Flow Free 游戏设计文档

前言

此为 Qt 设计游戏 FlowFree 的设计文档，本文档分为三个部分：

1. 游戏特色介绍
2. 游戏设计和各个类的简要介绍
3. 各个类的详细介绍
4. 游戏设计过程中遇到的问题及解决方法。

游戏模仿自 Windows Shop 和 Android Shop 同名游戏。用鼠标操作链接水管，胜利条件为填满屏幕。

一共用了 5 天，共 1974 行代码，期间遇到了很多 Qt 设计的问题，学到了不少东西。也是我第一次写这种规模的代码，感觉设计模式的思想对我最终完成这个游戏产生了很大的帮助。

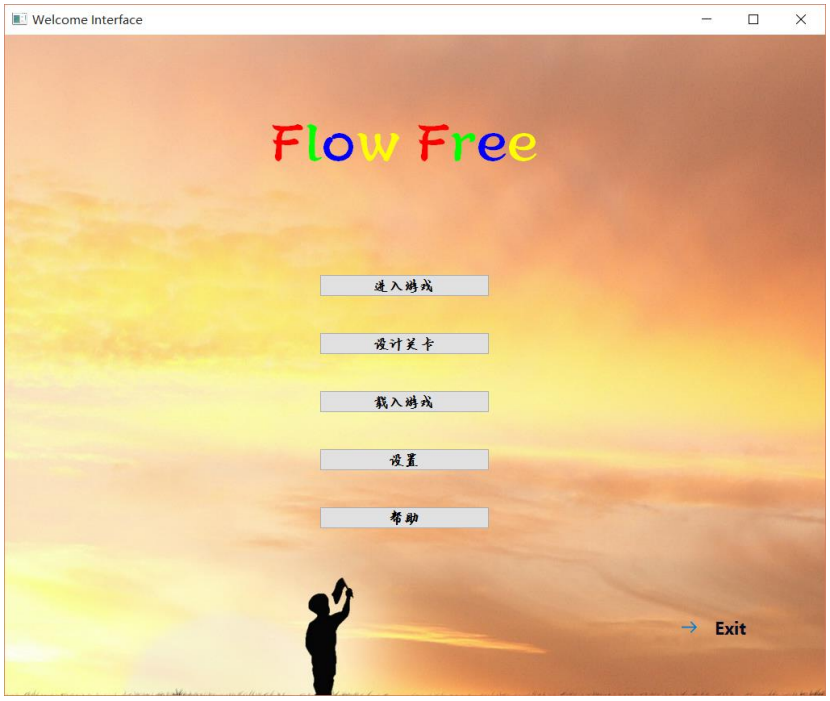
代码可在

