A星求解八数码问题

人工智能原理与技术课程

汇报人：龚哲飞 指导老师：王俊丽 2022.6.8

目录

content

第一部分

实验概述

实验概述

实验目的

熟悉和掌握启发式搜索策略的定义，评价函数f(n)和算法过程，并利用A*算法求解8数码问题，理解求解流程和搜索顺序

实验内容

以8数码问题为例，实现A*算法的求解程序（编程语言不限）。要求设计至少两种不同的启发函数h(n)。

设置相同的初始状态和目标状态，针对不同的评价函数求得问题的解，比较它们对搜索算法性能的影响，包括拓展结点数，生成结点数和运行时间等。画出结果比较的图表，并进行性能分析。

实验要求

要求界面显示初始状态，目标状态和中间搜索步骤。

要求显示搜索过程，画出搜索过程生成的搜索树，并在每个节点显示对应节点的评价值f(n)，并标注出最终结果所选用的解路线。

撰写实验报告，提交源代码（进行注释），实验报告、汇报PPT

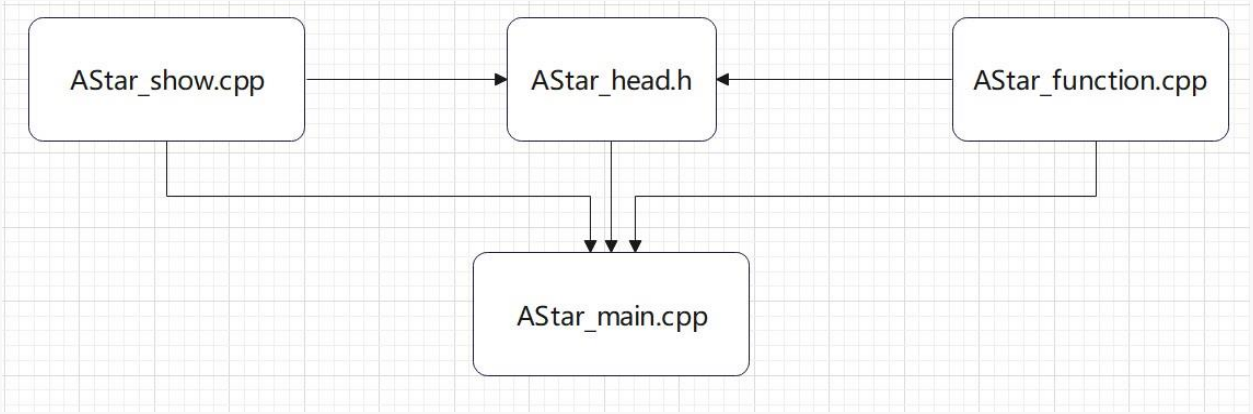
第二部分

实验设计方案

总体设计思路与总体框架

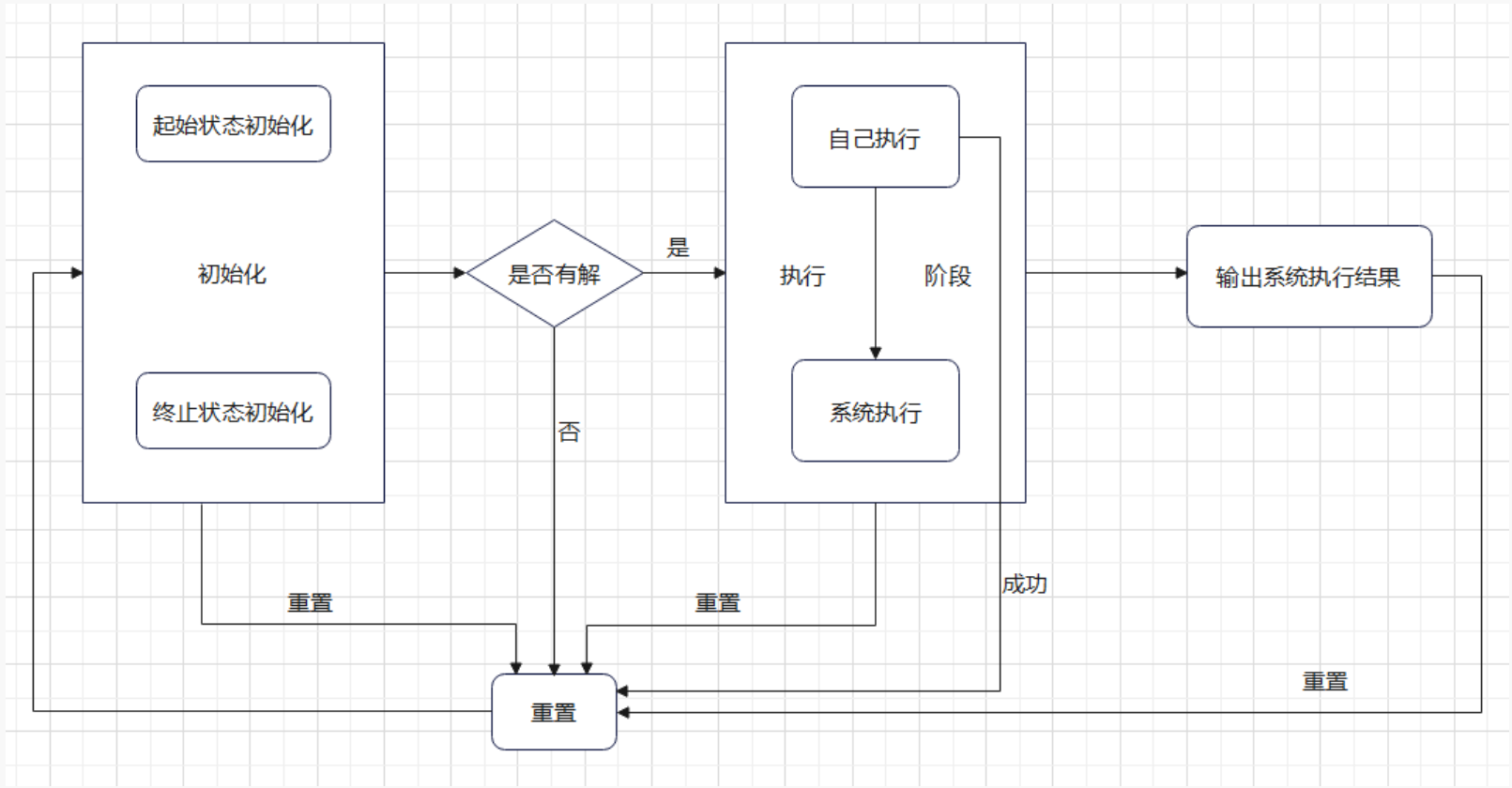
程序设计安排再4个cpp文件中实现，分别为：

AStar_head.h：存储整个项目类、结构体、函数、宏定义的声明，作为项目的头文件使用。
AStar_function.cpp：内含AStarEightDigital类的具体实现函数，实现八数码问题的内部核心算法的求解。
AStar_show.cpp：内含EasyXSHOW类的具体实现函数，实现八数码问题的可视化界面输入输出。
AStar_main.cpp：程序主函数所在地，调用各个类进行对八数码问题的求解。



总体框架

总体设计思路与总体框架



运行流程

核心算法及其原理 — A*算法

A*可以认为是添加了启发式函数的Dijkstra算法，在Dijkstra算法的基础上，构造一个函数h(n)，n为当前扩展结点，h(n)返回结点n到终点的开销估计。然后建立函数f(n)=g(n)+h(n)，其中g(n)为从起点到结点n已经使用了的代价，所以f(n)可以理解为是“从起点出发经过结点n再到终点的代价估计”

算法实现步骤

① 设置一个OPEN表用于存放那些搜索图上的叶节点，也就是已经被生成出来，但是还没被拓展的节点；初始化CLOSE表用于存放图中的非叶节点，也就是不但被生成出来，还已经被拓展的节点。

②OPEN中的节点按照 f 值从小到大排列。每次从OPEN表中取出第一个元素 n 进行扩展，如果 n 是目标节点，则算法找到一个解，算法结束，否则扩展 n。

③对于 n 的子节点 m ，如果 m 既不在OPEN也不在CLOSE，则将 m 加入OPEN；如果 m 在CLOSE，说明从初始节点到 m 有两条路径，如果新路径耗散值大，什么都不做；如果较小，则删除原来在CLOSE的，将新找到的放入OPEN。

④重复①，直到找到一个解结束；或者OPEN为空算法以失败结束，说明无解。

利用A*算法根据以上步骤进行求解，便轻松得到在算法层面利用A*算法求解八数码问题了。

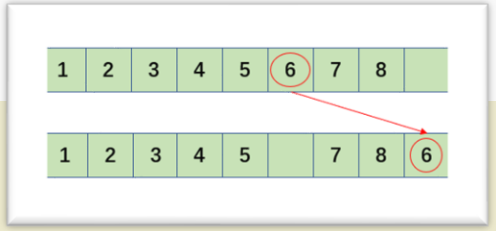
启发式函数算法设计

计算当前状态的不在位棋子数作为h(n)

将当前状态的各个棋子实际位置与期望位置的曼哈顿距离和作为h(n)

将当前状态的各个棋子实际位置与期望位置的欧拉距离和作为h(n)

将当前状态的各个棋子实际位置与期望位置的带系数的曼哈顿和作为h(n):



H(n)

