penWidth=1;
85 // return QRectF(0-penWidth/2,0-penWidth/2,50+penWidth,50+penWidth);
86 //}
87
88
89 //void ballitem::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
90 //{
91 // painter->setBrush(brushcolor);
92 // painter->drawEllipse(0,0,50,50);
93 //}
94
95 //void ballitem::setColor(const QColor &color)
96 //{
97 // brushcolor=color;
98 //}

```

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

```

1  #ifndef BALLSHAPE_H
2  #define BALLSHAPE_H
3
4  #include <QObject>
5  #include "shape.h"
6  #include "ballitem.h"
7  #include <QGraphicsScene>
8  #include <QList>
9  #include "sceneinfotrans.h"
10
11
12 class ballshape : public QObject
13 {
14     Q_OBJECT
15
16 public:
17     ballshape(QObject *parent =0);
18     ~ballshape();
19     void setballshape(int id,int n_ballsinshape,int ballpos[],QColor color);//构成当前形状
20     void addtoscene(sceneinfotrans &scene);//将形状加入场景
21     void removeballs(sceneinfotrans &scene);//将形状移出场景
22     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
23     void recordpos();//记录一下游戏开始时球放在网格外的坐标位置
24     void resetpos();//恢复一下游戏开始时球放在网格外的坐标位置
25     QPointF getballpos(int sn);//得到球真实序号为sn的坐标位置,sn从1开始计数
26     void setballdiameter(int diameter);//设置球的直径
27
28 private:
29     int shapeid;//当前形状id
30     int n_balls;//当前形状内的球数
31     QList<ballitem *> balls;//当前形状内球的列表
32     QList<QPointF> ballposinit;//游戏开始前放在框(网格)外的坐标位置
33     int balldiameter;//当前形状内球的直径
34
35 signals:
36     void shapeselected(int ballid,int shapeida);//当前形状被选中, 则传递给ballgird
37     void shapetoreset(int ballid,int shapeida);//当前形状被选中,需要还原为原来在网格外的位置, 则传递给ballgird
38     void shapetotrans(int shapeida,int transid);//当前形状被选中, 需要做旋转, 翻转变换
39
40 public slots://链接的槽与信号参数应是一致的。
41     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
42     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
43     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
44
45 };
46
47 #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

```

1  #include "ballshape.h"
2  #include <qDebug>
3  // #include <cstdlib>
4
5  ballshape::ballshape(QObject *parent)
6  {
7      Q_UNUSED(parent);
8  }
9
10 ballshape::~ballshape()
11 {}
12
13 void ballshape::setballdiameter(int diameter)//设置球的直径
14 {
15     balldiameter=diameter;
16 }
17
18 void ballshape::setballshape(int id,int n_ballsinshape,int ballpos[],QColor color)//注意输入的是实际的位置信息
19 {
20     shapeid=id;
21     n_balls=n_ballsinshape;
22
23     //QColor colora(qrand() % 256, qrand() % 256, qrand() % 256);
24
25     for(int i=0;i<n_balls;i++)
26     {
27         ballitem *balla=new ballitem();
28         balla->setshapeid(id);
29         balla->setballid(i+1);
30         balla->setRect(0, 0, balldiameter, balldiameter);
31         balla->setBrush(colora);
32         balla->setPos(ballpos[2*i],ballpos[2*i+1]);
33         balls.append(balla);
34         QObject::connect(balla,&ballitem::rotateormirror,this,&ballshape::transexec);
35         QObject::connect(balla,&ballitem::ballpressed,this,&ballshape::ballselected);
36         QObject::connect(balla,&ballitem::ballpressedright,this,&ballshape::ballstoresetpos);
37     }
38 }
39
40 void ballshape::setballpos(int i,int posx,int posy)
41 {
42     balls[i]->setPos(posx,posy);
43 }
44
45 void ballshape::recordpos()//
46 {
47     for(int i=0;i<n_balls;i++)
48     {

```

```

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(sceneinfo*trans &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(sceneinfo*trans &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 发出的信号到 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

```

1  #ifndef BALLGRID_H
2  #define BALLGRID_H
3
4  #include "shape.h"
5  #include "ballshape.h"
6  #include <QList>
7  #include <QGraphicsScene>
8  #include "boxitem.h"
9  #include "sceneinfotrans.h"
10 #include <QObject>
11 #include <QTime>
12 #include <QTimer>
13 #include <QtXml>
14 #include <QMediaPlayer>
15
16 class ballgrid : public QObject, public grid
17 {
18
19     Q_OBJECT
20
21 public:
22     ballgrid();
23

```

```

24     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
25     void setshapes(int arraysize);//利用划分算法生成划分形状
26     void addtoscenedefault();//将各划分形状加入到场景中
27     void setshaperandposdefault();//将形状移动到框外面, 以便开始游戏
28     void removeshapes();//将各划分形状移出场景
29
30     void setboxes();//生成空位(方框)网格
31     void addboxestoscene();//将方框网格加入场景
32     void removeboxes();//将方框网格移出场景
33     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
34
35     int getgridnum();//获取网格大小信息gridnum
36     int getlinknum();//获取划分形状的预设球数信息linknum
37     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
38     void setworkscene(sceneinfo &scene);//保存当前所在的场景信息
39     void isinrank(int gridnumber, int linknumber, float gametime);//判断本次游戏成绩是否进入排行榜, 进入则修改成绩记录xml
40
41 public slots:
42     void judgesitdown(int boxid);//根据接收到盒子信息去判断能否落位
43     void dealshapetosit(int ballid, int shapeida); //记录一下接收到的要落位的形状的信息
44     void shapereset(int ballid, int shapeida); //重新恢复形状在网格外的原位置
45     void shapetrans(int shapeida, int transid);//对当前形状进行旋转等操作
46     void shapeturnright(bool a);//按钮控制向右旋转
47     void shapeturnleft(bool a);//按钮控制向左旋转
48     void shapemirrorab(bool a);//按钮控制上下翻转
49     void shapemirrorlr(bool a);//按钮控制左右翻转
50     void shaperesetpos(bool a);//按钮控制移出网格
51     void gamestart(bool a);//游戏开始, 将所有形状移动到网格外
52     void gameexample(bool a);//游戏示例, 将所有形状恢复填入到网格中
53     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
54     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
55     void dealgridnumtext(QString text1);//根据界面输入信息记录网格大小
56     void deallinknumtext(QString text1);//根据界面输入信息记录划分形状的预设球数
57     void setballdiameter(QString text1);//根据界面输入信息记录球和方框的直径
58     void recordtime();//记录时间, 利用timer不断的显示流逝的游戏时间
59     void showcredit();//显示成绩排行榜
60
61 signals:
62     void shapecompleted(QString a);//当形状重新划分时, 划分完成的信号
63     void timerecorded(QString a);//记录一下游戏时间
64
65 private:
66     QList<ballshape *> ballshapingrid;//算法划分的形状图形的信息
67     QList<ballshape *> ballshapingriddefault;//默认保存的形状图形的信息
68     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
69     //的确也没有必要拷贝, 用指针可以指向它们
70
71     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
72     int gridnum, linknum;//记录输入的网格大小和链接形状的球数
73     int balldiameter;//设置球的直径
74     sceneinfo *sceneonwork;//当前网格所要加入的场景, 注意用指针是避免传值
75     int gamestarted;//记录游戏开始, =1表示已开始, 才会做落位判断
76     int flaggridchanged;//记录是否更改了网格大小和链接球数, 有更改, 则设置为1
77     int flagdiamchanged;//记录是否更改了球的直径, 有更改, 则设置为1

```

```

78  int flaggametimeon;//记录是否开始游戏计时，当游戏计时开始后，只有完成游戏才能更改计时，示例不影响计时
79  int flagexampleon;//记录是否处于示例状态，处于示例状态时设置为1，非处于时设置为0
80
81  QTime timecount;
82  QTimer timera;
83
84  QDomDocument doc;
85  float timegame;//用于记录本次游戏时间
86  QMediaPlayer *player;
87
88 };
89
90 #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 ballshapelsingriddefault 保存的默认形状图形的信息, ballshapelsingrid 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

```

1  #include "ballgrid.h"
2  #include <qDebug>
3  #include <QMessageBox>
4
5  #include <QInputDialog>
6  #include <QTableWidget>
7  #include <QDialog>
8  #include <QLineEdit>
9  #include <QDir>
10
11 ballgrid::ballgrid()

```

```

12 {
13     shapeidtosit=0; //
14     ballidtosit=0;
15     gridnum=10;
16     linknum=5;
17     flaggridchanged=0;
18     flagdiamchanged=0;
19     flaggametimeon=0;
20     flagexampleon=0;
21     balldiameter=70; //40 for desktop, 70 for apk
22     gamestarted=0;
23     timera.setInterval(1000);
24     QObject::connect(&timera,&QTimer::timeout,this,&ballgrid::recordtime); //
25     player=new QMediaPlayer;
26
27
28     //读取或创建一个xml文件用于记录游戏成绩
29     //for windows
30     //QString a=QCoreApplication::applicationDirPath();
31     //QString filepatha=a+"/info.xml";
32     //for andriod
33     QString filepatha="assets:/info.xml";
34     QFile file(filepatha);
35     if(file.exists())
36     {
37         if (!file.open(QIODevice::ReadOnly))
38             return;
39         if (!doc.setContent(&file)) {
40             file.close();
41             return;
42         }
43     }else
44     {
45         if(!file.open(QIODevice::WriteOnly|QIODevice::Truncate)) return;
46         QTextStream out(&file);
47         out <<"<?xml version='1.0' encoding='UTF-8'?'>\n";
48         out <<"<scores>\n";
49         out <<"<record user='zs' gridnum='10' linknum='5' id='1'>500</record>\n";
50         out <<"</scores>\n";
51         file.close();
52         if (!file.open(QIODevice::ReadOnly))
53             return;
54         if (!doc.setContent(&file)) {
55             file.close();
56             return;
57         }
58     }
59     file.close();
60 }
61
62 void ballgrid::setworkscene(sceneinfo*trans &scene)
63 {
64     sceneonwork=&scene;
65 }
66

```

```

67 int ballgrid::getgridnum()
68 {
69     return gridnum;
70 }
71 int ballgrid::getlinknum()
72 {
73     return linknum;
74 }
75 int ballgrid::getflag()
76 {
77     return flaggridchanged;
78 }
79
80 void ballgrid::setballdiameter(QString text1)
81 {
82     balldiameter=text1.toInt();
83     flagdiamchanged=1;
84     gamestarted=0;//当球的直径改变后,因为涉及到球的重新绘制,因此将不能进一步游戏,只能重新设置或重新划分。所以这里设置
      gamestarted=0
85 }
86
87 void ballgrid::dealgridnumtext(QString text1)
88 {
89     gridnum=text1.toInt();
90     flaggridchanged=1;
91     //qDebug()<<"gridnum:"<<gridnum;
92 }
93
94 void ballgrid::deallinknumtext(QString text1)
95 {
96     linknum=text1.toInt();
97     flaggridchanged=1;
98     //qDebug()<<"linknum:"<<linknum;
99 }
100
101 void ballgrid::recordtime()
102 {
103     int msa=timecount.elapsed();
104     float sa=msa/1000.0;
105     timegame=sa;
106     emit timerecorded(QString("%1").arg(sa));//发送信号,显示到界面中的文本框中
107 }
108
109 void ballgrid::gamestart(bool a)
110 {
111     Q_UNUSED(a);
112     if(flaggridchanged==0 && flagdiamchanged==0) {
113         setshaperandposdefault();
114         player->setMedia(QUrl("qrc:/sound/wav/trans.wav"));//行
115         player->play();
116         cleargridoccp();
117         gamestarted=1;
118         if(flaggametimeon==0) { //计时器开始工作,只有当新游戏开始时才计时
119             timecount.start();//开始时间记录
120             timera.start();

```



```

121         flaggametimeon=1;
122     }
123 }else{
124     if(flaggridchanged==1)
125     {
126         player->setMedia(QUrl("qrc:/sound/wav/warning.wav")); //行
127         player->play();
128         QMessageBox msgBox;
129         msgBox.setWindowTitle("WARNING");
130         msgBox.setText("Grid has been changed, need to partition, \nplease pull reset or reparation button");
131         msgBox.exec();
132     }
133     if(flagdiamchanged==1)
134     {
135         player->setMedia(QUrl("qrc:/sound/wav/warning.wav")); //行
136         player->play();
137         QMessageBox msgBox;
138         msgBox.setWindowTitle("WARNING");
139         msgBox.setText("Ball diameter has been changed, need to reset, \nplease pull reset or reparation button"
140             );
141         msgBox.exec();
142     }
143 }
144
145 void ballgrid::gameexample(bool a)
146 {
147     Q_UNUSED(a);
148     if(flagdiamchanged==0) { //当球的直径未修改时, 示例才能正确
149         if(flaggametimeon==1){
150             if(flagexampleon==0) {
151                 for(int i=0;i<n_ofshapes;i++)
152                 {
153                     //int id=veclinkshape[i].getshapen();
154                     int n_ballsinshape=veclinkshape[i].getshapenballs();
155                     //QDebug()<<"id:"<<id;
156                     //QDebug()<<"n_ball_inshape:"<<n_ballsinshape;
157
158                     for(int j=0;j<n_ballsinshape;j++) //直接根据记录的球的原始的行列信息重设位置
159                     {
160                         int row=veclinkshape[i].getptroworig(j);
161                         int col=veclinkshape[i].getptcolorig(j);
162                         ballshapesingriddefault[i]->setballpos(j,col*balldiameter,row*balldiameter);
163                     }
164                 }
165                 player->setMedia(QUrl("qrc:/sound/wav/trans.wav")); //行
166                 player->play();
167                 gamestarted=0;
168                 flagexampleon=1;
169             }else{
170                 setshaperandposdefault();
171                 player->setMedia(QUrl("qrc:/sound/wav/trans.wav")); //行
172                 player->play();
173                 cleargridoccp();
174                 gamestarted=1;

