推盘游戏的设计与实现

学　　院　计算机科学与技术

专　　业　计算机科学与技术

年　　级　2007 级

姓　　名　张　超

指导教师　张坤龙

2011 年 **6** 月 **15** 日

摘　　要

数字推盘游戏是一款休闲益智类游戏，其英文名称为 N-puzzle，在文曲星和手机游戏中经常能够看到其身影，有的也将数字换成图片成为一款拼图游戏。

N-puzzle 问题最早是由美国著名的游戏设计家 Sam Loyd 在 19 世纪 70 年代提出的。后来人们证明它是 NP 难问题，由于其有趣的性质和非常适合研究的状态空间，人们对于 N-puzzle 等棋盘类游戏的研究非常古老，正是对这些问题的研究才有了现今人工智能（AI）领域的蓬勃发展。

本文首先介绍了各种不同的搜索算法，并通过对 8-puzzle 问题的求解显示出了启发式搜索的优越性。然后我们选择了不同的启发函数重新对该问题求解，发现程序搜索节点的数量随解路径的增长呈指数性增加，即解路径越长，对于启发函数精确度的要求就越高。如果想要减少搜索时间，那么就必须寻求优化启发函数的方法。对于 15-puzzle 问题的求解，我们采用了模式数据库的方法，就是先计算出一部分子问题的精确解，将其当作启发值的数据库，这样的启发函数精确度很高，可以在几毫秒内解决一个随机的 15-puzzle 问题。我们还介绍了运用机器学习的方法可以对启发函数进行优化。最后是对该游戏软件的设计与实现，包括游戏需求分析、模块设计、界面设计和代码实现。

关键字：N-puzzle；启发式搜索；A*搜索；机器学习；模式数据库；软件设计

ABSTRACT

N-puzzle is a kind of puzzle game for leisure. In Wenquxing and mobile games can often see its shadow. Some of them change the numbers into pictures and become a jigsaw puzzle.

N-puzzle problem was first put forward by the famous American game designer Sam Loyd in 1970. It proved to be NP-hard problem later. Because of its interesting properties and suitable state space for study, people's study for N-puzzle and some chess games is very early, it is these studies that bring the current artificial intelligence (AI) field into flourish.

This paper first introduced some different search algorithms. And through solve the 8-puzzle problem, we shown that the superiority of heuristic search. Then we chose different heuristic functions to re-solve the problem. We found that the number of nodes increased exponentially when the solution path growth, that is the longer the path the higher accuracy will require for the heuristic function. So if we want to reduce the search time, we must find an optimization method to heuristic function. For the 15-puzzle problem solving, we used the mode database method. This method is first to calculate the exact solution of the sub-problems, and then use these solutions as the database of heuristic function. It is high accuracy that can solve a random 15-puzzle problem in several milliseconds. We also introduced that using machine learning methods can optimize the heuristic function. Finally, we design and complete this software, include the game needs analysis, module design, interface design and the program code.

Key words: N-puzzle; Heuristic search; A* search; Machine Learning; mode database; Software Design

目　　录

致　　谢

第一章　　文献综述

1.1　十五数码游戏

该问题叙述如下：在 4×4 的棋盘上放有 15 张编了号(1,2,…,15)的牌，空格牌的编号为 16，如图 1-1 所示。对于给定的初始格局，要求通过一系列的合法移动将初始格局转换成图 b 所示的目标格局。移动规则是：只有将与空格邻接的牌移到空格才是合法移动。