https://github.com/ytl13508111107/flowFree 下载

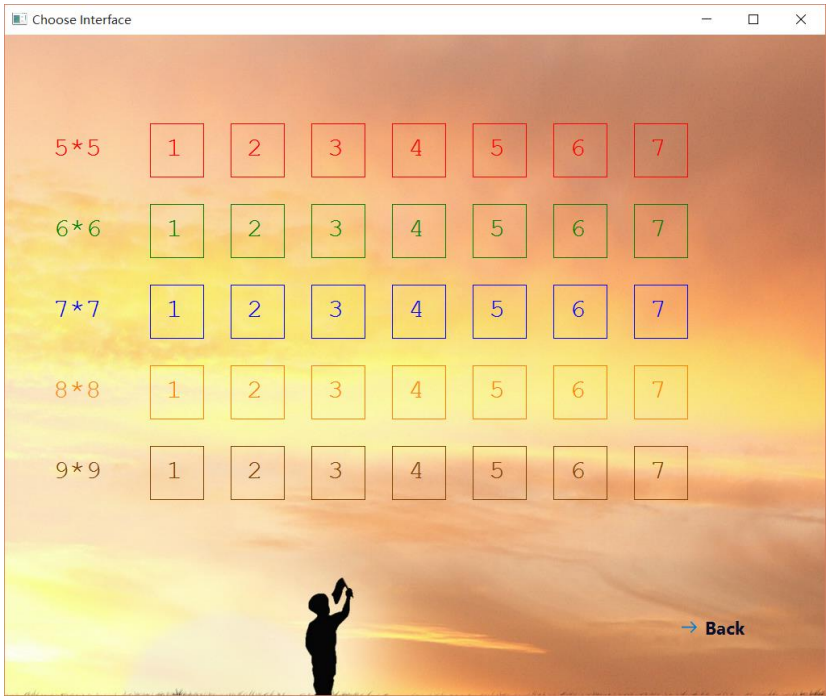
正文

特色介绍

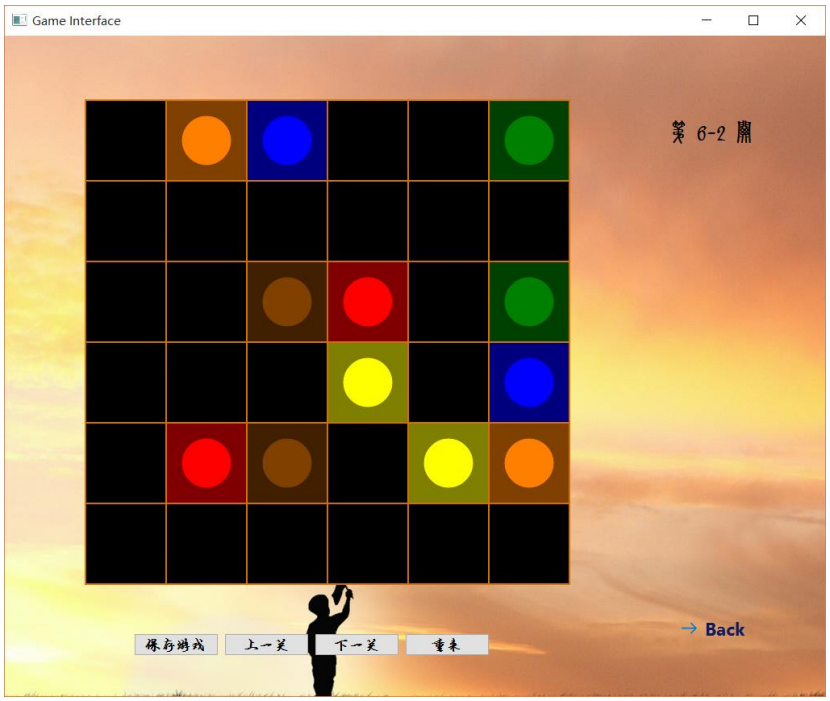
欢迎界面



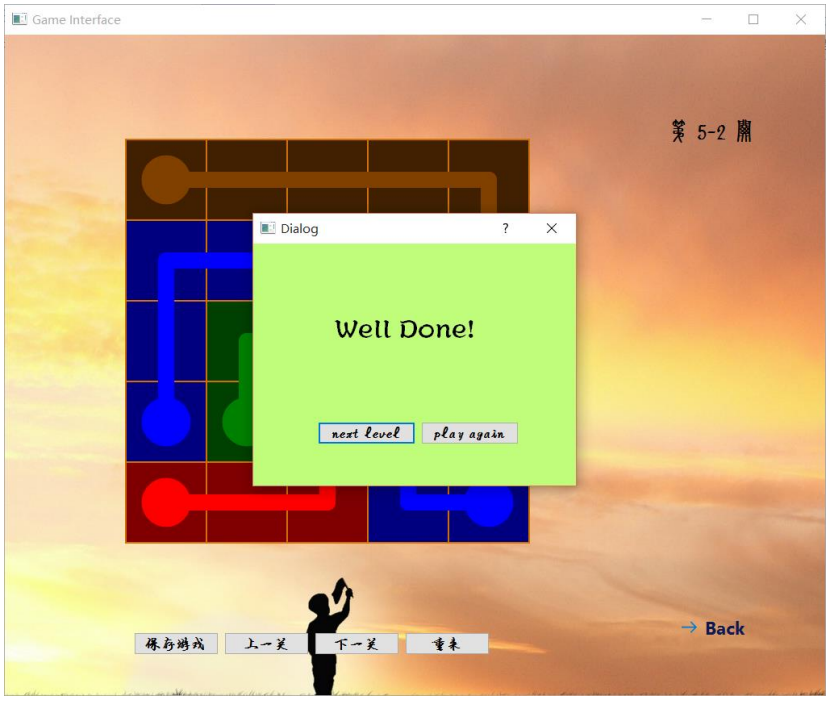
点击进入游戏进入选关界面


Choose Interface
5*5 1 2 3 4 5 6 7
6*6 1 2 3 4 5 6 7
7*7 1 2 3 4 5 6 7
8*8 1 2 3 4 5 6 7
9*9 1 2 3 4 5 6 7
Back

