同济大学人工智能原理课程实验报告

实验题目:八数码问题

小组成员: 1750844 周展田， 1752723 王松森

分工: 周展田：查找算法资料及文档编写
王松森：实现算法整合及图形化

一．实验概述

【实验目的】

1.理解和掌握 A*算法，并应用 A*算法解决问题

2.学习相关的图形化实现

【实验问题描述】

八数码问题也称为九宫问题。在 3×3 的棋盘，摆有八个棋子，每个棋子上标有 1 至 8 的某一数字，不同棋子上标的数字不相同。棋盘上还有一个空格，与空格相邻的棋子可以移到空格中。要求解决的问题是：给出一个初始状态和一个目标状态，找出一种从初始转变成目标状态的移动棋子步数最少的移动步骤，要求使用 A*算法实现。

【实验原理】

A*算法是一种常用的启发式搜索算法。

在 A*算法中，一个结点位置的好坏用估价函数来对它进行评估。A*算法的估价函数可表示为：

$f'(n) = g'(n) + h'(n)$

这里，$f'(n)$是估价函数，$g'(n)$是起点到终点的最短路径值（也称为最小耗费或最小代价），$h'(n)$是 n 到目标的最短路经的启发值。由于这个 $f'(n)$其实是无法预先知道的，所以实际上使用的是下面的估价函数：

$f(n) = g(n) + h(n)$

其中 $g(n)$是从初始结点到节点 n 的实际代价，$h(n)$是从结点 n 到目标结点的最佳路径的估计代价。在这里主要是 $h(n)$体现了搜索的启发信息，因为 $g(n)$是已知的。用 $f(n)$作为 $f'(n)$的近似，也就是用 $g(n)$代替 $g'(n)$，$h(n)$代替 $h'(n)$。这样必须满足两个条件：（1）$g(n)>=g'(n)$（大多数情况下都是满足的，可以不用考虑），且 f 必须保持单调递增。（2）h 必须小于等于实际的从当前节点到达目标节点的最小耗费 $h(n)<=h'(n)$。