```

```

175         flagexampleon=0;
176     }
177 }else{
178     player->setMedia(QUrl("qrc:/sound/wav/warning.wav")); //行
179     player->play();
180     QMessageBox msgBox;
181     msgBox.setWindowTitle("WARNING");
182     msgBox.setText("Game is not started, need not to show example,\nplease pull the start button!");
183     msgBox.exec();
184 }
185 }else
186 {
187     player->setMedia(QUrl("qrc:/sound/wav/warning.wav")); //行
188     player->play();
189     QMessageBox msgBox;
190     msgBox.setWindowTitle("WARNING");
191     msgBox.setText("Ball diameter is changed, need to reset,\nplease pull the reset or repartition button!");
192     msgBox.exec();
193 }
194 }
195
196 void ballgrid::gameset(bool a)
197 {
198     Q_UNUSED(a);
199     player->setMedia(QUrl("qrc:/sound/wav/trans.wav")); //行
200     player->play();
201     if(flaggridchanged==1 || flagdiamchanged==1) //只有网格大小或链接球数改变了才做
202     {
203         qDebug()<<"do new partition";
204         removeboxes();
205         removeshapes();
206         setgrid(gridnum,linknum);
207         setboxes();
208         addboxestoscene();
209         setshapesdefault(n_total);
210         addtoscedefault();
211     }else
212     {
213         player->setMedia(QUrl("qrc:/sound/wav/warning.wav")); //行
214         player->play();
215         QMessageBox msgBox;
216         msgBox.setWindowTitle("WARNING");
217         msgBox.setText("Grid not changed,need not to partition");
218         msgBox.exec();
219     }
220     flaggridchanged=0;
221     flagdiamchanged=0;
222     gamestarted=0;
223     if(flaggametimeon==1) { //计时器开始工作，只有当新游戏开始时才计时
224         timera.stop();
225         flaggametimeon=0;
226     }
227 }
228
229 void ballgrid::gamerepart(bool a)

```

```

230 {
231     Q_UNUSED(a);
232     player->setMedia(QUrl("qrc:/sound/wav/trans.wav")); //行
233     player->play();
234     flaggridchanged=1;
235     flagdiamchanged=1;
236     qDebug()<<"redo new partition";
237     removeboxes();
238     removeshapes();
239     setgrid(gridnum,linknum);
240     setboxes();
241     addboxestoscene();
242     setshapesdefault(n_total);
243     qDebug()<<"n_ofshapes="<<n_ofshapes;
244     qDebug()<<"n_total="<<n_total;
245     addtoscenedefault();
246     flaggridchanged=0;
247     flagdiamchanged=0;
248     gamestarted=0;
249     if(flaggametimeon==1) { //计时器开始工作，只有当新游戏开始时才计时
250         timera.stop();
251         flaggametimeon=0;
252     }
253 }
254
255
256 void ballgrid::setboxes() //根据setgrid生成的框生成对应的boxitem，并绘制所有的框
257 {
258     //boxitem boxb[n_total];
259     boxesingrid.clear();
260     for(int i=0;i<n_total;++i)
261     {
262         boxitem *boxa=new boxitem();
263         boxa->setRect(0, 0,balldiameter,balldiameter);
264         boxa->setBrush(Qt::lightGray); //cyan
265         coord boxpos=getrowcolformsn(i+1);
266         boxa->setPos(boxpos.col*balldiameter,boxpos.row*balldiameter);
267         boxa->setboxid(i+1);
268         boxesingrid.append(boxa);
269         QObject::connect(boxa,&boxitem::tojudgesitdown,this,&ballgrid::judgesitdown);
270     }
271     //drawboxes();
272 }
273
274 void ballgrid::addboxestoscene() //将绘制的框显示到场景中
275 {
276     int n_boxes=boxesingrid.size();
277     for(int i=0;i<n_boxes;++i)
278     {
279         sceneonwork->addItem(boxesingrid[i]);
280     }
281 }
282
283 int ballgrid::getboxidat(QPointF posa) //根据当前位置确定box的索引，从0开始计数
284 {

```

```

285     int flagingrid=0;
286     int i=0;
287     for(i=0;i<n_total;++i)
288     {
289         QPointF posb=boxesingrid[i]->pos();
290         if(QLineF(posa,posb).length() <0.00001)
291         {
292             flagingrid=1;
293             break;
294         }
295     }
296     if(flagingrid==0) //当不在网格内时返回网格中总的框数加1
297     {
298         return n_total+1;
299     }else
300     {
301         return i;
302     }
303 }
304
305 void ballgrid::removeboxes()//将绘制的框从场景中移除
306 {
307     int n_boxes=boxesingrid.size();
308     for(int i=0;i<n_boxes;++i)
309     {
310         sceneonwork->removeItem(boxesingrid[i]);
311     }
312 }
313
314 void ballgrid::judgesitdown(int boxid)//点击选中box时的形状落位判断
315 {
316     if(gamestarted==1) {
317         qDebug()<<"recieved info from box:"<<boxid;
318         //是否有shape要放的信息, 有的话, 根据shape信息调用
319         int flagyestosit=0;
320         int n_balls=0;
321         if(shapeidtosit>0) { //判断是否存在shape需要落位
322             QPointF posa=ballshapelsingriddefault[shapeidtosit-1]->getballpos(ballidtosit);
323             int boxida=getboxidat(posa)+1;//当前球所在的位置
324             if(boxida <= n_total)//当在网格内时
325             {
326                 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
327             }
328             qDebug()<<"ball in box id:"<<boxida;
329             qDebug()<<"ball id:"<<ballidtosit;
330             qDebug()<<"shape id:"<<shapeidtosit;
331
332             setfocus(boxid);
333             n_balls=veclinkshape[shapeidtosit-1].getshapenballs();
334             std::vector<coord> ballscords=veclinkshape[shapeidtosit-1].getvec_coord();
335             flagyestosit=canbesetdown(n_balls,ballidtosit, ballscords);//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
336         }
337     }

```

```

338 //如果能放入则发送信息给shape, 接着shape调整位置, //即绘图
339 if(flagvestosit>0) {
340     int rowfocusinshape=veclinkshape[shapeidtosit-1].getptrow(ballidtosit-1);
341     int colfocusinshape=veclinkshape[shapeidtosit-1].getptcol(ballidtosit-1);
342     int rowfocusingrid=getrowcolformsn(boxid).row;
343     int colfocusingrid=getrowcolformsn(boxid).col;
344     setshapeoccp(boxid,ballidtosit,shapeidtosit);//设置一下占用情况
345     for(int j=0;j<n_balls;j++)
346     {
347         int row=(veclinkshape[shapeidtosit-1].getptrow(j))-rowfocusinshape+rowfocusingrid;
348         int col=(veclinkshape[shapeidtosit-1].getptcol(j))-colfocusinshape+colfocusingrid;
349
350         ballshapingriddefault[shapeidtosit-1]->setballpos(j,col*balldiameter,row*balldiameter);
351     }
352     player->setMedia(QUrl("qrc:/sound/wav/dropdown.wav"));//行
353     player->play();
354 }
355
356 if(gridocupied())
357 {
358     player->setMedia(QUrl("qrc:/sound/wav/gamewon.wav"));//行
359     player->play();
360     QMessageBox msgBox;
361     msgBox.setWindowTitle("OK");
362     msgBox.setText("Goodjob! congratulations!! ");
363     msgBox.exec();
364     gamestarted=0;
365     timera.stop();
366     flaggametimeon=0;
367     isinrank(gridnum,linknum,timegame);//时间成绩处理
368 }
369 }
370 }
371
372
373 void ballgrid::isinrank(int gridnumber,int linknumber,float gametime)//判断本次游戏成绩是否进入排行榜, 进入则修改成绩记录
374     xml
375 {
376     QDomElement docElem = doc.documentElement();
377     QDomNode n = docElem.firstChild();
378     QList<QDomElement> elems;
379     while(!n.isNull())
380     {
381         elems.append(n.toElement());
382         n = n.nextSibling();
383     }
384     QList<int> elemelist;//记录此类元素的索引的列表
385     int idrec=1;
386     int elemlast=0;
387     for(int i=0;i<elems.count();++i)
388     {
389         qDebug()<< qPrintable(elems[i].tagName())<<elems[i].text();
390         qDebug()<< "gridnumber:"<<elems[i].attribute("gridnum").toInt();
391         if(gridnumber == elems[i].attribute("gridnum").toInt() && linknumber == elems[i].attribute("linknum").toInt())

```

```

392     {
393         elemlast=i;//记录下此类最后一个排名的元素的索引
394         idrec+=1;
395         elemlist.append(i);
396     }
397 }
398
399 bool flaginrank=false;
400 if(!elemlist.empty())//存在此类成绩记录，则判断排行榜
401 {
402     int elemins=0;
403     for(int j=0;j<elemlist.count();j++)
404     {
405         int i=elemlist[j];
406         if(gametime < elems[i].text().toFloat())//当游戏时间小于存在得记录，则做处理
407         {
408             if(idrec > elems[i].attribute("id").toInt()) { //主要是设置id值
409                 idrec=elems[i].attribute("id").toInt();
410                 elemins=i;//记录要插入位置的元素的索引
411                 elems[i].setAttribute("id",idrec+1);
412             }else
413             {
414                 elems[i].setAttribute("id",elems[i].attribute("id").toInt()+1);
415             }
416         }
417     }
418     //当游戏时间都大于版中的时，插入位置是当前类的最后一个成绩后面
419     if(idrec==elemlist.count()+1) elemins=elemlast+1;
420
421
422     qDebug()<<"current score's rank id :idrect"<<idrec;
423     QString gamer;
424     if(idrec<=elemlist.count() || idrec<=3) { //排行榜内
425         qDebug()<< "is in rank list";
426         flaginrank=true;
427
428         QDialog *rankdialog=new QDialog;//input dialog的parent需要是一个widget
429         QDialog inputdialog;
430         bool ok;
431         gamer = inputdialog.getText(rankdialog, tr("InputDialog::getText()"),
432                                     tr("gamer:"), QLineEdit::Normal,
433                                     tr("user"),&ok);
434
435         if (gamer.isEmpty())
436         {
437             gamer="no name";
438         }
439         qDebug()<<"gamer:"<< gamer;
440
441     }
442
443
444     if(flaginrank) {
445         QDomElement reca = doc.createElement("record");
446         reca.setAttribute("gridnum",gridnumber);

```

```

447     reca.setAttribute("linknum",linknumber);
448     reca.setAttribute("user",gamer);
449     reca.setAttribute("id",idrec);
450     QDomText recat = doc.createTextNode(QString("%1").arg(gametime));
451     reca.appendChild(recat);
452     //docElem.appendChild(reca);
453     qDebug()<<"elems:"<<elems.count()<<"insert pos:"<<elemins;
454     if(elemins < elems.count()) { //当插入位置在elems内时插入
455         docElem.insertBefore(reca,elems[elemins]);
456     } else //当插入位置在最后时
457     {
458         docElem.appendChild(reca);
459     }
460     if(elemlist.count()>2)
461     { //只有当一类排行数量大于3才会删除
462         docElem.removeChild(elems[elemlast]);
463     }
464 }
465 }
466 else //不存在此类成绩记录, 则创建排行榜
467 {
468     QString gamer;
469     qDebug()<< "is in rank list";
470     flaginrank=true;
471
472     QDialog *rankdialog=new QDialog;
473     QDialog inputdialog;
474     bool ok;
475     gamer = inputdialog.getText(rankdialog, tr("InputDialog::getText()"),
476                                tr("gamer:"), QLineEdit::Normal,
477                                tr("user"),&ok);
478
479
480     if (gamer.isEmpty())
481     {
482         gamer="no name";
483     }
484     qDebug()<< gamer;
485
486     QDomElement reca = doc.createElement("record");
487     reca.setAttribute("gridnum",gridnumber);
488     reca.setAttribute("linknum",linknumber);
489     reca.setAttribute("user",gamer);
490     reca.setAttribute("id",1);
491     QDomText recat = doc.createTextNode(QString("%1").arg(gametime));
492     reca.appendChild(recat);
493     docElem.appendChild(reca);
494
495 }
496
497 if(flaginrank){ //在排行榜内才保存
498     //QString a=QCoreApplication::applicationDirPath();
499     //QString filepathb=a+"/info.xml";
500     QString filepathb="assets:/info.xml";
501     QFile fileb(filepathb);