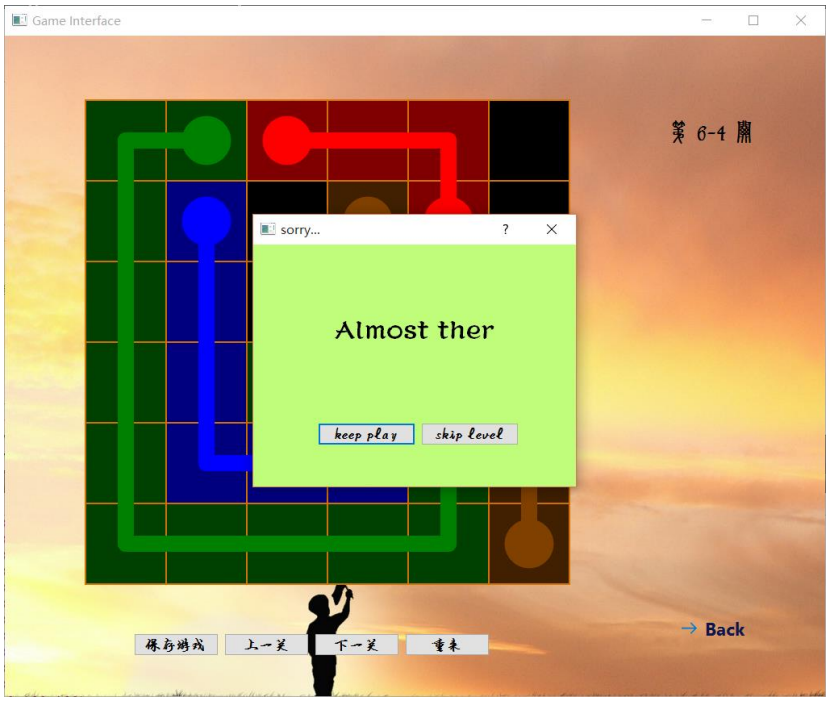
选择关卡进入游戏界面


Game Interface
第 6-2 关
保存游戏
上一关
下一关
重来
Back

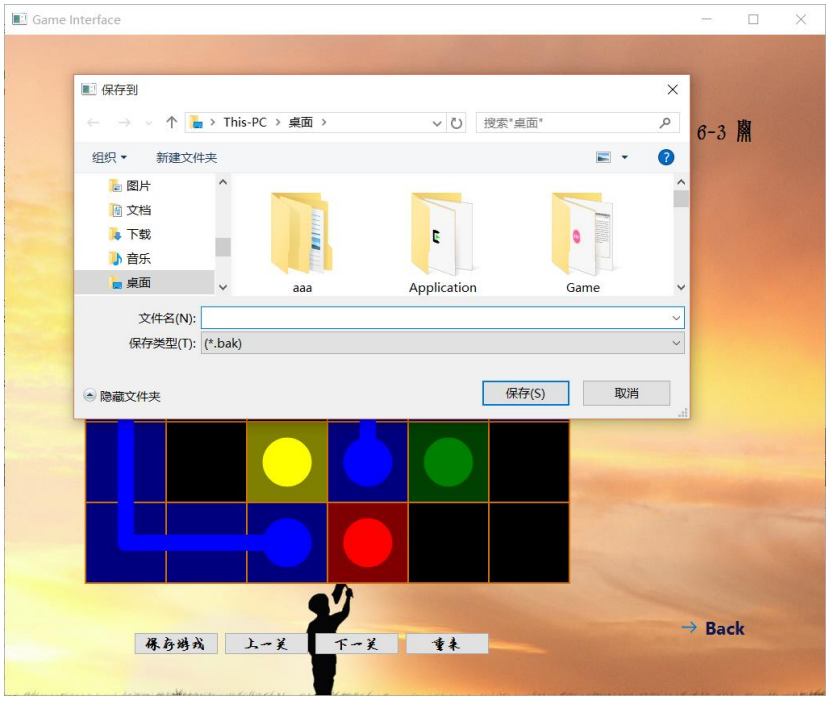
完成游戏



没有全部填满的情况

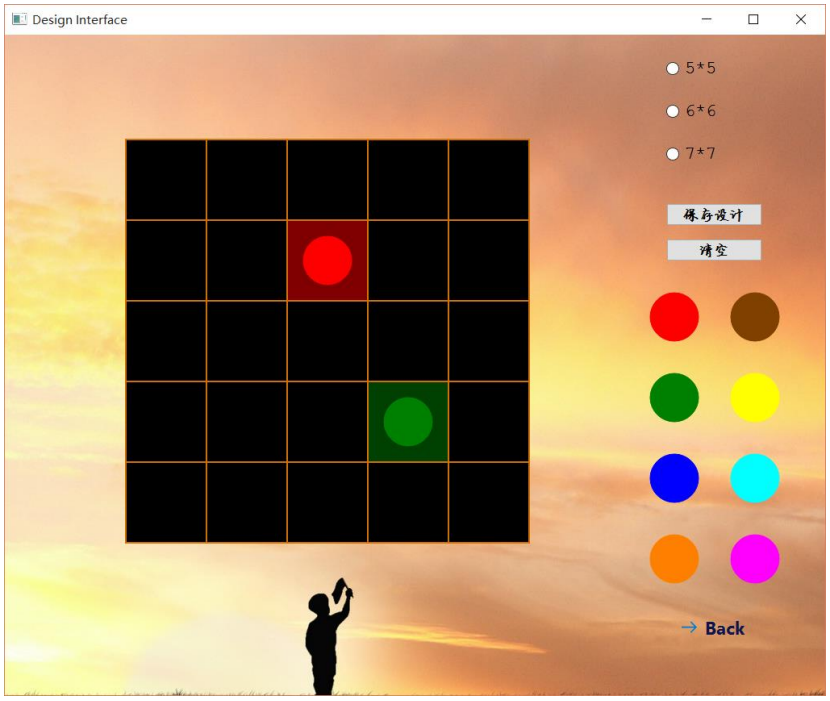


可以保存正在的玩的游戏局面

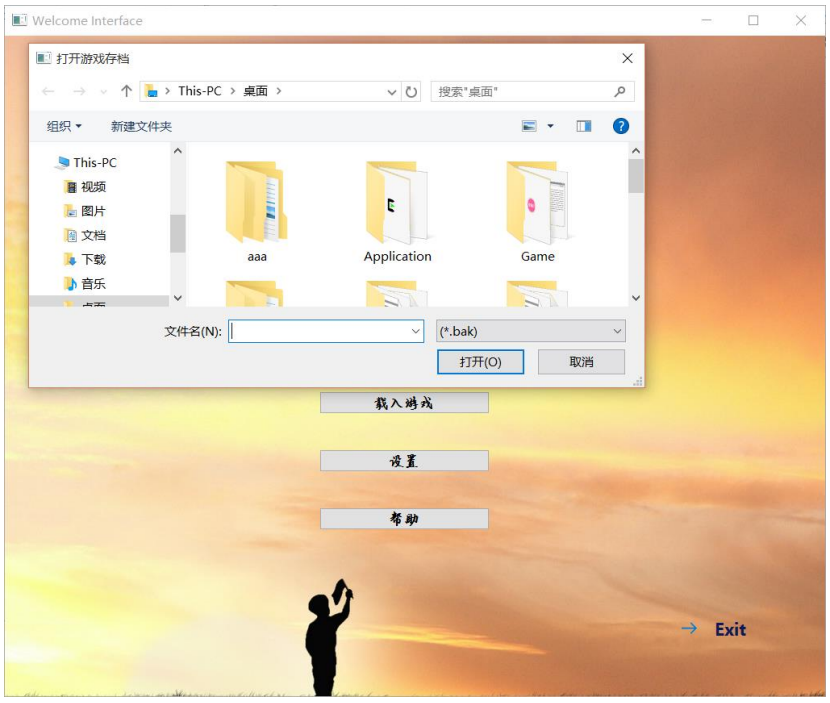


或者按住 Shift+XYZZY 直接显示答案

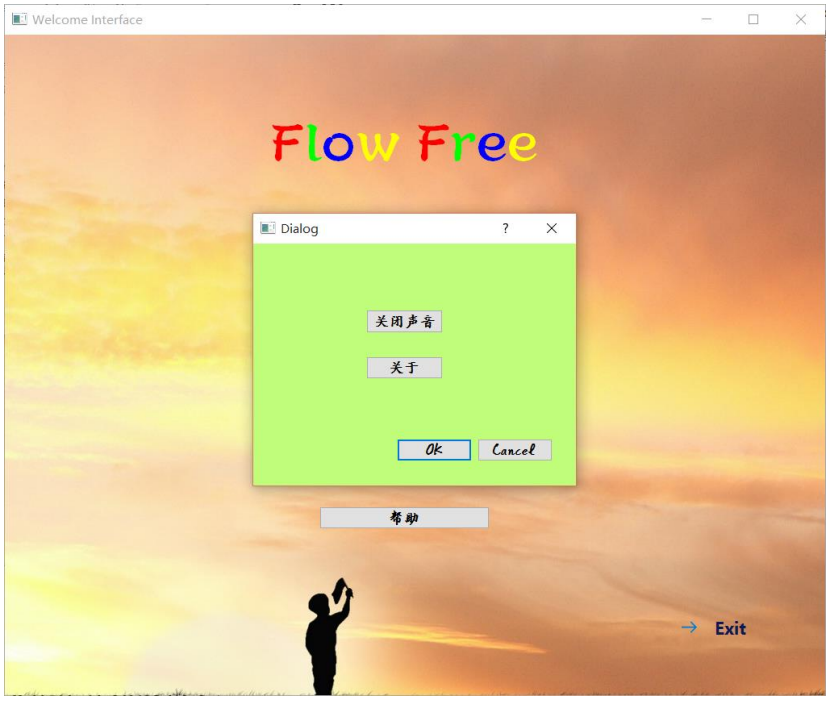
游戏设计及保存



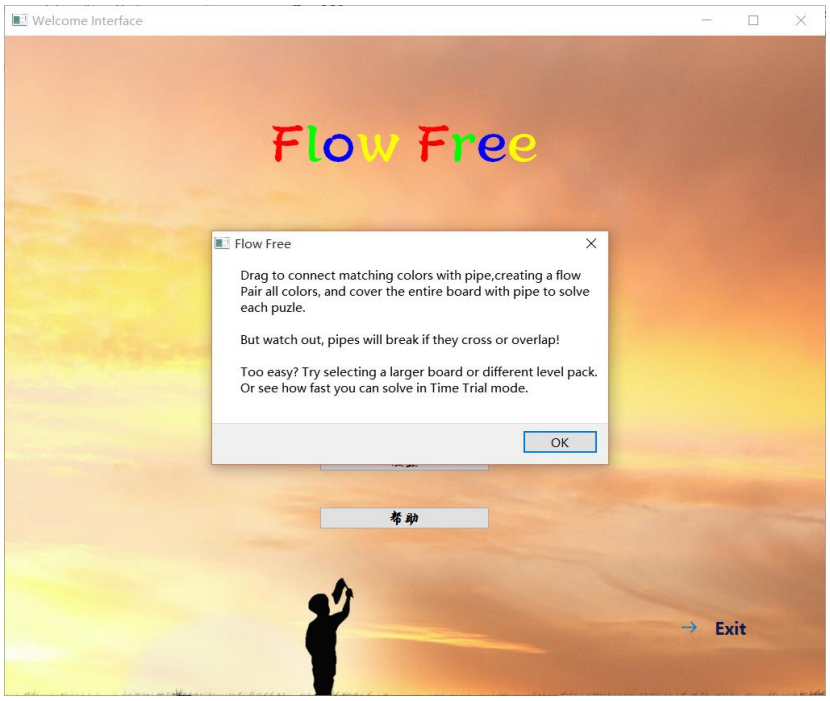
打开存档



游戏设置


Welcome Interface
Flow Free
Dialog
关闭声音
关于
Ok
Cancel
帮助
Exit

帮助


Welcome Interface
Flow Free
Flow Free
Drag to connect matching colors with pipe,creating a flow
Pair all colors, and cover the entire board with pipe to solve each puzle.
But watch out, pipes will break if they cross or overlap!
Too easy? Try selecting a larger board or different level pack.
Or see how fast you can solve in Time Trial mode.
OK
帮助
Exit

