大连理工大学

面向对象编程技术大作业

课程名称： 面向对象编程技术

院 系： 创新创业学院

学 号：

姓 名：

2017年12月11日

面向对象程序设计作业

----骑士周游问题

## 一、问题描述

马踏棋盘问题（又称骑士周游或骑士漫游问题）是算法设计的经典问题之一。

考虑国际象棋棋盘上某个位置的一只“马”，它是否可能只走63步，正好走过除起点外的其他63个位置各一次？如果有一种这样的走法，则称所走的这条路线为一条马的周游路线。

国际象棋的棋盘为8\*8的方格棋盘，现将“马”放在任意指定的方格中，按照“马”走棋的规则将“马”进行移动。要求每个方格只能进入一次，最终使得“马”走遍棋盘64个方格。

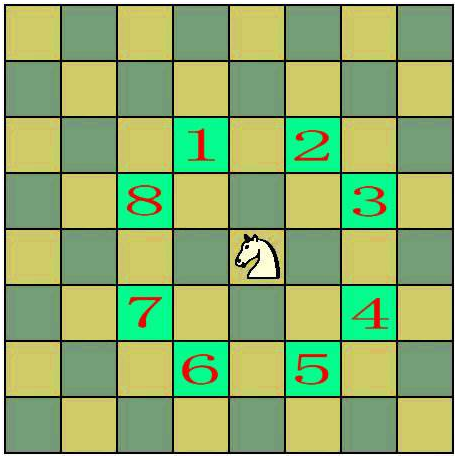


图1 国际象棋“马”的走法

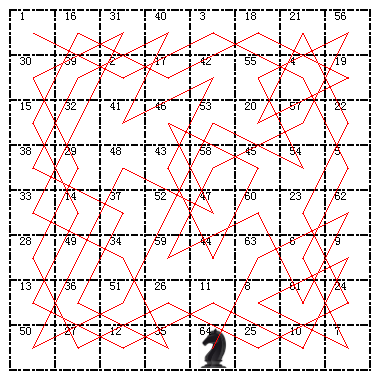


图2 骑士周游问题的一个解

## 二、问题分析

骑士周游问题是经典搜索问题，沿着一条路前进直到遍历全部的点，那就完成了整个的过程。如果不行，就回退一步，换个方向继续前进。这可以用递归很方便地实现。

注意到马在某个位置最多有8个方向可以走，因此需要对这8个方向进行试探。当然这8个方向不一定都存在，比如在边缘可能就会少一些。

## 三、算法选择

搜索算法是利用计算机的高性能来有目的的穷举一个问题的部分或所有的可能情况，从而求出问题的解的一种方法。搜索过程实际上是根据初始条件和扩展规则构造一棵解答树并寻找符合目标状态的节点的过程。

针对本问题，选用无回溯操作采的广度优先搜索算法，即使用入队和出栈的操作，牺牲内存空间来提升速度。

队列是特殊的线性表之一，它只允许在队的一端插入，在队的另一端删除。插入一端叫队尾（T），删除一端叫队首（H），没有任何元素的队叫做空队。队列遵循"先进先出"原则，排队购物、买票等，就是最常见的队。

广度优先搜索算法速度快但内存占用多，因此程序设计中，必须考虑溢出和节省内存空间得问题。

算法的主要参数有：

1.马的起始点，即左上角点坐标startX、startY，因为题目要求绘图演示过程，所以本文采用与屏幕坐标系相同的坐标系(即左上角点为(0,0)点，向右为x轴正方向，即列数；向下为y轴正方向，即行数)；

2.走的步数h\_step，初始时为0，每走一步，值加1；

3.马当前坐标位置point；

4.dir[8][2]中存储马理论上能走的8个位置。

算法程序如下：

//采用广度优先搜索算法

struct point { //用于存储马的位置

int x;//马的x方向

int y;//马的y方向

};

int Horse::\_step(int x,int y){

int i,xi,yi;

int count=0;

for( i=0;i<8;i++){

xi=x+dir[i][0];

yi=y+dir[i][1];

