同济大学计算机系

人工智能原理与技术综合实验报告

学　　号　1851381 1851726 1753087

姓　　名　赵中楷　汪一泓　杨宏辉

专　　业　计算机科学与技术

授课老师　武妍

目录

一、实验概述

(1) 实验目的

熟悉和掌握启发式搜索的定义、估价函数和算法过程，并利用 A*算法求解 8 数码难题，理解求解流程和搜索顺序。

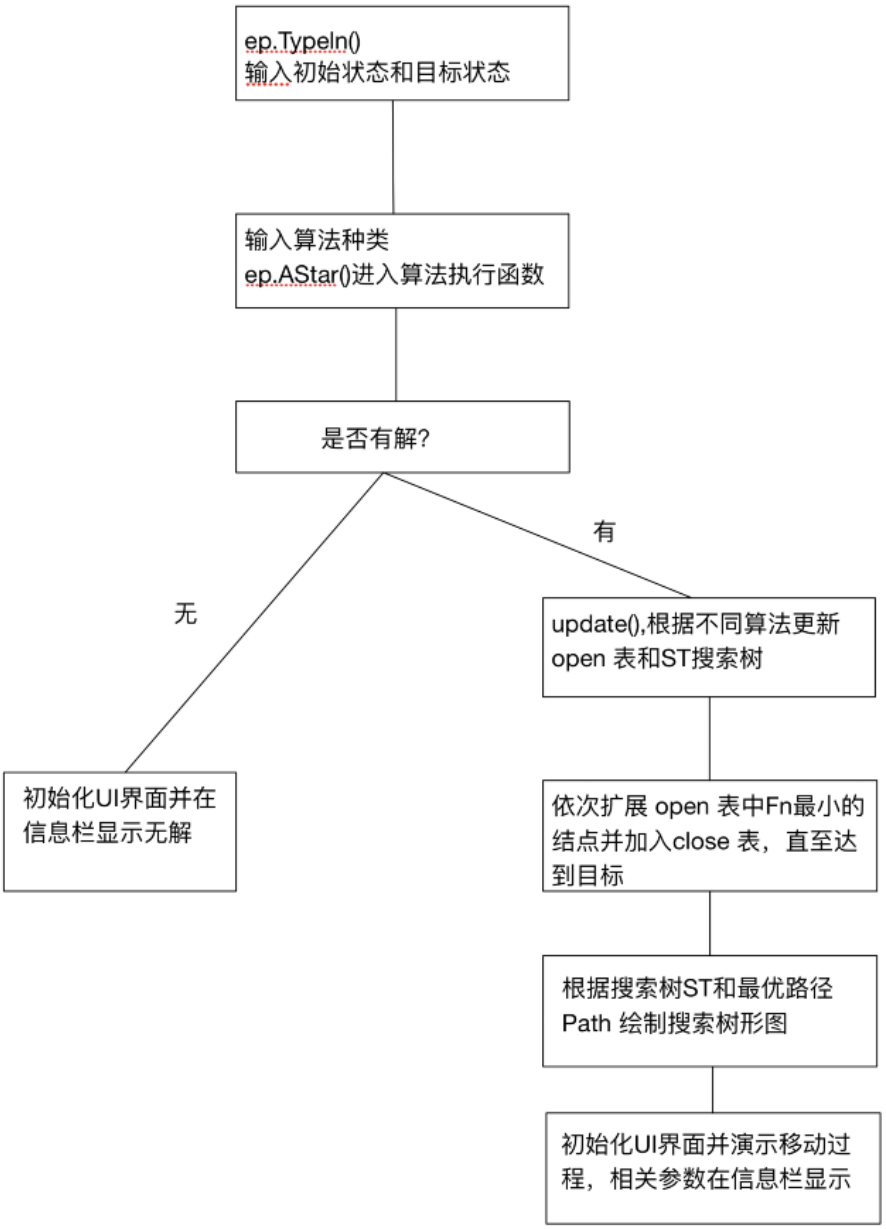
(2) 实验内容

1. 采用 C++语言实现 A*算法的求解八数码问题的程序，设计了两种不同的估价函数：第一种：计算不在位的棋子数；第二种：计算所有棋子到其目标的距离和。
2. 设置相同初始状态和目标状态，针对两种的估价函数，求得问题的解，并比较它们对搜索算法性能的影响，包括扩展节点数、生成节点数和运行时间。画出不同启发函数 h(n)求解 8 数码问题的结果比较表，进行性能分析。
3. 通过 UI 界面显示八数码问题的初始状态，目标状态和中间搜索步骤。
4. 画出搜索生成的树，在每个节点显示对应节点的 f (n)值，以显示搜索过程。以红色标注出最终结果所选用的路线。

二、实验方案设计

(1) 总体设计思路与总体架构

1. 总体设计思路：程序从设计上分为两个部分：算法实现部分和图形显示部分。两部分相互独立，仅通过函数进行数值传递或者指针传递的方式进行关联。当算法部分运行完成后，会得到关于一个八数码问题的最优路径解、搜索树以及求解过程中产生的例如生成结点数、扩展结点数的信息，将这些信息以数值、指针、字符串等形式传值至图形显示部分。
2. 总体架构：总体架构设计基于上述的设计思路。程序代码具体分为：算法实现部分、八数码九宫格绘制部分、搜索树绘制部分三份代码。每份代码具有独立的数据结构定义及相关功能实现过程，通过 main()函数进行主算法的调用、参数传递，并在主算法中对图形界面绘制函数进行调用和传参。
3. 流程图：（见下页）



(2) 核心算法及基本原理

核心算法采用的是 A*算法。A*算法是一种有信息（启发式）的搜索策略。它对于结点的评价函数 f(n)中结合了 g(n)，即到达此结点已经花费的代价，和 h(n)，即从该结点到目标结点所花代价。有公式：f(n)=g(n)+h(n)。

A*算法既是完备的也是最优的。它每次优先扩展 f(n)值最小的结点，直到找到目标结点，或是扩展完所有可扩展的结点（无解的情况）。h(n)是一个可采纳启发式，它从不会过高地估计到达目标的代价，因为 g(n)是当前路径到达结点 n 的实际代价，所以有结论：f(n)永远不会超过经过结点 n 的解的实际代价。