【实验环境】

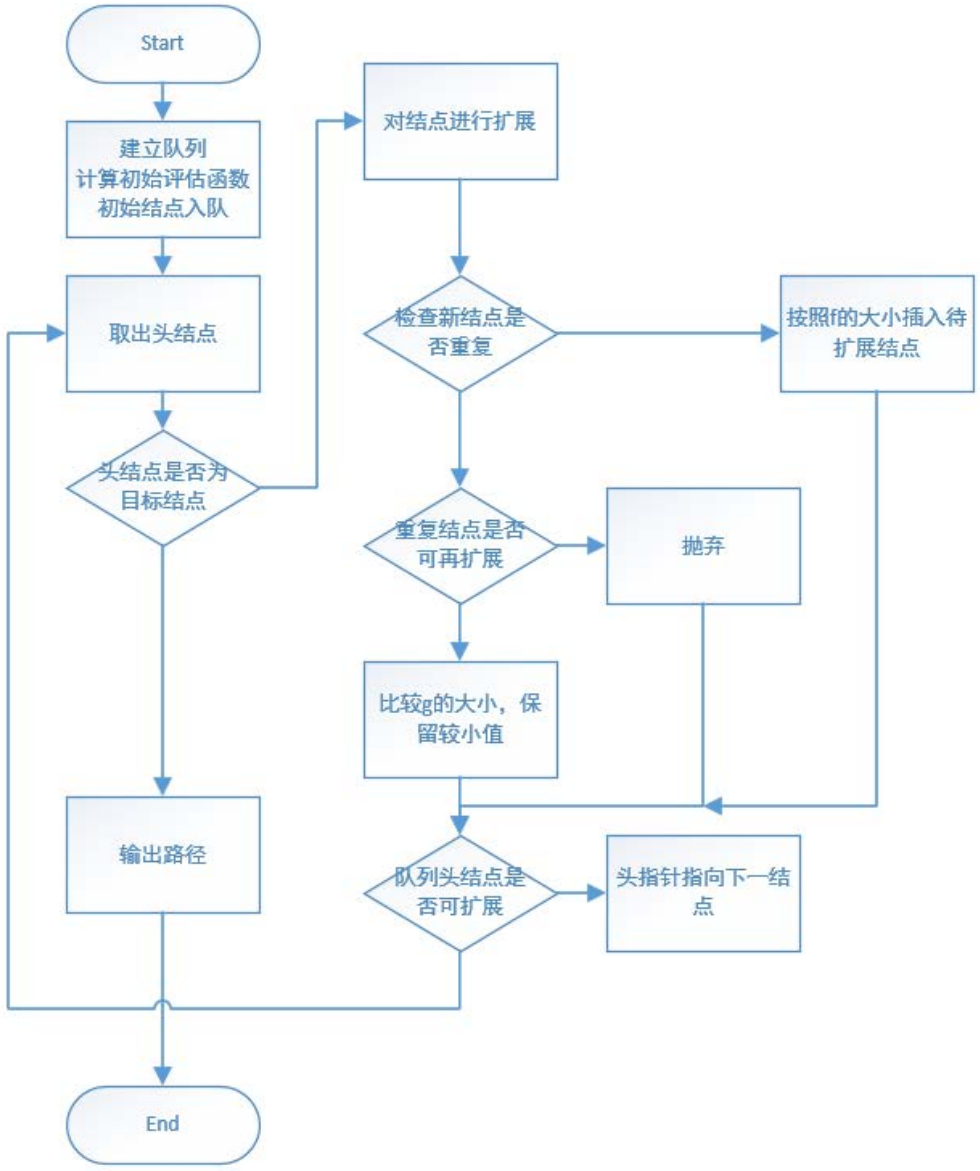
MS Visual Studio 2017（导入了 EasyX 库）

二．实验过程及结果

【算法详细设计】
算法结构设定：
使用 init.txt 文件初始化起始状态，设定目标状态
通过查询 dis.txt 文件，获取移动距离估计

A*算法的步骤如下：
(1)建立一个队列，计算初始结点的估价函数 f，并将初始结点入队，设置队列头指针和尾指针。
(2)取出队列头（队列头指针所指）的结点，如果该结点是目标结点，则输出路径，程序结束。否则对结点进行扩展。
(3)检查扩展出的新结点是否与队列中的结点重复，若与不能再扩展的结点重复（位于队列头指针之前），则将它抛弃；若新结点与待扩展的结点重复（位于队列头指针之后），则比较两个结点的估价函数中 g 的大小，保留较小 g 值的结点。跳至(5)。
(4)如果扩展出的新结点与队列中的结点不重复，则按照它的估价函数 f 大小将它插入队列中的头结点后待扩展结点的适当位置，使它们按从小到大的顺序排列，最后更新队列尾指针。
(5)如果队列头的结点还可以扩展，直接返回(2)。否则将队列头指针指向下一结点，再返回(2)。