//边界条件以及!map[xi][yi]表示未走过的位置

if(xi>=0&&xi<=MAXN-1&&yi>=0&&yi<=MAXN-1&&!map[xi][yi])

count++;

}

return count;

}

int Horse::BFS(point s)

{

int i,x,y;

QueueInitiate(&Q); //初始化队列

QueueAppend(&Q,s); //入队操作

point hd ;

//当队列非空时，进行广度优先搜索

while(QueueNotEmpty(Q))

{

QueuePop(&Q,&hd);//出队列

record[h\_step][0] = hd.x;

record[h\_step][1] = hd.y;

map[hd.x][hd.y] = h\_step; //标记该位置已走过

//在算法运行中执行绘制

QPainter \*painter = new QPainter;

msleep(m\_runFrame); // 修改此处设置刷新频率

h\_step++; //标记该位置已走过

//记录走过的位置

int minstep=10; //初始的最小步数

int flag=0; //标记

//8个方向进行搜索

for( i=0;i<8;i++){

x=hd.x+dir[i][0];

y=hd.y+dir[i][1];

//如果小于当前最小步数，则让队列元素出队列，并让当前的元素入队列

if(x>=0&&x<=MAXN-1&&y>=0&&y<=MAXN-1&&!map[x][y])

{

if(\_step(x,y)<minstep)

{

minstep=\_step(x,y) ;

point t ,th ;

t.x=x,t.y=y ;

//如果是第一次的话，就不用出队列了；

if(flag) QueuePop(&Q,&th) ;

QueueAppend(&Q,t) ;

flag=1;

}

}

}

}

return h\_step-1;

}

## 四、方案设计

关键事件类KenEvent的设计：

class KeyEvent : public QWidget

{

Q\_OBJECT

public:

KeyEvent(QWidget \*parent = 0);

~*KeyEvent*();

void drawPix(); // 图形绘制

void *keyPressEvent*(QKeyEvent \*); // WASD按键响应，修改马起始坐标

void *paintEvent*(QPaintEvent \*); // 绘制事件

private:

QPixmap \*pix;

QImage image; // 马的图标

int width;

int height;

int m\_step; // 马走过的步数

public:

volatile int startX;

volatile int startY;

void horserunEvent(); // 马坐标更新

public slots:

void slotStart(); // 各个按键对应的槽

void slotStop();

void slotPause();

void slotPrint();

void starValueChange();

void loadData();

void saveData();

void slotPlay();

void pop(); // 滑动slider时弹出彩蛋

public:

Horse \*workhorse;//WorkThread();

};

在KenEvent类的构造函数中，完成界面背景和按键界面的绘制，初始化各个参数防止调用空指针：

KeyEvent::KeyEvent(QWidget \*parent)

: QWidget(parent)

{

workhorse = new Horse(); //初始化马类Horse

//UI设计

setWindowTitle(tr("骑士周游问题"));

QPalette palette = this->palette(); //背景绘制

palette.setColor(QPalette::Window,Qt::white);

setPalette(palette);

setMinimumSize(570,400); // 窗口大小

setMaximumSize(570,400);

width=20+45\*8+1; //绘制棋盘

height=20+45\*8+1;

pix = new QPixmap(width,height);

pix->fill(Qt::white);

image.load(":/images/002.png"); // 载入马儿图标

startX=0; // 马初始位置

startY=0;

// 按键的布局和相应初始化

startBtn = new QPushButton(tr("开始"));

……

// 使用弹簧控制页面的布局

QVBoxLayout \*mainLayout = new QVBoxLayout();

// mainLayout->addStretch();

mainLayout->addWidget(label2);

mainLayout->addLayout(layout1);

mainLayout->addStretch();

……

// 将信号与槽函数联系起来 connect(startBtn,SIGNAL(clicked()),this,SLOT(slotStart()));

……

// 绘图

drawPix();

resize(450,600);

}

每次QPainter \*painter = new QPainter，都会引发绘图函数drawPix()