A*算法需要的时间复杂度是指数级的，存储结点信息也需要比较大的空间。因此需要耗费一定的时间和空间。对于选取不同的 h(n)，将对算法的效率有很大影响。

具体到八数码问题的实例中，首先需要判断给定的初始八数码和目标八数码是否可解。当且仅当初始八数码的逆序数之和与目标八数码的逆序数之和的奇偶性相同时，该

八数码问题有解，否则无解。

对于一个八数码结点，其可扩展的方向最多只有上、下、左、右四个方向。空格的每一次移动产生的路径耗散为 1。初始的结点加入 Open 表中，后续将扩展出的结点加入 Open 表中，将扩展过的结点加入 Close 表中。每次从 Open 表中取出 f(n)值最小的结点进行扩展，直到找到目标结点，或者 Open 表为空为止。

本实验中设计了两种不同的估价函数 h(n)：第一种：计算不在位的棋子数；第二种：计算所有棋子到其目标的最短距离之和。

- 第一种：计算不在位的棋子数：

  例如当前结点：1 4 3　目标结点：1 2 3
  2 5 8　　4 5 6
  6 7 0　　7 8 0

  有 2、4、6、7、8（0 计入或不计入不会产生影响，这里考虑不计入的情况）共 5 个数不在位，因此 h(n)=5。

- 第二种：计算所有棋子到其目标的最短距离和：

  例如当前结点：1 4 3　目标结点：1 2 3
  2 5 8　　4 5 6
  6 7 0　　7 8 0

  有 2、4、6、7、8（0 计入或不计入不会产生影响，这里考虑不计入的情况）共 5 个数不在位，它们各自对于目标位置的最短距离分别为 2、2、3、1、2，因此 h(n)=2+2+3+1+2=10。

关于搜索树，则采用孩子兄弟表示法的二叉树这一数据结构来表示。在 A*算法的运行过程中同时对搜索树进行添加结点的操作。当 A*算法完成后，搜索树就形成了。其中需要一个树的结点指针指向每次从 Open 表中取出的 f(n)值最小的结点在搜索树中的位置，需要每次对搜索树进行遍历。遍历采用的是深度受限的先序遍历方法，当搜素树的规模较大，即扩展的结点数较多时，遍历将消耗较多的时间。采用深度受限的方法能够提升一些效率。

综上所述，在 A*算法的框架下，第一种评估函数可能产生多个 h(n)值相同的结点，在问题比较复杂的时候会使结点规模变得巨大；而第二种评估函数产生的 h(n)值差异性较高，在相同规模下，产生的结点数会比第一种少。因此从效率角度，采用第二种评估函数的 A*算法的效率更高，速度更快。

(3) 模块设计

1. 算法实现模块：采用“8Puzzle”类进行算法实现部分的封装。具体内容如下表：

8Puzzle 类		
私有成员：		
变量/函数名	数据类型/函数参数	功能
initial	Matrix	存放初始八数码的数字及相关信息
target	Matrix	存放目标八数码的数字及相关信息

Open	priority_queue	存放待扩展结点的优先队列
Close	queue	存放已扩展结点的队列
ST	STree	搜索树
PI	PuzzleInfo	存放整个八数码问题中的整体参数
Path	stack	存放最优路径中的结点的栈
公有成员：		
变量/函数名	参数	功能
RandomAssignment	无	随机赋值生成初始八数码和目标八数码
TypeIn	无	键入初始八数码和目标八数码
IsSolvable	无	判断八数码问题是否可解
IsSolvable	Matrix m	判断当前结点是否为目标结点
Astar	int type	进行 A*算法求解
GetPos	Matrix m, int& x, int& y	找到空格在九宫格中的位置
Update	Matrix& m, int type	更新结点信息
Expand	Matrix& m, STNode* p, int type	扩展结点
AddToOpen	Matrix m	将结点加入 Open 表
AddToClose	Matrix m	将结点加入 Close 表
FindInOpen	Matrix m	查找 Open 表中是否有与参数结点相同的结点
FindInClose	Matrix m	查找 Close 表中是否有与参数结点相同的结点
CreatePath	STNode* last_node	寻找最优路径
GetInfo	string s[6]，int& expande, int& next, char Current[MatrixSize][MatrixSize], charExpandeble[MaxExpandableNode] [MatrixSize][MatrixSize]	获取八数码问题的信息，用于与显示模块数据的传输

2. 演示界面绘制模块：演示界面绘制要在程序窗口进行图形绘制、并显示文字信息（包括 9 宫格内数字和文字提示），在程序设计过程中分为三个部分：主框架（MAIN_FRAME），表示 8 数码问题的目标状态以及演示步骤的当前状态（初始情况为初始状态）；副框架（SUB_FRAME），表示当前状态的可扩展状态，并把被选中的可扩展状态以不同颜色显示；信息框（INFO_FRAME），表示当前状态的信息（当前步长，总步长，Fn，扩展结点树，生成结点数）主要的变量定义和函数实现如下：

演示界面		
重要变量定义：		
变量名	数据类型	说明
Peight	EIGHT_PUZZLE_UI（结构体类型）	成员由窗口大小、颜色、提示信息的参数，以及主框架、副框架、信息框结构体组成
Main_frame	MAIN_FRAME_REC（结构体类型）	表示主框架的结构体，成员由主框架图形位置、框架画线样式、文字提示信息等参数组成
Sub_frame	SUB_FRAME_REC（结构体类型）	表示副框架的结构体，成员由副框架图形位置、框架画线样式、文字提示信息、可扩展状态个数、被选中状态序号等参数组成
Info_frame	INFO_FRAME_REC（结构体类型）	表示信息框的结构体，成员由信息框位置、框架画线样式、提示信息（位置、内容、字体样式）等参数组成
主要函数实现：		
函数名	函数参数	功能
setui	EIGHT_PUZZLE_UI& Peight, int symbol	设置界面图形部分的各种参数（包括颜色、位置、样式等等）
init_graph	EIGHT_PUZZLE_UI Peight	初始化窗口
init_tip	EIGHT_PUZZLE_UI& Peight, int symbol	设置界面文字部分的各种参数（文字的字体、颜色、大小、显示位置等等）
draw_frame	EIGHT_PUZZLE_UI& Peight, int	显示图形框架部分

	symbol	
display_text	EIGHT_PUZZLE_UI& Peight, string info[INFO_NUM]	显示界面文字信息
display_number	EIGHT_PUZZLE_UI&Peight,char number[][SQRT_NINE_SQUARES], int symbol, int i = -1	显示数字
Draw_UI	EIGHT_PUZZLE_UI &Peight, bool Flag	初始化 UI 界面