【算法流程图】



【源程序】//提高可读性，标准文字解释

```
#include <iostream>
#include<fstream>
#include <string>
#include <graphics.h>  //图形化库

using namespace std;

//定义常量
#define SIZE 300 //八数码外框大小
#define Num 9

int step = 0; //步数
```

```
//绘图函数声明
void draw_struct(int left, int right, int top, int bottom); //画框线结构
void draw_number(int num, int x, int y, int size);  //输出数字 数字内容 起始位置 横向尺寸
void draw_move(int x, int y, int step); //方块移动

//八数码问题相关类及成员函数
class TEight
{
public:
    TEight() {}
    TEight(const char *fname);
    virtual void Search() = 0;
protected:
    int p[Num];
    int last, spac;
    static int q[Num], d[], total;
    void Printf();
    bool operator==(const TEight &T);
    bool Extend(int i);
};

int TEight::q[Num];
//空格向右、下、左和上移动后，新位置是原位置分别加上 1、3、-1、-3，
//如果将空格向右、下、左和上移动分别用 0、1、2、3 表示，并将-3、3、-1、1 放在静态数组 d[4]中，
int TEight::d[] = { 1,3,-1,-3 };
int TEight::total = 0;

//读取初始化文件
TEight::TEight(const char *fname)
{
    ifstream fin;
    fin.open(fname, ios::in | ios::_Nocreate);
    if (!fin)
    {
        cout << "不能打开数据文件!" << endl;
        return;
    }
    int i;
    //读取初始状态
    for (i = 0; i < Num;)
        fin >> p[i++];
```

```
    //读取空格
    fin >> spac;
    //读取目标状态
    for (i = 0; i < Num;)
        fin >> q[i++];
    fin.close();
    last = -1;
    total = 0;
}

//将结果输出到文件中
void TEight::Printf()
{
    ofstream fout;
    fout.open("result.txt", ios::ate | ios::app);
    fout << total++ << "step:";
    step = total; //取到步数
    for (int i = 0; i < Num;)
        fout << "  " << p[i++];
    fout << endl;
    fout.close();
}

//重载==
bool TEight::operator==(const TEight &T)
{
    for (int i = 0; i < Num;)
        if (T.p[i] != p[i++])
            return 0;
    return 1;
}

//判断是否可以扩展
bool TEight::Extend(int i)
{
    if (i == 0 && spac % 3 == 2 || i == 1 && spac > 5
        || i == 2 && spac % 3 == 0 || i == 3 && spac < 3)
        return 0;
    int temp = spac;
    //空格位置用spac表示,那么空格向方向i移动后,它的位置变为spac+d[i]
    spac += d[i];
    p[temp] = p[spac];
    p[spac] = 0;
    return 1;
```

```
}

template<class Type> class TList;            //线性表前视定义

//线性表结点类模板
template<class Type> class TNode
{
    friend class TList<Type>;
public:
    TNode() {}
private:
    TNode<Type>* Next;
    Type Data;
};

template<class Type> class TList
{
public:
    TList() { Last = First = 0; Length = 0; }   //构造函数
    int Getlen()const { return Length; }   //成员函数，返回线性表长度
    int Append(const Type& T);           //成员函数，从表尾加入结点
    int Insert(const Type& T, int k);       //成员函数，插入结点
    Type GetData(int i);                          //成员函数，返回结点
数据成员
    void SetData(const Type& T, int k); //成员函数，设置结点数据成员
private:
    TNode<Type> *First, *Last;             //数据成员，线性表首、尾指
针
    int Length;                                          //数据成员，线
性表长度
};

//从表尾加入结点
template<class Type> int TList<Type>::Append(const Type& T)
{
    Insert(T, Length);
    return 1;
}

//插入结点
template<class Type> int TList<Type>::Insert(const Type& T, int k)
{
    TNode<Type> *p = new TNode<Type>;
    p->Data = T;
```

```
    if (First)
    {
        if (k <= 0)
        {
            p->Next = First;
            First = p;
        }
        if (k > Length - 1)
        {
            Last->Next = p;
            Last = Last->Next;
            Last->Next = 0;
        }
        if (k > 0 && k < Length)
        {
            k--;
            TNode<Type> *q = First;
            while (k-- > 0)
                q = q->Next;
            p->Next = q->Next;
            q->Next = p;
        }
    }
    else
    {
        First = Last = p;
        First->Next = Last->Next = 0;
    }
    Length++;
    return 1;
}

//返回数据
template<class Type> Type TList<Type>::GetData(int k)
{
    TNode<Type> *p = First;
    while (k-- > 0)
        p = p->Next;
    return p->Data;
}

//插入数据
template<class Type> void TList<Type>::SetData(const Type& T, int k)
{
```

```
    TNode<Type> *p = First;
    while (k-- > 0)
        p = p->Next;
    p->Data = T;
}

//AStar 类声明
class AStar :public TEight
{
public:
    AStar() {}                                  //构造函数
    AStar(const char *fname1, const char *fname2);   //带参数构造函数
    virtual void Search();                      //A*搜索法
private:
    int f, g, h;                                  //估价函数
    int r[Num];                                 //存储状态中各个数字位置
的辅助数组
    static int s[Num];                          //存储目标状态中各个数字
位置的辅助数组
    static int e[];                               //存储各个数字相对距离
的辅助数组
    void Printl(TList<AStar> L);        //成员函数，输出搜索路径
    int Expend(int i);                            //成员函数，A*算法的状态
扩展函数
    int Calcuf();                                 //成员函数，计算估价函数
    void Sort(TList<AStar>& L, int k);  //成员函数，将新扩展结点按 f 从
小到大顺序插入待扩展结点队列
    int Repeat(TList<AStar> &L);        //成员函数，检查结点是否重复
};

int AStar::s[Num], AStar::e[Num*Num];

//两参构造，打开文件输入
AStar::AStar(const char *fname1, const char *fname2) :TEight(fname1)
{
    for (int i = 0; i < Num;)
    {
        r[p[i]] = i;   //存储初始状态数字的位置
        s[q[i]] = i++; //存储目标状态数字的位置
    }
    ifstream fin;
    fin.open(fname2, ios::in );//打开数据文件
    if (!fin)
    {
```

```
        cout << "不能打开数据文件!" << endl;
        return;
    }
    for (int i = 0; i < Num*Num; i++)   //读入各个数字相对距离值
        fin >> e[i];
    fin.close();
    f = g = h = 0;        //估价函数初始值
}

//输出路径
void AStar::Printl(TList<AStar> L)
{
    AStar T = *this;
    if (T.last == -1)
        return;
    else
    {
        T = L.GetData(T.last);
        T.Printl(L);
        T.Printf();
    }
}

//A*算法的状态扩展函数
int AStar::Expend(int i)
{
    if (Extend(i))   //结点可扩展
    {
        int temp = r[p[r[0]]];   //改变状态后数字位置变化，存储改变后的
位置
        r[p[r[0]]] = r[0];
        r[0] = temp;
        return 1;
    }
    return 0;
}

//评估函数
int AStar::Calcuf()
{
    h = 0;
    for (int i = 0; i < Num; i++)  //计算估价函数的 h
        h += e[Num*r[i] + s[i]];
    return ++g + h;
```

```
}

//将新扩展结点按 f 从小到大顺序插入待扩展结点队列
void AStar::Sort(TList<AStar>& L, int k)
{
    int n = L.Getlen();
    int i;
    for (i = k + 1; i < n; i++)
    {
        AStar T = L.GetData(i);
        if (this->f <= T.f)
            break;
    }
    L.Insert(*this, i);
}

//检查是否有重复结点
int AStar::Repeat(TList<AStar> &L)
{
    int i;
    int n = L.Getlen();
    for (i = 0; i < n; i++)
        if (L.GetData(i) == *this)
            break;
    return i;
}