```



```

502     if(!fileb.open(QIODevice::WriteOnly|QIODevice::Truncate)) return;
503     QTextStream out(&fileb);
504     doc.save(out,4);
505     fileb.close();
506 }
507 }
508
509
510 void ballgrid::showcredit()
511 {
512     QDomElement docElem = doc.documentElement();
513     QDomNode n = docElem.firstChild();
514     QList<QDomElement> elems;
515     while(!n.isNull())
516     {
517         elems.append(n.toElement());
518         n = n.nextSibling();
519     }
520
521
522     QDialog *rankdialog=new QDialog;
523     QTableWidgetItem *tableWidget;
524     tableWidget = new QTableWidgetItem(elems.count(),5, rankdialog);
525     tableWidget->setGeometry(0,0,500,300);
526
527
528     for(int row=0;row<elems.count();++row)
529     {
530         QTableWidgetItem *newItem = new QTableWidgetItem(elems[row].attribute("id"));
531         tableWidget->setItem(row, 0, newItem);
532         QTableWidgetItem *newItem1 = new QTableWidgetItem(elems[row].attribute("user"));
533         tableWidget->setItem(row, 1, newItem1);
534         QTableWidgetItem *newItem2 = new QTableWidgetItem(elems[row].text());
535         tableWidget->setItem(row, 2, newItem2);
536         QTableWidgetItem *newItem3 = new QTableWidgetItem(elems[row].attribute("gridnum"));
537         tableWidget->setItem(row, 3, newItem3);
538         QTableWidgetItem *newItem4 = new QTableWidgetItem(elems[row].attribute("linknum"));
539         tableWidget->setItem(row, 4, newItem4);
540     }
541
542     QTableWidgetItem *valuesHeaderItem = new QTableWidgetItem(tr("排名"));
543     tableWidget->setHorizontalHeaderItem(0, valuesHeaderItem);
544     QTableWidgetItem *valuesHeaderItem1 = new QTableWidgetItem(tr("游戏者"));
545     tableWidget->setHorizontalHeaderItem(1, valuesHeaderItem1);
546     QTableWidgetItem *valuesHeaderItem2 = new QTableWidgetItem(tr("完成时间"));
547     tableWidget->setHorizontalHeaderItem(2, valuesHeaderItem2);
548     QTableWidgetItem *valuesHeaderItem3 = new QTableWidgetItem(tr("网格大小"));
549     tableWidget->setHorizontalHeaderItem(3, valuesHeaderItem3);
550     QTableWidgetItem *valuesHeaderItem4 = new QTableWidgetItem(tr("形状大小"));
551     tableWidget->setHorizontalHeaderItem(4, valuesHeaderItem4);
552
553
554     QStringList labels;
555     for(int i=0;i<elems.count();++i)
556     {

```

```

557     labels<<" ";
558 }
559 tableWidget->setVerticalHeaderLabels(labels);
560
561
562 tableWidget->show();
563
564 rankdialog->exec();
565 }
566
567
568 void ballgrid::dealshapetosit(int ballid,int shapeida)
569 {
570     shapeidtosit=shapeida;
571     ballidtosit=ballid;
572     qDebug()<<"shape's balls:"<<veclinkshape[shapeida-1].getshapenballs();
573     for(int i=0;i<veclinkshape[shapeida-1].getshapenballs();++i)
574     {
575         qDebug()<<"coord:"<<veclinkshape[shapeida-1].getptrow(i)<<veclinkshape[shapeida-1].getptcol(i);
576         if(gamestarted==1) {
577             qDebug()<<" coord outer"<<veclinkshape[shapeida-1].getptrowouter(i)<<veclinkshape[shapeida-1].getptcolouter(i);
578         }
579     }
580     player->setMedia(QUrl("qrc:/sound/wav/selected.wav"));
581     player->play();
582     qDebug()<<"recieved info from shape:"<<shapeida;
583     qDebug()<<"shape:"<<shapeida<<" wants to be sit down!";
584 }
585
586 void ballgrid::shapeturnright(bool a)//按钮控制向右旋转
587 {
588     Q_UNUSED(a);
589     if(gamestarted==1) {
590         if(shapeidtosit>0){//当有形状选中的时候做
591             shapetrans(shapeidtosit,1);
592             player->setMedia(QUrl("qrc:/sound/wav/trans.wav"));//行
593             player->play();
594         }else{
595             player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
596             player->play();
597             QMessageBox msgBox;
598             msgBox.setWindowTitle("WARNING");
599             msgBox.setText("None shape was selected! \npress choose a shape!");
600             msgBox.exec();
601         }
602     }else{
603         player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
604         player->play();
605         QMessageBox msgBox;
606         msgBox.setWindowTitle("WARNING");
607         msgBox.setText("please start the game! \npress start button!");
608         msgBox.exec();
609     }
610 }
611

```

```

612 void ballgrid::shapeturnleft(bool a)//按钮控制向左旋转
613 {
614     Q_UNUSED(a);
615     if(gamestarted==1) {
616         if(shapeidtosit>0){//当有形状选中的时候做
617             shapetrans(shapeidtosit,2);
618             player->setMedia(QUrl("qrc:/sound/wav/trans.wav"));//行
619             player->play();
620         }else{
621             player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
622             player->play();
623             QMessageBox msgBox;
624             msgBox.setWindowTitle("WARNING");
625             msgBox.setText("None shape was selected! \npress choose a shape!");
626             msgBox.exec();
627         }
628     }else{
629         player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
630         player->play();
631         QMessageBox msgBox;
632         msgBox.setWindowTitle("WARNING");
633         msgBox.setText("please start the game! \npress start button!");
634         msgBox.exec();
635     }
636 }
637
638 void ballgrid::shapemirrorab(bool a)//按钮控制上下翻转
639 {
640     Q_UNUSED(a);
641     if(gamestarted==1) {
642         if(shapeidtosit>0){//当有形状选中的时候做
643             shapetrans(shapeidtosit,3);
644             player->setMedia(QUrl("qrc:/sound/wav/trans.wav"));//行
645             player->play();
646         }else{
647             player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
648             player->play();
649             QMessageBox msgBox;
650             msgBox.setWindowTitle("WARNING");
651             msgBox.setText("None shape was selected! \npress choose a shape!");
652             msgBox.exec();
653         }
654     }else{
655         player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
656         player->play();
657         QMessageBox msgBox;
658         msgBox.setWindowTitle("WARNING");
659         msgBox.setText("please start the game! \npress start button!");
660         msgBox.exec();
661     }
662 }
663
664 void ballgrid::shapemirrorlr(bool a)//按钮控制左右翻转
665 {
666     Q_UNUSED(a);

```

```

667     if(gamestarted==1) {
668         if(shapeidtosit>0){//当有形状选中的时候做
669             shapetrans(shapeidtosit,4);
670             player->setMedia(QUrl("qrc:/sound/wav/trans.wav"));//行
671             player->play();
672         }else{
673             player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
674             player->play();
675             QMessageBox msgBox;
676             msgBox.setWindowTitle("WARNING");
677             msgBox.setText("None shape was selected! \npress choose a shape!");
678             msgBox.exec();
679         }
680     }else{
681         player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
682         player->play();
683         QMessageBox msgBox;
684         msgBox.setWindowTitle("WARNING");
685         msgBox.setText("please start the game! \npress start button!");
686         msgBox.exec();
687     }
688 }
689
690 void ballgrid::shaperesetpos(bool a)//按钮控制移出网格
691 {
692     Q_UNUSED(a);
693     if(shapeidtosit>0){//当有形状选中的时候做
694         shapereset(ballidtosit,shapeidtosit);
695         player->setMedia(QUrl("qrc:/sound/wav/moveout.wav"));//行
696         player->play();
697     }else{
698         player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
699         player->play();
700         QMessageBox msgBox;
701         msgBox.setWindowTitle("WARNING");
702         msgBox.setText("None shape was selected! \npress choose a shape!");
703         msgBox.exec();
704     }
705 }
706
707
708 void ballgrid::shapereset(int ballid,int shapeida)
709 {
710     if(gamestarted==1) {
711         //QDebug()<<"recieved info from shape:"<<shapeida;
712         //QDebug()<<"shape:"<<shapeida<<" wants to be sit down!";
713         shapeidtosit=shapeida;
714         ballidtosit=ballid;
715
716         QPointF posa=ballshapingriddefault[shapeida-1]->getballpos(ballid);
717         int boxid=getboxidat(posa)+1;//当前球所在的位置
718         if(boxid < n_total)
719         {
720             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
721         }

```

```

722 //qDebug()<<"boxid:"<<boxid;
723
724 ballshapessingriddefault[shapeida-1]->resetpos();
725 veclinkshape[shapeida-1].resetoutercoord();
726 player->setMedia(QUrl("qrc:/sound/wav/moveout.wav"));//行
727 player->play();
728
729 }else{
730     player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
731     player->play();
732     QMessageBox msgBox;
733     msgBox.setWindowTitle("WARNING");
734     msgBox.setText("please start the game! \npress start button!");
735     msgBox.exec();
736 }
737 }
738
739
740 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
741 {
742     QColor colora[256];
743     for(int i=0;i<256;i++)
744     {
745         colora[i]=QColor(qrand()%256,qrand()%256,i);
746     }
747
748     ballshapessingriddefault.clear();//初始化一下
749     std::vector<linkshape> veclinka;
750     veclinkshape=veclinka;
751
752     qDebug()<<"gridnum:"<<gridnum;
753     if(flaggridchanged==0 && gridnum ==10)
754     {
755
756         n_ofshapes=12;
757         coord vectmp1[5]={6,0},{7,0},{7,1},{7,2},{7,3};
758         int vecnumber1[5]={22, 29, 30, 31, 32};
759         linkshape shapea=linkshape(1,5,vectmp1,vecnumber1);
760         veclinkshape.push_back(shapea);
761
762         coord vectmp2[5]={5,5},{5,4},{5,3},{4,3},{4,4};
763         int vecnumber2[5]={21, 20, 19, 14, 15};
764         shapea=linkshape(2,5,vectmp2,vecnumber2);
765         veclinkshape.push_back(shapea);
766
767         coord vectmp3[5]={3,3},{3,2},{4,2},{5,2},{3,1};
768         int vecnumber3[5]={10, 9, 13, 18, 8};
769         shapea=linkshape(3,5,vectmp3,vecnumber3);
770         veclinkshape.push_back(shapea);
771
772         coord vectmp4[5]={8,7},{7,7},{9,7},{9,6},{9,5};
773         int vecnumber4[5]={44, 36, 53, 52, 51};
774         shapea=linkshape(4,5,vectmp4,vecnumber4);
775         veclinkshape.push_back(shapea);

```

```

776
777 coord vectmp5[5]={9,4},{9,3},{9,2},{9,1},{9,0}};
778 int vecnumber5[5]={50, 49, 48, 47, 46};
779 shapea=linkshape(5,5,vectmp5,vecnumber5);
780 veclinkshape.push_back(shapea);
781
782 coord vectmp6[5]={8,5},{8,6},{7,6},{6,6},{6,5}};
783 int vecnumber6[5]={42, 43, 35, 28, 27};
784 shapea=linkshape(6,5,vectmp6,vecnumber6);
785 veclinkshape.push_back(shapea);
786
787
788 coord vectmp7[3]={9,9},{9,8},{8,8}};
789 int vecnumber7[3]={55, 54, 45};
790 shapea=linkshape(7,3,vectmp7,vecnumber7);
791 veclinkshape.push_back(shapea);
792
793 coord vectmp8[5]={6,3},{6,2},{6,1},{5,1},{5,0}};
794 int vecnumber8[5]={25, 24, 23, 17, 16};
795 shapea=linkshape(8,5,vectmp8,vecnumber8);
796 veclinkshape.push_back(shapea);
797
798 coord vectmp9[5]={8,3},{8,4},{7,4},{7,5},{6,4}};
799 int vecnumber9[5]={40, 41, 33, 34, 26};
800 shapea=linkshape(9,5,vectmp9,vecnumber9);
801 veclinkshape.push_back(shapea);
802
803 coord vectmp10[5]={2,1},{2,2},{1,1},{1,0},{0,0}};
804 int vecnumber10[5]={5, 6, 3, 2, 1};
805 shapea=linkshape(10,5,vectmp10,vecnumber10);
806 veclinkshape.push_back(shapea);
807
808 coord vectmp11[4]={4,0},{4,1},{3,0},{2,0}};
809 int vecnumber11[4]={11, 12, 7, 4};
810 shapea=linkshape(11,4,vectmp11,vecnumber11);
811 veclinkshape.push_back(shapea);
812
813 coord vectmp12[3]={8,1},{8,2},{8,0}};
814 int vecnumber12[3]={38, 39, 37};
815 shapea=linkshape(12,3,vectmp12,vecnumber12);
816 veclinkshape.push_back(shapea);
817
818 int ballpos[arraysize];
819
820 for(int i=0;i<n_ofshapes;i++)
821 {
822     int id=veclinkshape[i].getshapen();
823     int n_ballsinshape=veclinkshape[i].getshapenballs();
824     //QDebug()<<"id:"<<id;
825     //QDebug()<<"n_ball_inshape:"<<n_ballsinshape;
826
827     for(int j=0;j<n_ballsinshape;j++)
828     {
829         int row=(veclinkshape[i].getptrow(j)+veclinkshape[i].getrowofzero());
830         int col=(veclinkshape[i].getptcol(j)+veclinkshape[i].getcolofzero());