，在该功能中，主要完成马跳路径的绘制，并显示路径上相应数字：

void KeyEvent::drawPix()

{

pix->fill(Qt::white);

QPainter \*painter = new QPainter;

QPen pen(Qt::black, 1.5, Qt::DashDotLine, Qt::RoundCap);

for(int i=20;i<width;i=i+m\_step) // 绘制棋盘

{

painter->begin(pix);

painter->setPen(pen);

painter->drawLine(QPoint(i,20),QPoint(i,height));

painter->end();

}

for(int j=20;j<height;j=j+m\_step)

{

painter->begin(pix);

painter->setPen(pen);

painter->drawLine(QPoint(20,j),QPoint(width,j));

painter->end();

}

painter->begin(pix);

painter->drawImage(QPoint(startX+28,startY+23),image); // 画小马图标

painter->end();

if(flag\_startBtn || flag\_loadBtn) // 如果开始按键被按下，即进去算法执行中，再画一个数字记录路径

{

flag\_loadBtn = 0;

// 绘制马跳路径上的数字

for(int i=0;i<MAXN;i++)

{

for(int j=0;j<MAXN;j++)

{

if(workhorse->map[i][j]<= workhorse->h\_step && workhorse->map[i][j]!=0)

{

painter->begin(pix);

painter->drawText(QPoint(j\*45+30,i\*45+30), QString::number(workhorse->map[i][j]));

painter->end();

}

}

}

// 绘制马跳路径的折线

for(int i = 1; i <= workhorse->h\_step; i++)

{

painter->begin(pix);

painter->setPen(Qt::red);//设置钢笔颜色

painter->drawLine(QPoint(workhorse->record[i-1][1]\*45+43,

workhorse->record[i-1][0]\*45+43),

QPoint(workhorse->record[i][1]\*45+43,

workhorse->record[i][0]\*45+43));

painter->end();

}

// 在文本框中实时打印当前位置

if(flag\_startBtn == 1 && flag\_pauseBtn == 0)

{

int temp = workhorse->h\_step;

textEdit->setText("当前步数:"+QString::number(temp-1)+

"\n"+"当前位置:"+QString::number(workhorse->record[temp][0]+1)+"行,"

+QString::number(workhorse->record[temp][1]+1)+"列");

}

}

}

在各个按键的响应函数中，完成相应的功能：

//"开始"按键响应函数

void KeyEvent::slotStart()

//"停止"按键响应函数

void KeyEvent::slotStop()

//"暂停继续"按键响应函数

void KeyEvent::slotPause()

//"打印"按键响应函数

void KeyEvent::slotPrint()

//保存程序运行的结果

void KeyEvent::saveData()

//加载程序运行保存的结果

void KeyEvent::loadData()

//"play"按键响应函数，用于打开串口发送数据进行下位机演示

void KeyEvent::slotPlay()

马类Horse的设计：

class Horse : public QThread,public QPainter

{ Q\_OBJECT

public:

Horse();

int iX; // 起始点

int iY;

void setStartValue(int , int ); // 设置起始点

protected:

void *run*();

public:

struct point {

int x;//马的x方向

int y;//马的y方向

};

private:

typedef struct Queue{

struct point queue[MaxQueueSize];

int front;//头指针

int rear;//尾指针

int tag;//设置标记位

}SeqCQueue;

SeqCQueue Q;

//初始化队列操作

void QueueInitiate(SeqCQueue \*Q);

//判断队列是否为空

int QueueNotEmpty(SeqCQueue Q);

//入队操作

void QueueAppend(SeqCQueue \*Q,point x);

//出队操作

void QueuePop(SeqCQueue \*Q,point \*d);

public:

//int m\_startX,m\_startY;//起始位置

volatile int h\_step;//记录马走的步数

int map[MAXN][MAXN];//8\*8的国际象棋棋盘

volatile int record[64][2];//记录每步的位置

dir[][]={{-2,-1}，{-2,1},{-1,-2},{-1,2},{1,-2},{1,2},{2,-1},{2,1}}

//马理论上所能走全部的位置

private:

//求出该位置的马所能走的步数

int \_step(int x,int y);

int BFS(point s);

public:

int work(); //算法执行函数

int m\_runFrame; //播放速度

QMutex pause; //信号量pause用于锁住循环执行部分

};

设计保存和载入功能，保存时，只需记录马跳的步数和响应坐标参数即可，主要参数包括：

1. 马跳的步数 volatile int h\_step;
2. 棋盘坐标 int map[MAXN][MAXN]；
3. 每一步位置 volatile int record[64][2];

载入时，恢复相应的数据，重新进行一次覆盖，即可恢复到保存前的状态，可以继续执行。

系统UI与算法的多线程通过以下方式实现，先执行完算法保存结果在map和record中，再进行图形绘制，在while循环中便于增加信号量锁并保存中间结果：

if (flag\_loadBtn\_start ==1 )

while(h\_step < 64)

{

pause.lock();

QPainter \*painter = new QPainter; // 进行绘制

msleep(m\_runFrame); // 刷新动画频率

h\_step ++;

pause.unlock();

}

CDialog类中实现初始化串口，并发送马的路径信息给下位机，串口应用了QSerialPort类，未避免繁琐的工作，串口只提供端口和波特率的选项，其他参数如校验位等在程序中默认写死：

class CDialog : public QDialog

{

Q\_OBJECT

public:

~*CDialog*();

public:

explicit CDialog(QWidget \*parent = 0);

QStringList AvailablePorts(); // 端口是够可用

QString playOutput; // 发送的信息

public slots:

void OnOpenSerialClicked(); // 点击信号对应的槽函数来

void OnCloseSerialClicked();

void OnPlaySerialClicked();

private:

QSerialPort \*mySerialPort;

QLabel \*port; // 端口

QComboBox \*portBox;

QLabel \*baudRate; // 波特率

QComboBox \*baudRateBox;

// 只提供端口和波特率的选项，其他参数如校验位等在程序中默认写死

QPushButton \*openSerial; // 打开串口

QPushButton \*playSerial; // 演示

QPushButton \*closeSerial; // 关闭串口

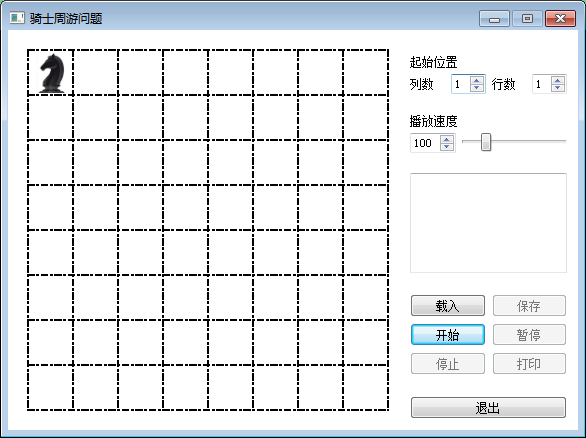
public:

void mySetBaudRate(int baudRate); // 设置波特率

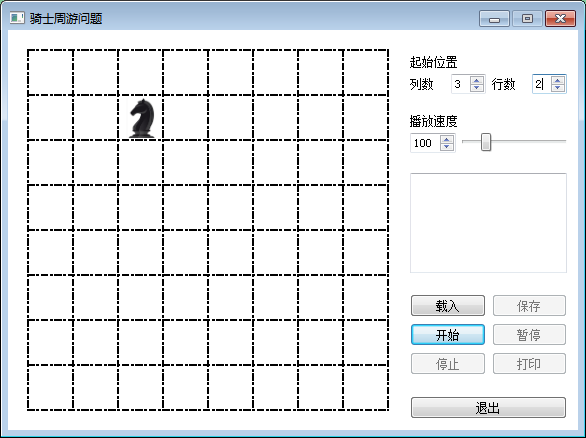
};