//A*搜索实现
void AStar::Search()
{
    AStar T = *this;                //初始结点
    T.f = T.Calcuf();                 //初始结点的估价函数
    TList<AStar> L;              //建立队列
    L.Append(T);                    //初始结点入队
    int head = 0, tail = 0;              //队列头和尾指针
    while (head <= tail)              //队列不空则循环
    {
        for (int i = 0; i < 4; i++)              //空格可能移动方向
        {
            T = L.GetData(head);    //去队列头结点
            if (T.h == 0)                       //是目标结点
            {
                T.Printl(L);//输出搜索路径
                T.Printf();     //输出目标状态
```

```
                return;         //结束
            }
            if (T.Expend(i))           //若结点可扩展
            {
                int k = T.Repeat(L); //返回与已扩展结点重复的序号
                if (k < head)                 //如果是不能扩展的结点
                    continue; //丢弃
                T.last = head;       //不是不能扩展的结点，记录父结点
                T.f = T.Calcuf(); //计算 f
                if (k <= tail)          //新结点与可扩展结点重复
                {
                    AStar Temp = L.GetData(k);
                    if (Temp.g > T.g)           //比较两结点 g 值
                        L.SetData(T, k); //保留 g 值小的
                    continue;
                }
                T.Sort(L, head);              //新结点插入可扩展结点队列
                tail++;         //队列尾指针后移
            }
        }
        head++; //一个结点不能再扩展，队列头指针指向下一结点
    }
}

/*图形化界面相关函数*/
void GraphMain(int width, int length, int x, int y, int step) { //图形化界面主函数 起始坐标 步数

    initgraph(width, length);   // 创建绘图窗口，大小为 w*l 像素

    setbkcolor(WHITE); //设置背景色
    cleardevice(); //清理屏幕,才能显示背景色

    setlinecolor(BLUE);
    draw_struct(x, x + SIZE, y, y + SIZE); //画框线结构

    settextcolor(RED);
    settextstyle(16, 0, _T("宋体"));
    RECT r = { 0, 0, width, 200 };
    drawtext(_T("八数码问题求解"), &r, DT_CENTER | DT_VCENTER | DT_SINGLELINE);//输出文字

    draw_move(x, y, step); //方块移动
```

```
    settextcolor(RED);
    settextstyle(16, 0, _T("宋体"));
    RECT re = { 0, 0, width, 200 };
    drawtext(_T("移动结束,请按ENTER结束!"), &re, DT_CENTER | DT_VCENTER
| DT_SINGLELINE);//输出文字

    getchar();
    closegraph();//关闭图形界面

}

void draw_struct(int left, int right, int top, int bottom) { //画框线
结构

    setlinestyle(PS_DASH);//设定线性为虚线

    rectangle(left, top, right, bottom);//画底板

    setlinestyle(PS_SOLID); //恢复实线
    setfillcolor(YELLOW); //设置填充颜色
    setfillstyle(BS_SOLID);//固实填充

    for (int i = 1; i <= 3; i++) {
        for (int j = 1; j <= 3; j++) {
            fillrectangle(left + 100 * (j - 1), top + 100 * (i - 1), left
+ 100 * j, top + 100 * i);//画九个填充方块
        }
    }

}

void draw_number(int num, int x, int y, int size) {//输出数字 数字内容
起始位置 横向尺寸

    setlinecolor(BLACK);
    setlinestyle(PS_SOLID); //实线

    switch (num) {
        case 1:
            line(x + size, y, x + size, y + 2 * size);//画线
            break;
        case 2:
            line(x, y, x + size, y);
            line(x + size, y, x + size, y + size);
```

```
            line(x + size, y + size, x, y + size);
            line(x, y + size, x, y + 2 * size);
            line(x, y + 2 * size, x + size, y + 2 * size);
            break;
        case 3:
            line(x, y, x + size, y);
            line(x + size, y, x + size, y + size);
            line(x, y + size, x + size, y + size);
            line(x + size, y + size, x + size, y + 2 * size);
            line(x + size, y + 2 * size, x, y + 2 * size);
            break;
        case 4:
            line(x, y, x, y + size);
            line(x, y + size, x + size, y + size);
            line(x + size, y, x + size, y + 2 * size);
            break;
        case 5:
            line(x + size, y, x, y);
            line(x, y, x, y + size);
            line(x, y + size, x + size, y + size);
            line(x + size, y + size, x + size, y + 2 * size);
            line(x + size, y + 2 * size, x, y + 2 * size);
            break;
        case 6:
            line(x + size, y, x, y);
            line(x, y, x, y + 2 * size);
            line(x, y + size, x + size, y + size);
            line(x + size, y + size, x + size, y + 2 * size);
            line(x, y + 2 * size, x + size, y + 2 * size);
            break;
        case 7:
            line(x, y, x + size, y);
            line(x + size, y, x + size, y + 2 * size);
            break;
        case 8:
            line(x, y, x + size, y);
            line(x, y, x, y + 2 * size);
            line(x + size, y, x + size, y + 2 * size);
            line(x, y + size, x + size, y + size);
            line(x, y + 2 * size, x + size, y + 2 * size);
            break;
        default:
            break;
```

```
    }

}

void draw_move(int x, int y, int step) { //方块移动

    ifstream input("result.txt", ios::in | ios::_Nocreate);//读文件

    string a;
    int order[9]; //记录排列顺序
    int orderBefore[9] = { 0 }; //记录前一组的顺序
    int count;//计数

    for (int i = 1; i <= step; i++) {
        input >> a;//读过序号
        count = 0;
        for (int j = 1; j <= 9; j++) {//读入顺序
            if (count == 3)
                count = 1;
            else
                count = count + 1;//计数

            input >> order[j - 1];

            if (orderBefore[j - 1] != order[j - 1]) { //方块移动处
                orderBefore[j - 1] = order[j - 1]; //赋为新值

                if (order[j - 1] == 0) {//空位
                    setlinestyle(PS_SOLID); //恢复实线

                    setfillcolor(YELLOW); //设置填充颜色
                    setfillstyle(BS_SOLID);//固实填充

                    fillrectangle(x + 100 * (count - 1), y + 100 * ((j -
1) / 3), x + 100 * count, y + 100 * ((j - 1) / 3 + 1));//输出空位
                }
                else {//0 位置为非 0 位
                    setlinestyle(PS_SOLID); //恢复实线
                    setfillcolor(GREEN); //设置填充颜色
                    setfillstyle(BS_SOLID);//固实填充

                    fillrectangle(x + 100 * (count - 1), y + 100 * ((j -
1) / 3), x + 100 * count, y + 100 * ((j - 1) / 3 + 1));//输出底色
```

```
                    draw_number(order[j - 1], x + 100 * (count - 1) + 37,
y + 100 * ((j - 1) / 3) + 25, 25);//输出数字标记
                }
            }
        }
        getchar();
    }

    input.close(); //关闭文件

}

int main()
{
    AStar aStar("init.txt", "dis.txt");
    aStar.Search();

    GraphMain(550, 550, 125, 125, step);//图形化

    return 0;
}
```

