**骑士周游程序设计文档**

院 系：

班 级：

学 生 姓 名：

学 号：

完 成 日 期： 2018.12.13

大连理工大学

# 1 骑士周游问题概述

## 1.1 问题描述

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。试设计一个算法找出这样一条马的周游路线。

此外，程序需要提供完整的图形界面，可设置棋盘初始位置、动画速度、实时显示执行过程、多线程运行、暂停和继续算法、断点保存与读取、打印当前执行结果等功能。

## 1.2 问题分析

1、棋盘的构建

在该问题中首先需要构建棋盘，国际象棋的棋盘由一个8×8的矩阵表示，需要标注该位置是否已走过以及经过的次序。

棋盘初始值均为0，从第一步开始，将经过的单元格置为当前步数，在标记的同时表明了行走次序。另外，在可视化界面中每个棋盘格对应的是像素坐标值18-482，而为了方便64格棋盘的访问，将其抽象为索引号0-7，所以需要建立索引到像素坐标之间的映射。

2、棋子的构建

由于棋盘格已经抽象为索引号，棋子只需要记录在棋盘中的位置索引即可，此外，需要定义棋子的行动方式，也就是国际象棋中骑士的方式，共有八个方向：（1，2）、（2，1）、（2，-1）（1，-2）（-1，-2）（-2，-1）（-2，1）（-1，2），将运动方式预先设定好，由于这些也是基于索引号，x+offset[i]、y+offset[i]这样直接加上增量即可完成棋子的运动。

3、多线程

由于本程序的图形化界面中有比较丰富的操作，周游路径的计算也是持续进行的。在打开模态对话框或比较占用运算资源的操作时，程序会发生阻塞情况，显然通过计时器回调函数是解决不了这一问题的，多线程就很必要。

本程序中将算法置于新的线程中运行，GUI及其他操作运行在主线程中，根据用户操作动态创建与销毁线程。

4、算法参数设置

骑士初始位置、算法运行速度等参数由界面对象设置，并通过Qt的信号槽机制，传递给算法对象。

# 2 算法设计说明

## 2.1 算法原理分析

本程序采用贪心策略的递归算法实现，具体过程如下：

1. 初始化棋盘：棋盘矩阵、起始位置、移动方向、运行速度及状态初始化。
2. 调用递归函数：根据当前点Xn计算周围八方向可行点的可行价值，根据价值进行降序排序，按照该顺序遍历，得到Xn+1 ，进入下一层递归，递归步数step加一。

可行价值计算函数：遍历当前点周围的八个点，如果在棋盘中且未走过，则价值加一，即价值区间[0, 8]，所以0表示周围没有可行点。

1. 如果点Xn+1周围八点均不可行，即价值均为0，则跳出该层归层，当前步数step减一，直至回溯到可行位置，继续进行。
2. 递归步数step达到64步时，整个棋盘被走遍，递归结束，逐层跳出。

## 2.2算法核心代码

递归函数体：

void BackTrace(int x, int y, int \_step)

{

if (isStopped) return;

if (\_step>=64) //执行到第64层时停止，跳出递归函数

{

isStopped = true; //stop标志位，控制每一层递归跳出

isPause = true;

return; //结束运行

}

else //递归主体

{

vector<node> outway(8);

outway = ALGThread::sort\_Step(x, y);//获取八可行点价值

for (auto node : outway) //八方向遍历搜索

{

int next\_x = x + next\_jump[node.index][0];

int next\_y = y + next\_jump[node.index][1];

if (canJump(next\_x, next\_y))

{

\_step++;

path[next\_x][next\_y] = \_step;//记录占用及次序

updateIndexes();//根据当前次序，更新显示队列

emit updateDisplay(\_step - 1);//通知界面显示更新

msleep(1600 - 15 \* algSpeed);//运行速度控制

while(isPause) { //暂停控制

msleep(1000);

}

BackTrace(next\_x, next\_y, \_step); //继续下一层递归

}

}

}

}

价值计算函数：

vector<node> ALGThread::sort\_Step(int r , int c)

{

vector<node> outway(8);

// 周围八点价值初始化为0

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

outway[i].value = 0;

outway[i].index = i;

}

int nextr, nextc;

