

数独大作业实验报告

姓名：林宸昊

学号：PB20000034

实验题目与要求

本次实验主要内容是实现一个简单数独软件，具体要求如下：

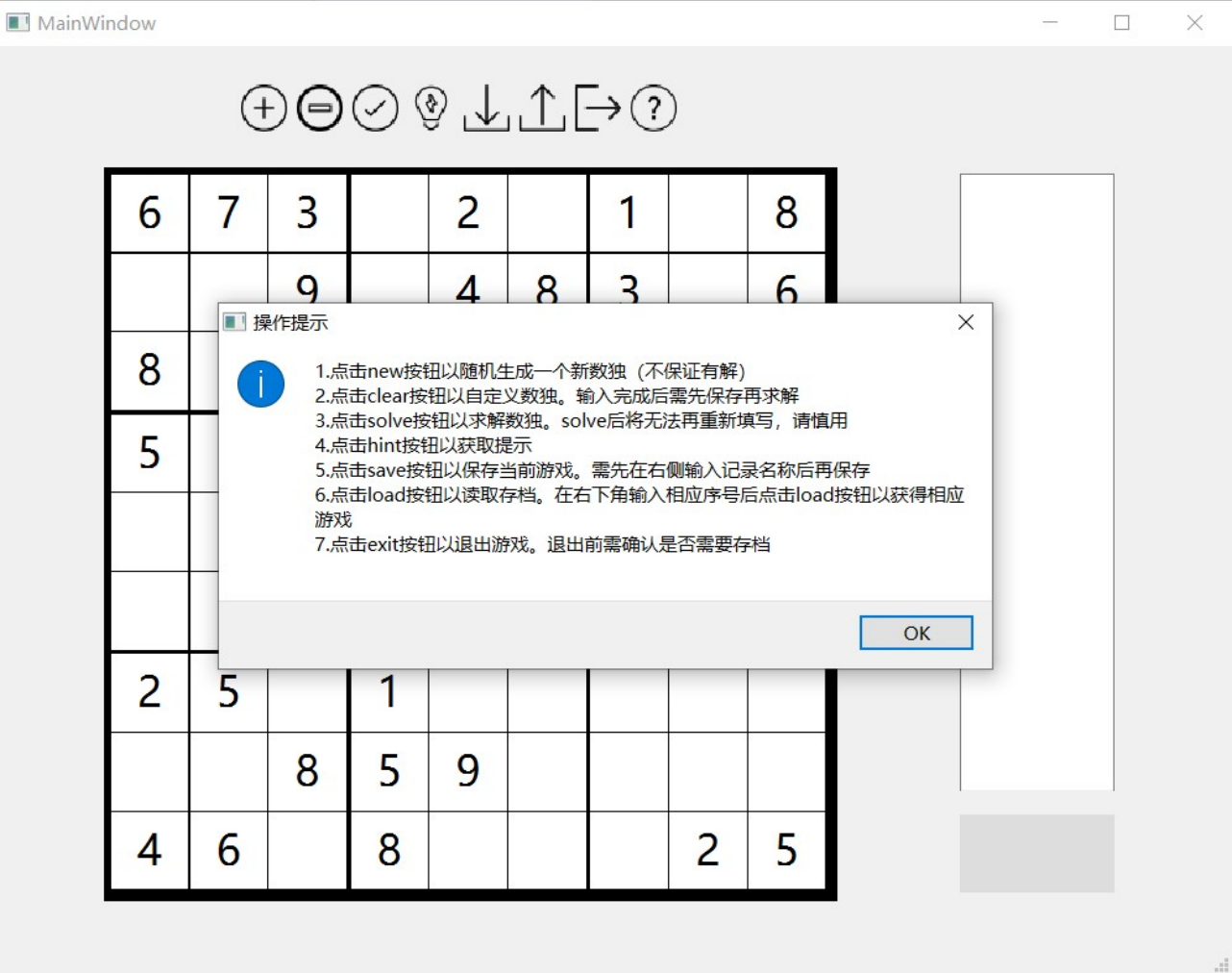
基本功能部分

放在前面

所有输入通过键盘实现，不合法的输入将无法填入数字，即程序不会给出响应。通过键盘方向键及鼠标点击获取目标方块以进行输入。

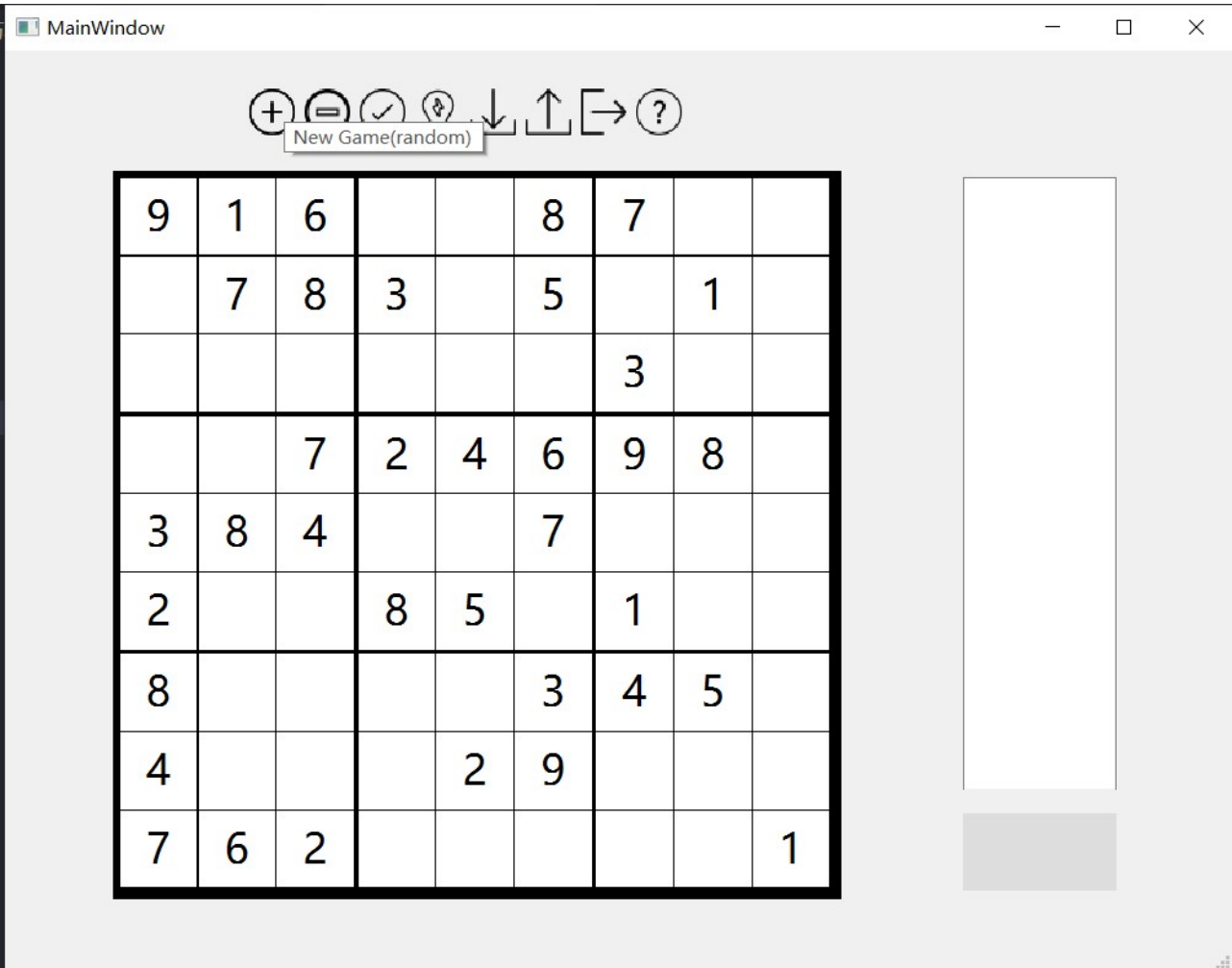
- 程序说明信息及交互

点击第八个按钮



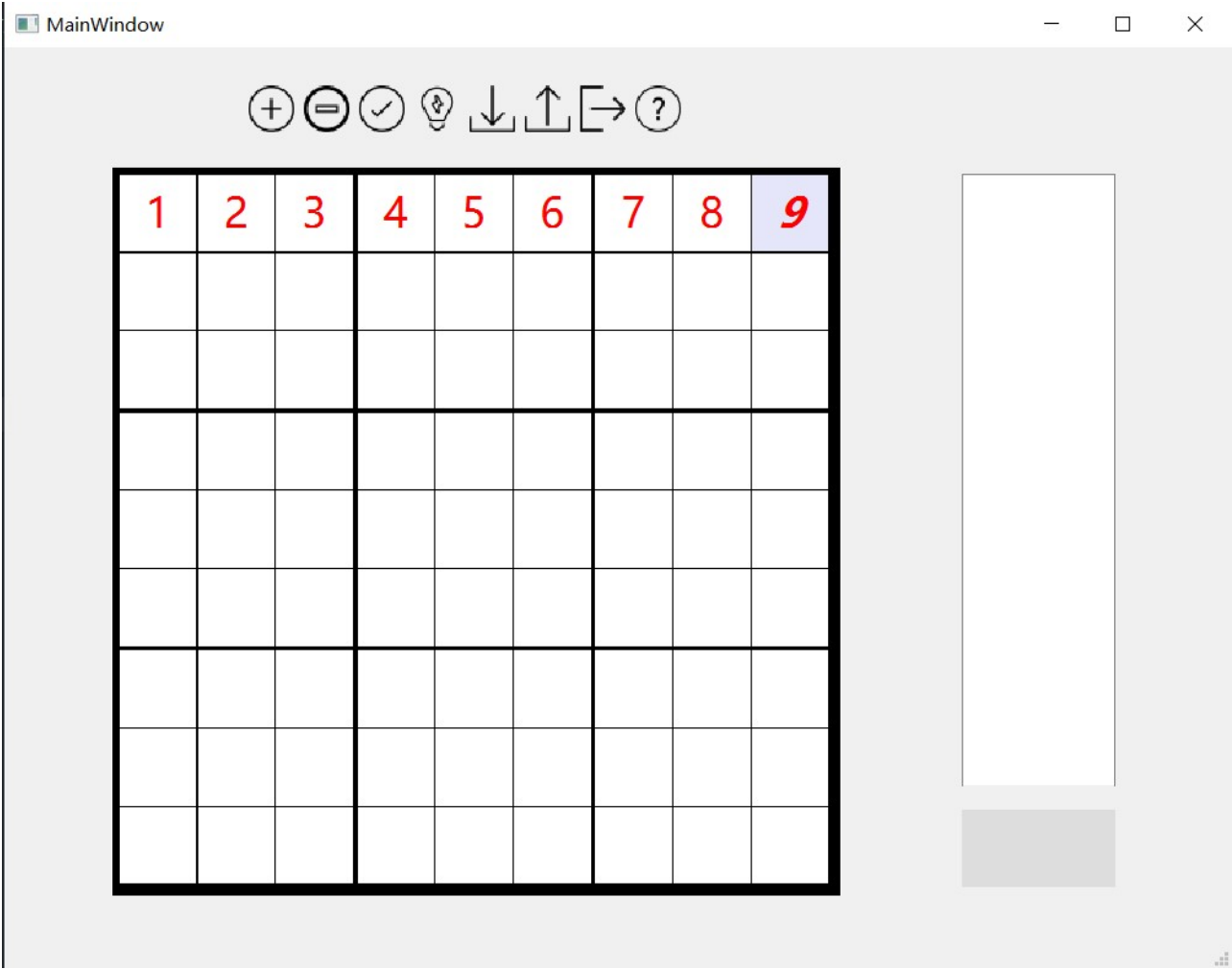
- 数独生成（不考虑是否有解）

点击第一个按钮




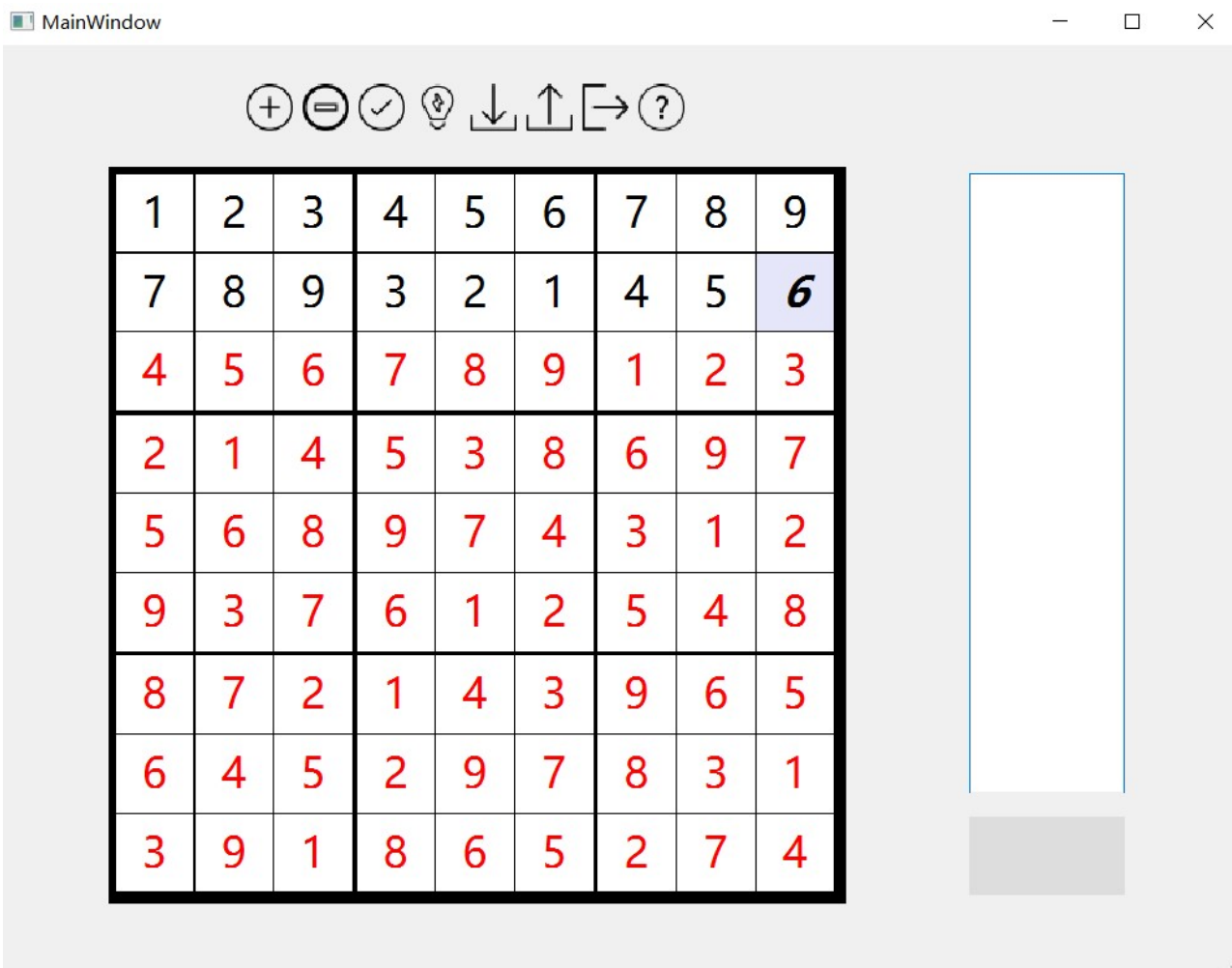
- 数独输入

点击第二个按钮清空后自行输入



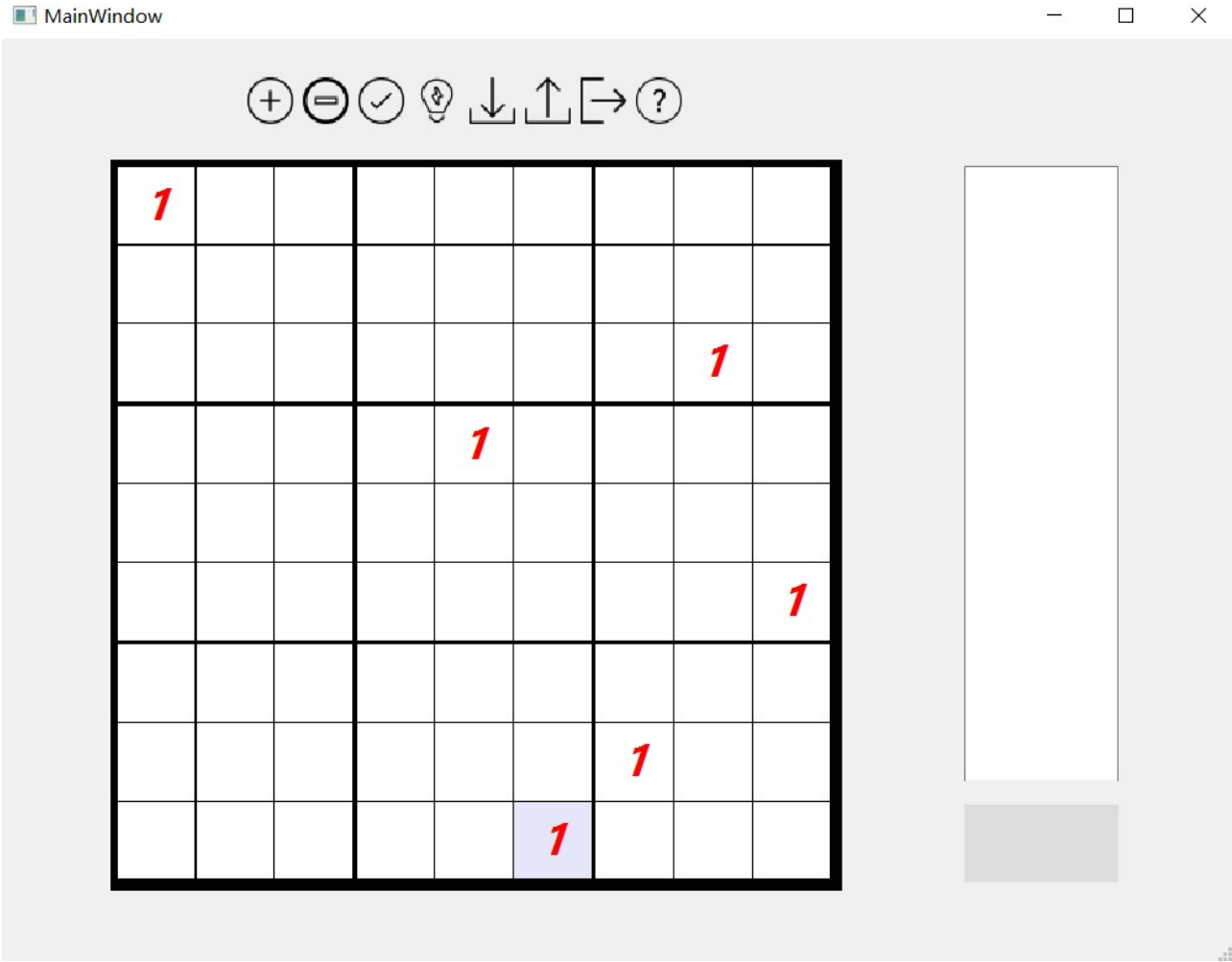
- 输出数独的解

 MainWindow



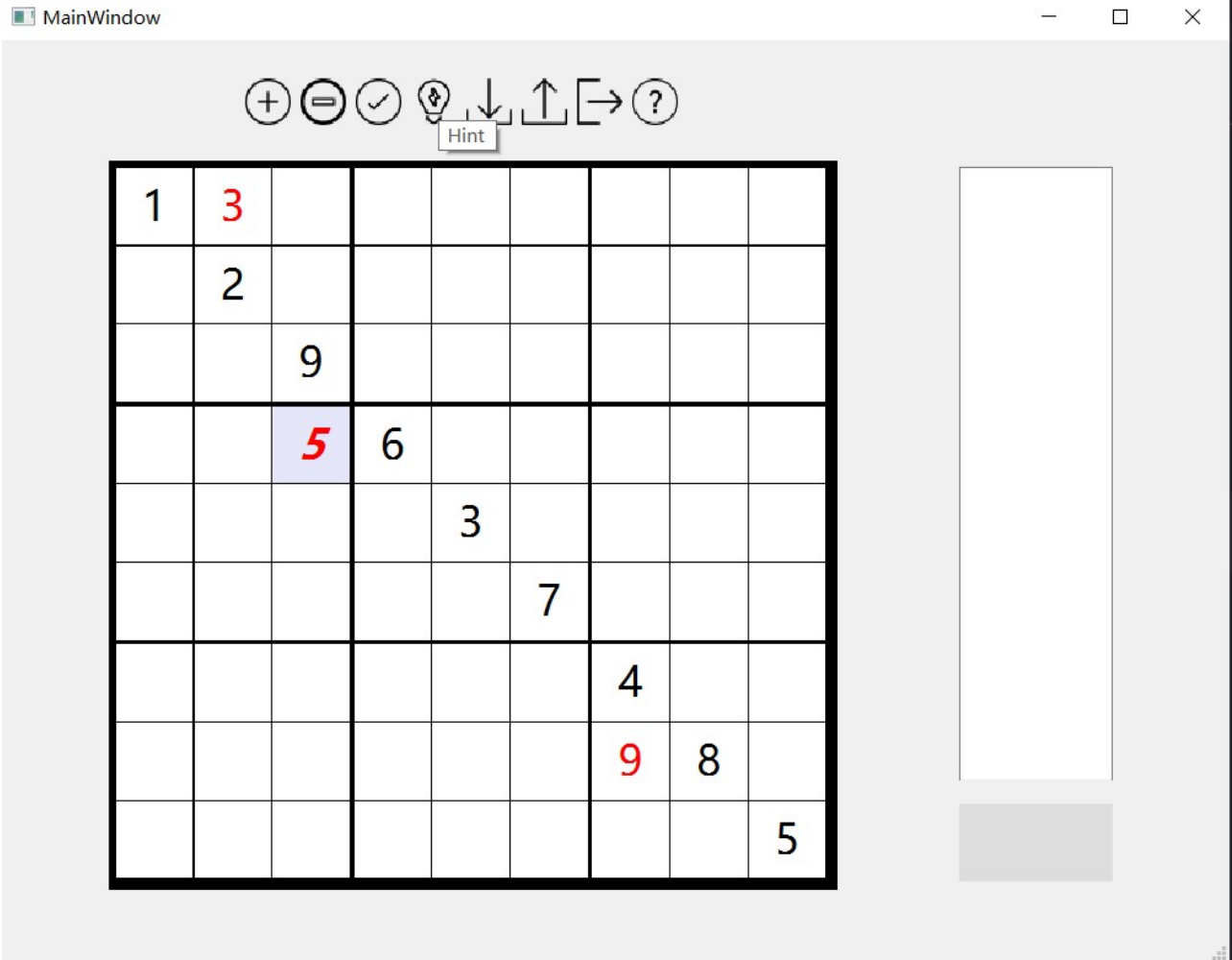
关于合法输入