```

```

831         //qDebug()<<"ball sn="<<veclinkshape[i].getballsn(j);
832         //qDebug()<<"row="<<row<<" col="<<col;
833         ballpos[2*j]=(col*balldiameter);
834         ballpos[2*j+1]=(row*balldiameter);
835     }
836
837     ballshape *shapea=new ballshape();
838     shapea->setballdiameter(balldiameter);
839     shapea->setballshape(id,n_ballsinshape,ballpos,colora[qrand()%256]);
840     ballshapessingriddefault.append(shapea);
841     QObject::connect(shapea,&ballshape::shapeselected,this,&ballgrid::dealshapetosit);
842     QObject::connect(shapea,&ballshape::shapetoreset,this,&ballgrid::shapereset);
843     QObject::connect(shapea,&ballshape::shapetotrans,this,&ballgrid::shapetrans);
844 }
845 } else
846 {
847     partition(n_total);
848
849     qDebug()<<"n_ofshapes="<<n_ofshapes;
850     qDebug()<<"n_total="<<n_total;
851     if(n_ofshapes<n_total) {
852         setshapes(n_total);
853         ballshapessingriddefault=ballshapessingrid;
854         QString stra=QString("%1").arg(n_ofshapes);
855         emit shapecompleted(stra);
856     }else
857     {
858         player->setMedia(QUrl("qrc:/sound/wav/warning.wav")); //行
859         player->play();
860         QMessageBox msgBox;
861         msgBox.setWindowTitle("WARNING");
862         msgBox.setText(" partition is not successful! \n please reset the grid scale and link ball number, \n
            and redo partition!");
863         msgBox.exec();
864         qDebug()<<"partition is not successful! please redo!"<<n_ofshapes;
865     }
866 }
867 }
868
869 void ballgrid::addtoscedefault()//将默认的划分的形状加入到场景中
870 {
871     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
872         for(int i=0;i<n_ofshapes;i++)
873         {
874             ballshapessingriddefault[i]->addtoscene(*sceneonwork);
875         }
876     }
877 }
878
879 void ballgrid::removesshapes()//将划分形状从场景移除
880 {
881     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
882         for(int i=0;i<n_ofshapes;i++)
883         {
884             ballshapessingriddefault[i]->removeballs(*sceneonwork);

```



```

885     }
886     }
887 }
888
889
890 void ballgrid::setshaperandposdefault()//将划分形状放到网格外
891 {
892     int colnow=0;
893     int rownow=-5;
894     for(int i=0;i<n_ofshapes;i++)
895     {
896         //全部在顶部竖放
897         if(colnow > gridnum) {
898             colnow=0;
899             rownow=-5;
900         }
901
902         int rows=veclinkshape[i].getrows();
903         int cols=veclinkshape[i].getcols();
904         if(cols > rows )
905         {
906             veclinkshape[i].rotate_right();
907             rows=veclinkshape[i].getrows();
908             cols=veclinkshape[i].getcols();
909         }
910
911         for(int j=0;j<veclinkshape[i].getshapenballs();j++)
912         {
913             int row=rownow+(veclinkshape[i].getptrow(j));
914             int col=colnow+(veclinkshape[i].getptcol(j));
915
916             ballshapelsingriddefault[i]->setballpos(j,col*balldiameter,row*balldiameter);
917         }
918         colnow=colnow+cols+1;
919
920         /* 一排竖放, 然后一列横放
921         if(colnow < gridnum+2 ) { //一排排完前 //一半竖着放 i < n_ofshapes/2
922             int rows=veclinkshape[i].getrows();
923             int cols=veclinkshape[i].getcols();
924             if(cols > rows )
925             {
926                 veclinkshape[i].rotate_right();
927                 rows=veclinkshape[i].getrows();
928                 cols=veclinkshape[i].getcols();
929             }
930
931             for(int j=0;j<veclinkshape[i].getshapenballs();j++)
932             {
933                 int row=(veclinkshape[i].getptrow(j))-rows-1;
934                 int col=colnow+(veclinkshape[i].getptcol(j));
935
936                 ballshapelsingriddefault[i]->setballpos(j,col*balldiameter,row*balldiameter);
937             }
938             colnow=colnow+cols+1;
939         }

```

```

940     else//另一半横着放
941     {
942         int rows=veclinkshape[i].getrows();
943         int cols=veclinkshape[i].getcols();
944         if(cols < rows )
945         {
946             veclinkshape[i].rotate_right();
947             rows=veclinkshape[i].getrows();
948             cols=veclinkshape[i].getcols();
949         }
950         for(int j=0;j<veclinkshape[i].getshapenballs();j++)
951         {
952             int row=rownow+(veclinkshape[i].getptrow(j));
953             int col=colnow+(veclinkshape[i].getptcol(j));
954
955             ballshapelsingriddefault[i]->setballpos(j,col*balldiameter,row*balldiameter);
956         }
957         rownow=rownow+rows+1;
958     }
959     */
960
961     ballshapelsingriddefault[i]->recordpos();
962     veclinkshape[i].recordoutercoord();
963 }
964 }
965
966
967 void ballgrid::shapetrans(int shapeida,int transid)//形状变换:旋转和翻转
968 {
969     if(shapeidtosit!=shapeida){
970         qDebug()<<"error:"<<shapeidtosit<<shapeida;
971     }
972
973     QPointF posa=ballshapelsingriddefault[shapeida-1]->getballpos(ballidtosit);
974     int boxid=getboxidat(posa)+1;//当前球所在的位置
975     qDebug()<<"pos  :"<<posa.x()<<posa.y();
976
977     switch(transid)
978     {
979     case 1://Qt::Key_Right:
980         veclinkshape[shapeida-1].saveoldcoord();//保存原始坐标信息
981         if(boxid <= n_total)//当在网格内时
982         {
983             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
984             veclinkshape[shapeida-1].rotate_right();
985             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
986                 从1开始计数
987             if(flagcanbe==1)
988             {
989                 setshapeoccp(boxid,ballidtosit,shapeida);
990             }else
991             {
992                 player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
993                 player->play();
994                 QMessageBox msgBox;

```

```

994         msgBox.setWindowTitle("WARNING");
995         msgBox.setText("can not rotate right");
996         msgBox.exec();
997         vecLinkshape[shapeida-1].resetoldcoord();//恢复原始坐标信息
998         setshapeoccp(boxid,ballidtosit,shapeida);
999     }
1000 }
1001 else
1002 {
1003     vecLinkshape[shapeida-1].rotate_right();
1004 }
1005 break;
1006
1007 case 2://Qt::Key_Left:
1008     vecLinkshape[shapeida-1].saveoldcoord();//保存原始坐标信息
1009     if(boxid <= n_total)//当在网格内时
1010     {
1011         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
1012         vecLinkshape[shapeida-1].rotate_left();
1013         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
1014         if(flagcanbe==1)
1015         {
1016             setshapeoccp(boxid,ballidtosit,shapeida);
1017         }else
1018         {
1019             player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
1020             player->play();
1021             QMessageBox msgBox;
1022             msgBox.setWindowTitle("WARNING");
1023             msgBox.setText("can not rotate left");
1024             msgBox.exec();
1025             vecLinkshape[shapeida-1].resetoldcoord();//恢复原始坐标信息
1026             setshapeoccp(boxid,ballidtosit,shapeida);
1027         }
1028     }
1029     else
1030     {
1031         vecLinkshape[shapeida-1].rotate_left();
1032     }
1033
1034     break;
1035
1036 case 3://Qt::Key_Up:
1037     //QDebug()<<"mirror up and down";
1038     vecLinkshape[shapeida-1].saveoldcoord();//保存原始坐标信息
1039     if(boxid <= n_total)//当在网格内时
1040     {
1041         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
1042         vecLinkshape[shapeida-1].mirror_ab();
1043         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
1044         if(flagcanbe==1)
1045         {
1046             setshapeoccp(boxid,ballidtosit,shapeida);

```

```

1047     }else
1048     {
1049         player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
1050         player->play();
1051         QMessageBox msgBox;
1052         msgBox.setWindowTitle("WARNING");
1053         msgBox.setText("can not mirror up and down");
1054         msgBox.exec();
1055         //QDebug()<<"can not mirror up and down";
1056         veclinkshape[shapeida-1].resetoldcoord();//恢复原始坐标信息
1057         setshapeoccp(boxid,ballidtosit,shapeida);
1058     }
1059 }
1060 else
1061 {
1062     veclinkshape[shapeida-1].mirror_ab();
1063 }
1064 break;
1065
1066 case 4://Qt::Key_Down:
1067     //QDebug()<<"mirror up and down";
1068     veclinkshape[shapeida-1].saveoldcoord();//保存原始坐标信息
1069     if(boxid <= n_total)//当在网格内时
1070     {
1071         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
1072         veclinkshape[shapeida-1].mirror_lr();
1073         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
1074         if(flagcanbe==1)
1075         {
1076             setshapeoccp(boxid,ballidtosit,shapeida);
1077         }else
1078         {
1079             player->setMedia(QUrl("qrc:/sound/wav/warning.wav"));//行
1080             player->play();
1081             QMessageBox msgBox;
1082             msgBox.setWindowTitle("WARNING");
1083             msgBox.setText("can not mirror up and down");
1084             msgBox.exec();
1085             //QDebug()<<"can not mirror up and down";
1086             veclinkshape[shapeida-1].resetoldcoord();//恢复原始坐标信息
1087             setshapeoccp(boxid,ballidtosit,shapeida);
1088         }
1089     }
1090     else
1091     {
1092         veclinkshape[shapeida-1].mirror_lr();
1093     }
1094     break;
1095
1096 default:
1097     break;
1098 }
1099
1100 int n_balls=veclinkshape[shapeida-1].getshapenballs();

```

```

1101     int rowfocusinshape=veclinkshape[shapeidtosit-1].getptrow(ballidtosit-1);
1102     int colfocusinshape=veclinkshape[shapeidtosit-1].getptcol(ballidtosit-1);
1103
1104     for(int j=0;j<n_balls;j++)
1105     {
1106         int row=(veclinkshape[shapeidtosit-1].getptrow(j))-rowfocusinshape;
1107         int col=(veclinkshape[shapeidtosit-1].getptcol(j))-colfocusinshape;
1108
1109         ballshapingriddefault[shapeidtosit-1]->setballpos(j,col*balldiameter+posa.x(),row*balldiameter+posa.y());
1110     }
1111 }
1112
1113 void ballgrid::setshapes(int arraysze)//根据grid函数partition后得到的std::vector<linkshape> veclinkshape;
1114 {
1115     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
1116     //int ballpos[arraysize];
1117
1118     QColor colora[256];
1119     for(int i=0;i<256;i++)
1120     {
1121         colora[i]=QColor(qrand()%256,qrand()%256,i);
1122     }
1123
1124     ballshapingrid.clear();
1125     for(int i=0;i<n_ofshapes;i++)
1126     {
1127         int id=veclinkshape[i].getshapen();
1128         int n_ballsinshape=veclinkshape[i].getshapenballs();
1129         //qDebug()<<"id:"<<id;
1130         //qDebug()<<"n_ball_inshape:"<<n_ballsinshape;
1131
1132         for(int j=0;j<n_ballsinshape;j++)
1133         {
1134             int row=(veclinkshape[i].getptrow(j)+veclinkshape[i].getrowofzero());
1135             int col=(veclinkshape[i].getptcol(j)+veclinkshape[i].getcolofzero());
1136             //qDebug()<<"ball sn="<<veclinkshape[i].getballsn(j);
1137             //qDebug()<<"row="<<row<<" col="<<col;
1138             ballpos[2*j]=(col*balldiameter);
1139             ballpos[2*j+1]=(row*balldiameter);
1140         }
1141
1142         ballshape *shapea=new ballshape();
1143         shapea->setballdiameter(balldiameter);
1144         shapea->setballshape(id,n_ballsinshape,ballpos,colora[qrand()%256]);
1145         ballshapingrid.append(shapea);//形状图形列表ballshapingrid, 对应的生成的形状veclinkshape
1146
1147         QObject::connect(shapea,&ballshape::shapeselected,this,&ballgrid::dealshapetosit);
1148         QObject::connect(shapea,&ballshape::shapetoreset,this,&ballgrid::shapereset);
1149         QObject::connect(shapea,&ballshape::shapetotrans,this,&ballgrid::shapetrans);
1150     }
1151 }

```

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

```
1  int n_scale;//网格的大小，行列数均为n_scale
2      int n_balls_ofshape;//链接形状中的球数
3      int n_ofshapes;//划分成的形状数
4  int n_total;
5      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
6  std::vector<balltopo> vecgrid;//球列表
7      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, trigrigidinit 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。gridoccupied 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

```
1  #ifndef SHAPE_H
2  #define SHAPE_H
3
4  #include <stdio.h>
5  #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(trigrig[i][j]);
56                 nij++;
57                 //printf("%d ",nij);
58             }
59         }
60         //printf("\n");
61         trigridinit();
62     }
63

```

```

64 void trigradinit(){//下三角网格信息初始化
65     balltopo trigrad[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
78             //左侧邻居
79             int left=0;
80             if(j==1){
81                 left=-1;//j=1时,左侧无邻居,所以设置为-1
82             }
83             else{
84                 left=sn-1;
85             }
86
87             //右侧邻居
88             int right=0;
89             if(j==i){
90                 right=-1;//j=i时,右侧无邻居,所以设置为-1
91             }
92             else{
93                 right=sn+1;
94             }
95
96             //上方邻居
97             int above=0;
98             if(i==j){
99                 above=-1;//i=j时,上方无邻居,所以设置为-1
100             }
101             else{
102                 above=sn-i+1;
103             }
104
105             //下方邻居
106             int below=0;
107             if(i==n){
108                 below=-1;//i=n时,下方无邻居,所以设置为-1
109             }
110             else{
111                 below=sn+i;
112             }
113
114             //printf("sn:%d,left:%d,right:%d,above:%d,below:%d\n",sn,left,right,above,below);
115             int topset[5]={sn,left,right,above,below};
116             trigrad[i-1][j-1].settopo(topset);
117         }
118     }
119     n_total=sn;

```