界面绘制内容变量定义以及函数实现较多，上表只是列出最主要的部分，详细内容请见函数清单以及 cpp 文件。

3. 搜索树绘制模块：搜索树绘制借助了 Easyx 库带有的图形绘制和文字输出函数，呈现效果为在程序界面内显示整个算法完整的搜索树形态，并用红线标注最佳路径（算法选择的路径），同时显示每个结点的 Fn 值

绘制搜索树		
主要函数实现：		
函数名	函数参数	功能
Mint_to_Mchar	char c[MatrixSize][MatrixSize], int s[MatrixSize][MatrixSize]	非负整数矩阵转化为字符矩阵
Matrix_to_Matrix	char a[MatrixSize][MatrixSize], char b[MatrixSize][MatrixSize]	字符矩阵相互赋值
Draw_STree	STree &ST, const int width, const int height, const int X, const int Y, const int maxdepth	初始化绘制窗口并打印搜索树
PrintSTree	STree ST, int x, int y, int lastx, int lasty, int space	搜索树的显示输出
Depth_of_STree	STree ST, int& depth	记录搜索树的最大深度
BestPath	STNode* last_node	从目标叶子结点回溯到根结点找出最优路径，并在树中做标记

(4) 其它创新内容或优化算法

1. 关于 Open 表：A*算法需要每次从 Open 表中取出 f(n)值最小的结点进行扩展。因此将 Open 表设计成优先队列，并通过重载运算符实现八数码数据间的直接比较。每次直接取 Open 表的表头结点进行扩展即可，简化了算法流程。
2. 关于搜索树与最优路径：搜索树的存储方式为孩子兄弟表示法的二叉树，并另外对每个树结点添加了一个 parent 指针，指向当前结点在搜索树结构中的父结点。每次用一个指针 p 指向从 Open 表中取出的结点在搜索树中的位置。当 A*算法运行结束后，p 所指的即使目标结点，再通过 parent 指针逆向回溯至搜索树的根结点，将回溯到的结点压入一个栈中，则得到一条正向的最优路径。
3. 关于演示界面显示：界面演示主要体现工程思维层面，其基础函数（如画一个矩形、显示一段文字）主要是通过调用 easyx 库中的基础图形绘制和文本显示函数，但是

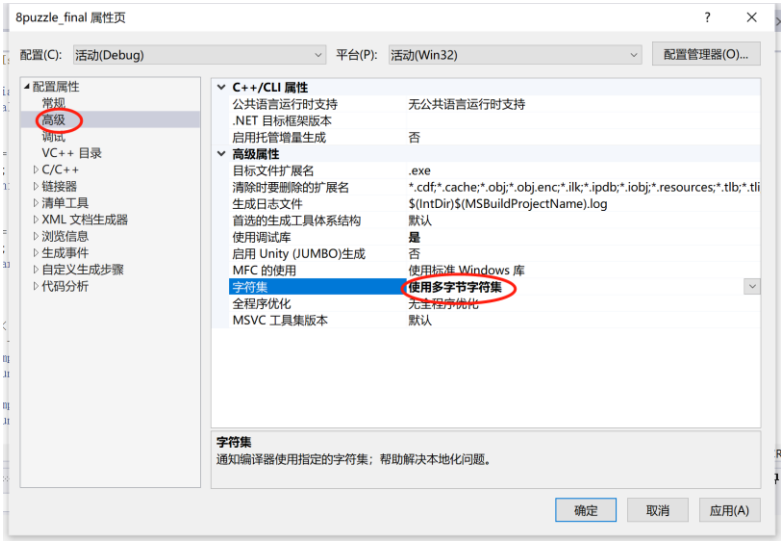
在实现过程中，通过比较合理的变量定义、函数提取与嵌套调用，以及合理使用 #define 将大部分参数数值进行宏定义，使得程序具有很高的可读性和可维护性，能够很方便的根据需要更改界面上的图形位置、样式，文字字体、大小，甚至对 15 数码问题也能通过简单的修改宏定义来整体适应。

4. 绘制搜索树时，采用记录相对结点位置的方法来简化连线的操作，省去了根据子树结点数计算相对位置的过程，大大减少了绘制时间。

三、实验过程

(1) 环境说明

1. 操作系统：Windows 10
2. 开发语言：C++
3. 开发环境：Visual Studio 2019、Visual Studio Code
4. 核心使用库：C++标准库、EasyX Library for C++（需安装，可见压缩包中 EasyX_20200315(beta).exe）
5. 特殊说明：为保证正常编译，创建项目时须在项目属性—>高级—>字符集改为“使用多字节字符集”。



(2) 源文件代码清单、主要函数清单

项目由三部分、五个文件组成:
头文件：8Puzzle_UI.h　8Puzzle.h
函数模块：8Puzzle_UI.cpp　8Puzzle.cpp
调用测试模块：demo.cpp
此处仅给出头文件函数清单，详见.cpp 和.h 文件