【实验结果及结论】

文件中保存的初始状态:

```
8 3 5 1 2 7 4 0 6
7
1 2 3 4 5 6 7 8 0
```

（可对初始化文件进行修改）

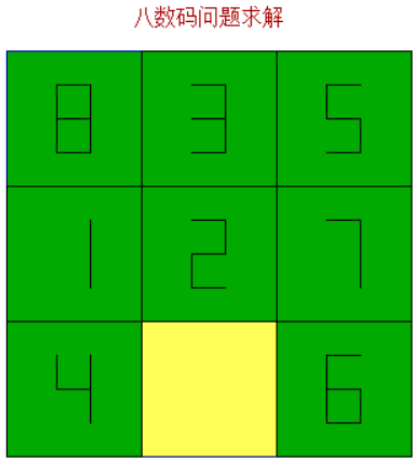
结点距离估计表：

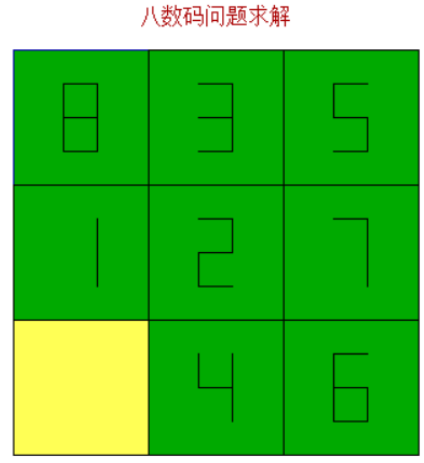
```
0 1 2 1 2 3 2 3 4
1 0 1 2 1 2 3 2 3
2 1 0 3 2 1 4 3 2
1 2 3 0 1 2 1 2 3
2 1 2 1 0 1 2 1 2
3 2 1 2 1 0 3 2 1
2 3 4 1 2 3 0 1 2
3 2 3 2 1 2 1 0 1
4 3 2 3 2 1 2 1 0
```