启发式函数听起来很有学问，其实可以很简单的理解为从源点到目标的所需要消耗的总代价**f(n)**（和适应度函数比较相像），这个总代价可以分成两个部分从源点到中间节点（搜索的中间状态）已经消耗的实际代价**g（n）**，另一个部分就是对从中间节点到目标的预测**h(n)**。

模块设计

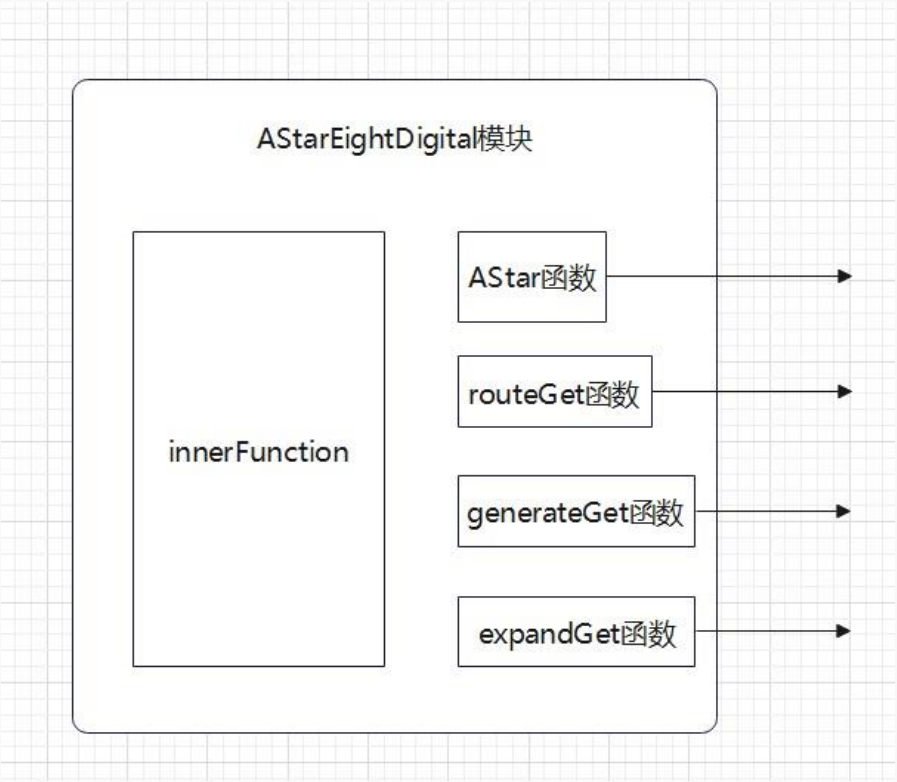
```
class AStarEightDigital {
public:
    char start [BoardSize][BoardSize] = { '\0' };              //起始八数码形式
    char end   [BoardSize][BoardSize] = { '\0' };              //终点八数码形式
    vector<AStarQueNote> OpenStorage;                          //open数组vector
    vector<AStarQueNote> CloseStorage;                         //close数组vector
    vector<AStarTreeNote> TreeList;                            //存储搜索过程中的线索树
public:
    AStarEightDigital();                                                    //构造函数
    int setup(char(*s)[BoardSize], char(*e)[BoardSize]);                    //初始化
    int DataChangeGraph(unsigned long long& in , char(*arry)[BoardSize]);   //unsigned long long 转 char格式化数组数据
    int GraphChangeData(char(*arry)[BoardSize], unsigned long long& in);    //unsigned long long 转 unsigned long long
    int setInspire(AStarQueNote& in);                                       //获取启发式函数的值
    int FindOpenMin(int& site);
    bool FindOpen(unsigned long long& in, int& site);
    bool FindClose(unsigned long long& in, int& site);
    int AStarMove(char(*arry)[BoardSize], AStarTreeNote& father);
    int FindFather(AStarQueNote& node, AStarTreeNote& father);
    int AStar(void);

    int routeGet(vector<unsigned long long>& routeResult);                  //根据求出解路径
    int generateGet(int& get);                                              //返回生成结点数
    int expandGet(int& get);                                                //返回拓展结点数

};
```