```

119     int nij=0;
120     for (int i=0;i<n_scale;i++)
121     {
122         for(int j=0;j<=i;j++)//[i][j]与矢量中序号是一一对应的。
123         {
124             trigrd[i][j].setoccp(0);
125             trigrd[i][j].setshpd(0);
126             vecgrid[nij]=trigrd[i][j];
127             nij++;
128         }
129     }
130 }
131
132 //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
133 int checksameshape(linkshape a,linkshape b)
134 {
135     if(a.getshapenballs() != b.getshapenballs()) {
136         return 0;
137     }else
138     {
139         for(int i=0;i<a.getshapenballs();i++)
140         {
141             if(fabs(a.getshapeeigen(i)-b.getshapeeigen(i))>0.000001) return 0;
142         }
143     }
144
145     //printf("eigen is the same!\n");
146
147     int flagsame=0;
148     //printf("ball a rows=%d , cols=%d\n",a.getrows(),a.getcols());
149     //printf("ball b rows=%d , cols=%d\n",b.getrows(),b.getcols());
150     if(a.getrows() != a.getcols()) { //行列不相等时，只要做翻转就可以比较了
151         if(a.getrows()==b.getrows() && a.getcols()==b.getcols() )//当行数列数完全相同
152         {
153             int sumrowsa=0;
154             int sumcolsa=0;
155             int sumrowsb=0;
156             int sumcolsb=0;
157             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
158             {
159                 sumrowsa=sumrowsa+a.getptrow(i);
160                 sumcolsa=sumcolsa+a.getptcol(i);
161                 sumrowsb=sumrowsb+b.getptrow(i);
162                 sumcolsb=sumcolsb+b.getptcol(i);
163             }
164             //printf("ball a pos sum of rows=%d , of cols=%d\n",sumrowsa,sumcolsa);
165             //printf("ball b pos sum of rows=%d , of cols=%d\n",sumrowsb,sumcolsb);
166             if(sumrowsa==sumrowsb && sumcolsa==sumcolsb) flagsame=1;
167
168             b.mirror_lr();//左右翻转一下
169             sumrowsb=0;
170             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相当于给出了常数。
379     linkshape shapevec[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;
390     shapenumberrec=ntotal+1;
391     int ndopart;
392     for(ndopart=0;ndopart<2000;ndopart++){

```

```

393         //printf("-----\n");
394         //printf("times of partitions:%d\n",ndopart);
395         //qDebug()<<"times of partitions:"<<ndopart;
396
397         for(i=0;i<ntotal;i++)//清除划分状态，便于再次划分
398         {
399             vecgrid[i].setshpd(0);
400         }
401
402         shapenumber=0;
403         flagnoleft=0;
404         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
405
406         while (flagnoleft==0 && ndoshapes<200)
407         {
408             ndoshapes++;
409             //printf("times of create different shape to cover all the grid :%d\n",ndoshapes);
410             coord ptpos[n_balls_ofshape]={0,0};
411             int ptnum[n_balls_ofshape]={0};
412
413
414             //确定起点
415             // k=0;
416             // i=0;
417             // while(i<ntotal)
418             // {
419             //     if(vecgrid[i].getshpd()==0){
420             //         sn_now=i;
421             //         posrow_now=0;
422             //         poscol_now=0;
423             //         vecgrid[sn_now].setshpd(1);
424             //         ptnum[k]=sn_now;
425             //         ptpos[k].row=posrow_now;
426             //         ptpos[k].col=poscol_now;
427             //         //printf("i=%d",i);
428             // i=ntotal;
429             // } else {
430             //     i++;
431             //     //printf("i=%d",i);
432             // if(i >= ntotal) flagnoleft=1;
433             //     //printf("flagnoleft=%d",flagnoleft);
434             // }
435             // }
436
437             //确定起点机制换一种,采用随机选取的方式进行
438             k=0;
439             do
440             {
441                 flagnoleft=1;//首先确定是否存在没有连接的球
442                 for(i=0;i<n_total;i++)
443                 {
444                     if(vecgrid[i].getshpd()==0) flagnoleft=0;
445                 }
446                 sn_now=rand()%n_total;
447             }

```

```

448 while(vecgrid[sn_now].getshpd()==1 && flagnoleft==0);
449 if(vecgrid[sn_now].getshpd()==0 && flagnoleft==0)
450 {
451     vecgrid[sn_now].setshpd(1);
452     coord coorda=getrowcolformsn(sn_now+1);
453     posrow_now=coorda.row;
454     poscol_now=coorda.col;
455     ptnum[k]=sn_now;
456     ptpos[k].row=posrow_now;
457     ptpos[k].col=poscol_now;
458 }
459
460 if(flagnoleft==0) {
461
462     // printf("\ninitial ball numbers:");
463     // for(j=0;j<=k;j++)
464     // {
465         // printf("%d ",ptnum[j]+1); //球真实序号比矢量中的序号大1
466     // }
467     // printf("\ninitial ball postions:");
468     // for(j=0;j<=k;j++)
469     // {
470         // printf("(%d,%d) ",ptpos[j].row,ptpos[j].col);
471     // }
472     // printf("\n");
473
474
475     int nstep=0;
476     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
477         nstep++;
478         randtemp=(rand()%100/100.0);
479         //printf("steps of moving try to create a shape:%d. ",nstep);
480         //printf("nstep=%d\n",nstep);
481         //printf("randt=%f\n",randtemp);
482
483         if(randtemp < 0.25) //右移
484         {
485             //printf("sn_now=%d\n",sn_now);
486             //printf("right of sn_now=%d\n",vecgrid[sn_now].getright()-1);
487             sn_ball=vecgrid[sn_now].getright();
488             if(sn_ball>0) {
489                 if(vecgrid[sn_ball-1].getshpd() == 0){
490                     sn_now=sn_ball-1; //矢量中的序号比球真实序号小1
491                     poscol_now++;
492                     k++;
493                     ptnum[k]=sn_now;
494                     ptpos[k].row=posrow_now;
495                     ptpos[k].col=poscol_now;
496                     vecgrid[sn_now].setshpd(1);
497                 } else { //处理往回走的情况, 但不能走到其它形状中
498                     flaginshape=0;
499                     for(j=0;j<=k;j++)
500                     {
501                         if(sn_ball-1==ptnum[j]) flaginshape=1;
502                     }

```

```

503         if(flaginshape==1) { //当移动到的球在形状内则确定移动
504             sn_now=sn_ball-1;
505             poscol_now++;
506         }
507     }
508 }
509 }
510 else if(randtemp < 0.5) //左移
511 {
512     //printf("sn_now=%d\n",sn_now);
513     //printf("left of sn_now=%d\n",vecgrid[sn_now].getleft()-1);
514     sn_ball=vecgrid[sn_now].getleft();
515     if(sn_ball>0) {
516         if(vecgrid[sn_ball-1].getshpd() == 0){
517             sn_now=sn_ball-1; //矢量中的序号比球真实序号小1
518             poscol_now--;
519             k++;
520             ptnum[k]=sn_now;
521             ptpos[k].row=posrow_now;
522             ptpos[k].col=poscol_now;
523             vecgrid[sn_now].setshpd(1);
524         } else { //处理往回走的情况,但不能走到其它形状中
525             flaginshape=0;
526             for(j=0;j<=k;j++)
527             {
528                 if(sn_ball-1==ptnum[j]) flaginshape=1;
529             }
530             if(flaginshape==1) {
531                 sn_now=sn_ball-1;
532                 poscol_now--;
533             }
534         }
535     }
536 }
537 else if(randtemp < 0.75) //上移
538 {
539     //printf("sn_now=%d\n",sn_now);
540     //printf("above of sn_now=%d\n",vecgrid[sn_now].getabove()-1);
541     sn_ball=vecgrid[sn_now].getabove();
542     if(sn_ball>0) {
543         if(vecgrid[sn_ball-1].getshpd() == 0){
544             sn_now=sn_ball-1; //矢量中的序号比球真实序号小1
545             posrow_now--;
546             k++;
547             ptnum[k]=sn_now;
548             ptpos[k].row=posrow_now;
549             ptpos[k].col=poscol_now;
550             vecgrid[sn_now].setshpd(1);
551         } else { //处理往回走的情况,但不能走到其它形状中
552             flaginshape=0;
553             for(j=0;j<=k;j++)
554             {
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);
610 // }
611 // printf("\n");
612 // 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。

```

```

613         //int tempa;
614         //std::cin>>tempa;
615
616
617         //将当前形状放到形状数组中
618         coord vectmp[k+1];
619         int vecnumber[k+1];
620         for(j=0;j<=k;j++)
621         {
622             vectmp[j].row=ptpos[j].row;
623             vectmp[j].col=ptpos[j].col;
624             vecnumber[j]=ptnum[j]+1;//输出真实的球序号，而不是从0开始的序号。
625         }
626         linkshape shapea=linkshape(shapenumber+1,k+1,vectmp,vecnumber);
627         //先判断形状是否与原先构成的相同
628         if(shapenumber==0) {
629             shapevec[shapenumber]=shapea;
630             shapenumber++;
631         }else{
632             flagsameshape=0;
633             for(j=0;j<shapenumber;j++)
634             {
635                 if(checksameshape(shapea,shapevec[j]))
636                 {
637                     flagsameshape=1;
638                     //printf("flagsameshape=%d\n",flagsameshape);
639                 }
640             }
641             if(flagsameshape==0){//和之前的形状不相同
642                 shapevec[shapenumber]=shapea;
643                 shapenumber++;
644             }else{
645                 for(j=0;j<=k;j++)
646                 {
647                     vecgrid[ptnum[j]].setshpd(0);
648                 }
649             }
650         }
651         //int tempa;
652         //std::cin>>tempa;
653     }
654 }
655
656 //输出当前次，划分的形状
657 // printf("\n");
658 // printf("do the %d th time's partition\n",ndopart);
659 // printf("number of shapes in grid is %d\n",shapenumber);
660 //qDebug()<<"total number of shapes in grid is "<<shapenumberrec;
661 // for(j=0;j<shapenumber;j++)
662 // {
663     // printf("j=%d ,shape sn: %d\n",j,shapevec[j].getshapen());
664     // 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].getshapen());
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     }
761     return flagcanbe;
762 }
763 void cleargridoccp()//游戏完成后清除所有网格的占用
764 {
765     for(int i=0;i<n_total;i++)
766     {
767         vecgrid[i].setoccp(0);
768         //qDebug()<<"sn_ball clear occp:"<<sn_ball;
769     }
770 }
771
772 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
773 {
774
775     setfocus(boxid);
776     int n_balls=veclinkshape[shapeid-1].getshapenballs();

```

```

777     int row=veclinkshape[shapeid-1].getptrow(ballid-1);
778     int col=veclinkshape[shapeid-1].getptcol(ballid-1);
779     for(int i=0;i<n_balls;i++)
780     {
781         int drow=veclinkshape[shapeid-1].getptrow(i)-row;
782         int dcol=veclinkshape[shapeid-1].getptcol(i)-col;
783         int sn_ball=getnumdrowdcola(drow,dcol);
784         vecgrid[sn_ball-1].setoccp(0);
785         //QDebug()<<"sn_ball clear occp:"<<sn_ball;
786     }
787 }
788
789 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
790 {
791
792     setfocus(boxid);
793     int n_balls=veclinkshape[shapeid-1].getshapenballs();
794     int row=veclinkshape[shapeid-1].getptrow(ballid-1);
795     int col=veclinkshape[shapeid-1].getptcol(ballid-1);
796     for(int i=0;i<n_balls;i++)
797     {
798         int drow=veclinkshape[shapeid-1].getptrow(i)-row;
799         int dcol=veclinkshape[shapeid-1].getptcol(i)-col;
800         int sn_ball=getnumdrowdcola(drow,dcol);
801         vecgrid[sn_ball-1].setoccp(1);
802     }
803 }
804
805 int gridoccupied() //判断网格是否全被占据
806 {
807     int flaggridoccp=1;
808     for(int i=0;i<n_total;i++)
809     {
810         if(vecgrid[i].getoccp()==0) {
811             qDebug()<<"box["<<i<<"]"<<vecgrid[i].getoccp();
812             flaggridoccp=0;
813         }
814     }
815     return flaggridoccp;
816 }
817
818 int getntotal()
819 {
820     return n_total;
821 }
822
823 coord getrowcolformsn(int sn_ball){//输入是球的真实序号
824     int nij=0;
825     coord a={0,0};
826     for(int i=0;i<n_scale;i++)
827     {
828         for(int j=0;j<=i;j++)
829         {
830             nij++;
831             if(nij==sn_ball)

```