可以通过点击相应块获知当前有哪些块中有相同数字



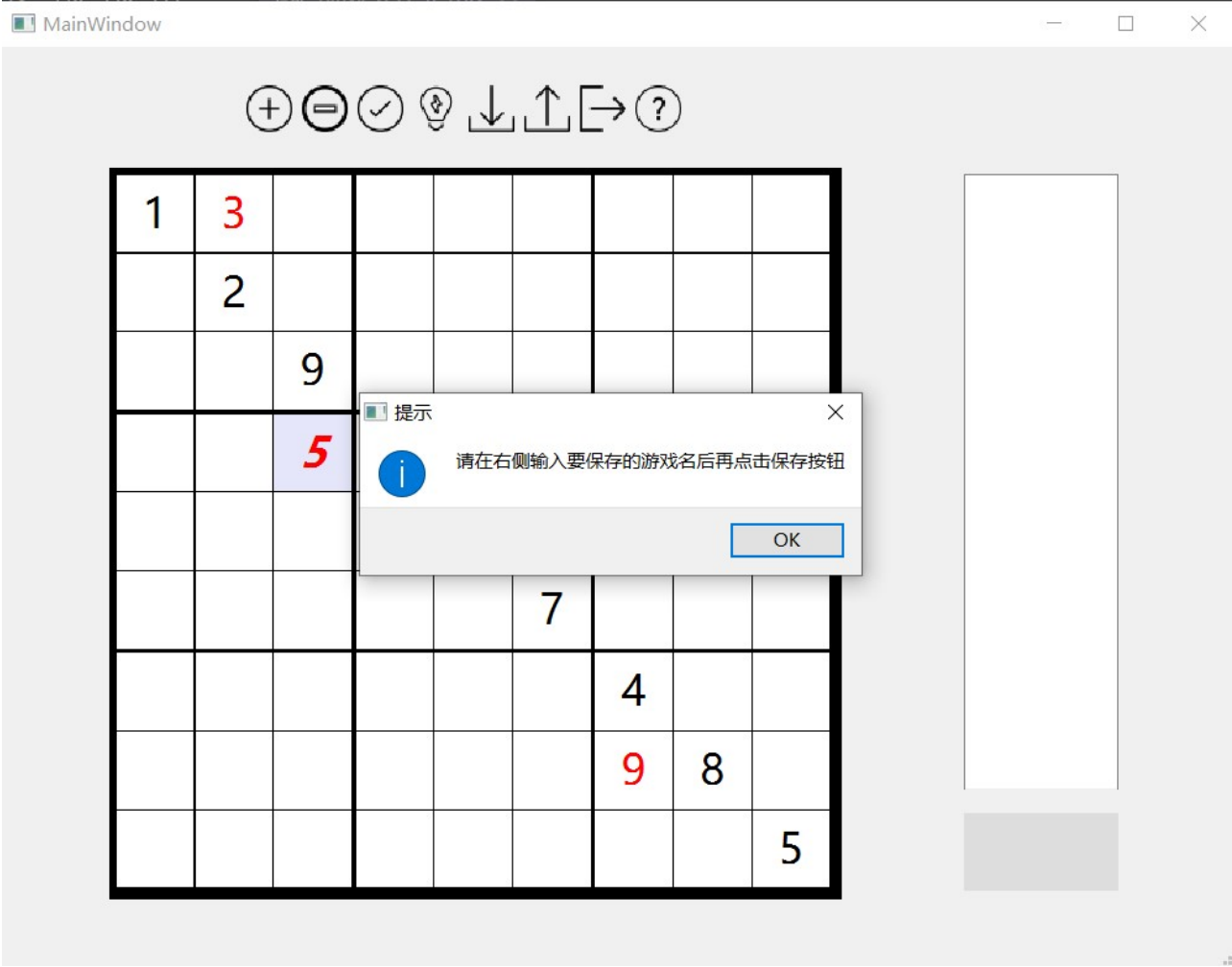
数独提示

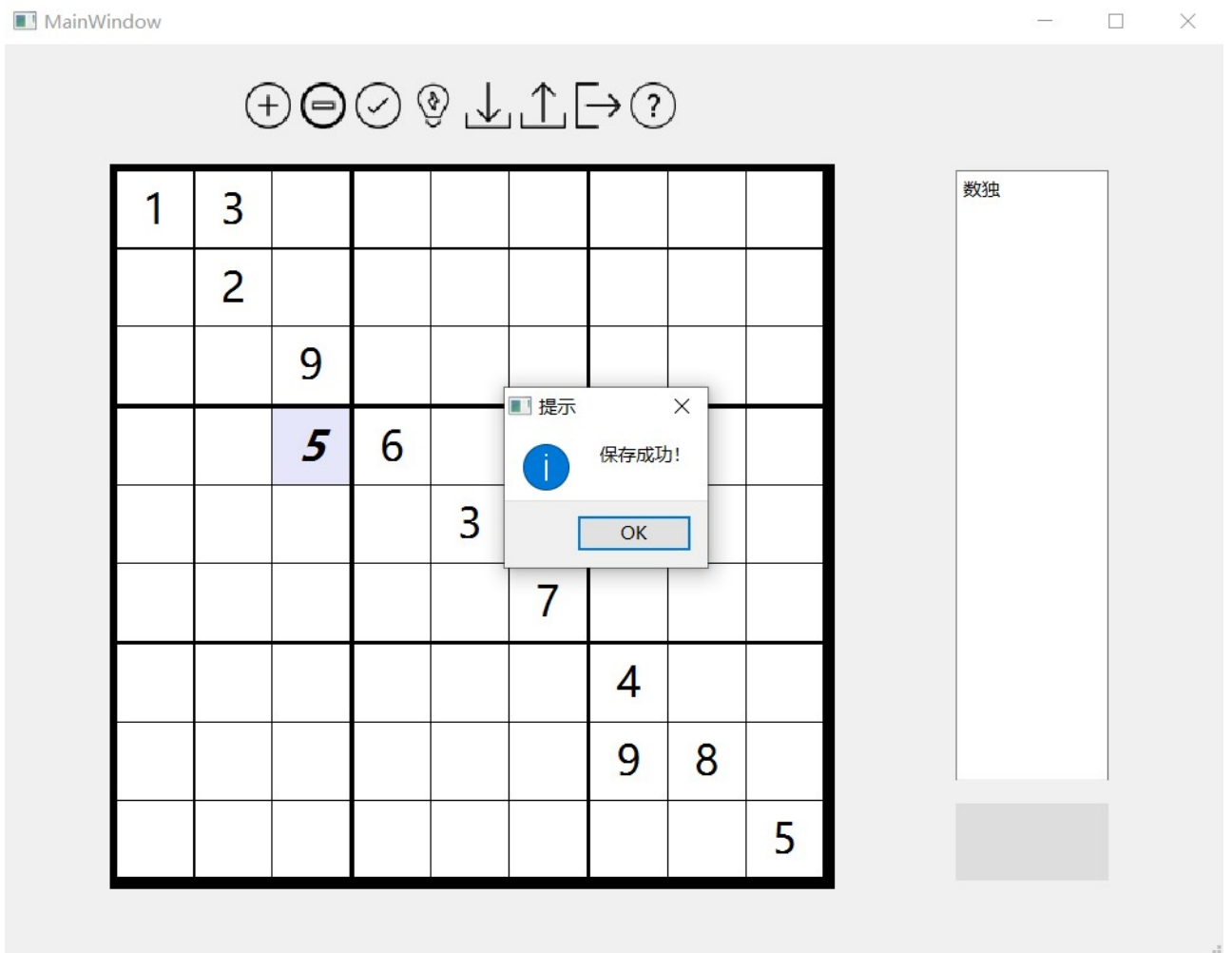
点击第四个按钮获取随机提示



- 保存游戏

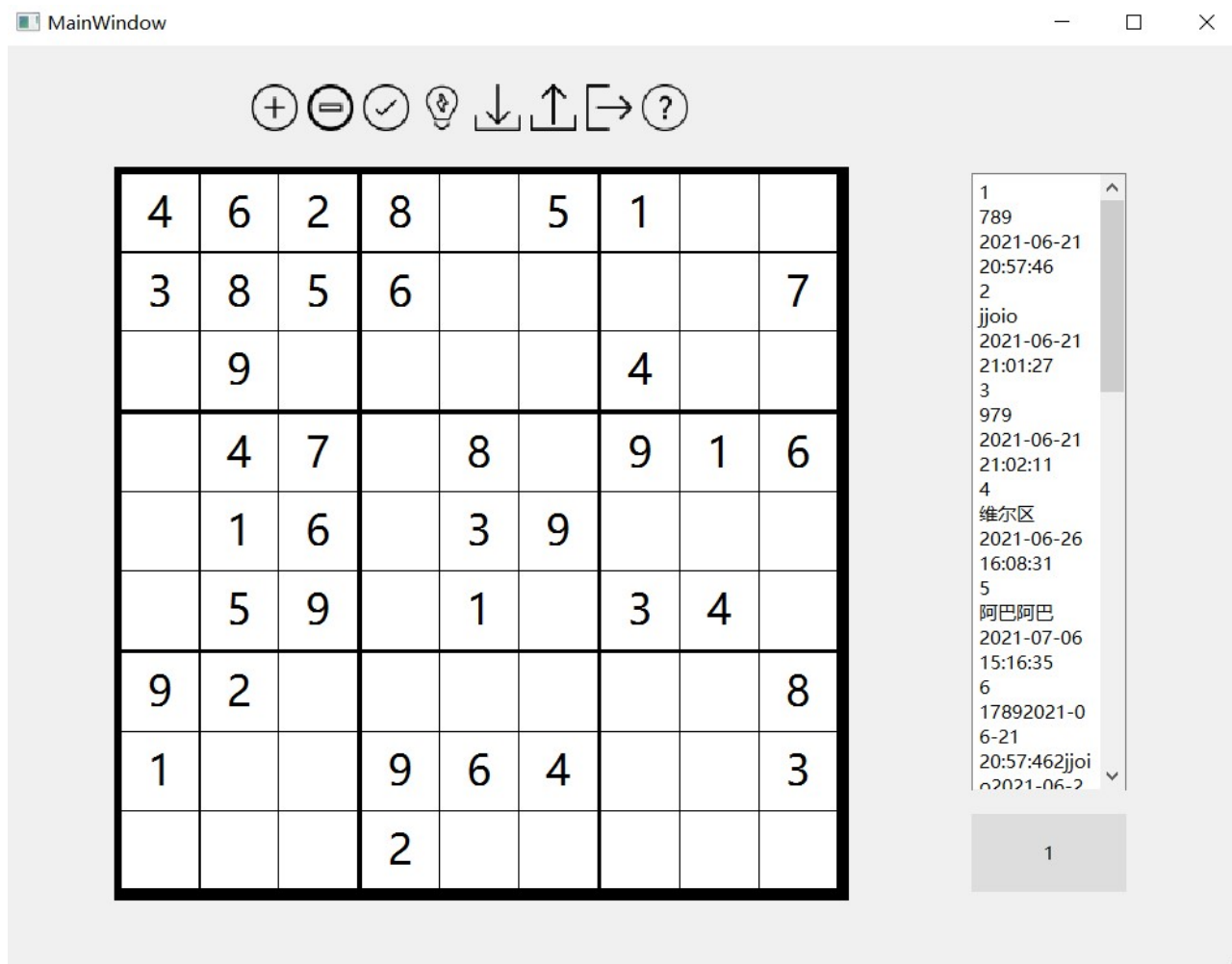
在右侧输入游戏名称后保存（名称默认不可为空）





-
- MainWindow





扩展部分

- 高级搜索

```
bool Sudoku::solve(int maxdep)
{
    for (int i = 0; i <= 8; i++)
    {
        for (int j = 0; j <= 8; j++)
        {
            if (grids_[i*span_+j])
            {
                continue;
            }
            if(i*span_+j > maxdep)
                return false;