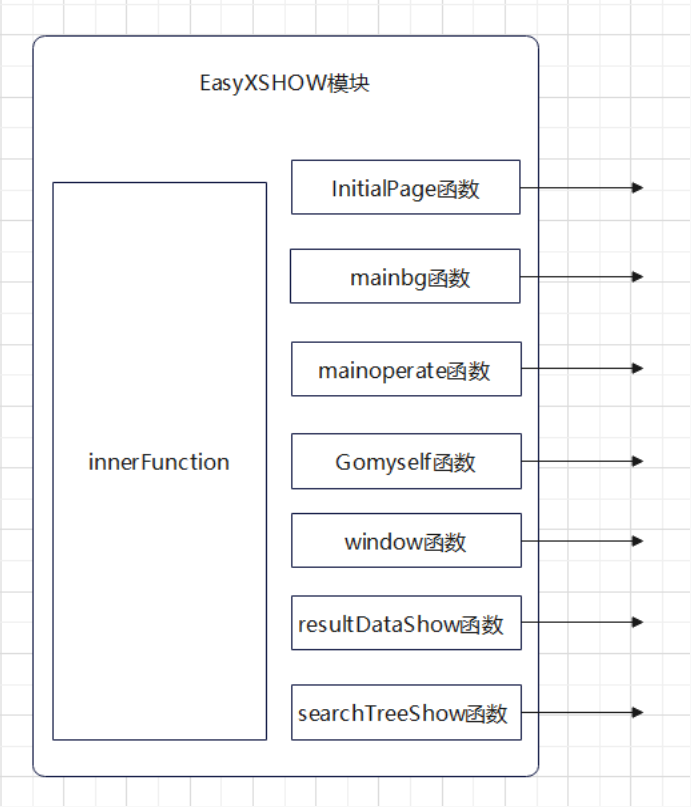
```
class EasyXSHOW {
public:
    IMAGE num[9];
    AREA General, cherkerboard, dataout, stateshow, buttonoperate;
    AREA eightborad, startstate, endstate;
    AREA systemgo, myselfgo;
    AREA upbutton, rightbutton, downbutton, leftbutton;
    AREA reset,initialbegin, initialend;
public:
    EasyXSHOW();                                                //可视化初始构造函数
    int Initialbg(void);
    int Boardshow(char(*array)[BoardSize]);
    int blockshow(int x, int y, char(*array)[BoardSize]);
    int block(int x, int y, int num);
    int button(int x, int y, int length, int width, TCHAR text[]);
    int stateinitial(char(*st)[BoardSize], char(*end)[BoardSize]);
    int moveinitial(char(*state)[BoardSize], AREA& site);
    int InitialPage(char(*start)[BoardSize], char(*end)[BoardSize]);
    int mainbg(char(*start)[BoardSize], char(*state)[BoardSize], char(*end)[BoardSize]);
    int mainoperate(void);
    int Gomyself(char(*state)[BoardSize], char(*end)[BoardSize]);
    void winshow(void);

    int processShow(vector<unsigned long long>& routeResult);
    int resultDataShow(clock_t time, int expandNum, int generateNum, vector<unsigned long long> routeResult);
    int searchTreeShow(vector<AStarTreeNote>& TreeList, unsigned long long  end);
    int searchTreeShowIn(vector<AStarTreeNote>& TreeList, unsigned long long  end);
    int searchTreeShowNode(int x, int y, AStarTreeNote& node, unsigned long long  end);
    int resultNone(void);
};
```