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13

(a) 一个初始格局

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(b) 一个目标格局

图 1-1　　15 码游戏

该游戏是由美国著名的游戏设计家 Sam Loyd（1959）在 19 世纪 70 年代设计的。15 数码游戏当时在美国很快变得非常流行，可以与较近的魔方引起的轰动相媲美。它也很快吸引了许多数学家（Johnson 和 Story，1879；Tait，1880）的注意力，如对于解是否存在的问题。后来人们发现在状态空间中存在两棵不相连的状态树，即任意状态有百分之五十的概率有解，可以通过初始状态的一个排列的奇偶性进行判断。《美国数学期刊》（American Journal of Mathematics）的主编声称“15 数码游戏在上几周里突然出现在美国公众面前，也许可以比较保险地说它吸引了百分之九十的人的注意，无论男女老少以及社会地位。”1967 年，Schofield 借助计算机的帮助对八数码问题（15 数码问题的小型版本）进行了详细的分析。1986 年，Ratner 和 Warmuth(1986)证明了 15 数码问题推广得到的一般的 n*n 版本则属于 NP 完全问题。

N-puzzle 是一个 NP 难度的组合数学问题，其算法思想的研究对于简单的博弈和人工智能（AI）方面非常基础有帮助，如启发式搜索和机器学习。4×4 的棋盘可以有 16!种不同的格局，所以它的状态空间也非常适合于研究，对于 3×3 的版本则状态空间更小。

1.2　状态空间的搜索算法

1.2.1　无信息搜索

问题的搜索策略一般分为无信息搜索和有信息搜索两类。其中无信息搜索[11]中包括：

1、广度优先搜索

广度优先搜索也称为宽度优先搜索，它首先扩展根节点，接着扩展根节点的所有后继，然后再扩展它们的后继，以此类推。广度优先搜索保证可以找到从起始状态到目标状态的最短路径，因为它先考虑每一层的所有节点然后再向图的更深空间前进，所以距离起始节点最短路径的所有状态先被访问。但对于分支多的问题搜索量巨大。

2、深度优先搜索

深度优先搜索总是扩展搜索树的当前边缘中最深的节点。搜索直接推进到搜索树的最深层，那里的节点没有后继节点。当那些节点扩展完之后，它们被从边缘中去掉，然后搜索算法“向上回到”下一个还有未扩展后继节点的稍浅的节点。它的优点为对内存的需求很少，只需要存储一条从根节点到叶节点的路径，以及该路径上每个节点的所有未被扩展的兄弟节点即可。深度优先搜索可以迅速地深入搜索空间。如果已知解路径很长，那么深度优先搜索不会浪费时间来搜索图中的大量“浅层”状态。但是深度优先搜索可能在深入空间时迷失，错过了到达目标的更短路径，甚至陷入不能产生目标的无限长路径。

3、迭代深入搜索

对于无边界的搜索树问题可以通过对深度优先搜索提供一个预先设定的深度限制来解决，这种办法称为深度有限搜索。对深度的限制解决了无穷路径的问题，不幸的是，如果我们选择的深度限制小于目标状态的深度，则将找不到解。因此一般采用迭代深入搜索，它的做法是不断的增大深度限制——首先为 0，然后为 1，然后为 2，以此类推，直到找到目标节点，当然增加的速度可以自己选择。迭代深入搜索结合了深度优先搜索和广度优先搜索的优点，它的空间需求同深度优先搜索一样小。虽然有些状态被生成多次，但事实上它确实比广度优先搜索更快。原因是在一棵每层的分支因子都相同的搜索树中，绝大多数的节点都在底层。我们可以分别计算两种方法生成的节点总数。在迭代深入搜索中，底层节点只被生成一次，倒数第二层的节点被生成两次，以此类推，生成节点的总次数为：

$$N(IDS)=(d)b+(d-1)b^2+\cdots+b^d \tag{1-1}$$

而广度优先搜索生成节点的次数

$$N(BFS)=b+b^2+\cdots b^d+(b^{d+1}-b) \tag{1-2}$$

注意广度优先搜索生成了一些第 d+1 层地节点，然而迭代深入搜索不会如此。

4、双向搜索

双向搜索的思想是运行两个同时的搜索——其中一个从初始状态向前搜索，而另一个从目标状态向后搜索，当它们在中间相遇时搜索终止。这个思想的动机

是 $b^{d/2}+b^{d/2}$ 要比 b^{d} 小得多，或者说两个小圆的面积之和比起以起点为中心达到目标的大圆的面积要小得多。可以通过哈希表检测一个节点是否属于另一棵搜索树，这样虽然时间复杂度大大降低，但代价是哈希表的空间需求。并且我们需要从目标节点向后搜索，这一点有时并不简单。

