数独游戏开发文档

计 65

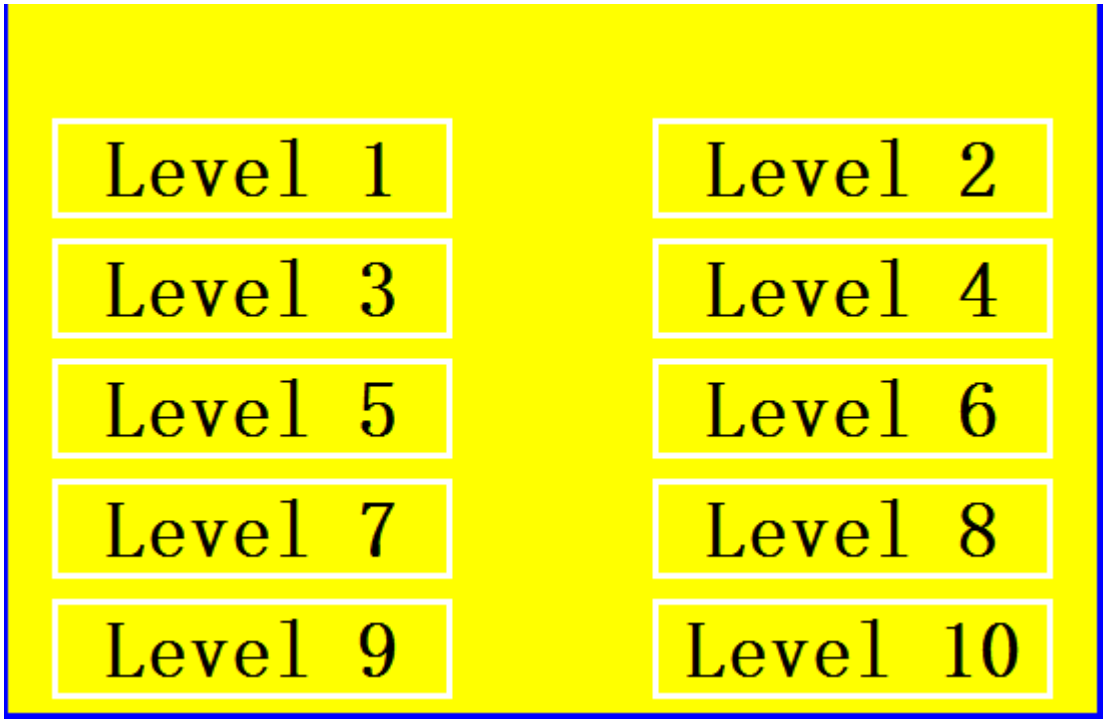
李文凯

2016011369

目录

一、页面总览

1.关卡选择页面：



2. 游戏进行页面：


3 2 4 8 6 9 1
5 9 1 7 2 4 8
6 9 5 2 7
4 6 1 5 7 9 3
1 2 5 7 3 9 8 4
9 6 4 2 5
1 7 2 3 8 4 6
2 3 6 4 1 9
6 4 1 9 2
TIME:
P/C
00: 04
Win?
Ans
Menu
<<
>>
DEL
R
1 2 3
4 5 6
7 8 9

二、设计思路：

1.游戏界面部分：

游戏界面主窗口为 Widget 类对象，共设有七个主要模块:

ui->widget: 放置数独宫格

ui->widget_2: 放置数字键盘

ui->widget_3: 放置功能按钮，从左到右依次为后退，前进，清空，重玩

ui->widget_4: 放置功能按钮，从左到右依次为提交，答案，菜单

timetable: 计时器，包括显示器和暂停/重启计时的按钮

infoBoard: 信息栏，显示当前是否完成数独表格的填充

welcomeWIndow: 欢迎界面，选择关卡

以下逐一进行描述。

ui->widget: 放置数独宫格

使用类：my_button 继承自 QPushButton，在原有控件基础上设置新的函数功能
包含成员：81 个 my_button 类的成员

my_button 类：

```
protected:
    virtual void enterEvent(QEvent* event);//鼠标经过操作
    virtual void leaveEvent(QEvent* event);//鼠标离开操作
    void focusInEvent(QFocusEvent* e);//鼠标点击，焦点选中操作
    void focusOutEvent(QFocusEvent *e);//焦点转移操作
public slots:
    void setText_wrapper(QString);//修改按钮文本的装饰函数
    void highlight();//高亮操作，包含数字高亮以及对应行列选中高亮
    void getGeo(int,int,QString);//后退，前进按钮对应槽函数，前两个int记录位置，将对应位置的按钮内容置为QString
    void reSetBtn();//重置按钮操作
    void delBtn();//清空按钮中的多个数据
signals:
     void textChanged();
     void sendAction(int,int,QString);
private:
    static int xpos,ypos;//记录焦点框所在位置
    static QString cur_text;//记录当前焦点框内按钮文本
```