01 AStarEightDigital模块

02 EasyXSHOW模块

模块设计

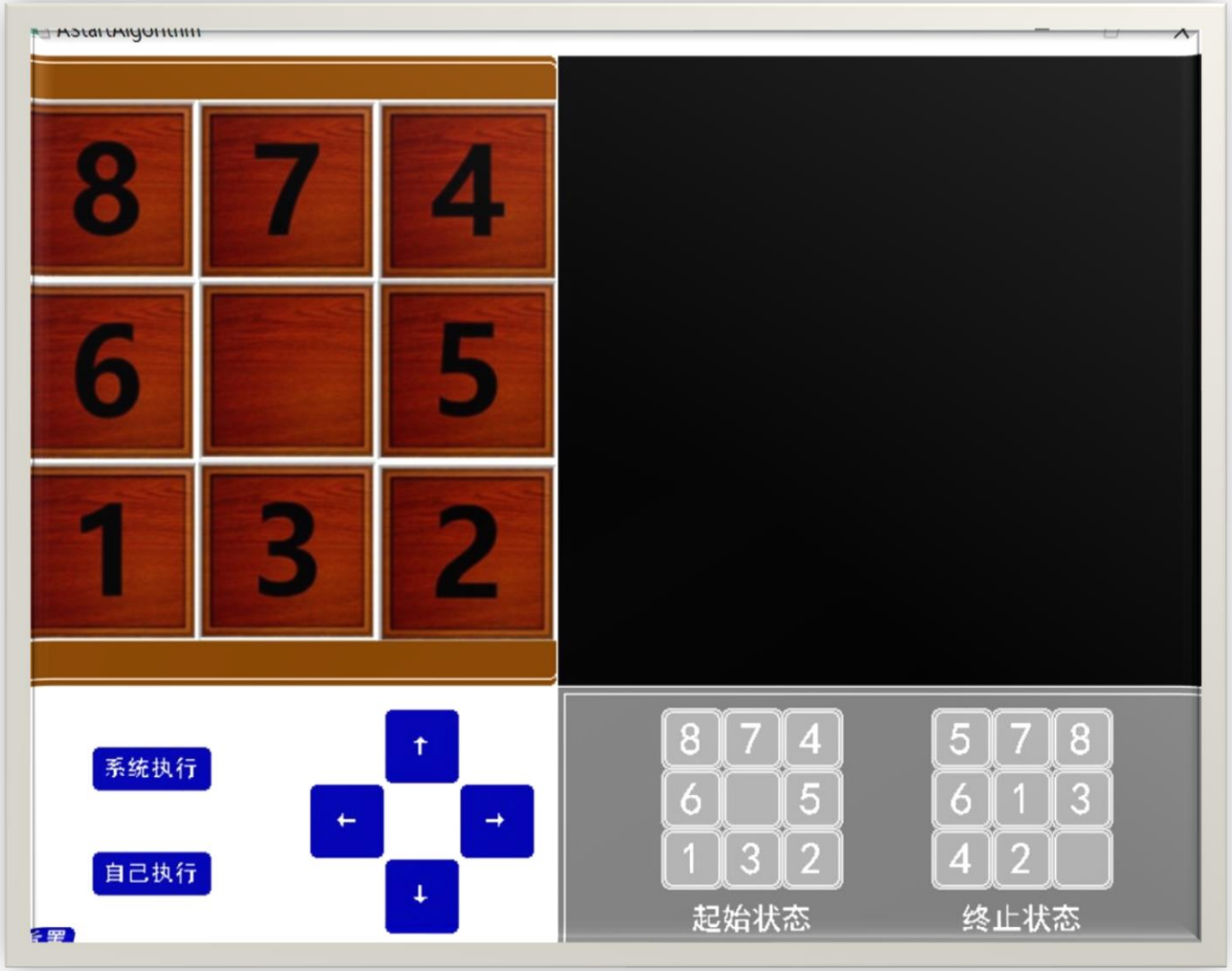




01 AStarEightDigital模块

02 EasyXSHOW模块

创新点--自己执行+系统执行双模式



在系统寻找最优解的基础上，添加玩家自玩模式，增加趣味性

界面交互友好，操作方便

Gonzalez

创新点--启发函数 灵活变更

充分运用工程化思维

利用宏定义+条件编译的方式，组合多种启发函数于一体，仅需在头文件中简单操作，便可变更启发函数

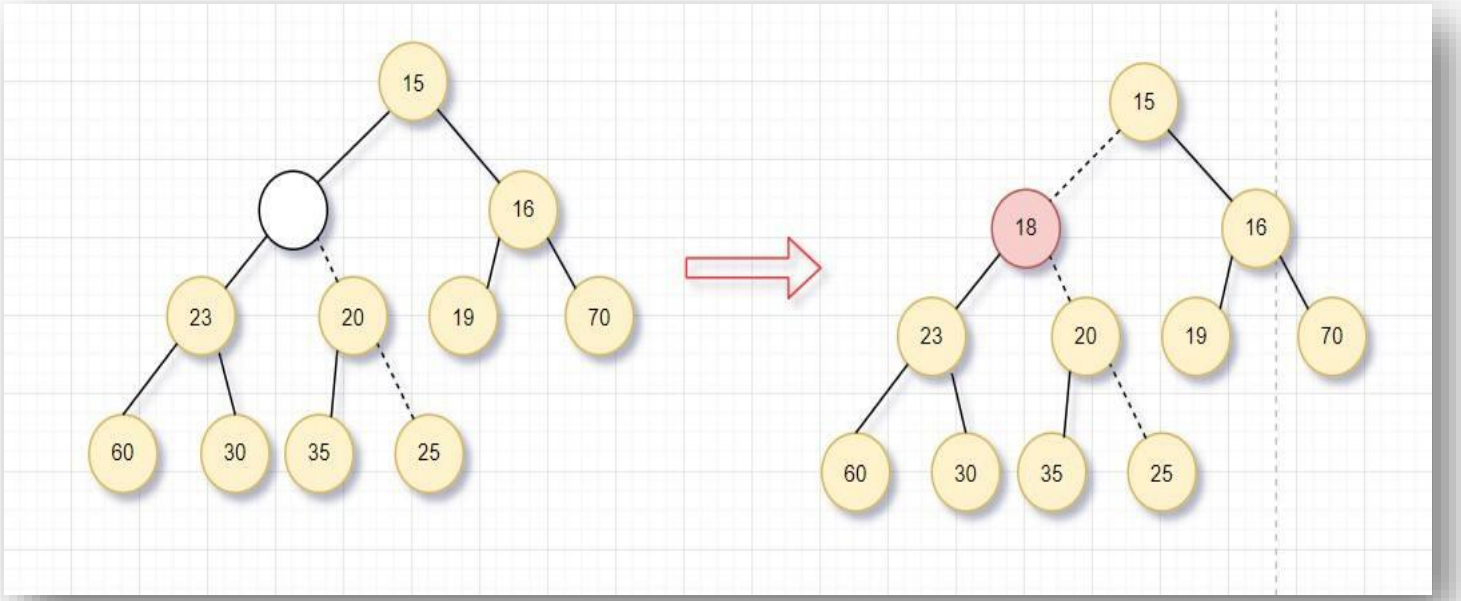
```
/*！！！影响算法关键因素的启发式函数！！！*/

//@function : 不在位的棋子数作为启发式函数值
#ifdef OUFOFPOSITION 非活动预处理器块
#endif // OUFOFPOSITION

//@function : 所有棋子到其目标位置的距离和作为启发式函数值（曼哈顿法）
#ifdef DISTANCETOTARGET 活动预处理器块
#endif // DISTANCETOTARGET

//@function : 所有棋子到其目标位置的直线距离(欧氏距离)作为启发式函数值
#ifdef EULERDISTANCE 非活动预处理器块
#endif//EULERDISTANCE

//@function : 系数曼哈顿距离法
#ifdef COEFFICIENTMANHANTTAN
int AStarEightDigital::setInspire(AStarQueNote& in)...
#endif//COEFFICIENTMANHANTTAN
```