## 五、运行截图

### 1.初始界面



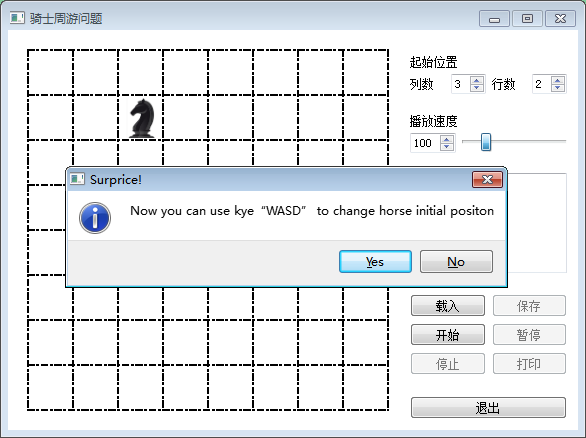
### 2.设置棋盘初始位置，此处为3行4列



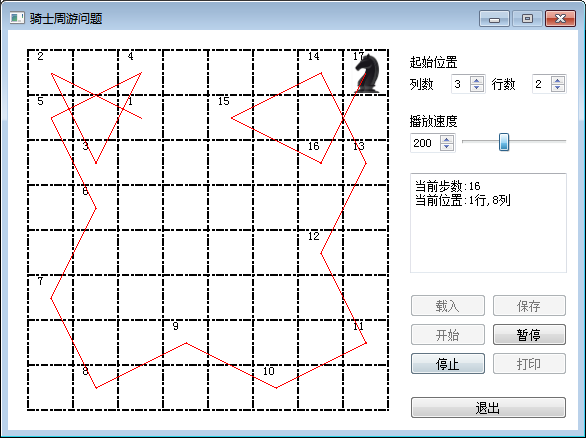
### 3.设置动画速度，此处速度值为200

### 

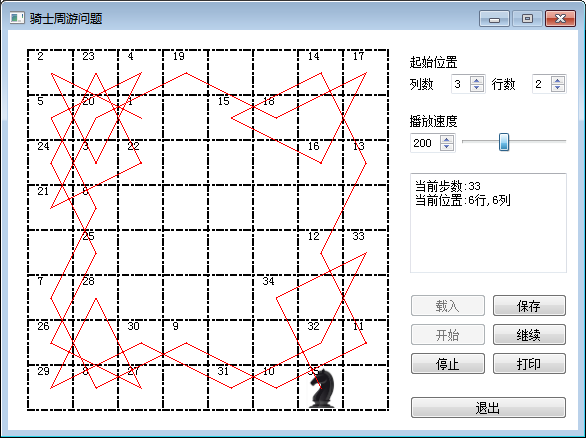
### 如果用户使用slider设置动画速度，会弹出一个彩蛋，提示可以使用WASD键设置马的初始位置了。



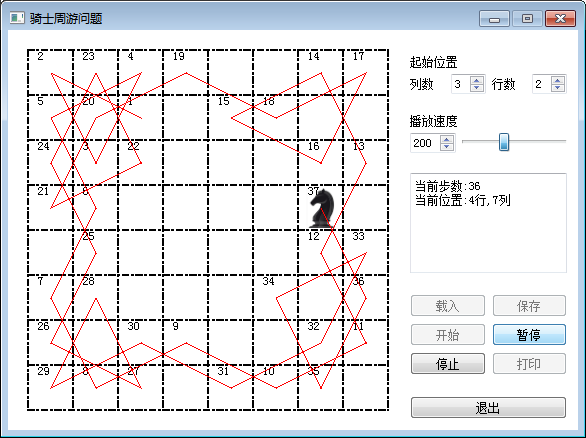
### 4.点击开始按键，进行演示，实时图形化显示执行结果，并实时在文本框中打印当前步数和位置