各类基本介绍及用途

游戏被分为6个模块。分别为

1.游戏信息（GameInfo,Block,Way）负责记录游戏信息

2.界面(WelcomeInterface,ChooseInterface,GameInterface,
DesignInterface)负责显示界面和按钮事件

3.文本信息(TextInfo)负责游戏信息和文本信息的转换和文件存取

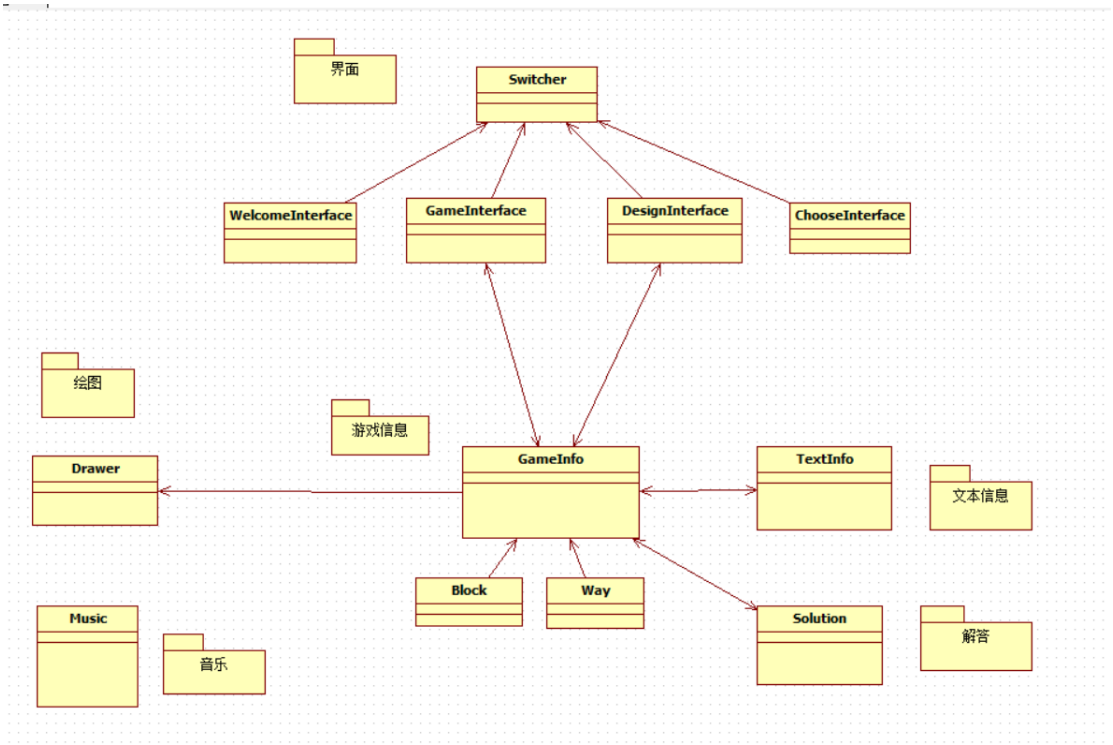
4.解答(Solution) 负责计算答案

5.绘图(Drawer)负责按照游戏信息绘制图形

6.音乐(Music)负责音乐播放

UML 图如下

(对 UML 不是很熟，只是列出各类的关系，UML 有使用不恰当的地点、请谅解)



GameInfo 类——游戏信息

核心部件，记录绘制游戏图形(图 1)需要的信息

包含 Block 和 Way 两个子类，用来记录小方块和路径的信息

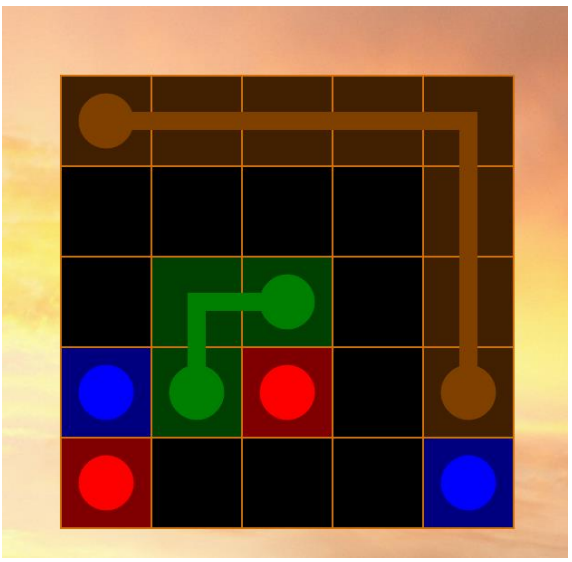


图 1

TeextInfo 类——文本信息

记录文本信息和打开保存游戏文件(*.bak),以及和GameInfo 之间的相互转换

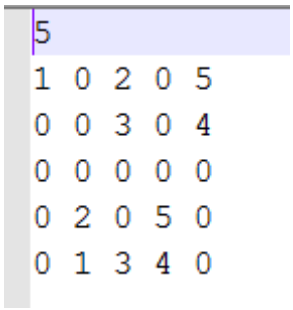


图 2

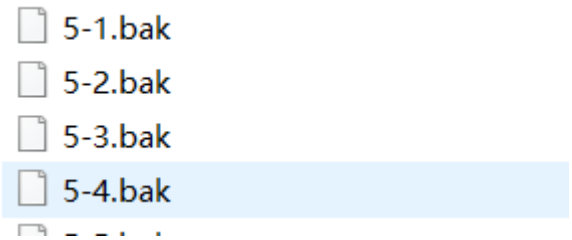
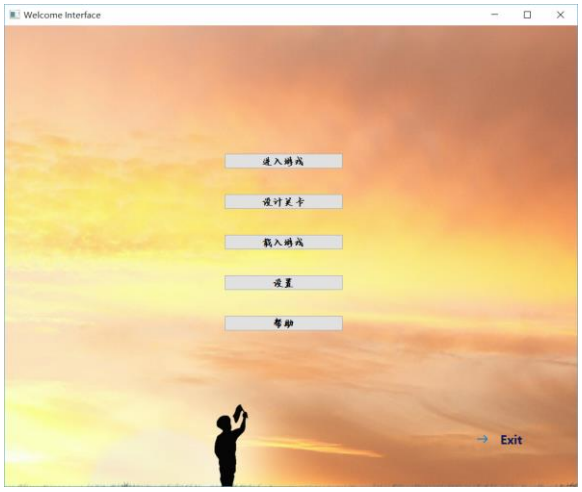