            for (int k = 1; k <= 9; k++)
            {
                if (this->is_right(i, j, k))
                {
                    this->set(i,j,k);
                    if (this->solve(maxdep))
                        return true;
                    this->set(i,j,0);
                }
            }
        }
    }
}
```

```

        }
        return false;
    }
}
return true;
}

```

```

void Sudoku::solve_iddfs()
{
    int dep=1;
    while(!solve(dep)){
        dep++;
    }
}

```

```

void Sudoku::update_GBFS()
{
    avail.clear();
    int sum = 0;
    for(int i = 0; i < span_; i++){
        for(int j = 0; j < span_; j++){
            if(grid_[i*span_+j] && !candidates[i*span_+j][0]){
                continue;
            }
            else{
                sum=0;
                for(int k=1; k<=span_; k++){
                    if(candidates[i*span_+j][k]<0) continue;
                    if(is_right(i,j,k)){
                        candidates[i*span_+j][k]=1;
                        candidates[i*span_+j][0]=1;
                        sum++;
                    }
                    else
                        candidates[i*span_+j][k]=0;
                }
                if(grid_[i*span_+j] && candidates[i*span_+j][0])
                    candidates[i*span_+j][grid_[i*span_+j]]=0;
                avail.push_back(sum*100+i*span_+j);
                sort(avail.begin(), avail.end());
            }
        }
    }
    //for(int i=0; i<=80; i++) qDebug()<<avail[i]<<' ';
    //qDebug()<<'\n';
}

bool Sudoku::solve_GBFS()
{