8Puzzle_UI.h：

```
void init_graph(EIGHT_PUZZLE_UI Peight);
void set_sub_square_num(EIGHT_PUZZLE_UI& Peight, int num);
void set_selected_sub_square(EIGHT_PUZZLE_UI& Peight, int select_num);
void is_solved(EIGHT_PUZZLE_UI& Peight, bool solve);
```

```
void setgraph(EIGHT_PUZZLE_UI& Peight);
void setmain_frame_graph(EIGHT_PUZZLE_UI& Peight);
void setsub_frame_graph(EIGHT_PUZZLE_UI& Peight);
void setinfo_frame_graph(EIGHT_PUZZLE_UI& Peight);
void setui(EIGHT_PUZZLE_UI& Peight, int symbol);

void set_graph_tip(EIGHT_PUZZLE_UI& Peight);
void set_mainframe_tip(EIGHT_PUZZLE_UI& Peight, bool is_init);
void set_subframe_tip(EIGHT_PUZZLE_UI& Peight);
void set_info_frame_none(EIGHT_PUZZLE_UI& Peight);
void set_info_frame_info(EIGHT_PUZZLE_UI& Peight, string info[INFO_NUM]);
void init_tip(EIGHT_PUZZLE_UI& Peight, int symbol);

void draw_9squares(int divide, SQUARE Psq);
void draw_main_frame(EIGHT_PUZZLE_UI Peight);
void draw_sub_frame(EIGHT_PUZZLE_UI& Peight);
void draw_info_frame(EIGHT_PUZZLE_UI Peight);
void draw_frame(EIGHT_PUZZLE_UI& Peight, int symbol);

void print_tips(DESCRIPTION& des, bool is_center);
void print_graph_frame_tip(EIGHT_PUZZLE_UI Peight);
void print_main_frame_tip(EIGHT_PUZZLE_UI Peight);
void print_sub_frame_tip(EIGHT_PUZZLE_UI Peight);
void print_info_frame_tip(EIGHT_PUZZLE_UI& Peight, string info[INFO_NUM]);
void display_text(EIGHT_PUZZLE_UI& Peight, string info[INFO_NUM]);

void set_number_style(DESCRIPTION& des, SQUARE Psq);
void print_number(DESCRIPTION des, char
number[SQRT_NINE_SQUARES][SQRT_NINE_SQUARES], SQUARE Psq);
void display_number(EIGHT_PUZZLE_UI& Peight, char
number[SQRT_NINE_SQUARES][SQRT_NINE_SQUARES], int symbol, int i = -1);
```

8Puzzle.h：

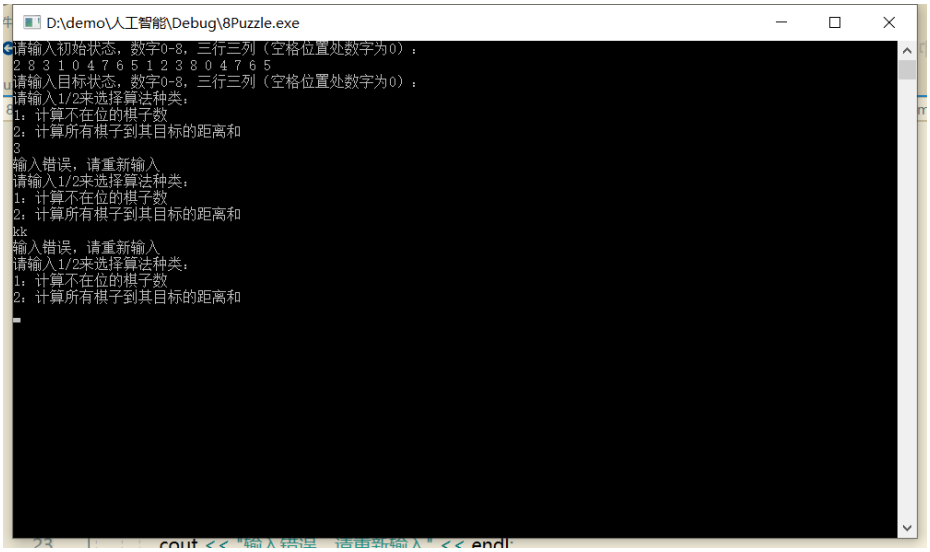
```
/*
    八数码类定义：
    private：
        initial：初始的数码排列。
        target：目标数码排列。
        Open：Open表，为优先队列，Fn值小的结点优先。记录当前可扩展的结点。
        Close：Close表，为队列，记录已扩展过的结点。
        ST：搜索树。
```