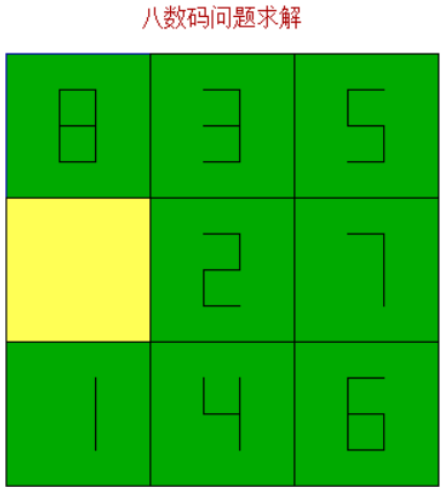
程序运行结束后输出文件状态：

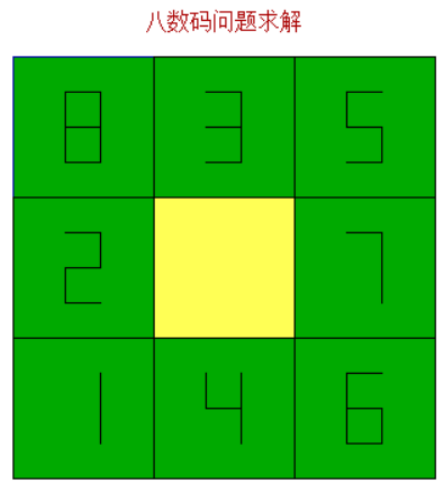
```
0step: 8 3 5 1 2 7 4 0 6
1step: 8 3 5 1 2 7 0 4 6
2step: 8 3 5 0 2 7 1 4 6
3step: 8 3 5 2 0 7 1 4 6
4step: 8 3 5 2 7 0 1 4 6
5step: 8 3 0 2 7 5 1 4 6
6step: 8 0 3 2 7 5 1 4 6
7step: 0 8 3 2 7 5 1 4 6
8step: 2 8 3 0 7 5 1 4 6
9step: 2 8 3 1 7 5 0 4 6
10step: 2 8 3 1 7 5 4 0 6
11step: 2 8 3 1 0 5 4 7 6
12step: 2 0 3 1 8 5 4 7 6
13step: 0 2 3 1 8 5 4 7 6
14step: 1 2 3 0 8 5 4 7 6
15step: 1 2 3 4 8 5 0 7 6
16step: 1 2 3 4 8 5 7 0 6
17step: 1 2 3 4 0 5 7 8 6
18step: 1 2 3 4 5 0 7 8 6
19step: 1 2 3 4 5 6 7 8 0
```