操作方便、易于维护

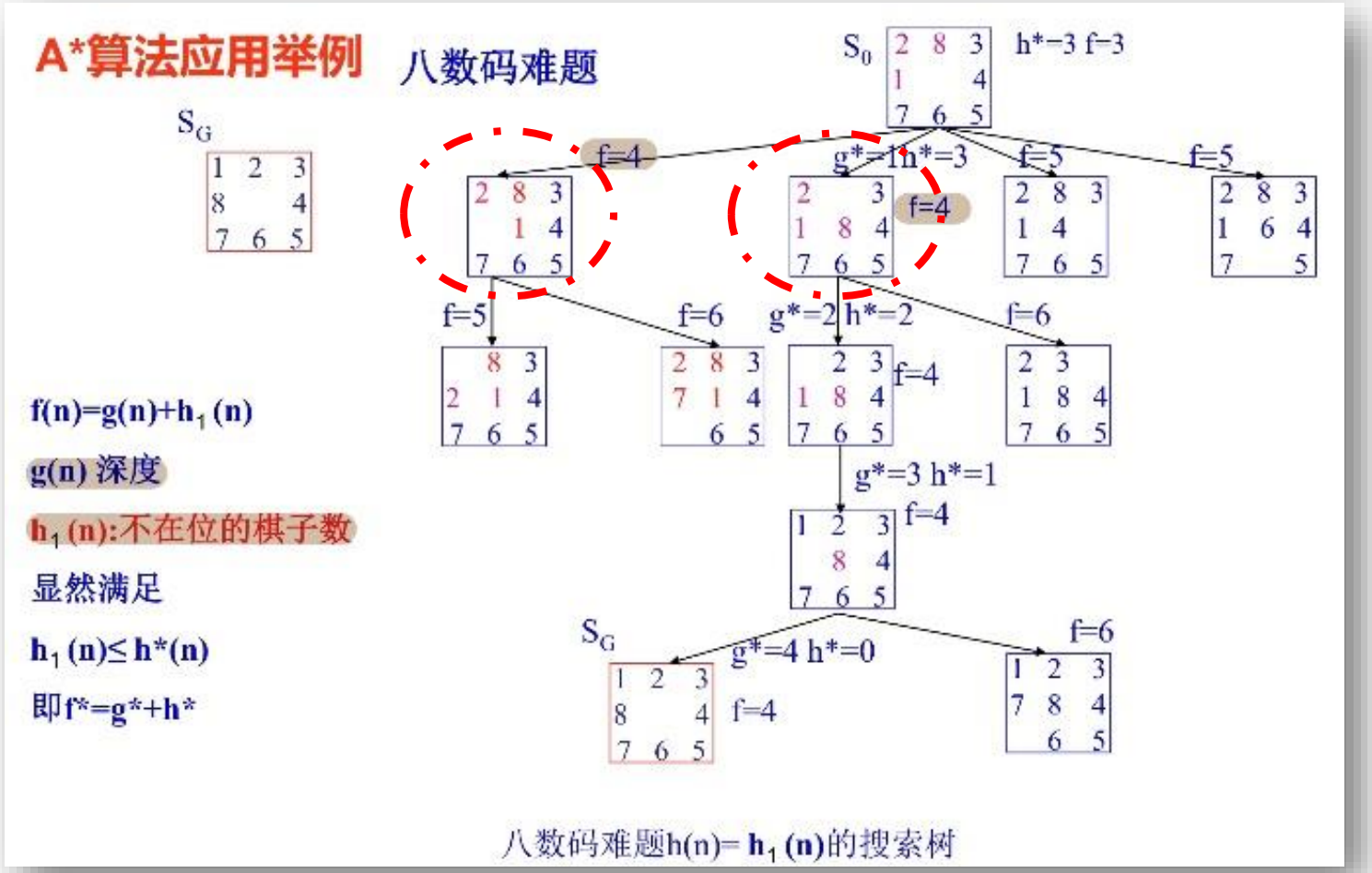
```
/*条件编译选择启发式函数*/
//#define OUFOFPOSITION                 //不在位的棋子数
//#define COEFFICIENTMANHANTTAN         //系数曼哈顿距离法——>可能存在高估的情况，因此并非可采纳的，结果可能不是最优解
//#define EULERDISTANCE                 //欧拉距离法
#define DISTANCETOTARGET              //曼哈顿距离法所有棋子到其目标位置的距离和
```

创新算法--优先队列加速OPEN/CLOSE表

由于本题采用了OPEN和CLOSE表实现A*算法，每次均需要从OPEN表中提取f(n)最小的元素，因此，若每次进行遍历寻找最小值，当程序数据量增大时，程序运行效率将大大降低。因此可以使用优先队列的方式对OPEN表进行存储，每次只需要pop出最优节点即可。