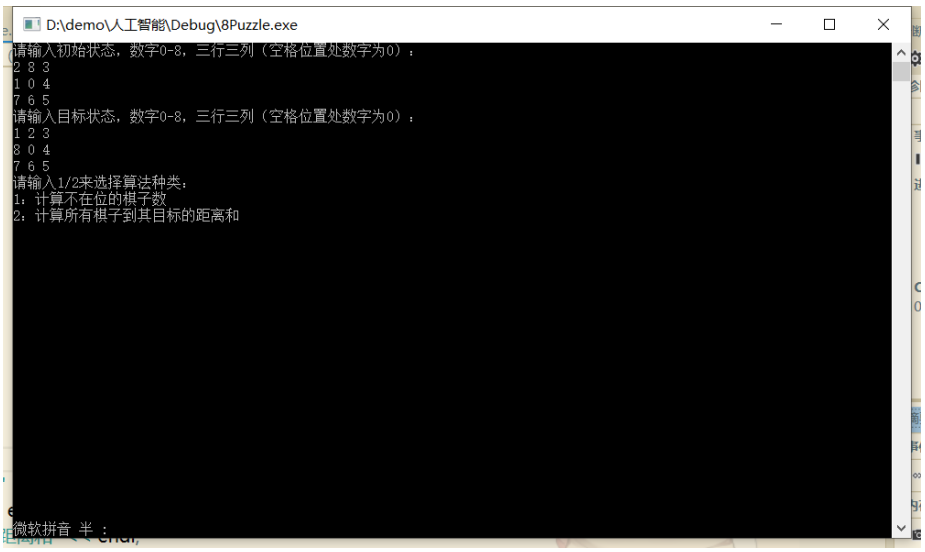
```
        PI：存放关于整个八数码问题的相关信息。
        Path：存放最优路径的栈，栈顶元素为初始结点。
    public：
        EightPuzzle()：构造函数，将initial和target置0，清空Open表和Close表,搜索树ST置空。
        ~EightPuzzle()：析构函数，清空Open表和Close表，销毁搜索树ST。
        RandomAssignment()：以随机赋值的方式生成initial和target。
        TypeIn()：以键盘输入的方式对initial和target赋值。
        IsSolvable()：判断该八数码问题是否可解。
        IsOver()：判断当前结点是否与target相同。
        AStar()：A*算法。
        GetPos()：找到当前结点中空格所在的位置。
        Move()：对空格进行上、下、左、右方向的移动，移动成功后对结点的Gn值进行更新。
        Update()：对结点Hn值、Fn值进行更新。
        Expand()：对当前结点进行上、下、左、右方向的扩展，添加搜索树ST的结点。
        AddToOpen()：将结点加入Open表。
        AddToClose()：将结点加入Close表。
        FindInOpen()：查找Open表中是否有相同的结点，若有则更新结点信息。
        FindInClose()：查找Close表中是否有相同的结点。
        CreatePath()：将最优路径存储至栈Path中。
*/
class EightPuzzle {
private:
    Matrix initial;
    Matrix target;
    priority_queue<Matrix, vector<Matrix>, greater<Matrix>> Open;
    queue<Matrix> Close;
    STree ST;
    PuzzleInfo PI; //八数码游戏信息
    stack<MatrixInfo> Path; //最优路径
public:
    EightPuzzle();
    ~EightPuzzle();
    void RandomAssignment();
    void TypeIn();
    bool IsSolvable();
    bool IsOver(Matrix m);
    void AStar(int type);
    void GetPos(Matrix m, int& x, int& y);
    Status Move(Matrix& m, int direction);
    void Update(Matrix& m, int type);
    void Expand(Matrix& m, STNode* p, int type);
    Status AddToOpen(Matrix m);
    Status AddToClose(Matrix m);
    Status FindInOpen(Matrix m);
```

```
    Status FindInClose(Matrix m);
    void CreatePath(STNode* last_node);
    void GetInfo(string s[6], int& expande, int& next, char Current[MatrixSize][MatrixSize], char
Expandable[MaxExpandableNode][MatrixSize][MatrixSize]);
    //新增：用于传至画图部分的信息字符串
};

/* Show()：显示当前结点的所有信息 */
void Show(Matrix m);

/*
    搜索树进行的操作：
        InitSTree()：初始化搜索树。
        DestroySTree()：销毁搜索树。
        CreateSTNode()：创建新的搜索树结点，并连接至搜索树。
        PreOrderTraverse()：先序遍历搜索树。
        PreOrderSearch()：先序遍历搜索树查找与e值相同的结点，指针p指向其位置。
        MyVisit()：先序遍历搜索树中的显示函数。
*/
Status InitSTree(STree& ST);
Status DestroySTree(STree& ST);
Status CreateSTNode(STree& ST, STNode* Node, TElemType e, int type);
Status PreOrderTraverse(STree ST, Status(*visit)(TElemType e));
Status PreOrderSearch(STree ST, STNode*& p, TElemType e);
Status MyVisit(TElemType e);

/* 将非负整数转化为字符串 */
string int_to_char(int n);
/* 非负整数矩阵转化为字符矩阵 */
void Mint_to_Mchar(char c[MatrixSize][MatrixSize], int s[MatrixSize][MatrixSize]);
/* 字符矩阵相互赋值 */
void Matrix_to_Matrix(char a[MatrixSize][MatrixSize], char b[MatrixSize][MatrixSize]);

/* 初始化绘制窗口并打印搜索树 */
void Draw_STree(STree &ST, const int width, const int height, const int X, const int Y, const int
maxdepth);
/* 搜索树的显示输出 */
void PrintSTree(STree ST, int x, int y, int lastx, int lasty, int space);
/* 记录搜索树的最大深度 */
Status Depth_of_STree(STree ST, int& depth);
/* 从目标叶子结点回溯到根结点找出最优路径，并在树中做标记 */
void BestPath(STNode* last_node);
/* 初始化UI界面 */
void Draw_UI(EIGHT_PUZZLE_UI &Peight, bool Flag);
```