图 3

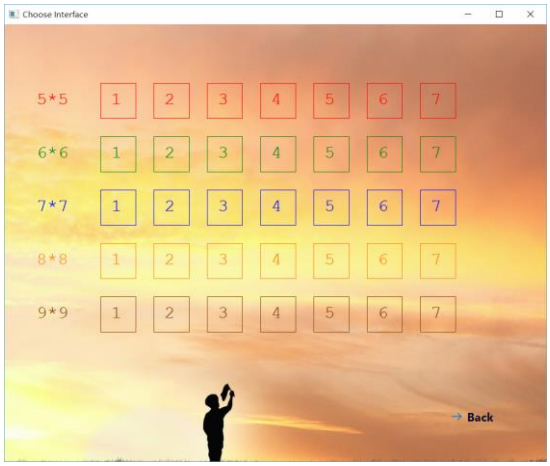
界面

前端，主要负责 UI 显示和按钮操作部分

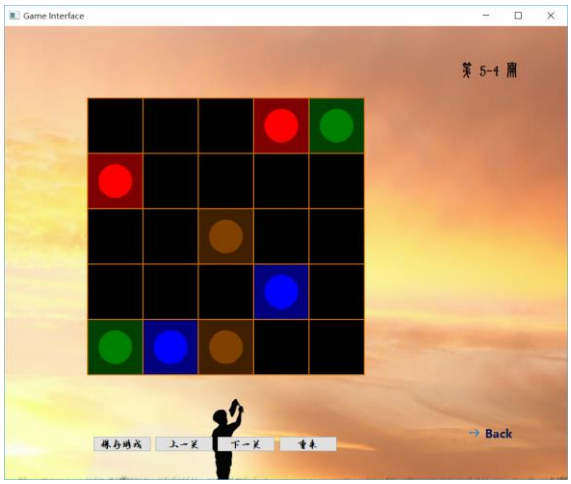
WelcomeInterface 类——欢迎界面



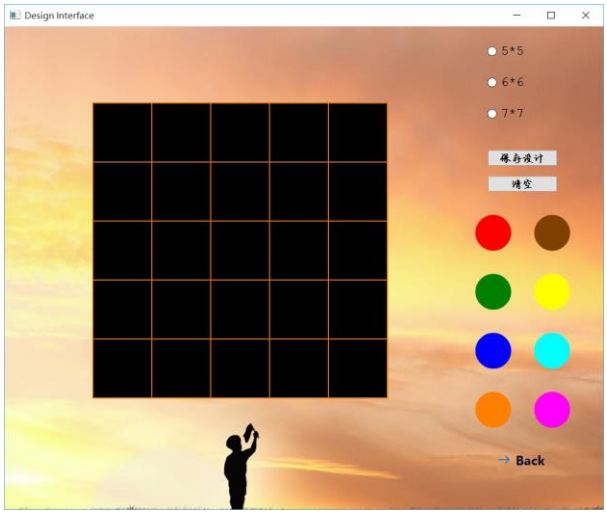
ChooseInterface 类——选关界面



GameInterface 类——游戏界面



DesignInterface 类——设计界面



Switcher 类——用于各界面的切换

Drawer 类——绘图

将游戏信息 GameInfo 绘制到对应的界面（GameInterface 和 DesignInterface）上

Music 类——音乐

控制不同事件播放音乐

Solution 类——解答

搜索计算游戏答案

environment.h——环境变量

存储如颜色，界面坐标等不变的环境信息

各类的详细介绍

游戏信息

对于游戏的记录分为两个部分，一个记录每一个游戏小格（Block）的状态，记录小格的状态，二是记录所有有效的管道连接（Way）

Block 类

记录每个小格的状态

```
class Block
{
public:
    Block();
    QPoint loc;//坐标
    int color;//颜色
    Status status;//状态 status={source,gothrough,unmark}

    QRect getRect();//Block所在的矩形框坐标
    QPoint getCenter();//中心坐标
    bool isNeighbour(const Block &a);//和某一个Block处于游戏界面相邻位置
    bool isSame(const Block &a){return loc.x()==a.loc.x() && loc.y()==a.loc.y();}
    bool initBlock();//重置Block信息
};
```

Block 分为 status，loc，color

status：分为三种

status={source,gothrough,unmark},分别表示

源点  ，有路径经过的点  ，没有标记的点 

color:记录该 Block 的颜色

loc: 记录该 Block 是处于游戏界面的第几行第几列

Way 类

记录一条从起点到终点的路径

```
8   //记录一条从起点到终点的路径
9   class Way
10  {
11  public:
12      Way();
13      vector<Block*> link;
14
15      int size()const{return link.size();}//包含的Block个数
16      int getColor()const{return (*link[0]).color;}//路径颜色
17      Block& last(){return *link[link.size()-1];}//路径最后一个Block
18      Block& head(){return *link[0];}//路径第一个Block
19      void add(Block* a);//添加一个Block
20      void clearTail();//去掉除头部的所有Block(比如路径被切断时)
21      void initWay();//重置way
22      void print();//打印way信息到qDebug
23  };
```