1.2.2 有信息搜索

有信息的搜索策略也称为启发式搜索[2,3]。启发式搜索是指利用问题的特定的知识对节点进行评价，选择表现的最佳的节点进行扩展，这样能够更有效的找到解。启发式搜索是人工智能研究的一个主要领域，这些算法的一个关键元素是启发函数（heuristic），记为 h(n)：h(n)=从节点 n 到目标节点最佳路径的估计代价。

图 1-2 为启发式搜索与无信息搜索的对比

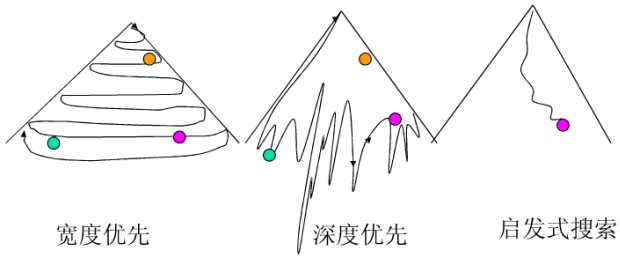

图 1-2　　启发式搜索与无信息搜索对比

1、贪婪最佳优先搜索

贪婪最佳优先搜索试图扩展离目标最近的节点，因此它只用启发函数 f(n)=h(n)来评价节点。该搜索方法同深度优先搜索一样都更倾向于沿着一条路径搜索下去直到目标，但是在遇到死路的时候会退回。并且它和深度优先搜索由同样的缺陷——它不是最优的，也是不完备的。

2、A*搜索：最小化总的估计代价

A*搜索把到达节点的实际代价 g(n)和从该节点到目标节点的估计代价 h(n)结合起来对节点进行评价：

$$f(n)=g(n)+h(n) \tag{1-3}$$

因为 g(n)给出了从起始节点到节点 n 的实际代价，而 h(n)是从节点 n 到目标节点最佳路径的估计代价，因此 f(n)=经过节点 n 的最佳解路径的估计代价。倘若启发函数 h(n)是一个可采纳的启发式，即不会过高的估计到达目标的耗散，那么 A*搜索既是完备的也是最优的。

另外还有一些 A*算法的变种[4-6]：

3、迭代深入 A*算法（IDA*）

IDA*算法是将迭代深入的思想用在启发式搜索上，它与经典迭代深入算法的

主要区别是所用的截断值是 f 耗散值而不是搜索深度，即选取上次迭代的最后一层节点中最小的 f 耗散值为下一次的迭代深度。该算法与迭代深入算法一样可以减少对存储的需求，只保留当前的 f 耗散值限制。

4、递归最佳优先搜索（RBFS）

该算法结构和深度优先搜索类似，但是它不沿着当前的不确定路径继续下去，而是记录从当前节点的祖先可得到的最佳可替换路径的 f 值。如果当前节点超过了这个限制，递归将转回到替换路径上。当递归回溯时，对回溯前的当前路径上的每个节点，用其子节点的最佳 f 值替换其 f 值。

RBFS 仅记录了上一层的 f 值信息，无法检测除了当前路径上的状态之外的重复状态，因此，它可能反复对一个状态探索多次。

5、存储限制 A*（MA*）

以上两种方案均是为了减少存储需求，但并不能保证充分利用内存。MA*算法同 A*算法一样，扩展最佳叶节点直到内存放满为止。此时删除最老的最差叶节点，以便继续扩展最新的最佳节点，这样可以保证比 A*搜索丢失的信息尽可能的少。