程序运行截图：

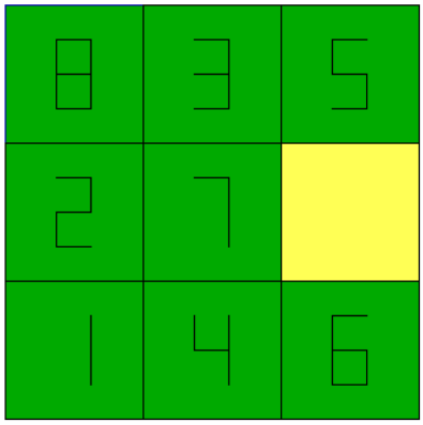




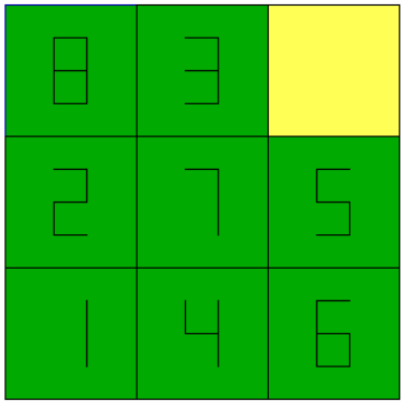




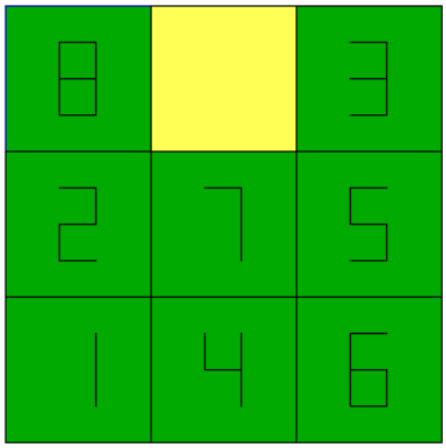
八数码问题求解



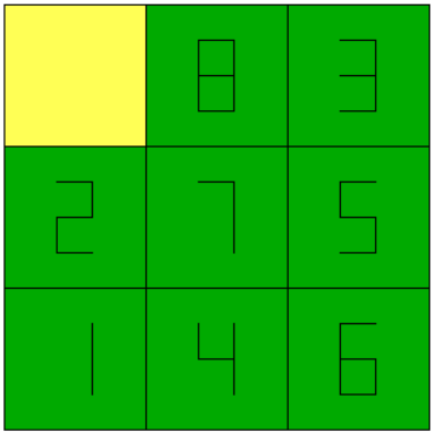
八数码问题求解



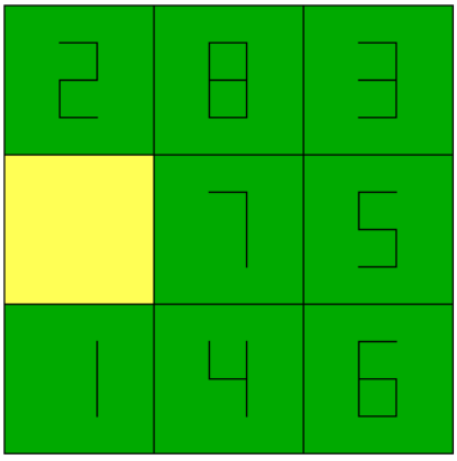
八数码问题求解



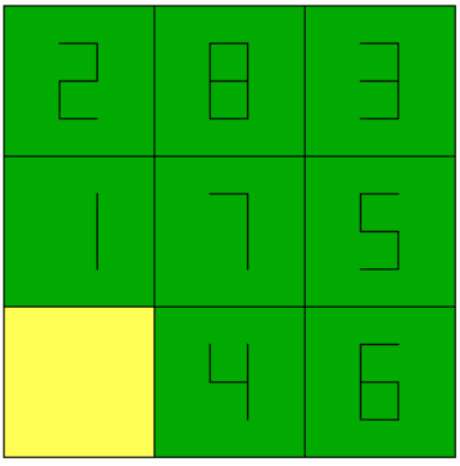
八数码问题求解



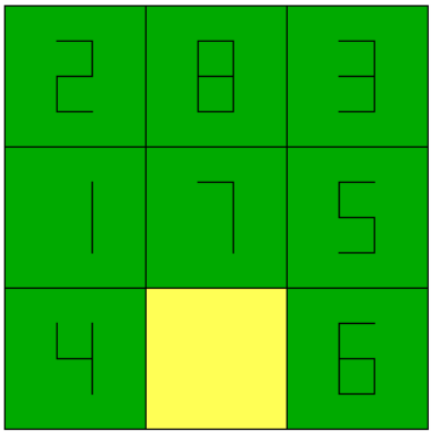
八数码问题求解



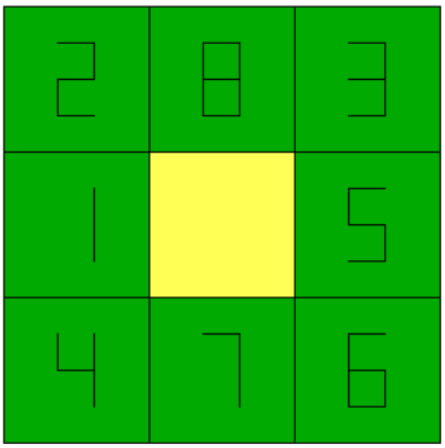
八数码问题求解



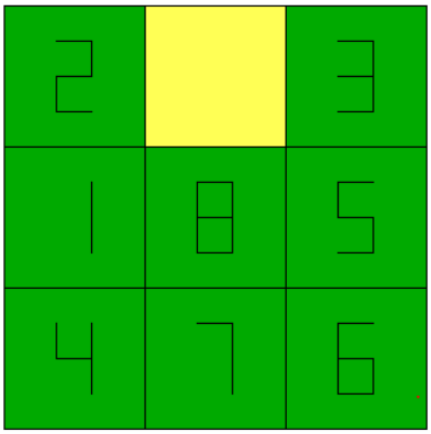
八数码问题求解



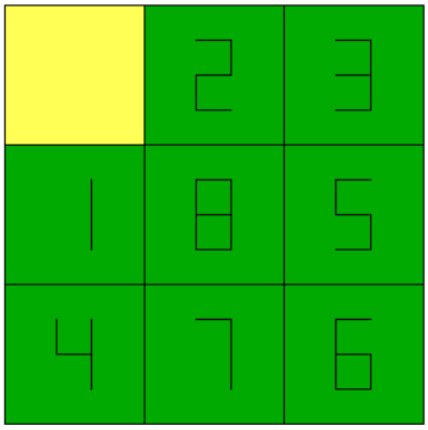
八数码问题求解



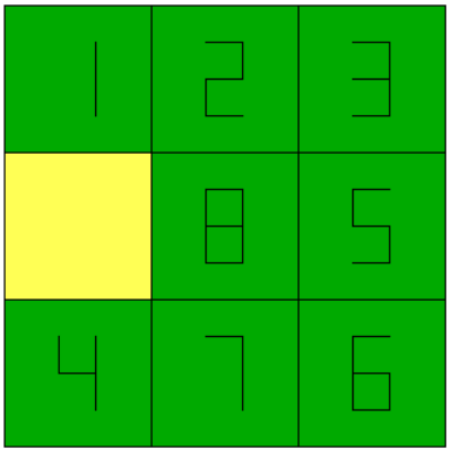
八数码问题求解



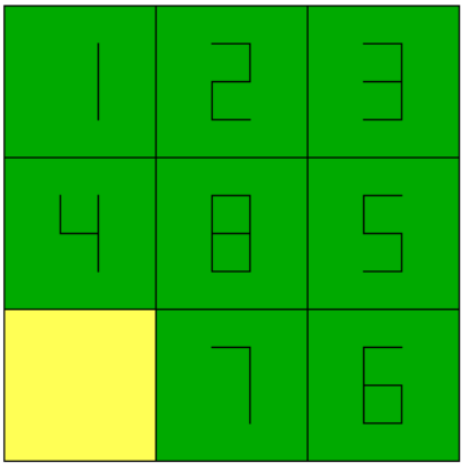
八数码问题求解

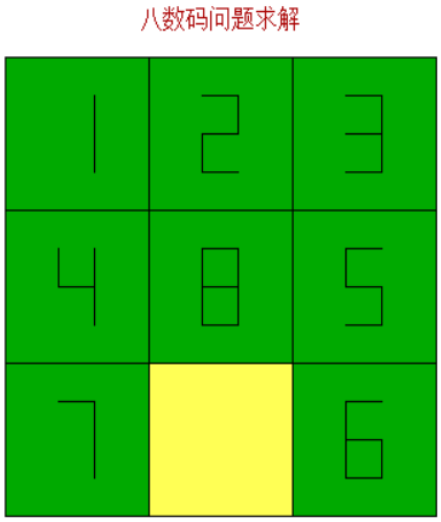