```

832         {
833             a.row=i;
834             a.col=j;
835             break;
836         }
837     }
838 }
839 return a;
840 }
841
842 int getsnfromrowcol(int row,int col){//输入的行和列是以0为起点的
843     if(row<0) return -1;
844     if(col<0) return -1;
845     if(row>n_scale) return -1;
846     if(col>row) return -1;
847
848     int nij=0;
849     for(int i=0;i<row;i++)
850     {
851         for(int j=0;j<=i;j++)
852         {
853             nij++;
854         }
855     }
856     return nij+col+1;//sn_ball
857 }
858
859 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
860
861     coord a=getrowcolformsn(n_focus);//获取焦点球的行列
862     int row=a.row+drow;
863     int col=a.col+dcol;
864     return getsnfromrowcol(row,col);
865 }
866
867
868 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
869
870     //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，
871     //当drow>0则所以先走行，再走列，不会出现先越界的情况
872     //当drow<=0则先走列，再走行，则不会出现先越界的情况
873     int flagoutrange=0;
874     int sn_now=n_focus-1;//sn_now是在矢量中的序号，等于球真实序号-1
875     if(drow >0) {
876         for(int j=0;j<fabs(drow);j++)
877         {
878             if(sn_now>=0) {//注意行增加是往下走，即below
879                 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
880             }else
881             {
882                 flagoutrange=1;
883                 j=fabs(drow);
884             }
885         }
886     }

```

```

887     if(dcol <=0) {
888         for(int j=0;j<fabs(dcol);j++)
889             {
890                 if(sn_now>=0) {
891                     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
892                 }else
893                 {
894                     flagoutrange=1;
895                     j=fabs(dcol);
896                 }
897             }
898     }else{
899         for(int j=0;j<fabs(dcol);j++)
900             {
901                 if(sn_now>=0) {
902                     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
903                 }else
904                 {
905                     flagoutrange=1;
906                     j=fabs(dcol);
907                 }
908             }
909     }
910     if(drow <=0) {
911         for(int j=0;j<fabs(drow);j++)
912             {
913                 if(sn_now>=0) {
914                     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
915                 }else
916                 {
917                     flagoutrange=1;
918                     j=fabs(drow);
919                 }
920             }
921     }
922     if(flagoutrange==1){
923         return -1;
924     }else{
925         return sn_now+1;
926     }
927 }
928
929
930 void trigriddisplay(){//显示下三角网格信息函数
931     balltopo trigrid[n_scale][n_scale];
932     int nij=0;
933     for (int i=0;i<n_scale;i++){
934         for(int j=0;j<=i;j++){
935             {
936                 //矩阵和矢量的关系是
937                 trigrid[i][j]=vecgrid[nij];
938                 nij++;
939             }
940         }
941     }

```

```

942     for(int i=1; i<=n_scale; i++)//行
943     {
944         for(int j=1; j<=i; j++)//列
945         {
946             if(j==i){printf(" %5d\n",trigrd[i-1][j-1].getsn());}
947             else{printf(" %5d",trigrd[i-1][j-1].getsn());}
948         }
949     }
950
951     // for(int i=0;i<n_total;i++)
952     // {
953         // printf("%5d ",vecgrid[i].getsn());
954     // }
955     // printf("\n");
956
957 }
958
959 void trigrdtest(){//查看当前球及其邻居序号测试
960     balltopo trigrd[n_scale][n_scale];
961     int nij=0;
962     for (int i=0;i<n_scale;i++)
963     {
964         for(int j=0;j<=i;j++)
965         {
966             trigrd[i][j]=vecgrid[nij];
967             //printf("%d sn=%d\n",nij,vecgrid[nij].getsn());
968             nij++;
969         }
970     }
971     int n=n_scale;
972
973     for(int i=1; i<=n; i++)//行
974     {
975         for(int j=1; j<=i; j++)//列
976         {
977             trigriddisplay();
978             int topoget[5]={0};
979             trigrd[i-1][j-1].gettopo(topoget);
980             printf("%5c%5d%5c\n",' ',topoget[3],' ');
981             printf("%5d%5d%5d\n",topoget[1],topoget[0],topoget[2]);
982             printf("%5c%5d%5c\n",' ',topoget[4],' ');
983
984             //char ch;
985             //printf("please put in a key to continue:");
986             //ch = getchar();
987         }
988     }
989 }
990
991 };
992
993
994 #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 axiscoord{//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 axiscoord 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
15 {
16 private:
17     int sn_shape; //<当前链接形状序号,从1开始计数,这是真实的数字
18     int n_balls_inshape; //<形状的链接的球数
19
20     std::vector<int> vec_balls_number; //球的序号
21     int rows,cols; //<链接形状当前的总行数和总列数
22     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
23     std::vector<coord> vec_coord_shape; //位置坐标矢量与形状矩阵相关
24     std::vector<int> vec_matrix_adjacent; //邻接矩阵矢量
25     std::vector<float> vec_eigen_matrix; //邻接矩阵特征值列表
26
27     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
28     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
29     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
30
31 public:
32
33     //默认的构造函数,如果要形成一个数组,是必须的。
34     linkshape()
35     {}
36
37     //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
38     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
39     {
40         printf("\n");
41         printf("construct a new shape \n");
42         sn_shape=sn;
43         n_balls_inshape=n_balls;
44         int i=0;
45         int j=0;
46
47         //对位置坐标矢量做从0开始的规范化
48         int rowmin=100;
49         int rowmax=-100;
50         int colmin=100;
51         int colmax=-100;
52         for(i=0;i<n_balls;i++){
53             if(coordmatrix[i].row < rowmin) rowmin=coordmatrix[i].row;
54             if(coordmatrix[i].row > rowmax) rowmax=coordmatrix[i].row;
55             if(coordmatrix[i].col < colmin) colmin=coordmatrix[i].col;
56             if(coordmatrix[i].col > colmax) colmax=coordmatrix[i].col;
57         }
58         const int jm=colmax-colmin; //使用了const后才使得matrix_shape定义正确
59         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
60         rows=im; //为方便起见总行数也从0开始计数

```

```

61         cols=jm;
62         rowofzero=rowmin;
63         colofzero=colmin;
64         for(i=0;i<n_balls;i++){
65             coordmatrix[i].row=coordmatrix[i].row-rowmin;
66             coordmatrix[i].col=coordmatrix[i].col-colmin;
67         }
68         printf("position of balls: \n");
69         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)
84                        ==1) || (coordmatrix[i].col==coordmatrix[j].col && abs(coordmatrix[i].row-
85                        coordmatrix[j].row)==1) ) {
86                         matrix_adjacent[i][j]=1;
87                     }else{
88                         matrix_adjacent[i][j]=0;
89                     }
90                 }
91             }
92         }
93
94         printf("adjacent matrix of shape: \n");
95         for(i=0;i<n_balls;i++){
96             for(j=0;j<n_balls;j++){
97                 printf("%5d%5c",matrix_adjacent[i][j], ' ');
98             }
99             printf("\n");
100         }
101
102         //由邻接矩阵得到其特征值列表
103         double eigen_matrix[n_balls];
104         double eps=0.000001;
105         double mtemp[n_balls][n_balls];
106         double vtemp[n_balls][n_balls];
107         for(i=0;i<n_balls;i++){
108             for(j=0;j<n_balls;j++){
109                 mtemp[i][j]=matrix_adjacent[i][j];
110             }
111         }
112         double *vpt=&vtemp[0][0];
113         double *mpt=&mtemp[0][0];
114
115         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
116         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
134 号
135 {
136     //printf("\n");
137     //printf("construct a new shape \n");
138     sn_shape=sn;
139     n_balls_inshape=n_balls;
140
141     int i=0;
142     int j=0;
143
144     for(i=0;i<n_balls;i++){
145         vec_balls_number.push_back(ballnumbers[i]);
146         vec_coord_origin.push_back(coordmatrix[i]);
147     }
148
149     //对位置坐标矢量做从0开始的规范化
150     int rowmin=100;
151     int rowmax=-100;
152     int colmin=100;
153     int colmax=-100;
154     for(i=0;i<n_balls;i++){
155         if(coordmatrix[i].row < rowmin) rowmin=coordmatrix[i].row;
156         if(coordmatrix[i].row > rowmax) rowmax=coordmatrix[i].row;
157         if(coordmatrix[i].col < colmin) colmin=coordmatrix[i].col;
158         if(coordmatrix[i].col > colmax) colmax=coordmatrix[i].col;
159     }
160     const int jm=colmax-colmin;//使用了const后才使得matrix_shape定义正确
161     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
162     rows=im;//为方便起见总行数也从0开始计数
163     cols=jm;
164     rowofzero=rowmin;
165     colofzero=colmin;
166     for(i=0;i<n_balls;i++){
167         coordmatrix[i].row=coordmatrix[i].row-rowmin;
168         coordmatrix[i].col=coordmatrix[i].col-colmin;

```

```

168     }
169     //printf("position 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)
185                     ==1) || (coordmatrix[i].col==coordmatrix[j].col && abs(coordmatrix[i].row-
186                         coordmatrix[j].row)==1) ) {
187                     matrix_adjacent[i][j]=1;
188                 }else{
189                     matrix_adjacent[i][j]=0;
190                 }
191             }
192         }
193     }
194
195     // printf("adjacent matrix of shape: \n");
196     // for(i=0;i<n_balls;i++){
197     //     for(j=0;j<n_balls;j++){
198     //         // printf("%5d%5c",matrix_adjacent[i][j],' ');
199     //     }
200     //     // printf("\n");
201     // }
202
203     //由邻接矩阵得到其特征值列表
204     double eigen_matrix[n_balls];
205     double eps=0.000001;
206     double mtemp[n_balls][n_balls];
207     double vtemp[n_balls][n_balls];
208     for(i=0;i<n_balls;i++){
209         for(j=0;j<n_balls;j++){
210             mtemp[i][j]=matrix_adjacent[i][j];
211         }
212     }
213     double *vpt=&vtemp[0][0];
214     double *mpt=&mtemp[0][0];
215
216     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
217     if(flag>0){
218         //printf("eigen elements=");
219         for (i=0; i<n_balls; i++) {
220             eigen_matrix[i]=mtemp[i][i];
221             //printf("%13.7e ",mtemp[i][i]);
222         }
223     }

```

```

221         //printf("\neigen in order=");
222         maopao(eigen_matrix, n_balls);
223         for (i=0; i<n_balls; i++) {
224             //printf("%13.7e ",eigen_matrix[i]);
225             vec_eigen_matrix.push_back(eigen_matrix[i]);
226         }
227         //printf("\n");
228
229
230     }
231
232     void saveoldcoord()//记录放置在网格外的坐标
233     {
234         vec_coord_save=vec_coord_shape;
235     }
236
237     void resetoldcoord()//记录放置在网格外的坐标
238     {
239         vec_coord_shape=vec_coord_save;
240     }
241
242     void recordoutercoord()//记录放置在网格外的坐标
243     {
244         vec_coord_outer=vec_coord_shape;
245     }
246
247     void resetoutercoord()//恢复放置在网格外的坐标
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 coord getptcoord(const int i)//给出球在形状中的坐标
332 {
333     return vec_coord_shape[i];
334 }
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 cjbci(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 (l==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))
450             { fm=d; p=i; q=j;}
451         }
452       if (fm<eps) return(1);
453       if (l>jt) return(-1);
454       l=l+1;
455       u=p*n+q; w=p*n+p; t=q*n+p; s=q*n+q;
456       x=-a[u]; y=(a[s]-a[w])/2.0;
457       omega=x/sqrt(x*x+y*y);
458       if (y<0.0) omega=-omega;
459       sn=1.0+sqrt(1.0-omega*omega);
460       sn=omega/sqrt(2.0*sn);
461       cn=sqrt(1.0-sn*sn);
462       fm=a[w];
463       a[w]=fm*cn*cn+a[s]*sn*sn+a[u]*omega;
464       a[s]=fm*sn*sn+a[s]*cn*cn-a[u]*omega;
465       a[u]=0.0; a[t]=0.0;
466       for (j=0; j<=n-1; j++)
467       if ((j!=p)&&(j!=q))
468         { u=p*n+j; w=q*n+j;
469           fm=a[u];
470           a[u]=fm*cn+a[w]*sn;
471           a[w]=-fm*sn+a[w]*cn;
472         }
473       for (i=0; i<=n-1; i++)
474       if ((i!=p)&&(i!=q))
475         { u=i*n+p; w=i*n+q;
476           fm=a[u];
477           a[u]=fm*cn+a[w]*sn;
478           a[w]=-fm*sn+a[w]*cn;
479         }
480       for (i=0; i<=n-1; i++)
481       { u=i*n+p; w=i*n+q;
482         fm=v[u];
483         v[u]=fm*cn+v[w]*sn;
484         v[w]=-fm*sn+v[w]*cn;
485       }
486     }
487     return(1);
488 }
489
490 /*冒泡算法, 对数组进行排序, 从大到小排序
491 输入:数组stu
492 数组长度n
493 */
494 void maopao(double stu[], int n)
495 {

```