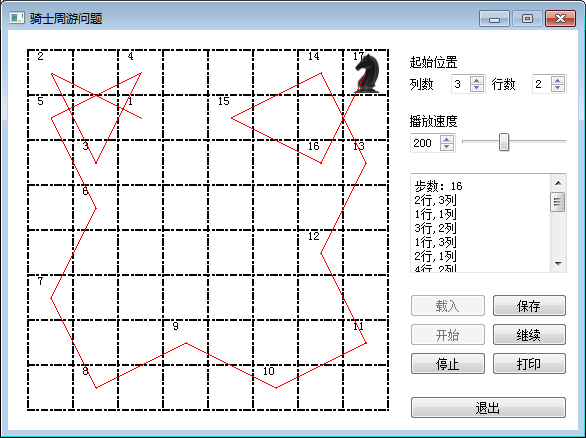
### 5.可以暂停



### 6.可以继续

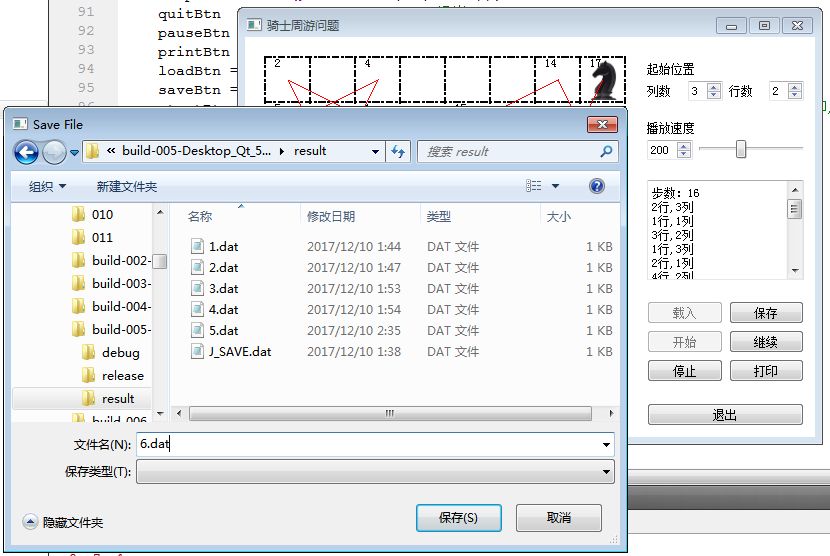


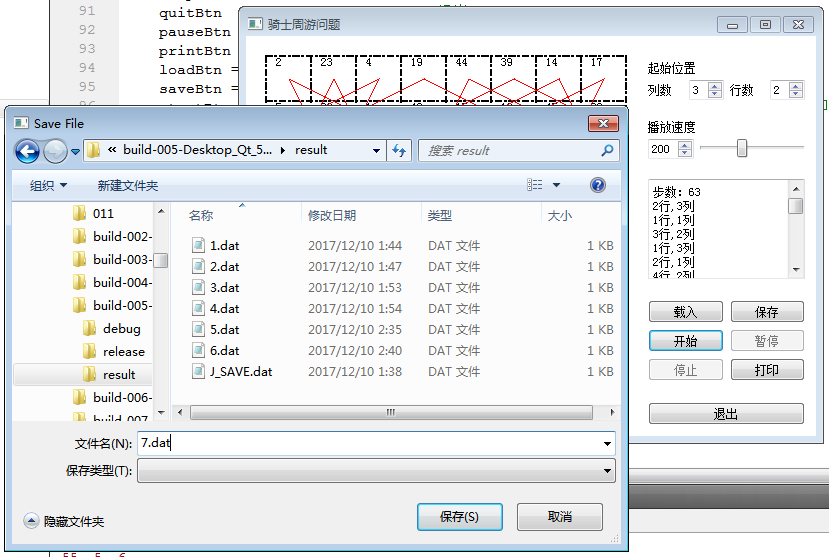
### 7.可在执行完成后，或暂停时，在文本框中打印执行的当前结果