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

nextr = r + next\_jump[i][0];

nextc = c + next\_jump[i][1];

if(nextr>=0 && nextc>=0 && nextr<8 && nextc<8 && path[nextr][nextc] == 0) {

//判断是否可行

for(int j = 0 ; j < 8 ; j++) {

int a = nextr + next\_jump[j][0];

int b = nextc + next\_jump[j][1];

if(a>=0 && b>=0 && a<8 && b<8 && path[a][b]==0)

outway[i].value++;//有可行点则价值加一

}

}

}

stable\_sort(outway.begin(),outway.end(),cmp);//降序排序

return outway;//返回可行价值降序序列

}

# 3 程序设计说明

## 3.1 界面设计

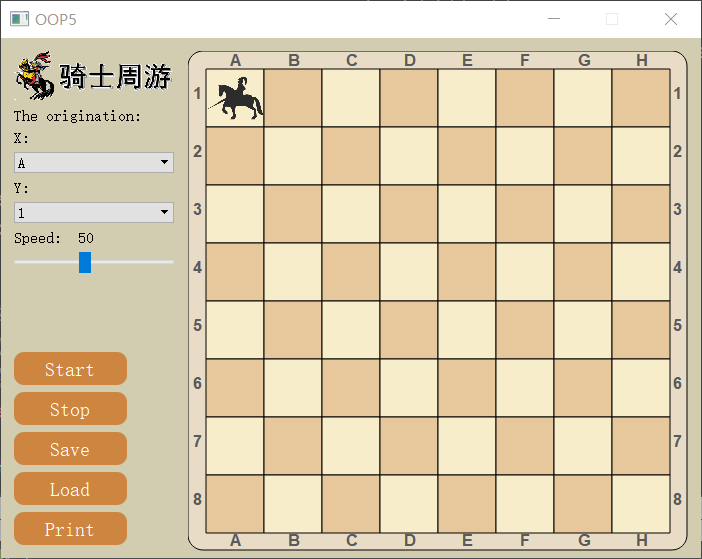


图3.1 程序界面

本程序由Qt编写，主界面有左侧工具栏以及右侧显示窗口。

工具栏包含初始位置设置、速度设置、开始/暂停、进度保存、进度读取、打印当前结果等功能。

Start可以开始运行算法，期间会在Continue、Pause之间变化；Stop可以停止当前算法，开始按键恢复Start标志；速度设置可以在任意时刻设置；初始位置设置在算法开始后无效。