```

496     int i,j,k,s;
497     double temp1,temp2;
498     for(j=1;j<=n-1;j++){//遍历次数是n-1次
499         if(fmod(j,2) != 0) { //j为奇数次, 从前向后遍历
500             k=(j+1)/2;
501             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
502                 {
503                     s=i-1;
504                     if( stu[s] > stu[s-1]){ //从大到小排序, 若要从小到大则应改为<号
505                         temp1=stu[s];
506                         stu[s]=stu[s-1];
507                         stu[s-1]=temp1;
508                     }
509                 }}
510             else
511                 { //j为偶数次, 从后向前遍历
512                     k=j/2;
513                     for(i=n-k;i>=k+1;i--){ //注意这里i>=k+1, 因为i是递减的
514                         {
515                             s=i-1;
516                             if( stu[s] > stu[s-1]){ //从大到小排序, 若要从小到大则应改为<号
517                                 temp2=stu[s];
518                                 stu[s]=stu[s-1];
519                                 stu[s-1]=temp2;
520                             }
521                         }}
522                     //printf("\n");
523                     //for(i=0;i<n;i++) printf("%13.7e ",stu[i]);
524                 }
525     }
526 };
527
528
529
530 #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

```

1  #ifndef SCENEINFOTRANS_H
2  #define SCENEINFOTRANS_H
3
4  #include<QGraphicsScene>
5
6  class sceneinfotrans : public QGraphicsScene

```

```

7 {
8 public:
9     sceneinfotrans();
10
11 public slots:
12     void scenetobeset(bool a);
13
14 signals:
15     void toshowgrid(sceneinfotrans &b);
16
17 };
18
19 #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

```

1  // #include "mainwindow.h"
2  #include "ballitem.h"
3  #include "ballshape.h"
4  #include "ballgrid.h"
5  #include "sceneinfotrans.h"
6  #include <QApplication>
7  #include <QGraphicsScene>
8  #include <QGraphicsView>
9  #include <QGraphicsWidget>
10 #include <QGraphicsProxyWidget>
11 #include <QPushButton>
12 #include <QLabel>
13 #include <QLineEdit>
14 #include <QGraphicsGridLayout>
15 #include <QGraphicsLinearLayout>
16 #include <qDebug>
17 #include <QTimer>
18
19 int main(int argc, char *argv[])
20 {
21     QApplication a(argc, argv);
22
23     //MainWindow w;
24     //w.show();
25
26     sceneinfotrans scene;
27
28     QPushButton *buttoncredit=new QPushButton(QObject::tr("历史排行榜"));
29     QPushButton *buttonstart=new QPushButton(QObject::tr("开始"));
30     QPushButton *buttonexample=new QPushButton(QObject::tr("示例"));
31
32     //buttonstart->setFixedSize(50, 30);
33
34     QPushButton *buttonset=new QPushButton(QObject::tr("重新设置"));

```

```

35     QPushButton *buttonpart=new QPushButton(QObject::tr("重新划分"));
36     QLabel * labelgridnum=new QLabel(QObject::tr("网格大小"));
37     QLabel * labellinknum=new QLabel(QObject::tr("链接球数"));
38     QLineEdit * textgridnum=new QLineEdit("10");
39     textgridnum->setMaximumWidth(100);//设置最大尺度长度为50像素
40     QLineEdit * textlinknum=new QLineEdit("5");
41     textlinknum->setMaximumWidth(100);
42     //textlinknum->setMaximumHeight(30);
43
44     QLabel * label_nshapes=new QLabel(QObject::tr("形状数量"));
45     QLabel * labelediameter=new QLabel(QObject::tr("球的尺寸"));
46     QLineEdit * text_nshapes=new QLineEdit("12");
47     text_nshapes->setMaximumWidth(100);//设置最大尺度长度为50像素
48     QLineEdit * textdiameter=new QLineEdit("70");
49     textdiameter->setMaximumWidth(100);
50
51     QPushButton *turnright=new QPushButton(QObject::tr("向右旋转"));
52     QPushButton *turnleft=new QPushButton(QObject::tr("向左旋转"));
53     QPushButton *mirrorab=new QPushButton(QObject::tr("上下翻转"));
54     QPushButton *mirrorlr=new QPushButton(QObject::tr("左右翻转"));
55     QPushButton *resetpos=new QPushButton(QObject::tr("移出网格"));
56
57     QLabel * labeltime=new QLabel(QObject::tr("游戏时间:"));
58     QLineEdit * texttime=new QLineEdit("0");
59     texttime->setMaximumWidth(100);//设置最大尺度长度为50像素
60
61     buttonstart->setMaximumHeight(50);
62     buttoncredit->setMaximumHeight(50);
63     buttonexample->setMaximumHeight(50);
64     labelgridnum->setMaximumHeight(50);
65     labellinknum->setMaximumHeight(50);
66     buttonset->setMaximumHeight(50);
67     buttonpart->setMaximumHeight(50);
68     label_nshapes->setMaximumHeight(50);
69     labelediameter->setMaximumHeight(50);
70     text_nshapes->setMaximumHeight(50);
71     textdiameter->setMaximumHeight(50);
72     turnright->setMaximumHeight(50);
73     mirrorab->setMaximumHeight(50);
74     mirrorlr->setMaximumHeight(50);
75     turnleft->setMaximumHeight(50);
76     resetpos->setMaximumHeight(50);
77
78     QGraphicsWidget *button9=scene.addWidget(buttoncredit);
79     QGraphicsWidget *button10=scene.addWidget(mirrorlr);
80     QGraphicsWidget *button1=scene.addWidget(buttonstart);//用代理的方式将部件加入场景中
81     QGraphicsWidget *button2=scene.addWidget(buttonexample);
82     QGraphicsWidget *button3=scene.addWidget(buttonset);
83     QGraphicsWidget *button4=scene.addWidget(buttonpart);
84
85     QGraphicsWidget *button5=scene.addWidget(turnright);//用代理的方式将部件加入场景中
86     QGraphicsWidget *button6=scene.addWidget(turnleft);
87     QGraphicsWidget *button7=scene.addWidget(mirrorab);
88     QGraphicsWidget *button8=scene.addWidget(resetpos);
89

```

```

90     QGraphicsWidget *label1=scene.addWidget(labelgridnum);
91     QGraphicsWidget *label2=scene.addWidget(labellinknum);
92     QGraphicsWidget *text1=scene.addWidget(textgridnum);
93     QGraphicsWidget *text2=scene.addWidget(textlinknum);
94
95     QGraphicsWidget *label3=scene.addWidget(label_nshapes);
96     QGraphicsWidget *label4=scene.addWidget(labeldiameter);
97     QGraphicsWidget *text3=scene.addWidget(text_nshapes);
98     QGraphicsWidget *text4=scene.addWidget(textdiameter);
99
100    QGraphicsWidget *label5=scene.addWidget(labeltime);
101    QGraphicsWidget *text5=scene.addWidget(texttime);
102
103    QGraphicsGridLayout *layout = new QGraphicsGridLayout;
104
105    QGraphicsLinearLayout *left=new QGraphicsLinearLayout;
106    left->addItem(label1);
107    left->addItem(text1);
108
109    QGraphicsLinearLayout *left0=new QGraphicsLinearLayout;
110    left0->addItem(label5);
111    left0->addItem(text5);
112
113    QGraphicsLinearLayout *right=new QGraphicsLinearLayout;
114    right->addItem(label2);
115    right->addItem(text2);
116
117    QGraphicsLinearLayout *left1=new QGraphicsLinearLayout;
118    left1->addItem(label3);
119    left1->addItem(text3);
120
121    QGraphicsLinearLayout *right1=new QGraphicsLinearLayout;
122    right1->addItem(label4);
123    right1->addItem(text4);
124
125    layout->addItem(left0, 0, 0);
126    layout->addItem(button9, 0, 1);
127    layout->addItem(button1, 1, 0);
128    layout->addItem(button2, 1, 1);
129    layout->addItem(left, 2, 0);
130    layout->addItem(right, 2, 1);
131    layout->addItem(button3, 3, 1);
132    layout->addItem(button4, 3, 0);
133    layout->addItem(left1, 4, 0);
134    layout->addItem(right1, 4, 1);
135    layout->addItem(button8, 5, 0);
136    layout->addItem(button5, 5, 1);
137    layout->addItem(button6, 6, 1);
138    layout->addItem(button7, 7, 1);
139    layout->addItem(button10, 8, 1);
140
141    //layout->setMaximumWidth(200);//没有效果, 估计策略是按钮自身是第一优先级
142    //layout->setMaximumHeight(200);
143    //layout->setColumnMaximumWidth(1,50);
144    //layout->setRowMaximumHeight(1,30);

```



```

145
146   QGraphicsWidget *form = new QGraphicsWidget;
147   form->setLayout(layout);
148   scene.addItem(form);
149   form->setPos(textdiameter->text().toInt()*4,-10);
150
151   ballgrid gridballs;
152
153   QObject::connect(buttoncredit,&QPushButton::clicked,&gridballs,&ballgrid::showcredit);
154   QObject::connect(&gridballs,&ballgrid::shapecompleted,text__nshapes,&QLineEdit::setText);
155   QObject::connect(&gridballs,&ballgrid::timerecorded,txttime,&QLineEdit::setText);
156   QObject::connect(buttonstart,&QPushButton::clicked,&gridballs,&ballgrid::gamestart);
157   QObject::connect(buttonexample,&QPushButton::clicked,&gridballs,&ballgrid::gameexample);
158   QObject::connect(textgridnum,&QLineEdit::textChanged,&gridballs,&ballgrid::dealgridnumtext);
159   QObject::connect(textlinknum,&QLineEdit::textChanged,&gridballs,&ballgrid::deallinknumtext);
160   QObject::connect(textdiameter,&QLineEdit::textChanged,&gridballs,&ballgrid::setballdiameter);
161   QObject::connect(buttonset,&QPushButton::clicked,&gridballs,&ballgrid::gameset);
162   QObject::connect(buttonpart,&QPushButton::clicked,&gridballs,&ballgrid::gamerepart);
163
164   QObject::connect(turnright,&QPushButton::clicked,&gridballs,&ballgrid::shapeturnright);
165   QObject::connect(turnleft,&QPushButton::clicked,&gridballs,&ballgrid::shapeturnleft);
166   QObject::connect(mirrorab,&QPushButton::clicked,&gridballs,&ballgrid::shapemirrorab);
167   QObject::connect(mirrorlr,&QPushButton::clicked,&gridballs,&ballgrid::shapemirrorlr);
168   QObject::connect(resetpos,&QPushButton::clicked,&gridballs,&ballgrid::shaperesetpos);
169
170   if(gridballs.getflag()==0 && gridballs.getgridnum()==10)
171   {
172       gridballs.setgrid(gridballs.getgridnum(),gridballs.getlinknum());
173       gridballs.setboxes();
174       gridballs.setshapedefault(gridballs.getnttotal());
175   }
176
177   gridballs.setworkscene(scene);
178   gridballs.addboxestoscene();
179   gridballs.addtoscenedefault();
180
181   QGraphicsView view(&scene);
182   view.resize(750,650);
183   view.show();
184
185   return a.exec();
186 }

```

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 ctrl+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 creator 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。

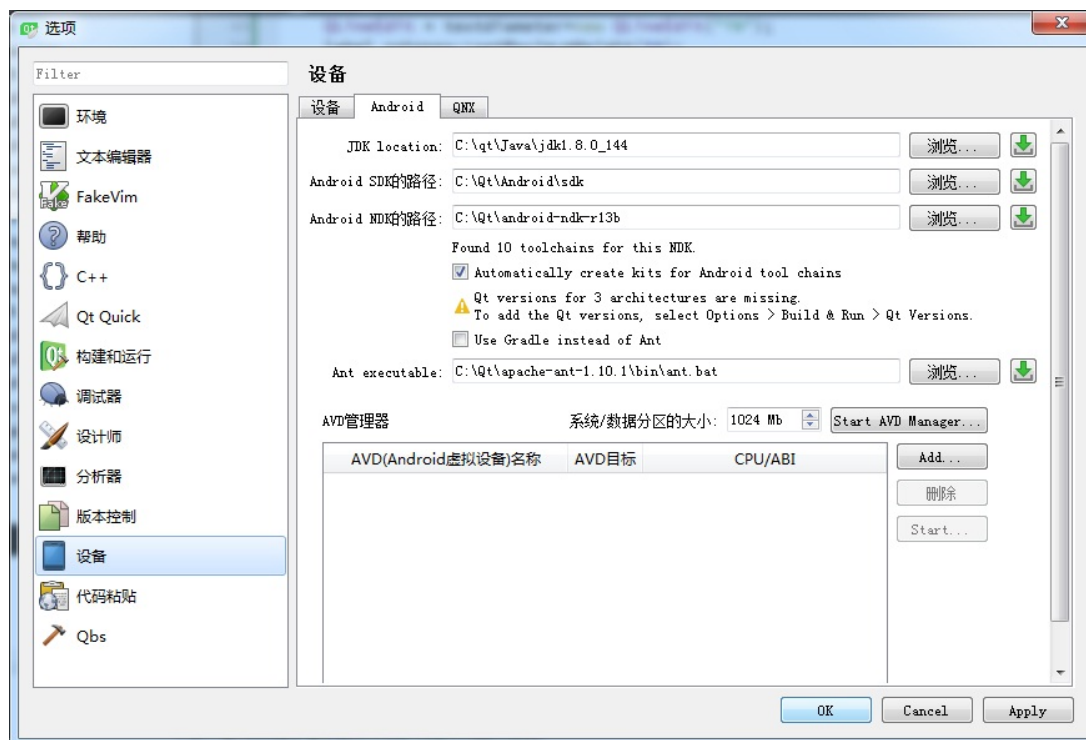


图 9 编译 android 程序工具链

然后设置工程选项, 编译工具集, 如图10所示。

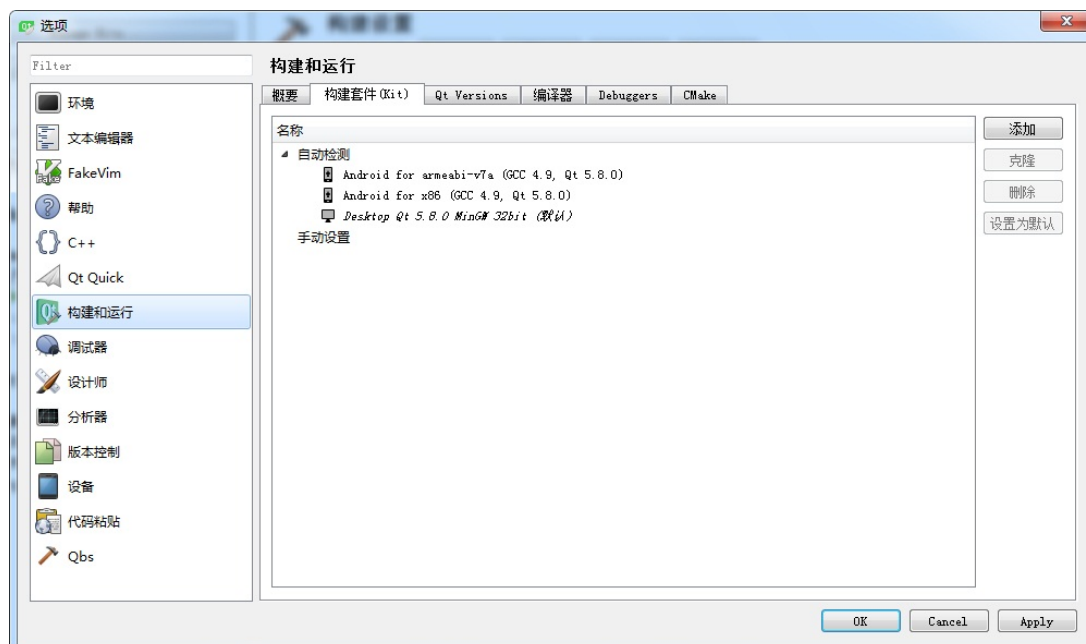


图 10 工程设置选择 android 工具链

最后, 按 `ctrl+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 android 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, 需要调整的代码包括:

3.2.1 球的默认大小

因为 android 下的屏幕的分辨率与 win 下的不同, 因此需要特别的调整, 比如:

示例 20 球的默认大小