1.3 博弈与人工智能

1.3.1 棋盘游戏

状态空间搜索的大多数早期研究都是针对常见的棋盘游戏来实现的，比如西洋跳棋、国际象棋以及我们所研究的 15 数码游戏等，棋盘游戏的很多属性使其成为这些早期研究的理想对象。

早在 1959 年，Samuel 的西洋跳棋程序使用了启发式搜索算法。Samuel 的程序在当时是非凡的，特别还受当时计算机能力的限制。Samuel 的程序不仅为走棋应用了启发式搜索，而且用了算法来优化有限内存的使用，还实现了简单形式的学习。实际上，这个程序开创的许多技术至今还用在很多博弈和机器学习程序中。Samuel 的程序用几个不同启发尺度的带权和来评估棋盘的状态 $\sum_i a_i x_i$，其中 xi 代表一些棋盘特征，比如棋子的优势、棋子的位置、棋盘中心的控制、牺牲子换取优势的机会，甚至是棋子关于棋盘轴线的惯性力矩。ai 是可以调整的权，这时可以通过机器学习的方法不断更新权值。机器如何进行学习？这也是人工智能研究的一个重要课题。其中赫赫有名的神经网络、马尔可夫模型等都是基于一个我们熟知的概率公式——贝叶斯公式，即如果一个事件发生了，那么我们模型的参数设定应该使这件事发生的概率尽可能的高，这恰好符合人类学习过程的特点。Samuel 的程序正是通过对对手走法的学习来不断更新权值，使其达到更好的评估作用。

1.3.2 人工智能

人工智能（Artificial Intelligence，AI）是计算机科学的一个分支，它被认为是二十一世纪（基因工程、纳米科学、人工智能）三大尖端技术之一。这是一个富有挑战的学科，在近几十年来获得了迅速的发展，在很多学科领域都获得了广泛应用。如博弈、自主规划和调度、自主控制、病情诊断、机器人技术、语言理解和问题求解等。其中棋类博弈可以说是人工智能的最早应用，1997 年 IBM 的“深蓝”电脑战胜卡斯帕罗夫可以说是人工智能研究的一个重要里程碑。至今，智能搜索仍然是人工智能研究的重要课题。

1.3.3 机器学习

机器学习方法是人工智能研究的核心，它直接来源于一门古老的概率学科：统计模型拟合。机器学习的目的是通过建立适当的统计模型，从一个数据集 D 中找出有用的数据特征，这对于启发式搜索中寻求一个好的启发式是非常有用的。下面我们首先介绍由贝叶斯公式[7-9]推导出的学习过程。由一组数据 D 导出一个参数化模型 M=M(w)，M 的建立是由背景信息完成的。由贝叶斯公式我们得到：

$$P(M \mid D) = \frac{P(D \mid M)P(M)}{P(D)} = P(M)\frac{P(D \mid M)}{P(D)} \tag{1-4}$$

其中先验概率 P(M)表示在没有任何数据之前所估计的模型 M 为真的概率。后验概率 P(M|D)表示我们观测到数据集 D 后重新计算的模型 M 为真的概率。P(D|M)是指似然度。对于顺序获得的数据，我们有

$$P(M \mid D^1,\cdots,D^t) = P(M \mid D^1,\cdots,D^{t-1})\frac{P(D^t \mid M,D^1,\cdots,D^{t-1})}{P(D^t \mid D^1,\cdots,D^{t-1})} \tag{1-5}$$

修正前的后验概率成为新的先验概率，这样就表述了一个类似学习的过程，通过输入大量的数据，我们将获得一组表征数据特征的参数集 w。

人工神经网络方法[10]（图 1-3）可以用于完成上述学习过程，它的提出源于模拟大脑的信息处理和学习过程。它设计了如反馈、前馈和分层等结构，并形象将数据的输入和输出表示为神经元，每个神经元都有一个阀值，当输入数据大于阀值时才可以通过传入下一层，这样通过大量实际的输入和输出数据，我们可以调整每个神经元的阀值，使其能够更好的预测下一次输入数据的输出结果。