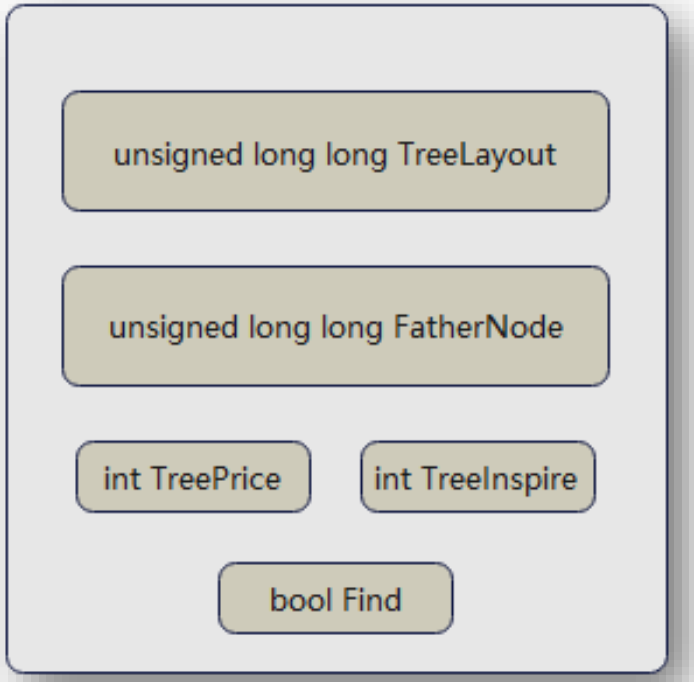
创新算法--优先队列加速OPEN/CLOSE表



但是这样也存在一个问题：

当OPEN表中存在两个相同最小值时，按照逻辑严密性进行分析，应当将两个最小节点均进行遍历。避免出现：随机遍历多个最小值之一，忽略另一个最小值子节点可能出现更小f(n)的情况。因此，使用优先队列的过程中也应该进行适当的修改。

创新算法--搜索树与最优路径




```
struct AStarTreeNote {                                  //A*遍历树存储结构体
    unsigned long long FatherNote = 0;
    unsigned long long Treelayout = 0;
    int Treeprice = 0;
    int Treeinspire = 0;
    bool Find = false;                                  //是否被遍历
};

vector<AStarTreeNote> TreeList;                         //存储搜索过程中的线索树
```

在搜索树的存储上，本题并未采用一般的树构造方式，而是采用结构体+数组的方式进行存储

简化计算步骤 提高访问效率

创新算法--UI界面展示

工程化思维

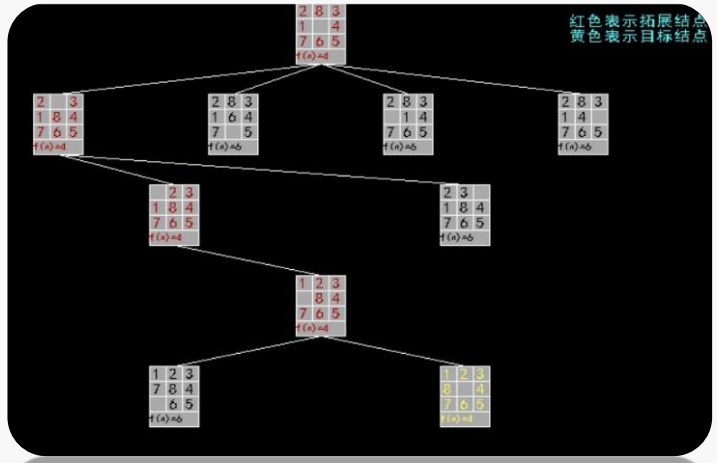
在EasyX库的基础上，编写适用于本项目的UI界面功能函数。通过合理的变量定义、函数提取与嵌套调用进行页面展示

```
// @function: 按钮输出函数
int EasyXSHOW::button(int x, int y, int length, int width, TCHAR text[])
{
    int HIGHT = BUTTONHIGHT;
    setbkmode(TRANSPARENT);
    setfillcolor(BLUE);
    fillroundrect(x, y, x + length, y + width, 10, 10);
    settextstyle(HIGHT, 0, _T("黑体"));
    int tx = x + (length - textwidth(text)) / 2;
    int ty = y + (width - textheight(text)) / 2;
    outtextxy(tx, ty, text);

    return FINE;
}
```

代码可读性高、可维护性强

线索树连线简化

绘制搜索树时，通过线索树结点数组记录的相对位置来简化连线的操作，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。



创新算法--八数码是否有解特判

```
//@function : 判断是否存在结果—>start和end的逆序数不变
bool answerExist(char(*arr1)[BoardSize], char(*arr2)[BoardSize])
{
    vector<int>list1;
    vector<int>list2;

    for (int i = 0; i < BoardSize; i++)
        for (int j = 0; j < BoardSize; j++) {
            list1.push_back(arr1[i][j] - '0');
            list2.push_back(arr2[i][j] - '0');
        }
    int num1 = 0;
    int num2 = 0;
    int Temp = 0;
    for (int i = 0; i < (int)list1.size(); i++) {
        if (list1[i] != 0) {//空格不检查
            Temp = list1[i];
            for (int j = 0; j < i; j++) {
                if (list1[j] > Temp)
                    num1++;
            }
        }
    }
    for (int i = 0; i < (int)list2.size(); i++) {
        if (list2[i] != 0) {//空格不检查
            Temp = list2[i];
```