```

```

    update_GBFS();
    int m = avail[0];
    int i = 0;
    int j = 0;
    int k;
    while(m>=0){
        for(i = 0,m = avail[0]; grids_[m%100/span_*span_ +
m%100%9]&& i<=80; i++) m = avail[i];
        if(i==81) return true;

        for(k=1; k<=9; k++){
            if(candidates[m%100][k]>0){
                queue.push_back(m);
                this->set(m%100/9, m%100%9, k);
                qDebug()<<m%100/9<<m%100%9<<k;
                break;
            }
        }
        if(k==10){
            qDebug()<<m;
            for(int i=1; i<=9; i++) candidates[m%100][i] = 0;
            int m_ = queue.back();
            queue.pop_back();
            qDebug()<<m_;
            int n = grids_[m_%100/span_*span_ + m_%100%9];
            this->reset(m%100/9, m%100%9);
            candidates[m_%100][n] = -1;
            update_GBFS();
            continue;
        }
        update_GBFS();
        j++;
    }
    return false;
}

void Sudoku::clear_GBFS()
{
    memset(candidates, 0, sizeof (candidates));
    avail.clear();
    queue.clear();
}

```

具体设计

函数部分

- 总览

```

class Sudoku
{

```

```

int size_, span_;
std::vector<int> avail;
//存放目前尚空的节点
std::vector<int> queue;
//存放被操作过节点, 形成队列
std::vector<int> grids_;
//存放数独
int candidates[81][10];
//存放所有位置的候选数

public:
    Sudoku(int size = 3);
    ~Sudoku() = default;
    bool is_empty() const;
    //判断是否非空
    bool is_solved() const;
    //判断是否已解出
    void clear();
    //清空存放数独的数组
    bool reset(int x, int y);
    //清除某个格子已放置的值
    bool set(int x, int y, int v);
    //设定值
    int get(int x, int y) const;
    //获取指定位置的值
    bool is_consistent(int x,int y,int v) const;
    //判断(x,y)填入v时数独是否合法
    bool solve(int dep) ;
    //求解函数 (dep为搜索深度)
    void solve_iddfs();
    //迭代加深dfs求解
    bool solve_GBFS();
    //GBFS求解
    void update_GBFS();
    //同上, 用于更新每个位置的候选数
    void clear_GBFS();
    //同上, 用于初始化
    int span() const;
    // 获取数独尺寸 (此处默认为9)
    bool is_right(int x,int y,int v);
    // 判断(x,y)位置的v是否正确合法, 用于数独的生成
    void random_sudoku();
    //随机生成数独
    static Sudoku generate(int size);
    //生成size尺寸的数独 (此处默认为3)
};

```

- 某些重要函数

生成函数

```

void Sudoku::random_sudoku()
{
    fill(grid_.begin(),grid_.end(),0);
    //初始化
    srand((unsigned)time(NULL));
    int size = 10;
    int row = 0,col = 0;
    for(;row!=9;++row){
        for(;col!=9;++col){
            int index = rand() % size-1;
            //生成0-9的随机数
            if(!is_consistent(row,col,index + 1))
                grids_[row * span_ + col] = 0;
            else grids_[row * span_ + col] = index + 1;
            //如果合法则填入, 否则填入0
        }
        col=0;
    }
}

```

求解函数 (dfs)

```

bool Sudoku::solve(int maxdep)
{
    for (int i = 0; i <= 8; i++)
    {
        for (int j = 0; j <= 8; j++)
        {
            if (grid_[i*span_+j])
            {
                continue;
            }
            /*if(i*span_+j > maxdep)
                return false;*/
            for (int k = 1; k <= 9; k++)
            {
                //1-9筛选
                if (this->is_right(i, j, k))
                {
                    this->set(i,j,k);
                    //目前合适则填入
                    if (this->solve(maxdep))
                        return true;
                    this->set(i,j,0);
                    //后续无法搜索到解则将改点重设为0
                }
            }
            return false;
        }
    }
    return true;
}

```

求解函数 (GBFS)

```

void Sudoku::update_GBFS()
{
    avail.clear();
    int sum = 0;
    for(int i = 0; i < span_; i++){
        for(int j = 0; j < span_; j++){
            if(grid_[i*span_+j] && !candidates[i*span_+j][0]){
                continue;
            }
            //candidates[][0]中的值表示是否初始状态下为空
            //1表示可以填入
            else{
                sum=0;
                for(int k=1; k<=span_; k++){
                    if(candidates[i*span_+j][k]<0) continue;
                    //当前某条路径下没有任何可能有解
                    //则该候选数设负数，不再考虑
                    //离开本次路径后将重设为0
                    if(is_right(i, j, k)){
                        candidates[i*span_+j][k]=1;
                        candidates[i*span_+j][0]=1;
                        sum++;
                    }
                    else
                        candidates[i*span_+j][k]=0;
                }
                if(grid_[i*span_+j] && candidates[i*span_+j][0])
                    candidates[i*span_+j][grid_[i*span_+j]]=0;
                avail.push_back(sum*100+i*span_+j);
                //将扩展出的可填入节点放入队列
                //sum表示候选数个数
                sort(avail.begin(), avail.end());
                //按候选数由少至多排序
            }
        }
    }
}

bool Sudoku::solve_GBFS()
{
    update_GBFS();
    //初始化
    int m = avail[0];
    int i = 0;
    int k;
    while(m>=0){
        for(i = 0, m = avail[0]; grid_[m%100/span_*span_ +
m%100%9] && i<=80; i++) m = avail[i];
        //找到位于队列最前面且尚未填入数字的节点
    }
}

```



```

        if(i==81) return true;

        for(k=1;k<=9;k++){
            if(candidates[m%100][k]>0){
                queue.push_back(m);
                //如果可填, 加入队列
                this->set(m%100/9, m%100%9, k);
                //填入当前合适的候选数, 并退出循环
                break;
            }
        }
        if(k==10){
            //如果没有候选数可填
            for(int i=1;i<=9;i++) candidates[m%100][i] = 0;
            //清除可能存在的负数, 全部归零
            int m_ = queue.back();
            //获取最后一个节点, 即目前节点的上一个
            queue.pop_back();
            //删除最后节点
            int n = grids_[m_%100/span_*span_ + m_%100%9];
            this->reset(m_%100/9, m_%100%9);
            //重设
            candidates[m_%100][n] = -1;
            //将上一个节点的该候选数设为-1, 重新考虑剩余其他候选数
            update_GBFS();
            //更新
            continue;
        }
        update_GBFS();
    }
    return false;
}

```

提示函数

```

void SudokuGrid::game_hint()
{
    Sudoku ans = *sudoku;
    //临时生成以求解
    if(ans.solve(81)){
        //若有解
        int hint[82];
        int k = 0;
        for (int i = 0; i < 9; i++)
        {
            for (int j = 0; j < 9; j++)
            {
                if (sudoku->get(i,j))
                    continue;
                hint[k++] = 9 * i + j;
                //获取初始空的位置
            }
        }
    }
}

```

```

    }
    if (!k){
        QMessageBox::warning(parentWidget(),"warning","already
solved");
        return ;
    }
    int zero = rand() % k;
    //随机生成
    int pos = hint[zero];
    cells[pos]->emit_selected_signal();
    cells[pos]->set_value(ans.get(pos/9,pos%9));
}
else QMessageBox::warning(parentWidget(),"warning","can't be solved");
}

