数据结构大作业报告
八数码

姚皓天 (2013011515)

2014 年 11 月

目录

1 基本原理

1.1 概述

本程序采用 Microsoft Visual Studio 2012 的“MFC 应用程序”开发，实现了简单地八数码的搜索算法和演示功能。实现了三种算法，深搜、广搜、和 A* 算法。

因为程序功能有限，使用了基本的单对话框模板实现程序功能。

1.2 原理

在 MFC 的基本框架之上，首先编写了 EightFigureState 类，用于统一使用一个 int 变量来描述八数码的的状态，同时还保存搜索中产生的相关信息。SearchCore 类作为作为基类派生所有的搜索方法。CEightFigureDlg 类用于控制图形界面。

1.3 特色

- 使用 SearchCore 类作为作为基类，派生出所有的搜索方法，每个方法复写虚函数 Search，使用统一的接口，方便编程与扩展。
- 开发过程中使用了 VS 的单元测试框架对几个重要类的方法进行了测试，大大提高了开发成功率，缩短了开发时间。
- 搜索中访问过的状态使用 Hash Set 保存，可以取得很快的速度。

1.4 界面

Figure 1: 界面

1.5 操作说明

1.5.1 始末状态

始末状态可以手动设置，也可以随机生成。（始末状态会通过逆序数方法检查是否可解，如果不可解就会提示用户；随机生成的组合都是可解的）

1.5.2 搜索方法

程序提供了 3 种搜索方法，可以通过下拉列表选择。
在 A* 算法中可以选择估值函数，在 DFS 算法中可以设置搜索深度。

1.5.3 结果演示

结果会在左侧显示，操作方法与普通视频播放器的风格类似，右侧给出了搜索用时和访问的状态数。

2 程序设计

2.1 需求分析

程序需要采用不同的搜索方法来求解八数码问题。

2.2 概要设计

在 MFC 的基本框架之上，首先编写了 EightFigureState 类，用于统一使用一个 int 变量来描述八数码的的状态，同时还保存搜索中产生的相关信息。SearchCore 类作为作为基类派生所有的搜索方法。CEightFigureDlg 类用于控制图形界面。

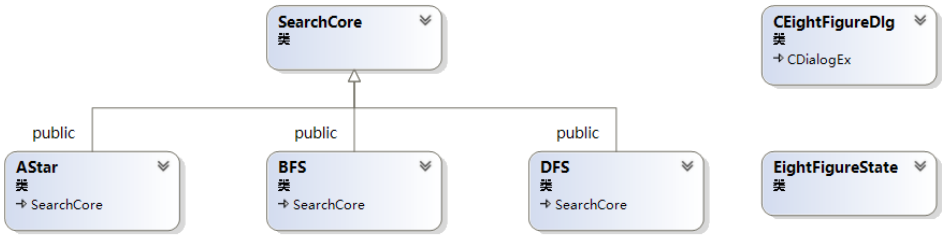

Figure 2: 类图

2.3 详细设计

2.3.1 EightFigureState 类

EightFigureState 类，用于统一使用一个 int 变量来描述八数码的的状态，同时还保存搜索中产生的相关信息，并配合封装了一些方法来实现基本的功能。

```
class EightFigureState
{
private:
        .....
public:
    int data;                    //使用一个int
       类型表示八数码的状态
    int depth;
    int selfIdx;
    int fatherIdx;               //搜索过程中父节点的编号
    int fVaule;                  //启发函数计算值
    ......
    bool Move(Direction direction);
        //控制空白块向某个方向移动
    bool CanSolve(EightFigureState c);
    bool operator == (EightFigureState c);
    void operator = (EightFigureState c);
};
```

2.3.2 SearchCore 类

SearchCore 类作为所有搜索方法的基类，用于规定统一的接口

```
class SearchCore
{
public:
    void SetStart(EightFigureState state);
    void SetTarget(EightFigureState state);
    void GetPath(std::vector<EightFigureState> &path);
    int GetTime();
    int GetStateCount();
    virtual bool Search() = 0;
        //定义了虚函数，在子类中实现搜索方法
protected:
    EightFigureState startState;
    EightFigureState targetState;
    std::vector<EightFigureState> path;
                //vector 保存搜索出的路径
    std::hash_set<unsigned int> close;
        //hash set 用于保存访问过的状态
    clock_t startTime;
                //记录算法运行时间
    clock_t stopTime;
};
```