GameInfo 类

由 n*n 个 Block 类组成的 blocks 和 Way 数组组成的 ways 以及其余若干和游戏有关的所有绘制信息

```
class GameInfo
{
public:
    Block blocks[15][15];
    int map[15][15];
    Way ways[50];
    int waysTot;//ways总数
    int gameFormat; //游戏大小(5*5 6*6 7*7)
    int noSolution;//游戏无解
    int sourceTot;//源点总数
    int nowWay;

    GameInfo();
    void makeBlocksInfo();//根据gameFormat初始化Block信息(如坐标位置等)
    void clearWay(int);//去除某条way
    void cutWay(int);//切断某条way
    bool isLegal();
    void makeColor(int &color);
    void makePos(int &x,int &y);
    void makeSource();//随机生成源点(用于随机地图)
    void reInit();//重置GameInfo(清空)
    void restart();//重新开始游戏(保留源点)
    bool getLoc(int &a,int &b,QPoint pos);
    bool legalDesign();//不合法的GameInfo
    void print();//打印GameInfo
};
```

map[i][j]:第 i 行 j 列的 Block 属于哪一条路径

waysTot: ways 的总数

gameFormat: 游戏的规模(gameFormat*gameFormat)

sourceTot:源点总数

noSolution:游戏无解标记

TextInfo 类

游戏存档的文本信息，可以和 GameInfo 实现互相转换，以及打开保存游戏存档文件

```
class TextInfo
{
public:
    int noSolution;//是否无解

    TextInfo();
    TextInfo(const GameInfo &game);//GameInfo转TextInfo
    bool transToGameInfo(GameInfo &game);//TextInfo转GameInfo
    bool openFile(QString fileDir);//打开游戏存档并载入信息
    bool saveFile(QString fileDir);//保存游戏到文件
    bool loadGame(int level,int id);//打开关卡(level-id.bak)
    void print();//打印TextInfo

private:
    int gameFormat;//游戏规模
    int map[15][15];//文本信息

    bool inMap(int x,int y);
    bool dfs(int x,int y,int num,GameInfo &game);
    bool checkColorLegal();
    bool checkWaysLegal(const GameInfo &game);
    bool haveGothrough(int x,int y);
};
```

transToGame(): 将 TextInfo 转换为 GameInfo-

openFile():打开游戏存档

saveFile(): 保存游戏存档

界面

WelcomInterface 类

用来显示欢迎界面以及处理相应的按钮事件

```
class WelcomeInterface : public QWidget
{
    Q_OBJECT
public:
    explicit WelcomeInterface(QWidget *parent = 0);
    ~WelcomeInterface();
signals:
    void sendTextInfo(TextInfo text,int level,int id);//将"载入游戏"读到的游戏存档发送给GameInterface处理
private slots:
    void on_goIntoGame_button_clicked();//"进入游戏"按钮事件
    void on_designGame_button_clicked();//"设计游戏"按钮事件
    void on_loadGame_button_clicked();//"载入游戏"按钮事件
    void on_setting_button_clicked();//"设置"按钮事件
    void on_help_button_clicked();//"帮助"按钮事件
    void on_exit_button_clicked();//"Exit"按钮事件
private:
    Ui::WelcomeInterface *ui;
};
```

ChooseInterface 类

用来显示选关界面以及处理相应的按钮事件

```
11  class ChooseInterface : public QWidget
12  {
13      Q_OBJECT
14
15  public:
16      explicit ChooseInterface(QWidget *parent = 0);
17      ~ChooseInterface();
18      void paintEvent(QPaintEvent *event);
19      void mousePressEvent(QMouseEvent *event);
20      void loadGame(int level,int id);//玩家点击第level-id关图标
21  signals:
22      sendTextInfo(TextInfo text,int level,int id);
23  private slots:
24      void on_back_button_clicked();//"Exit"按钮事件
25  private:
26      Ui::ChooseInterface *ui;
27  };
```

GameInterface 类

显示游戏界面以及处理相应的按钮事件

处理各种游戏事件(如上一关下一关，游戏完成等)

```
class GameInterface : public QWidget
{
    Q_OBJECT
public:
    explicit GameInterface(QWidget *parent = 0);
    ~GameInterface();
    void paintEvent(QPaintEvent *event);
    void mouseMoveEvent(QMouseEvent *event);
    void mousePressEvent(QMouseEvent *event);
    void mouseReleaseEvent(QMouseEvent *event);
    void keyPressEvent(QKeyEvent *event);
    void keyReleaseEvent(QKeyEvent *event);
    void startGame();//开始游戏
    void quitGame();//离开游戏
    bool checkGameComplete();//检查游戏完成
    void gameComplete();//游戏完成
    bool checkGameFinishHalf();//检查游戏完成(未填满)
    void gameFinishHalf();//游戏完成(未填满)
    void connectTextInfo(QWidget *choose);
    void showLevel();//显示当前关卡标签
    bool isTracking;//追踪状态(鼠标按下)
    int focusWay;//追踪的路径编号

private slots:
    void on_back_button_clicked();//"Exit"按钮事件
    void on_previous_level_button_clicked();//"上一关"按钮事件
    void on_next_level_button_clicked();//"下一关"按钮事件
    void on_restart_button_clicked();//"重来"按钮事件
    void on_save_button_clicked();//"保存游戏"按钮事件

private:
    int level,id;//处于第level-id关
    void cheat();//作弊模式开启

public slots:
    void getTextInfo(TextInfo text,int level,int id);//接收传来的游戏信息
    void nextLevel();//下一关
    void previousLevel();//上一关
    void restartGame();//重新开始游戏
    void keepGame();

private:
    Ui::GameInterface *ui;
    GameInfo game;
    Drawer *drawer;
    QString cheatRecord;
    int wayConnectEvent;
};
```

DesignInterface 类

用来显示设计界面以及处理相应的按钮事件

```
class DesignInterface : public QWidget
{
    Q_OBJECT

public:
    explicit DesignInterface(QWidget *parent = 0);
    ~DesignInterface();
    void paintEvent(QPaintEvent *event);
    void mousePressEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent *event);
    void mouseReleaseEvent(QMouseEvent *event);
    void reInit();

private slots:
    void on_back_button_clicked();//"Exit"按钮事件
    void on_radioButton_5_5_clicked();//"5*5"按钮事件
    void on_radioButton_6_6_clicked();//"6*6"按钮事件
    void on_radioButton_7_7_clicked();//"7*7"按钮事件
    void on_store_button_clicked();//"保存"按钮事件
    void on_clear_button_clicked();//"重置"按钮事件

private:
    Ui::DesignInterface *ui;
    GameInfo game;
    Drawer *drawer;
    int chooseSource;
    int sourceColor;
    QPoint movePoint;
    int haveMovePoint;
};
```