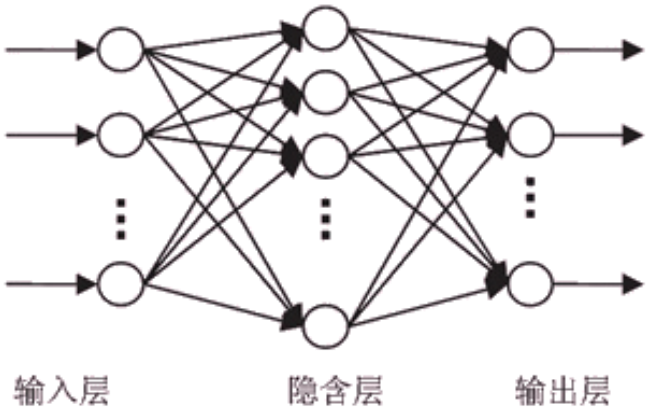

图 1-3　　人工神经网络

机器学习方法的应用包括：人脸识别、机器翻译、信息检索（google，baidu）、生物信息处理等。

1.4　论文工作

本文首先介绍了人工智能中的启发式搜索算法，并根据该算法实现了对 15 数码问题的求解，最后完成了该课题的目标——设计并实现这款数字推盘游戏。

第二章用启发式搜索算法对实际的 8 数码问题进行求解，并对比了无信息搜索和启发式搜索的性能差异。

第三章讨论了如何对 15 数码问题的启发函数进行优化，包括模式数据库方法和机器学习方法。

第四章为数字推盘游戏的设计与实现，包括需求分析、模块设计、窗体设计和代码实现等。

第五章为论文的总结。

第二章　八数码问题求解

对于我们要制作的数字拼图游戏，其核心功能为能够提供一个自动的解法。下面我们就用上一章讲到的搜索算法解决这一实际问题，算法使用 C#语言来实现。

2.1　宽度优先搜索

8-puzzle 的状态空间总数为 9!/2=181440，首先采用无信息的暴力搜索方式求解。

对于每个状态节点用一个 9 位的十进制数表示，并且每个节点都需要保存其父节点的信息，这样做有两个目的，一是可以避免重复的上下或左右移动，二是可以利用这些信息来返回最终解路径。另外，在搜索过程中需要避免重复状态的访问，因此需要建立 Open 表和 Closed 表分别存储待访问的节点和已访问过的节点。对每次产生的新节点，判断其是否已在 Open 表和 Closed 表中，若不在则加入 Open 表进行排队。由此就可以保证能够访问到 8-puzzle 问题的所有 181440 个状态。

1、宽度优先搜索的伪代码如下[1]：

```
begin
    Open:=[Start];
    Closed:=[];
    while Open≠[] do
        begin
            remove leftmost state from Open, call it X;
                if X=goal then return the path from Start to X
                else begin
                    generate children of X;
                    put X on Closed;
                    discard children of X if already on Open or Closed;  %循环检验
                    put remaining children on right end of Open          %排队
                end;
        end;
    return FAIL  %没有状态了
end.
```

2、程序实现：

State类为节点状态类，包含int data和State Father两个成员变量。

Program类为主程序，其成员变量为：

```
ArrayList OpenArr = new ArrayList();
ArrayList ClosedArr = new ArrayList();
int[] ten = { 100000000, 10000000, 1000000, 100000, 10000, 1000, 100, 10, 1 };
int[] Cnum = { 2, 3, 2, 3, 4, 3, 2, 3, 2 };
int[,] Cpoint = { { 1, 3, 0, 0 }, { 0, 2, 4, 0 }, { 1, 5, 0, 0 }, { 0, 4, 6, 0 }, { 1, 3, 5, 7 },
{ 2, 4, 8, 0 }, { 3, 7, 0, 0 }, { 4, 6, 8, 0 }, { 5, 7, 0, 0 } };
```