2.3.3 搜索算法

【BFS】广搜采用队列实现，部分代码如下：

```
bool BFS::Search()
{
    EightFigureState state,temp;
    route.push_back(startState);
    close.insert(startState.data);
    q.push(startState);
    do
    {
        state = q.front();
        q.pop();
        if (state == targetState)
        {...}
        for (int i = 0;i<4;i++)
        {
            temp = state;
            if (temp.Move((Direction)i))
            {
                if (!close.count(temp.data))
                {
                    q.push(temp);
                    close.insert(temp.data);
                }
            }
        }
    }while (!q.empty());
    return false;
}
```

【DFS】深搜采用堆栈实现，部分代码如下：

```
bool DFS::Search()
{
    EightFigureState state,temp;
    startState.depth = 0;
    route.push_back(startState);
    close.insert(startState.data);
    s.push(startState);
    do
    {
        state = s.top();
        s.pop();
        while((depth != 0) && (state.depth > depth))
        {...}
        if (state == targetState)
```

```
        {...}
        for (int i = 0;i<4;i++)
        {
            temp = state;
            if (temp.Move((Direction)i))
            {
                if (!close.count(temp.data))
                {
                    temp.depth = state.depth + 1;
                    s.push(temp);
                    close.insert(temp.data);
                }
            }
        }
    }while (!s.empty());
    return false;
}
```

【A*】A* 算法采用优先队列实现，使用函数值 f 为关键码的优先队列，部分代码如下：

```
 bool AStar::Search()
{
    EightFigureState state,temp;
    startState.depth = 0;
    startState.fVaule = 10*(this->*clac)(startState,targetState
       );
    close.insert(startState.data);
    q.push(startState);
    do
    {
        state = q.top();
        q.pop();
        if (state == targetState)
        {...}
        for (int i = 0;i<4;i++)
        {
            temp = state;
            if (temp.Move((Direction)i))
            {
                temp.depth = state.depth + 1;
                temp.fVaule = temp.depth + 10*(this->*clac)(
                   temp,targetState);
                if (!close.count(temp.data))
                {
                    q.push(temp);
                    close.insert(temp.data);
                }
```

```
            }
        }
    }while (!q.empty());
    return false;
}
```

3 设计心得

3.1 收获

感觉在这次程序设计中类的编写和封装比上一次的实现有了提高，整个编程思路显得清晰了一些。

而且在这次开发中使用了 vs 的单元测试框架，进行了一些单元测试，提高了开发的成功率，减少了调试消耗的时间，单元测试是一种很好的方法，以后会一直使用。

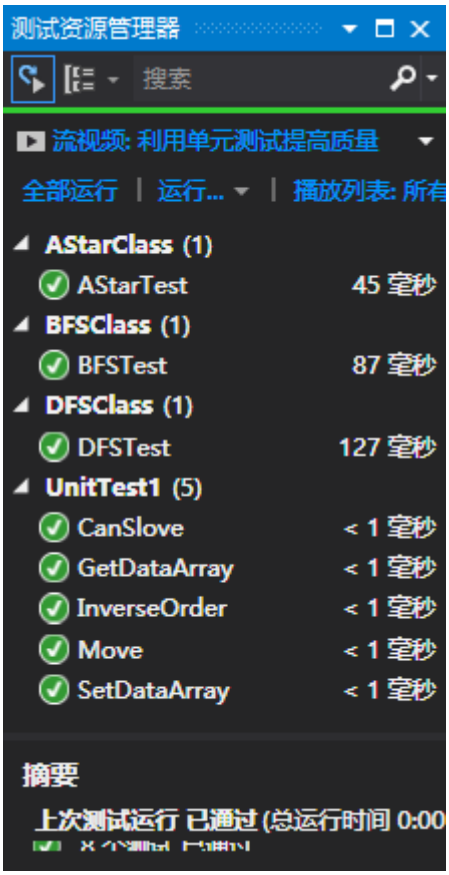

Figure 3: 单元测试

4 文件清单

- src\ 工程文件

  - src\ EightFigure \
    * EightFigureState.h/.cpp EightFigureState 类——用于描述八数码的状态
    * SearchCore.h/.cpp SearchCore 类——所有搜索方法的基类
    * DFS.h/.cpp ——深搜算法
    * BFS.h/.cpp ——广搜算法
    * AStar.h/.cpp ——A* 算法
    * EightFigureDlg.h/.cpp 界面对话框控制
  - src\ UnitTest \ 单元测试
- bin\ 可执行文件
- doc\ 文档

程序版本库：https://github.com/yht1995/EightFigure.git