1	2	3
4	5	6
7	8	

→

1	2	3	4	5	6	7	8	

一个状态表示成一维的形式，求出除空之外所有数字的逆序数之和，也就是每个数字前面比它大的数字的个数的和，称为这个状态的逆序。

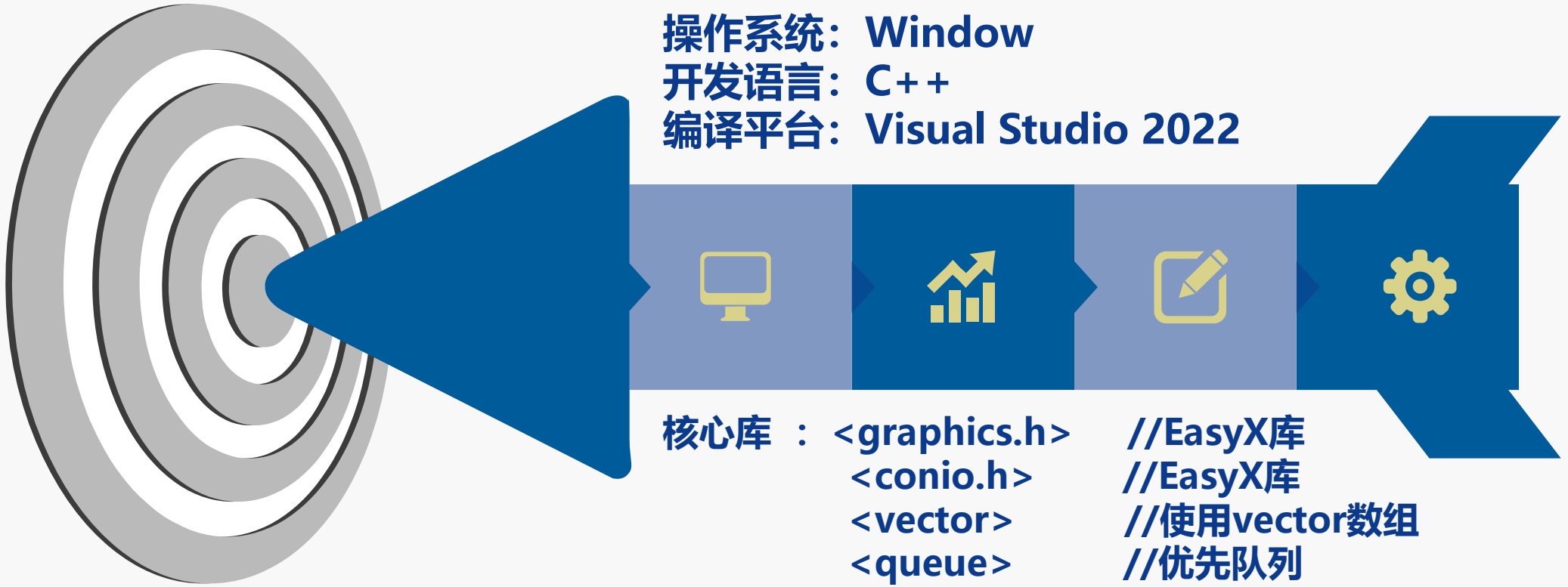
若两个状态的逆序奇偶性相同，则可相互到达，否则不可相互到达。

当左右移动空格时，逆序不变。当上下移动空格时，相当于将一个数字向前（或向后）移动两格，跳过的这两个数字要么都比它大（小），逆序可能±2；要么一个较大一个较小，逆序不变。

第三部分

实验过程

环境说明

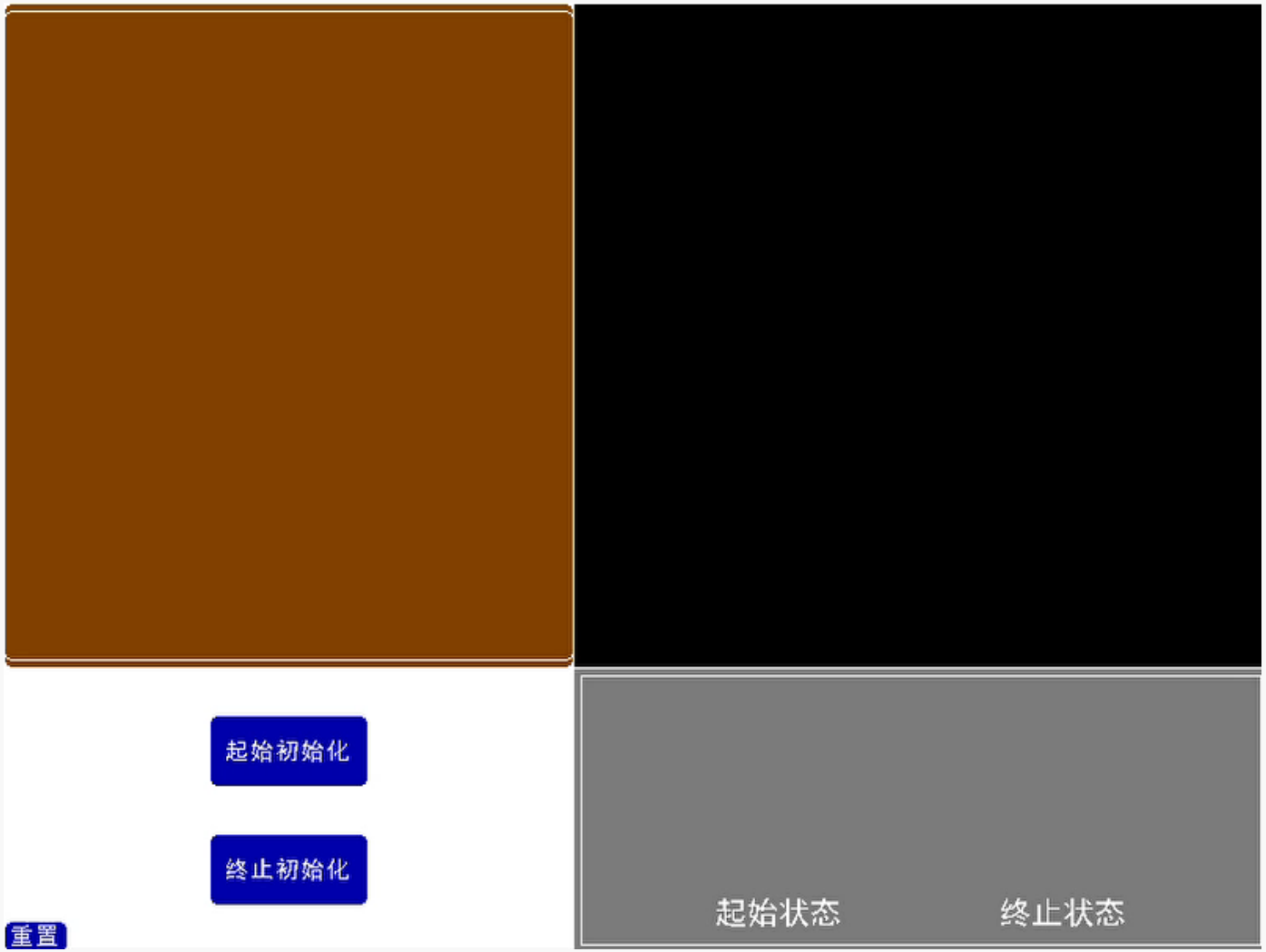


操作系统：Window
开发语言：C++
编译平台：Visual Studio 2022

核心库 ：

```
<graphics.h>    //EasyX库
<conio.h>       //EasyX库
<vector>        //使用vector数组
<queue>         //优先队列
<windows.h>     //sleep函数
<time.h>        //计算运行时间
```