(3) 实验结果展示

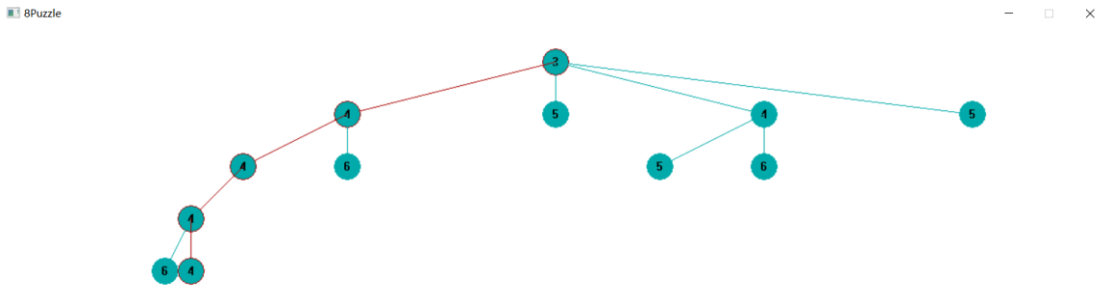
初始赋值和算法选择均进行了错误处理：



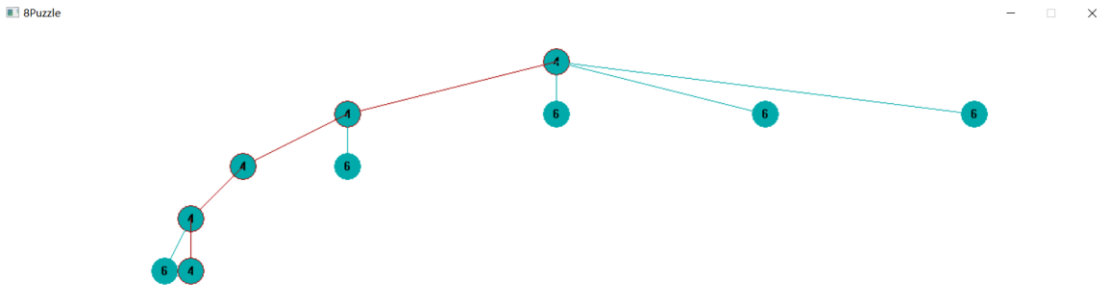


按回车后显示搜索树（若无解则跳过这一步骤）

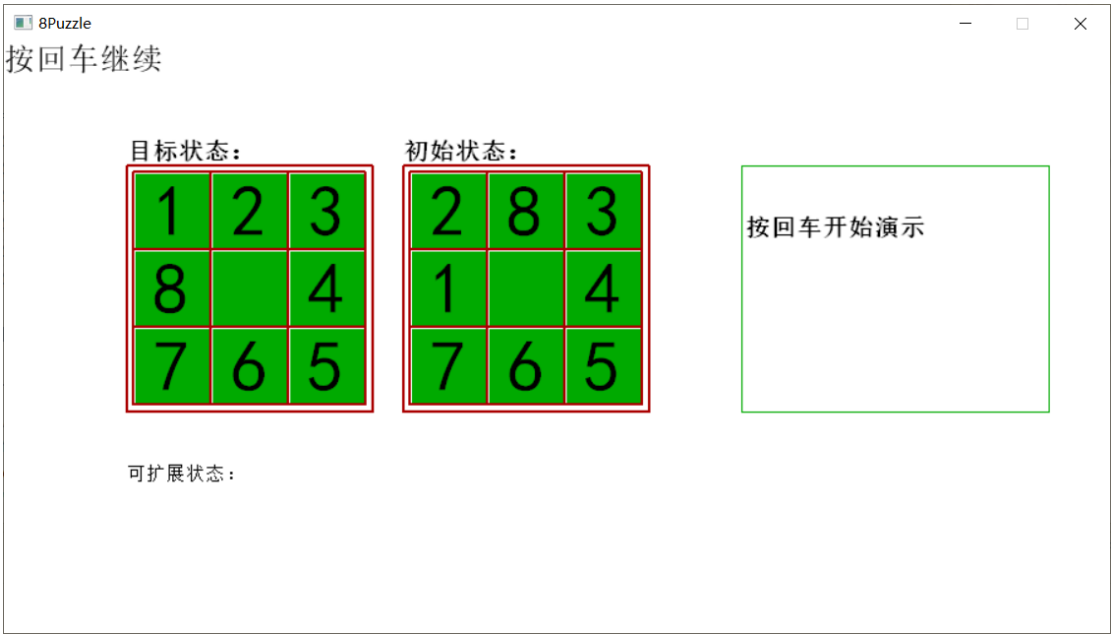
算法 1 的搜索树：

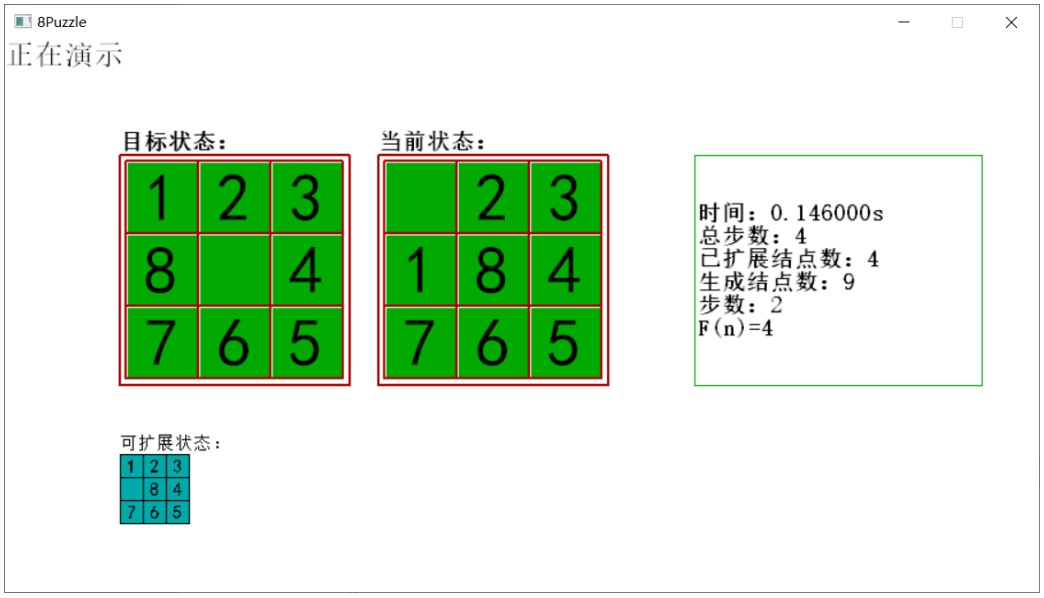


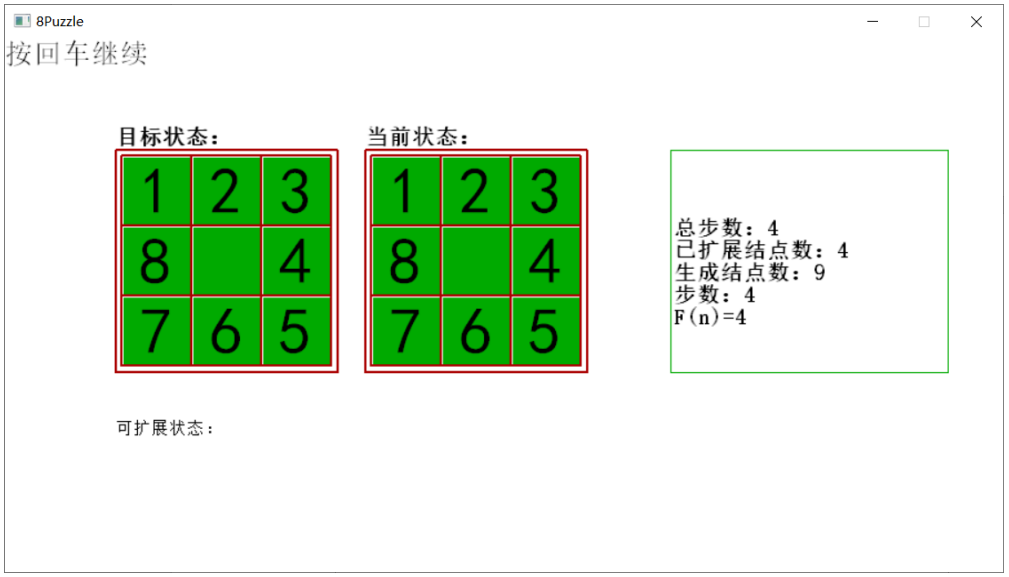
算法 2 的搜索树：



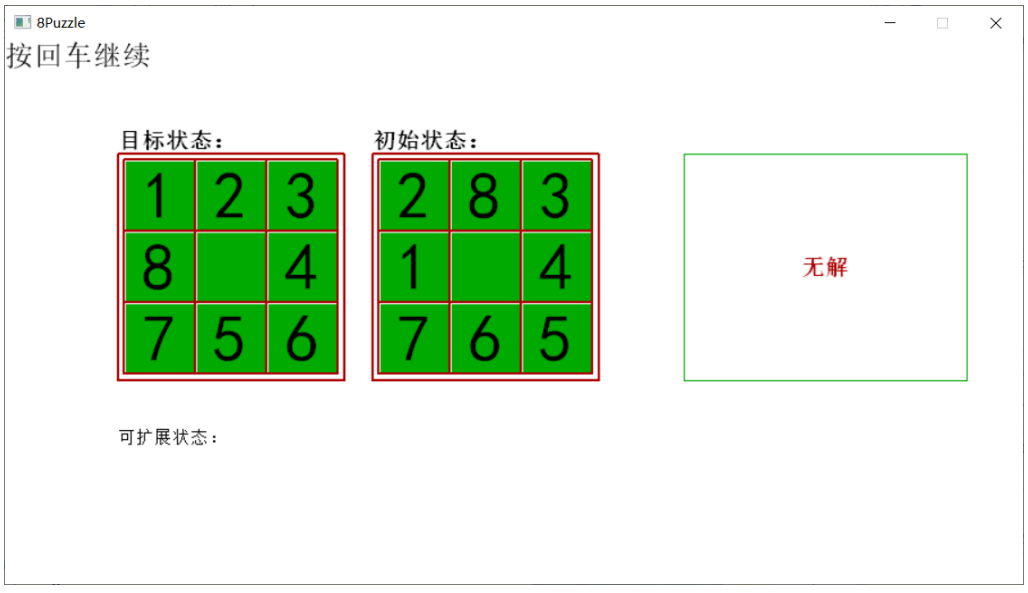
运行过程（算法 2 为例）：







无解则直接在右侧信息框显示：



两种算法的对比（采用统一数据）：

算法一（计算不在位的棋子数）：

初始数据	目标数据	耗时/s	步数	扩展结点数	生成结点数
2 8 3 1 0 4 7 6 5	2 8 3 0 1 4 7 6 5	0.057	1	1	4
2 8 3 1 0 4 7 6 5	1 2 3 8 0 4 7 6 5	0.162	4	5	11
2 8 1 0 3 4 7 5 6	3 0 4 1 2 8 7 5 6	1.587	10	51	90
2 8 3 1 0 4 7 6 5	2 8 0 3 1 4 7 6 5	3.709	12	152	248
2 8 1 0 3 4 7 5 6	3 0 4 1 8 2 7 6 5	668.327	20	3705	6272

算法二（计算所有棋子到其目标的最短距离和）：

初始数据	目标数据	耗时/s	步数	扩展结点数	生成结点数
2 8 3 1 0 4 7 6 5	2 8 3 0 1 4 7 6 5	0.053	1	1	4
2 8 3 1 0 4 7 6 5	1 2 3 8 0 4 7 6 5	0.152	4	4	9
2 8 1 0 3 4 7 5 6	3 0 4 1 2 8 7 5 6	0.476	10	13	26
2 8 3 1 0 4 7 6 5	2 8 0 3 1 4 7 6 5	2.287	12	75	131