```

图形化部分

- 生成子格

总览

```

class SudokuRoom : public QLabel
{
    //基类为QLabel, 可以进行读写操作
    Q_OBJECT
public:
    explicit SudokuRoom(int row, int col, std::shared_ptr<Sudoku> sudoku,
QWidget *parent = 0);

    void set_value(int v);
    void set_initial_status(int v);
    //设定初始状态便于后续判断
    bool is_initial_status() const;
    //判断是否为初始状态
    void emit_selected_signal();
    //发射被选中信号
    void remove_value();
    bool check_value(int v);
    //检查输入的值
    void light_value(int);
    //点亮数独中所有与选中值相同的数字

    int get_row() const { return row; }
    int get_col() const { return col; }
    int get_value() const {return sudoku->get(row,col);}

signals:

    void selected_signal(SudokuRoom*);

public slots:

```

```

void free_selection();
//释放当前选中的room
void selected();
//更新选中者的状态

protected:
    void mousePressEvent(QMouseEvent*);
    void keyPressEvent(QKeyEvent*);
    void paintEvent(QPaintEvent*);
    //自动调用，用于绘制Label

private:
    void update_text(int);
    //更新Label的显示数字
    void update_font();
    //更新字体
    void update_style();
    //更新Label的整体样式

private:
    int row, col, initial_status;
    std::shared_ptr<Sudoku> sudoku;
    //提供一个智能指针，使得其他所有子格都能拥有同一个数独状态
    //并能进行相关操作
    bool is_lighted;
    bool is_selected;
};

```

- 生成九宫格

```

class SudokuGrid : public QLabel
{
    Q_OBJECT

public:
    explicit SudokuGrid(int cell_size, int fixed_size, QWidget *parent =
0);
    ~SudokuGrid();

    int get_value(int row,int col) const {return sudoku->get(row,col);}
    void status_reset(int x,int y,int v);
    void thread_set(Sudoku);
    Sudoku initial_sudoku;
    //grid只需用一个，故无需指针

public slots:

    //按钮响应函数
    void light_value();
    void free_selection();
    void game_start();
    void game_reset();

```

```

void game_clear();
void game_solve();
void game_solve_plus();
void game_solve_GBFS();
void game_hint();

void cell_selected(SudokuRoom*);
void move_focus(int);

private:

int cell_size, cell_span, fixed_size;
QGridLayout *top_layer;
//Qt自带网格布局, 是九宫格生成核心
SudokuRoom *current_selected;
std::vector<SudokuRoom*> cells;
//指针数组, 存放81个子格
std::shared_ptr<Sudoku> sudoku;
};

```

构造函数

```

SudokuGrid::SudokuGrid(int cell_size, int fixed_size, QWidget *parent)
: QLabel(parent),
  cell_size(cell_size),
  cell_span(cell_size * cell_size),
  fixed_size(fixed_size),
  current_selected(nullptr),
  sudoku(std::make_shared<Sudoku>(cell_size))
{
  top_layer = new QGridLayout(this);
  //赋予空间
  cells.assign(cell_span * cell_span, nullptr);
  for(int r = 0; r != cell_size; ++r)
    for(int c = 0; c != cell_size; ++c)
    {
      QGridLayout *grid = new QGridLayout;
      top_layer->addLayout(grid, r, c);
      //先设置9个网格布局控件
      grid->setSpacing(1);

      for(int x = 0; x != cell_size; ++x)
        for(int y = 0; y != cell_size; ++y)
        {
          int row = r * cell_size + x;
          int col = c * cell_size + y;
          SudokuRoom *cell = new SudokuRoom(row, col, sudoku);
          cell->setFocusPolicy(Qt::ClickFocus);
          cell->setFixedSize(fixed_size, fixed_size);
          cell->setTextFormat(Qt::PlainText);
          cell->setAlignment(Qt::AlignCenter);
          grid->addWidget(cell, x, y);
        }
    }
}

```

```

        //再添加9个子格
        cells[row * cell_span + col] = cell;
    }
}
top_layer->setSpacing(2);
top_layer->setContentsMargins(4,4,4,4);
setStyleSheet(QString("background-color: ") + "#000000" + ";");

int size = (fixed_size + 1) * cell_span + cell_size * 2;
setFixedSize(size+10, size+10);
}

```

显示生成数独

```

void SudokuGrid::game_start()
{
    *sudoku = Sudoku::generate(cell_size);
    initial_sudoku = *sudoku;
    //设定数独初始状态
    game_reset();
}

void SudokuGrid::game_reset()
{
    free_selection();
    *sudoku = initial_sudoku;
    for(int r = 0; r != cell_span; ++r)
        for(int c = 0; c != cell_span; ++c)
        {
            int id = r * cell_span + c;
            cells[id]->set_initial_status(initial_sudoku.get(r, c));
        }
}

```

点亮函数

```

void SudokuGrid::light_value()
{
    if(current_selected)
    {
        int value = current_selected->get_value();
        for(SudokuRoom * cell : cells)
            cell->light_value(value);
    }
}
.....
void SudokuRoom::light_value(int v)
{
    if(v && this->get_value() == v){

```

```

        is_lighted = true;
    }
    else
        is_lighted = false;
    update_style();
}

```

其余函数思路大抵一致，通过操作每个cell以达到显示与数独状态同步

- 生成按钮

总览

```

class Button : public QPushButton
{
//QPushButton类，通过点按实现相应功能
    Q_OBJECT
public:
    Button(QWidget* parent = 0);
    void set_image(QString);
    //为按钮添加图标
protected:
    void paintEvent(QPaintEvent *);
    //绘制按钮
    void mousePressEvent(QMouseEvent *);
    void mouseReleaseEvent(QMouseEvent *);
    void enterEvent(QEnterEvent *);
    //鼠标移入
    void leaveEvent(QEvent *);
    //鼠标移出
private:
    bool is_mouse_pressed, is_mouse_over;
    QString image_path;
    //图标上载路径
};

```

按钮图标设置

```

void Button::paintEvent(QPaintEvent *)
{
    QRect rect = this->rect();
    QPainter p(this);
    QPixmap bg(image_path);
    bg = bg.scaled(rect.width(), rect.height(),
                   Qt::KeepAspectRatio, Qt::SmoothTransformation);
    //自适应改变图片比例并饱满填充(按比例填充)
    p.drawPixmap(QPoint(0, 0), bg);
    if(is_mouse_pressed)
    {
        p.setCompositionMode(QPainter::CompositionMode_Multiply);
        //按下按钮时图标颜色变深
    }
}

```

```

        p.drawPixmap(QPoint(0, 0), bg);
    }
    else if(is_mouse_over) {
        p.setCompositionMode(QPainter::CompositionMode_HardLight);
        //悬停时图标变亮
        p.drawPixmap(QPoint(0, 0), bg);
    }
}

```

- 生成状态保存输入框

总览

```

class recbtn : public QLabel
{
    //需要输入故使用QLabel类
    Q_OBJECT
public:
    explicit recbtn(int rec, QWidget *parent = 0);
    void update_text(int x);
    //更新显示, 对输入做出反应
    QString str;
    //主函数通过将str转化为数字判断需要读取的存档
signals:
    void selected_signal(recbtn*);
public slots:
    void selected();
protected:
    void mousePressEvent(QMouseEvent*);
    void keyPressEvent(QKeyEvent*);
private:
    bool is_selected;
    int rec;
};