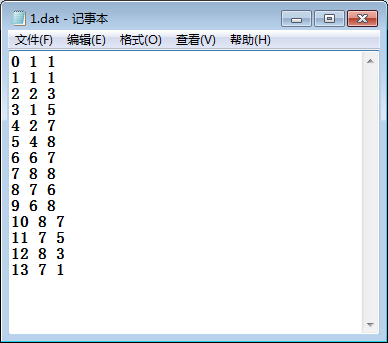
### 

### 8.保存当前状态

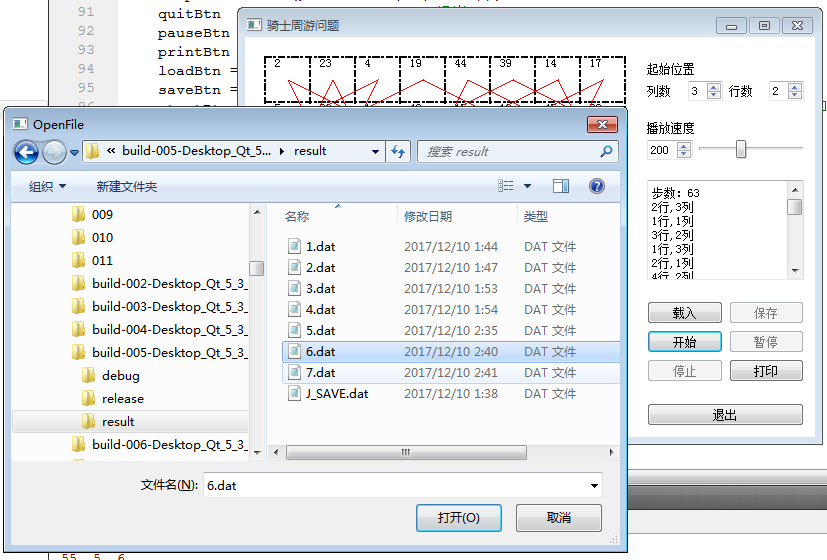


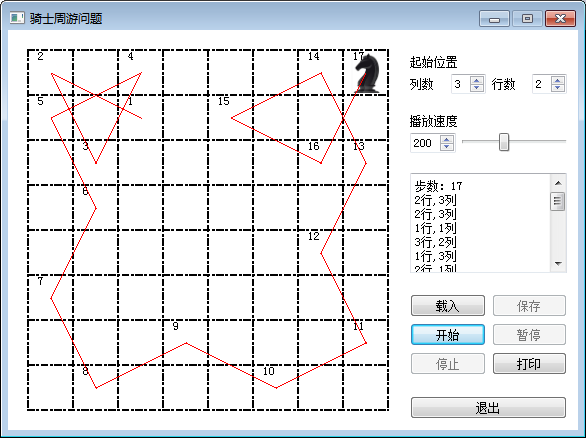


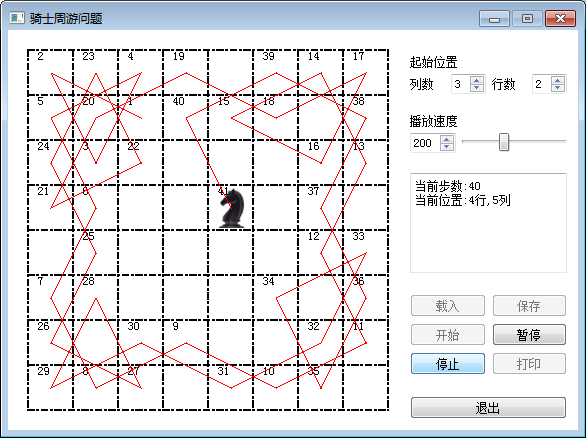