2 8 1 0 3 4 7 5 6	3 0 4 1 8 2 7 6 5	15.674	20	558	916

(4) 实验结论

小组成功实现了利用两种不同评估函数解决 8 数码问题，根据实验数据可以得出以下结论：

1. 由于 A*算法的完备性和最优性，采用两种评估函数的程序均能算出 8 数码问题的解并给出最优路径。
2. 从表格中对比，在问题规模相同的情况下，采用第二种评估函数的算法相比第一种评估函数耗时短、扩展和生成的结点数少，是效率更高的选择。当问题规模小的时候，二者差距不明显；但是随着问题规模的扩张，二者在消耗时间、空间上的差距越来越大。
3. 两种评估函数产生差异的原因在于：第一种评估函数可能产生多个 h(n)值相同的结点，在问题比较复杂的时候会使结点规模变得巨大；而第二种评估函数产生的 h(n)值差异性较高，在相同规模下，产生的结点数会比第一种少。因此从效率角度，采用第二种评估函数的 A*算法的效率更高，速度更快。
4. 根据实验结果，可以得出在 A*算法中，面对大规模的问题时，采用更优的评估函数对算法的效率有着巨大的提升的结论。因此，在设计 A*算法的过程中，考虑一种较优的，产生的评估值对于不同的结点值、不同的路径值有着较大差异性（即产生相同评估函数值的不同结点数较小）的评估函数时算法优化过程中的重要部分。

四、总结

(1) 实验中存在的问题及解决方案

1. 存在的问题：算法部分中对于寻找每次从 Open 表中取出的结点在搜索树中的位置一开始采用的是先序遍历搜索树查找结点的方法。该方法一定能找到结点的位置，但在搜索树规模较大的情况下会消耗大量的时间，甚至会超过算法其它重要部分运行的时间很多。

解决方案：在先序遍历的基础上添加深度限制。通过读取结点的 g(n)信息即可确定结点所在的深度，从而在遍历中不去遍历层次更深的结点，降低了时间消耗。

2. 存在的问题：界面绘制过程中会出现调整界面某一部分图形的位置而导致整个界面错乱的情况，或者需要更改多处代码，十分容易出现错误。

   解决方案：在程序设计时尽量将不同功能都提取为函数，尽量不要出现基本的函数实现在功能上会有重叠的情况；图形绘制选取某些点作为基准点，这样只修改基准点位置可以对演示界面进行整体修改；合理使用宏定义，不要把参数写死，只通过修改头文件就可以对程序进行更改，可维护性更强。