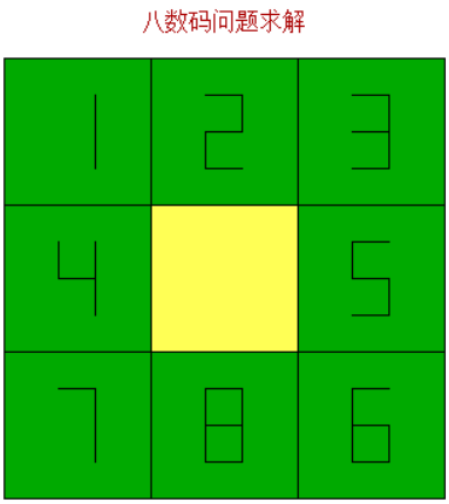
八数码问题求解

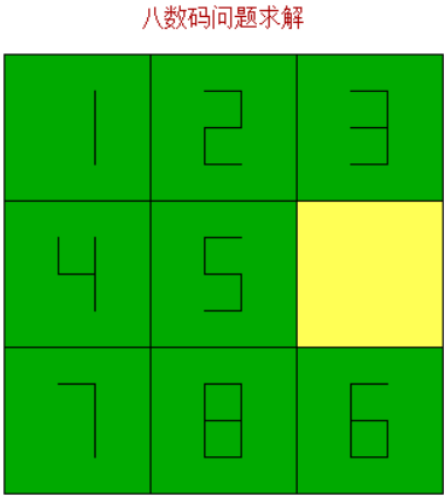


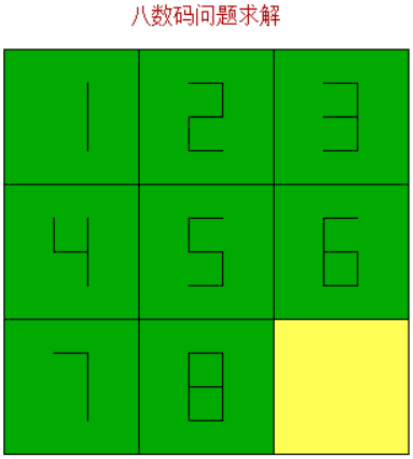
八数码问题求解

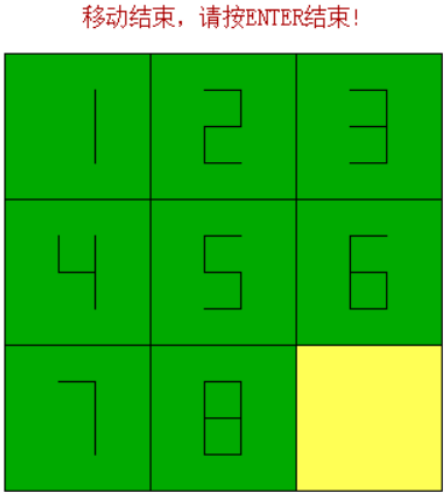












结论：

实现了 8 数码问题的 A*算法求解，并成功将求解过程图形化，实验结果符合预期，达到要求

三．参考文献

A* algorithm from wikipedia https://en.wikipedia.org/wiki/A*_search_algorithm

四．小结

通过本次实验，对 A*算法有了更加深入的认识，学会了使用 A*算法求解相应的问题。

本次实验的图形化界面使用 EasyX 库函数进行实现，Easy 库函数实现相应的图形化界面较为简单。在对代码进行编译时，需要安装相应的库。

本次实验的数据均以文件的形式给出，包括初始化数据，距离表，以及求解路径。这样只需要在运行程序前对相应的文件进行修改，即可改变初始的条件，无需重新编译，更加简单方便。

通过八数码的具体例程的实现，增强了我们的 C++编程能力，同时也真正体会到了搜索算法在生活学习中的广泛应用，整体程序采用类与继承的方式实现，封装良好，图形化部分单独完成，查错方便，更改容易，具备进一步提升性能和移植的条件。