成果展示

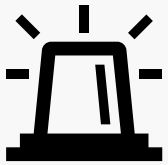


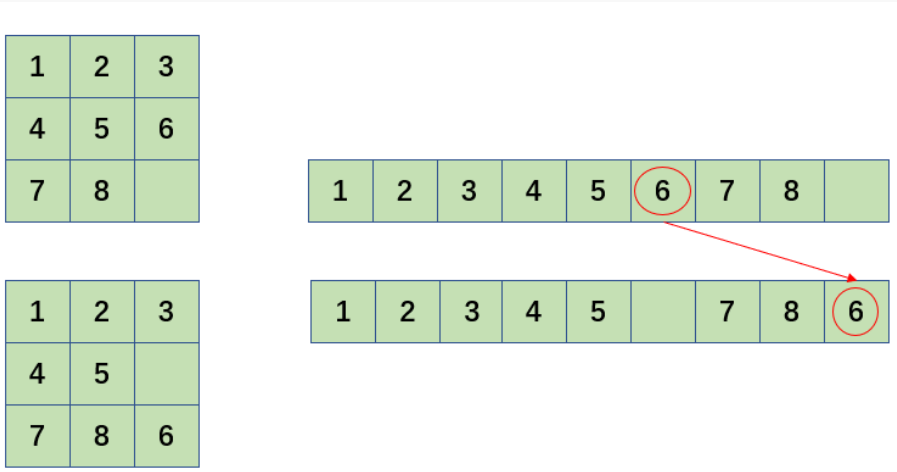
结果分析

方法名	运行时间	生成结点	拓展结点	解路径步骤
不在位棋子数	28395ms	28340	18767	25
欧拉距离和	1789ms	6985	4474	25
曼哈顿距离和	170ms	2099	1289	25
系数曼哈顿法	9ms	343	198	31

根据上列表格信息，我们可以得出以下结论：

首先系数曼哈顿法并非可采纳的，求得解路径并非最优结果，这也是可以推证的：





我们可以发现，以上两状态真实距离只需要一步，但是系数曼哈顿法算出距离为3，h(n)>h*(n)，因此也就出现了系数曼哈顿法无法得到最优解，其并非可采纳的。

结果分析

方法名	运行时间	生成结点	拓展结点	解路径步骤
不在位棋子数	28395ms	28340	18767	25
欧拉距离和	1789ms	6985	4474	25
曼哈顿距离和	170ms	2099	1289	25
系数曼哈顿法	9ms	343	198	31

而其余三种方法，我们也可发现，均是可采纳的，但**曼哈顿距离法** 优于 **欧拉距离法** 优于 **不在位棋子数法**。这也符合，三种启发式函数同真实h*(n)之间的接近关系。随机推算多个状态对比可知，曼哈顿距离法h(n)更加接近于实际距离，欧拉距离法次之，不在位棋子数相差最远。

综上，实验得出结论有：

①系数曼哈顿法是不可采纳的，其余h(n)均可采纳

②启发式函数性能对比：**曼哈顿距离法 > 欧拉距离法 > 不在位棋子数法**

第四部分

总结

总结

问题

整个实验过程中存在的问题不多，A*算法的实现过程也较为顺利。主要问题在于考虑到算法实现的逻辑严密性上。即当OPEN表中最小值有多个时，如果随机选择一个进行遍历，而其子节点小于此最小值，则下一次遍历将在子节点产生。因此将出现同为最小值，但由于遍历的随机性问题，导致某些节点在整个过程中出现未被遍历的情况，这样是否会对最后的结果产生影响？对于这个问题思考了许久，仍无法得到令人信服的证明。

体会

经过本次实验，我对于A*算法的实现过程有了更加清晰的认识。启发式函数的选择极大程度影响A*算法的效率，并且如果选择不可采纳的算法，尽管最终结果可能并非最优解，但可以极大程度地提升效率，降低运行时间。因此，我们在使用A*算法时，应当考虑当时的实际情况，选择恰当地启发式函数，以达到更好的实际效果。

今后

后续可以在算法效率上提升：本次采用OPEN表和CLOSE表进行A*过程的搜索，并且还添加了一个TreeList表记录搜索过程。我们可以考虑是否可以仅使用一个表格即可完成A*搜索，并且记录搜索树，这将极大地降低A*算法实现地空间复杂度。

除此之外，在算法的时间复杂度方面，也可以查阅更多的相关文献，提升算法效率。

参考文献

1. 邓伟.基于C#WinForm的AStar寻路算法交互设计[J].信息技术与信息化,2021(08):80-83.
2. 谭威,胡新荣,雷伟.基于Unity的A-Star算法在游戏中的具体实现[J].计算机产品与流通,2018(11):121.陈桂娥，樊行雪，许振良.线性滴定中稳定常数测定方法比较[J].华东理工大学学报，1996，22(5): 620-625.
3. 刘好. 基于优化Astar与MPC的自动驾驶车辆路径规划研究[D]. 东南大学,2019.DOI:10.27014/d.cnki.gdnau.2019.001055.

感谢聆听

人工智能原理与技术课程

汇报人：龚哲飞　指导老师：王俊丽　2022.6.8