3. 存在的问题：在使用 easyx 库调用 outtextxy()函数时会编译出现“没有参数列表匹配的重载函数‘outtextxy’的实例”错误。

   解决方案：通过调整项目属性->配置属性->高级->字符集，改为使用多字节字符集，即可解决编译出现的错误。（运行环境为 Visual Studio 2019）

4. 存在的问题：在绘制搜索树的时候需要获取存储在 Path 中的最优路径结点信息，但由于整个 8Puzzle 类已被封装完毕，Path 属于 private 属性，因此无法直接在 main 函数中调用，此类情况还出现在对运行界面进行绘制时

   解决方案：在 A*算法中一并调用树形绘制函数和运行界面绘制函数，避免在外部调用类中私人成员，并根据是否有解进行不同调用操作

5. 存在的问题：进行图形更新时会产生新旧图像重叠覆盖导致字体变粗或者图像错误的情况，并且部分颜色也会重叠

   解决方案：使用 eaxyx 库中的清除画面函数，根据绘图参数来在更新画面前清除原画面，实现无重叠更新

(2) 心得体会

1. 在程序设计的初期必须先明确程序功能，做一定的需求分析，划分程序模块，并进行相应的分工，才能保证程序的有序性，也方便后期程序的合并、修改。
2. 编写程序的过程中尽量做到高内聚低耦合。模块化的设计方便了参数的设定和修改，也在排除 Bug 的过程中更容易确定问题出处，减少合并不同人代码的工作量。
3. 小组合作过程中工程意识和合作意识十分重要。每个人在完成自己部分的过程中，应充分考虑变量定义和函数实现的合理性，给出自己负责部分需要的参数和传出的参数，面向对象编程要有合理的封装。每个人都对自己的部分的功能、接口给出详细说明，方便其他成员的对自己部分的设计以及最后的整合。

(3) 后续改进方向

1. 算法方面：关于寻找每次从 Open 表中取出的结点在搜索树中的位置，如今的解决方法在结点数量大、扩展的结点在树中的距离长的情况下依然会消耗大量的时间。

存在更加快捷的方法，但需要改变数据结构类型，让 Open 表、Close 表等数据结构直接存储搜索树结点，便能实现指针对当前树结点的快速访问。

2. 界面方面：限于实现手段，程序演示是基于控制台的窗口界面，在未来掌握更多工具后可以对界面的美观程度进行优化。
3. 绘制树形图方面：控制台界面能容纳的树形十分有限，当层数比较高时搜索树不能完全展示，当结点代价太大时，Fn 值可能会超出结点显示框。可以考虑采用其他 c++相关工具来单独对树进行绘制，而不是拘泥于控制台显示。
4. 整体实现方面：本程序只是对求解 8 数码进行了步骤演示，可以考虑加入使用者自行操作进行求解的功能，使程序拥有更多的交互。

(4) 总结

在本次采用 A*算法解决八数码问题的实验中，小组通过明确的分工，分别完成了采用 A*算法实现八数码问题、采用两种不同的评估函数进行对比、设计图形界面显示八数码问题的最优路径、设计画出求解八数码问题过程中产生的搜索树等诸多实验要求。小组成员在此次实验中加深了对于 A*算法的理解，提升了采用 C++语言的编程技巧，实践了面向对象编程的思想，学习了 UI 图形设计方面的知识，锻炼了在团队中分工合作实现项目的能力。

五、参考文献

[1] 《人工智能：一种现代的方法（第 3 版）》，Stuart J.Russell，Peter Norvig 著
[2] 《Easyx 官方文档》

六、成员分工与自评

1. 成员分工：
   - 赵中楷：主要负责界面整体设计以及实现，并对部分算法进行了优化；
   - 汪一泓：主要负责算法设计及算法优化，对界面设计的一部分进行调整；
   - 杨宏辉：主要对项目各部分进行整合、调试，负责界面的一部分设计。
2. 成员自评：
   - 赵中楷：本次实验我在界面设计的过程中负责主要部分，通过自学查找资料能熟练掌握利用 C++显示图形界面的方法；在课上的学习中对 A*算法以及 8 数码问题有了一定的掌握，在本次实验过程中参与部分算法的实现，也增进了对算法的理解。总体上能积极进行学习，认真完成了自己的所有工作，设计功能完整、可维护性好的模块，并在实现过程中对于出现的问题及时进行解决。在程序设计和实现过程中也让我对如何与其他组员沟通、对实验进行更合理的模块划分和任务分配有了新的认识。
   - 汪一泓：本次实验参与最多是算法实现部分。通过动手对算法的实现，对 8 数码问题和 A*算法有了更深的理解，掌握了 A*算法中的不同估价函数；参与界面的一部分调整也让我学习到了有关界面可视化的设计知识。在实验过程中我认真完成了自己负责的部分，积极与其他两位同学沟通，根据反馈及时调整自己的实现方法和传递给其他组员的参数，锻炼了在多人合作模式下如何设计封装性好，功能齐全的程序模块的能力。

- 杨宏辉：在项目的整体整合和调试过程中，让我在算法的原理知识掌握基础上关注到了许多在动手实现过程中的细节，提高了动手实践的能力；在根据算法进行搜索树的绘制实现也丰富了我对界面知识的了解。在整个实验进程中，能认真设计了自己负责的部分，同时也对实验的各个模块进行了合理的整合，调试过程中也始终保持耐心，关注每一个细节，根据运行情况对各个部分进行适应性的调整，或者及时给其他两位组员反馈，共同修改，解决问题。

3. 总体评价与收获

在本实验过程每位同学都能积极参与，并对实验的各个部分（算法、界面）都进行认真的学习，然后按照分工完成自己的任务，充分发挥自己的能力。通过这次作业我们对 A*算法有了更多的了解，也提升了小组合作的团队能力。在程序设计上我们也有一些值得改进的地方，以后也会更加努力，一直进步！