Switcher 类

界面切换器，负责各个界面的 show,hide 控制

可调用 add(“game”,gameInterface);将名字为”game”的界面加入 switcherr

调用 showInterface(“game”)即可显示对应的界面，隐藏其他界面以及调整窗口的位置和大小等

```
class Switcher
{
public:
    Switcher();
    void add(QString name,QWidget *interface);
    void showInterface(QString name,QString mode="menuclick");
private:
   map<QString,QWidget*> switcher;
   QWidget *now;
};
```

Drawer 类

用于游戏图形绘制，指定界面 QWidget *devide 和游戏信息 GameInfo *game, 即可在 device(如 GameInterface 和 DesignInterface)上按照 game 的信息绘制图像

```
class Drawer
{

public:
    Drawer(QPaintDevice *device,GameInfo *game);
    void draw();//在device界面上按照game的信息绘制游戏界面
    void drawSource(QColor color,QPoint pos);
    QPainter *painter;


private:
    void drawGridding();//画网格线
    void drawBlocks();//画方块
    void drawWays();//画路径
    QPaintDevice *device;
    GameInfo *game;

};

```

Music 类

后台实现对音乐的控制，游戏开始时载入音乐，调用playMusic(QString name)播放指定音乐,changeStatus()控制音乐的播放/暂停

```
class Music
{
public:
    Music();
    void changeStatus();//音乐开/关
    void loadMusic();//载入音乐
    void playMusic(QString name);//播放音乐name
    MusicStatus getStatus();//音乐播放状态(播放/停止)

private:
    MusicStatus status;
    map<QString,QString> musicList;
    QSound *background;
};
```

Solution 类

传递 GameInfo 信息给 Solution 类，调用 getSolution()可得到一个完成解答的 GameInfo

```
class Solution
{
public:
    Solution();
    TextInfo getSolution(const GameInfo &game);//计算答案

private:
    void greedyAlgorithm();//贪心算法
    bool dfs(int colorID,QPoint pos,int step);//暴力搜索
    bool inMap(int x,int y);
    bool fillAll();

    GameInfo game;
    QPoint pointRecord[maxColorTot][2];
    int colorRecord[maxColorTot];
    QPoint record[100];
};
```

环境变量

environment.h

```
8   //source圆点的半径
9   static const int sourceWide=30;
10  //游戏界面的左上坐标
11  static const int defaultStartX=30,defaultStartY=50;
12  extern int startX,startY;//坐标为(y,x),其中x为坐标轴x轴正方向,y为坐标轴y轴负方向
13  //游戏界面小格宽度
14  static const int BlockLen=100;
15  //网格线宽度
16  static const int penLen=2;
17  //游戏中心点
18  static const QPoint GameCenter(400,380);
19  //设计界面8个待选颜色左上坐标
20  static const int selectionStartX=730,selectionStartY=250;
21  //设计界面8个待选颜色间距
22  static const int selectionLen=100;
23  //作弊码
24  //static const int cheatKeys[]={16777248,90,89,90,90,89};//shift+xyzzy
25  //static const int cheatTot=6;
26  static const QString cheatKeys="XYZZY";
27
28  //总颜色数
29  static const int myColorTot = 11;
30  static const int maxColorTot=20;
31  static const QColor myColor[15]={
32      QColor(0,0,0),//黑 1
33      QColor(255,0,0),//红 2
34      QColor(0,128,0),//绿 3
35      QColor(0,0,255),//蓝 4
36      QColor(255,128,0),//橙 5
37      QColor(128,64,0),//棕 10
38      QColor(255,255,0),//黄 6
39      QColor(0,255,255),//青 7
40      QColor(255,0,255),//粉 8
41      QColor(128,0,255),//紫 9
42      QColor(128,128,0),//屎黄 11
43      QColor(192,192,193),//灰 12
44
45  };
```

遇到的问题和解决方案

1. 界面切换的时候如果之前移动了窗口，切换后的界面还会在原来的位置

   解决方法：设计了 Switcher 类来处理界面切换的问题

2. 使用了多个.ui 文件在编译时会报错

   解决方法：设计 ui 文件对应的类，修改.ui 的类名

3. 多次调整窗口布局，经常要改很多地方，一个漏了游戏界面就会全乱套

   解决方法：所有坐标通过一个基本的全局变量窗口起始坐标计算得到，避免直接使用数字，绘图部分抽离出来用专门的 Drawer 类来完成

4. 需要弹出一些简单的自定义窗口，专门为此创建.ui 和对应的类有点小题大做，但是直接 new QDialog 又需要调整窗口的大小，位置和按钮的大小过于麻烦。使用 QMessageBox 又没法修改按钮的文字

   解决方法：学会只 include “ui_xx.h”，用

```
mydialog = new QDialog(this);
Ui::GameDialog ui;
ui.setupUi(mydialog);
```

   直接在创建的 QDialog 上套用 ui 信息

5. 使用 QMeultiPlayList 无法加载音乐,加载背景音乐程序崩溃

   解决方法：使用 QSound 类，Qt 的 Resource 资源文件会

全部写入内存不能放大的文件，实在需要加载只能打开本地路径