ui->widget_2：放置数字键盘

使用类： QPushButton
包含成员： 9 个 QPushButton

ui->widget_3：放置功能按钮，从左到右依次为后退，前进，清空，重玩

使用类： QPUshButton
包含成员： 4 个 QPushButton 实例

ui->widget_4：放置功能按钮，从左到右依次为提交，答案，菜单

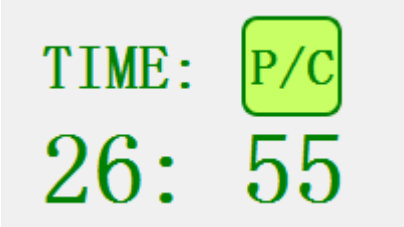
使用类： QPushButton

包含成员： 3 个 QPushButton 实例

timetable：计时器，包括显示器和暂停/重启计时的按钮

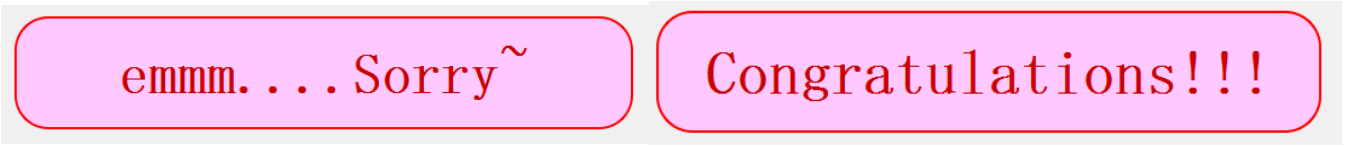
使用类： QPushButton，QLabel

包含成员： 1 个 QPushButton 实例，4 个 QLabel 实例



infoBoard：信息栏，显示当前是否完成数独表格的填充

使用类：QPUshButton
包含成员： 1 个 QPUshButton 实例



welcomeWIndow：欢迎界面，选择关卡

使用类：QPUshButton
包含成员：10 个 QPUshButton 实例

游戏界面所使用其他类：

QString： 使用其中字符串处理函数 left 等
QStringList： 使用其中字符串处理函数 split 等
QSighnalMapper：信号映射，用于数字键盘与关卡选择与对应槽函数连接

2. 数独逻辑部分：

数独逻辑部分分为 Generator 和 solver

Solver：

方法：回溯

算法思路：

读入矩阵后，进行九宫格检查，SudokuGame 类中的 bool blockCheck(int block_id, int num)函数能够在第 id 个九宫格中查找是否含有数字 num，如果没有则返回 true，将 block_id 和 num 压入 vector 存储，组成需要填写的序列，在利用回溯法即可方便解决。

整个 solve 过程分为三个部分：

第一部分：获取当前数独矩阵

第二部分：通过 blockCheck(int,int)函数构建操作序列 vector

第三部分：对 vector 进行回溯处理

将这种做法与逐一格子遍历判断做一个对比。如果暴力回溯，每个格子尝试 9 种，每填一格都需要对其行列以及所在九宫格进行检查，结合数字查重，整个工序非常繁冗。

所以在这里采取一种思想，从 1 到 9，一个数字一个数字的填，这样做一方面规避了数字的查重，另一方面，填补的对象是每一个九宫格，只需要对行列进行检查即可，体现一种分治的思想。最后构建的 vector 最多也只有 64 组数据，每组数据记录需要在第几个九宫格填入几，回溯空间大大缩小，极为便利。

Generator：

方法：生成随机终盘，之后设定挖洞数目随机挖洞

思路概述：

生成随机终盘和解数独有很多相似性，如果暴力填会有很大的回溯空间，耗时也很多，同样使用分治的思想，每次对每个九宫格填入一个数字，所以对 solver 进行改造就可以变成终盘生成器，和 solver 相比，不同进行九宫格检查，不同构造工序 vector，只需要每次打乱 1 到 9 个数，直接回溯即可。