### 9.保存的结果



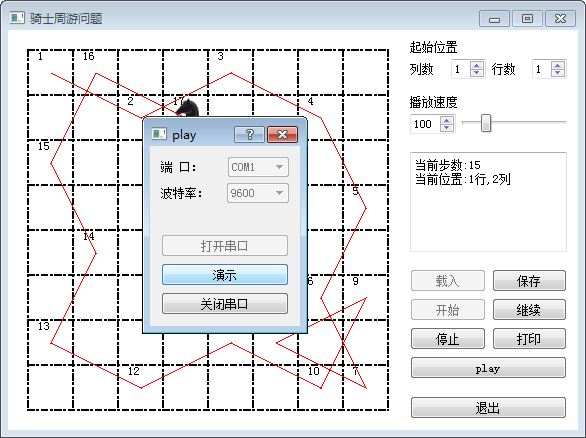
### 10.载入保存的结果



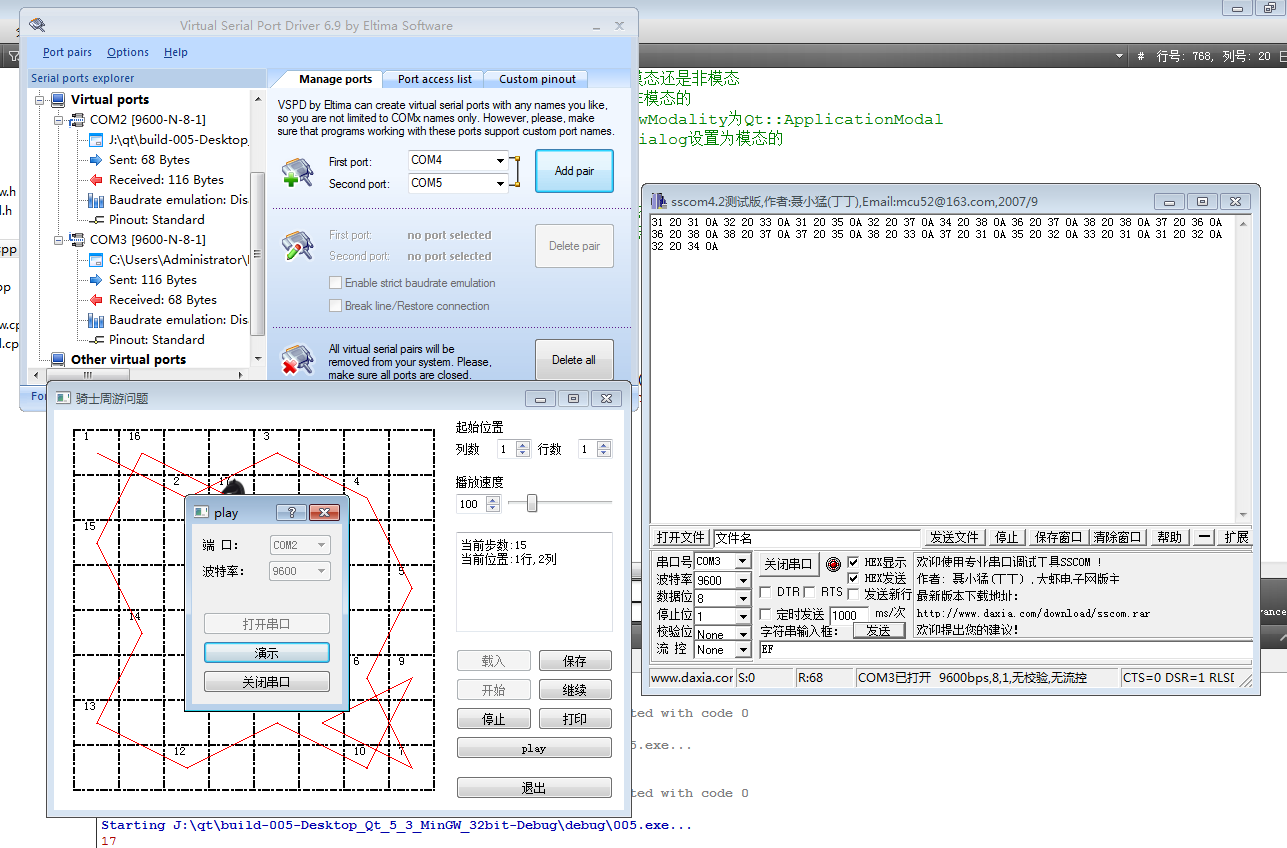




### 11.发挥项，通过串口发送记录的路径给下位机



### 通过虚拟串口软件，查看演示功能中发送给下位机的信息。



### 11.下位机解析数据后在LCD屏显示结果