```

- 生成主界面

总览

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <vector>
#include <QtWidgets>
#include <grid_of_sudoku.h>
#include <button.h>
#include <rec_button.h>
#include <thread.h>

QT_BEGIN_NAMESPACE

```

```
namespace Ui { class MainWindow; }

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

public slots:
    //与不同按钮关联
    void game_start();
    void game_clear();
    void game_exit();
    bool game_save();
    //储存游戏状态
    void game_solve();
    void game_solve_plus();
    void game_load();
    //读取游戏存档与上传
    void game_hint();
    void game_help();
    void rec_selected(recbtn*);
    void rec_chosen(QString);

private:
    void init_widgets(int);
    //生成主界面
    void set_tool_enable(bool);
    //将按钮设为可以选取状态

private:
    Ui::MainWindow *ui;
    QWidget *window;
    QVBoxLayout *layout, *rec_layout;
    //垂直布局，用于摆放存取操作所需文本框与Label
    QHBoxLayout *top_layout, *bottom_layout;
    //水平布局，用于摆放按钮
    SudokuGrid *grid;
    //生产九宫格
    Button *start_btn, *exit_btn, *hint_btn, *clear_btn;
    Button *save_btn, *load_btn, *solve_btn;
    Button *help_btn;
    QPlainTextEdit *save_rec;
    //文本框，用于显示与输入
    recbtn *rec_btn;
};
```

主界面生成函数


```

void MainWindow::init_widgets(int cell_size)
{
    int fixed_size=50;
    //设定固定尺寸
    window=new QWidget;
    setCentralWidget(window);

    layout = new QVBoxLayout(window);
    top_layout=new QHBoxLayout;
    bottom_layout=new QHBoxLayout;

    layout->addLayout(top_layout);
    layout->addLayout(bottom_layout);

    grid = new SudokuGrid(cell_size,fixed_size);
    bottom_layout->addWidget(grid);

    save_rec = new QPlainTextEdit;

    bottom_layout->setSpacing(15);
    //设置存取布局
    rec_layout = new QVBoxLayout;
    rec_layout->setContentsMargins(4,4,4,4);

    rec_btn = new QPushButton();
    QPushButton *rec = rec_btn;
    rec->setFocusPolicy(Qt::ClickFocus);
    rec->setTextFormat(Qt::PlainText);
    rec->setAlignment(Qt::AlignCenter);

    //设置按钮样式
    start_btn = new QPushButton;
    start_btn->set_image("D:/icons/new.png");
    start_btn->setToolTip("New Game(random)");
    connect(start_btn, SIGNAL(clicked()), this, SLOT(game_start()));
    .....
    //添加按钮
    top_layout->addWidget(start_btn);
    .....
    //初始化游戏
    grid->game_start();
}

```

退出游戏

```

void MainWindow::game_exit()
{
    int button;
    button = QMessageBox::question(this, tr("退出程序"),
                                   QString(tr("是否保存? ")),
                                   QMessageBox::Yes | QMessageBox::No);
}

```

```

        if (button == QMessageBox::No) {
            close();
        }
        else if (button == QMessageBox::Yes) {
            if(game_save()) close(); //接受退出信号，程序退出
        }
    }
}

```

保存游戏状态（较复杂故文字说明）

```

bool MainWindow::game_save()
{
    首先提示在右侧文本框中输入游戏名称；
    然后打开存放游戏记录的文件，读取目前已有多少存档，以便给目前游戏赋予一个新的序号；
    之后打开一个新的目标文件用于存放数独；
    输出保存成功。
}

```

此电脑 > Data (D:) > record

名称	修改日期	类型	大小
1.txt	2021/6/21 20:57	文本文档	1 KB
2.txt	2021/6/21 21:01	文本文档	1 KB
3.txt	2021/6/21 21:02	文本文档	1 KB
4.txt	2021/6/26 16:08	文本文档	1 KB
5.txt	2021/7/6 15:16	文本文档	1 KB
6.txt	2021/7/6 15:22	文本文档	1 KB
7.txt	2021/7/6 15:22	文本文档	1 KB
8.txt	2021/7/6 21:21	文本文档	1 KB
9.txt	2021/7/7 9:13	文本文档	1 KB
10.txt	2021/7/7 9:48	文本文档	1 KB
11.txt	2021/7/7 11:49	文本文档	1 KB
12.txt	2021/7/7 11:50	文本文档	1 KB
13.txt	2021/7/13 13:10	文本文档	1 KB
14.txt	2021/7/13 13:13	文本文档	1 KB
15.txt	2021/7/13 13:16	文本文档	1 KB
record.txt	2021/7/13 13:16	文本文档	1 KB

读取游戏状态

```

void MainWindow::game_load()
{
    QString str;
    //如果没有输入任何序号
    //没有存档则输出“没有存档”
}

```

```
//若有则在文本框中输出存档记录
if(rec_btn->str==""){
    QFile sfile("D:/record/record.txt");
    sfile.open(QIODevice::ReadOnly | QIODevice::Text);
    str = sfile.readAll();
    if(str=="") {
        save_rec->setPlainText("没有存档");
        sfile.close();
        return ;
    }

    else {
        sfile.open(QIODevice::ReadOnly | QIODevice::Text);
        save_rec->setPlainText(str);
        sfile.close();
        return ;
    }
}

game_clear();
//序号不对则输出错误信息
QFile rfile("D:/record/"+rec_btn->str+".txt");
if(!rfile.open(QIODevice::ReadOnly)) {
    QMessageBox::warning(this,"warning","请输入正确的序号");
    return;
}
//读取对应文件中数独状态
QString new_str = rfile.readAll();
for(int i=0;i<9;i++){
    for(int j=0;j<9;j++){
        grid->status_reset(i,j,QString(new_str[i*9+j]).toInt());
    }
}
rfile.close();
}
```

实验过程

由于太多（以及记不住了），以及主要难点在于实现图形化——即怎样实现目的而非实现之后报错（他根本不显示所以……），报错主要集中在算法实现。

GBFS

- 问题

每次运行都导致程序强制退出

- 原因

- 在尝试失败回溯至上一节点时一直卡在同一节点——因为没有构造队列使得每次最多回溯至上一个节点
- 没有剪枝。在每一次回溯到上一个节点时总会重复尝试理应被排除的候选数导致程序崩溃

- 解决方案
 - 构造队列

```
std::vector<int> queue;
```

以存放完整的填入顺序，方便回溯

- 多设置一层约束，将理应排除的候选数设为-1，

```
if(某节点下一个节点没有任何候选数)
candidates[m_%100][n] = -1; 认为该节点目前填入候选数有误
.....
if(candidates[i*span_+j][k]<0) continue;
```

实验总结

- 首先当然是Qt的使用。看教材属实是一头雾水，只能面向csdn编程（误）。实际实现时发现纯代码设计ui有时比利用qt自带图形设计更方便一些（估计是不费脑子的原因）。但遗憾的是学到的只是几个需要被使用的类的使用，日后如果想要使用还需大量的时间.....
- 其次是对c++的熟悉。qt是纯c++，所以事实上一开始最简单的程序都看着很吃力.....因为全都是通过类实现，并且还有不明所以的const类型变量或是成员函数。好在硬啃下来还是学到很多有趣有用的东西，比如容器的操作，比数组方便得多。等等等等。
- 最后是高级算法部分。写出来就是收获吧，以及了解到了几个典型的算法的结构，虽然不会写（说的就是A*）。