显示窗口显示当前算法进度，并且在算法开始前可响应按键选择初始位置。

## 3.2 加载与保存

本程序提供进度保存与加载功能。

在算法运行过程中可以点击Save按键将当前进度保存为如下格式：

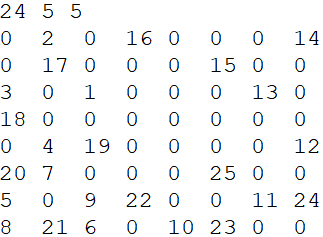


图3.2 进度存储格式

第一行信息依次为当前迭代步数、骑士当前行索引、骑士当前列索引。之后的8×8矩阵为棋盘信息，保存了骑士行走的次序信息。

点击Load按键可以加载如上格式的文件，复现所存进度，如果再点击Start可以继续断点继续运行算法，加载效果如下：

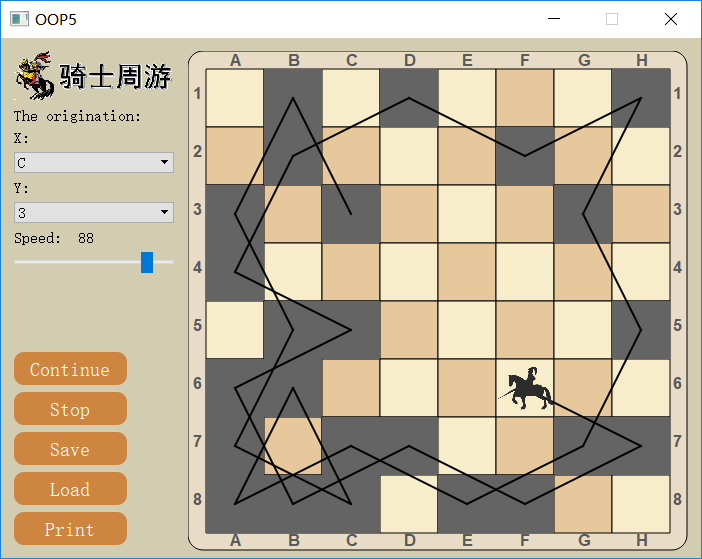


图3.3进度加载效果

加载图3.2中数据的显示结果如图3.3所示，再点击Start按键就可以在此基础上继续运行。程序中，算法可以正常接受start的信号，以默认值开始运行，即就是从初始位置开始运行，当读取断点信息后，会将迭代步数、骑士索引、棋盘信息传递给算法对象，复现文件中保存的运行状态，从而继续运行。

最终的运行结果如图3.4所示。

## 3.3 运行结果打印

如下图3.4所示，将运行结果打印至pdf文件中：

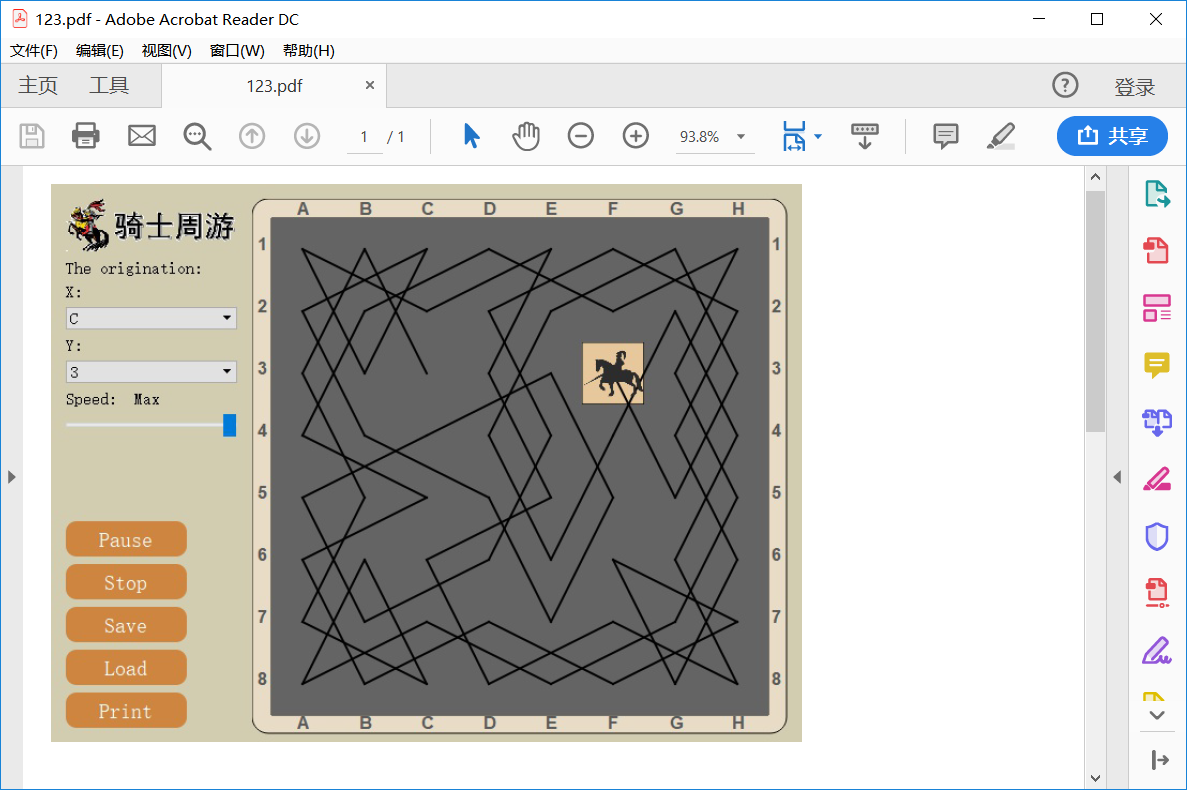


图3.4 运行结果图

点击Print按键进行打印，在程序的任何时刻都可以进行。程序将窗口当前的截图输出至pdf文件保存。