```
代码
1 QLineEdit * textdiameter=new QLineEdit("70");
2 textdiameter->setMaximumWidth(100);
```

这主要在 main.cpp 文件中修改。

3.2.2 操作界面上按钮的高度

因为同样的原因, 按钮的大小也需要设置调整, 比如:

示例 21 按钮大小

```
代码
1 buttonstart->setMaximumHeight(50);
2 buttoncredit->setMaximumHeight(50);
3 buttonexample->setMaximumHeight(50);
4 labelgridnum->setMaximumHeight(50);
5 labellinknum->setMaximumHeight(50);
```

这主要在 main.cpp 文件中修改。

3.2.3 形状在网格外的排布位置

因为手机屏幕是长方形, 因此当形状在网格外时, 放置的位置还需要进行调整, 不同于 win 下放在网格的一周, android 下主要放在网格的上方, 多排放置。比如:

示例 22 形状排布

```
代码
1 void ballgrid::setshaperandposdefault()//将划分形状放到网格外
2 {
3     int colnow=0;
4     int rownow=-5;
5     for(int i=0;i<n_ofshapes;i++)
6     {
7         //全部在顶部竖放
8         if(colnow > gridnum) {
9             colnow=0;
10            rownow=-5;
11        }
12
13        int rows=veclinkshape[i].getrows();
14        int cols=veclinkshape[i].getcols();
15        if(cols > rows )
16        {
17            veclinkshape[i].rotate_right();
18            rows=veclinkshape[i].getrows();
19            cols=veclinkshape[i].getcols();
20        }
21
22        for(int j=0;j<veclinkshape[i].getshapenballs();j++)
23        {
24            int row=rownow+(veclinkshape[i].getptrow(j));
25            int col=colnow+(veclinkshape[i].getptcol(j));
26
27            ballshapingriddefault[i]->setballpos(j,col*balldiameter,row*balldiameter);
```

```

28     }
29     colnow=colnow+cols+1;
30
31     /* 一排竖放，然后一列横放
32     if(colnow < gridnum+2 ) { //一排排完前 //一半竖着放 i < n_ofshapes/2
33         int rows=veclinkshape[i].getrows();
34         int cols=veclinkshape[i].getcols();
35         if(cols > rows )
36         {
37             veclinkshape[i].rotate_right();
38             rows=veclinkshape[i].getrows();
39             cols=veclinkshape[i].getcols();
40         }
41
42         for(int j=0;j<veclinkshape[i].getshapenballs();j++)
43         {
44             int row=(veclinkshape[i].getptrow(j))-rows-1;
45             int col=colnow+(veclinkshape[i].getptcol(j));
46
47             ballshapelsingriddefault[i]->setballpos(j,col*balldiameter,row*balldiameter);
48         }
49         colnow=colnow+cols+1;
50     }
51     else // 另一半横着放
52     {
53         int rows=veclinkshape[i].getrows();
54         int cols=veclinkshape[i].getcols();
55         if(cols < rows )
56         {
57             veclinkshape[i].rotate_right();
58             rows=veclinkshape[i].getrows();
59             cols=veclinkshape[i].getcols();
60         }
61         for(int j=0;j<veclinkshape[i].getshapenballs();j++)
62         {
63             int row=rownow+(veclinkshape[i].getptrow(j));
64             int col=colnow+(veclinkshape[i].getptcol(j));
65
66             ballshapelsingriddefault[i]->setballpos(j,col*balldiameter,row*balldiameter);
67         }
68         rownow=rownow+rows+1;
69     }
70     */
71
72     ballshapelsingriddefault[i]->recordpos();
73     veclinkshape[i].recordoutercoord();
74 }
75 }

```

这主要在 ballgrid.cpp 文件中修改。

3.2.4 排行榜的控制

关于排行榜的控制，在 android 下，所有的控件都能显示，但就是排行榜的内容没有，可能的原因是在 info.xml 文件没有读取出来。将 win 下原来设定在 exe 目录下的 info 文件改为用 assets

的方式存放后, 仍然没有得到正确的结果。

可能还是 qt for android 配置还不正确, 这个问题可以通过建个小的测试程序的方式进行测试, 还有一个重要问题是, 就是 qt for android 的调试, 以及 assets 的文件机制, 目前还不是很清楚, 无法确认 xml 文件的位置, 及其读写是否正确 (20171028)。

限于时间, 而且没有进一步开发的动力, 所以这种核心算法外的东西, 暂时无法再投入时间去研究了。因此, 排行榜功能暂时无效了。

4 游戏操作

最终构成的游戏界面如图11所示。

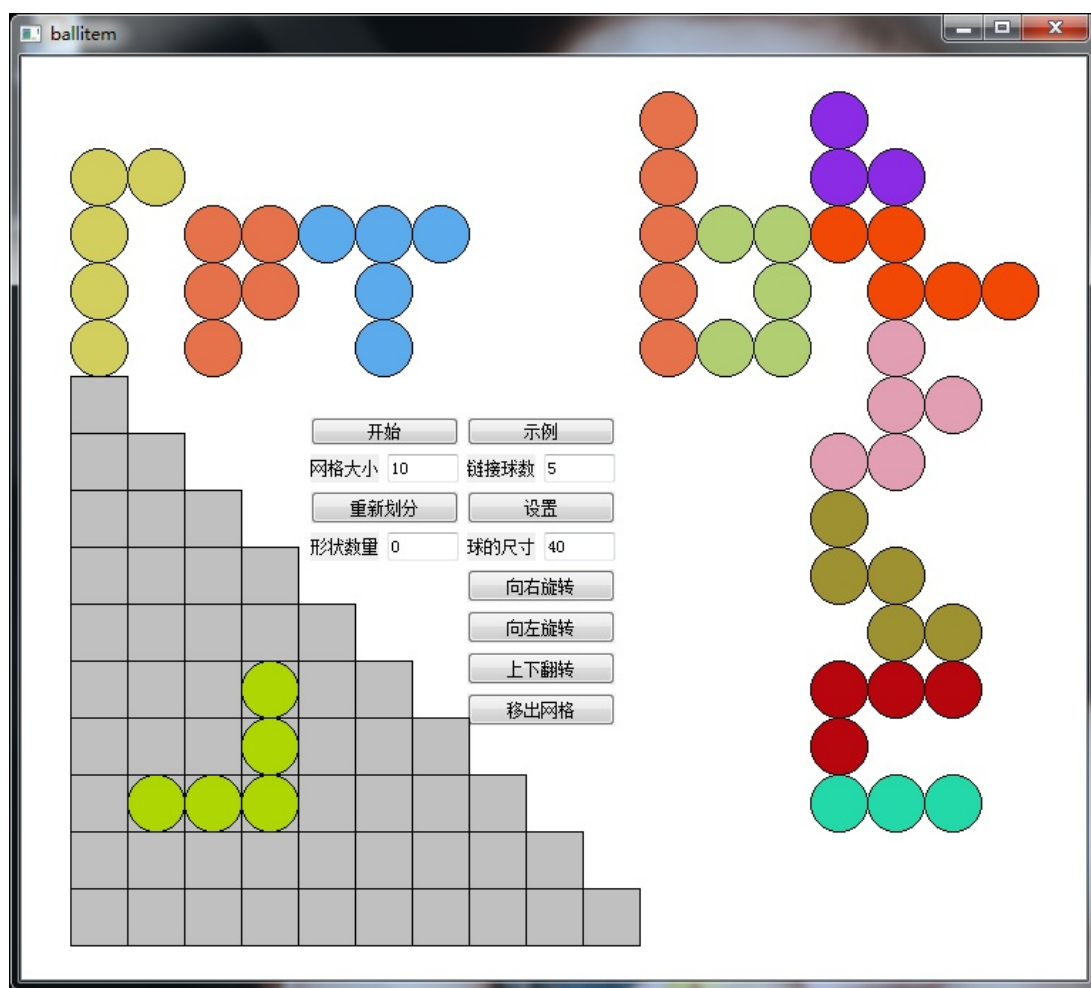


图 11 游戏界面

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，形状向左旋转 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 android 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!

ps: 遗憾的是，限于投入时间有限，android 下的排行榜功能可能因为文件的原因，暂时没有实现。同时需要注意，在做开发的时候，还是需要慢点把能注释的内容都要注释上，否则一旦长时间无法投入，再来看代码都会觉得困难，所以事情还是需要一件一件的做好，而不应求快，要为后续考虑。

目 录

0 引言	1
1 开发环境准备	3
1.1 C++ 环境	3
1.2 qt 环境	5
1.2.1 命令行方法	5
1.2.2 集成开发方法	6
1.3 qt/c++ 程序开发的理解要点	9
2 程序设计	9
2.1 boxitem 类	9
2.1.1 功能分析	9
2.1.2 代码实现	10
2.2 ballitem 类	11
2.2.1 功能分析	11
2.2.2 代码实现	12
2.3 ballshape 类	14
2.3.1 功能分析	14
2.3.2 代码实现	16
2.4 ballgrid 类	18
2.4.1 功能分析	18
2.4.2 代码实现	20
2.5 grid 类	41
2.5.1 功能分析	41
2.5.2 代码实现	42
2.6 balltopo 类	60
2.6.1 功能分析	60
2.6.2 代码实现	60
2.7 linkshape 类	62
2.7.1 功能分析	62
2.7.2 代码实现	63
2.8 sceneinfoTrans 类	73
2.9 主程序	73

3 编译为 window 程序和 andriod 的 apk 软件	77
3.1 编译为 window 程序	77
3.2 编译为 andriod 的 apk 软件	77
3.2.1 球的默认大小	78
3.2.2 形状在网格外的排布位置	78
3.2.3 操作界面上按钮的高度	78
3.2.4 排行榜的控制	78
4 游戏操作	78
4.1 游戏全局控制	79
4.2 形状选择和落位	79
4.3 形状的旋转翻转复位	79
5 结论	80
目录	81
插图	82
示例	82

插 图

1 游戏基本构想	1
2 选择 qt widget application	6
3 命名工程选择位置	7
4 选择编译工具	7
5 选择主界面基类	7
6 编辑 ui 界面添加信号槽	8
7 编译构建	8
8 运行结果	9
9 编译 andriod 程序工具链	77
10 工程设置选择 android 工具链	78
11 游戏界面	79

示 例

1 简单编译脚本	3
2 gcp 编译脚本	3
3 简单 qt 程序	5
4 boxitem 类声明	10
5 boxitem 类定义	10

6	ballitem 类声明	11
7	ballitem 类定义	12
8	ballshape 类声明	15
9	ballshape 类定义	16
10	ballgrid 类声明	18
11	ballgrid 类定义	20
12	grid 类的保护属性	41
13	grid 类定义	42
14	balltopo 类的属性	60
15	balltopo 类定义	60
16	linkshape 类的属性	62
17	linkshape 类定义	63
18	sceneinfotrans 类声明	73
19	主程序 main	73