其中Open表和Closed表采用C#自带的列表类ArrayList来实现，可以方便进行插入删除操作。Cnum数组记录了每个格子能产生子节点的个数，Cpoint二维数组为对应每个格子产生子节点的确切位置，这样可以方便生成孩子节点。

方法包括：

```
private bool IsAim(State Node)
private void SetPath(State Node)
private int Contain(ArrayList Arr, int data)
private void LoopSearch()
```

其中LoopSearch为搜索函数，可以根据上述伪代码进行编码，它负责调用其它3个方法。IsAim判断是否为目标节点，SetPath返回具体解路径，Contain函数判断data数据是否在列表中存在。

2.2 A*搜索

仍然采用 9 位的十进制数表示每个节点的状态，与宽度优先搜索不同的是对每个节点都需要计算其启发值 f(n)=g(n)+h(n)，这里我们选择启发函数 h=所有棋子到其目标位置的距离和。同宽度优先搜索一样使用列表来维护状态：用 Open 列表来记录搜索的当前搜索带；用 Closed 列表记录已经访问过的状态。A*搜索中新加的一步是对 Open 表中的状态按照启发值排序，因此，每次循环都是考虑 Open 表中最有希望的状态。

1、A*搜索的伪代码如下[1]：

```
begin
    Open:=[Start];
    Closed:=[];
    while open≠[]do      %还有状态
        begin
            remove the leftmost state from open, call it X;
            if X=goal then return the path from Start to X
```

```
            else begin
                generate children of X;
                for each child of X do
                case
                    the child is not on Open or Closed;
                        begin
                            assign the child a heuristic value;
                            add the child to Open
                        end;
                    the child is already on Open;
                        if the child was reached by a shorter path
                        then give the state on Open the shorter path
                    the child is already on Closed;
                        if the child was reached by a shorter path then
                            begin
                                remove the state from Closed;
                                add the child to Open
                            end;
                end;
                put X on Closed;
                re-order states on Open by heuristic merit(best leftmost)
        end;
    return FAIL  %Open 中为空了
end.
```

在每一次迭代中，A*搜索删除 Open 列表中的第一个节点，如果这个节点为目标节点，那么算法便返回产生这个目标的解路径，如果不是目标节点，则根据游戏规则（与空格邻接的牌移到空格）产生该节点的后继。如果孩子节点已经在 Open 或 Closed 表中，那么检查对应部分的解路径以确保这个状态记录的是较短的路径，重复状态不会被保留，以确保程序不会陷入死循环且找到的解是最优的。然后把 Open 表按照启发值进行排序，把表现得最佳的节点排在 Open 表的最前面，使程序每次都扩展最佳的叶节点从而尽可能减少搜索无用的节点。

2、程序实现

同宽度优先搜索类似，State 类为节点状态类，属性包括

```
public int data;
```

```
public int fvalue;//启发值
public int n;
public State Father;
```

比宽度优先搜索多记录了每个节点的启发值和搜索层数。

Program类为程序类，其方法包括：

```
private bool IsAim(State Node)
private void SetPath(State Node)
private int Contain(ArrayList Arr, int data)
private int Sortfvalue(ArrayList Arr, int fvalue)
private void LoopSearch()
```

其中Sortfvalue方法对Open表中元素按照启发值进行排序，为减少时间复杂度，我们可以按启发值顺序插入Open表，此时可以采用二分查找的方法确定插入位置，从而一直保持Open表有序。对于计算新节点的启发值，我们可以定义一个9*9的二维数组，用来记录每两个格子之间的曼哈顿距离。在产生子节点时，只需计算交换的格子所产生曼哈顿距离的变化即可，并且注意g(n)值+1（搜索层数）。

2.3 A*搜索与无信息搜索比较

我们选择初始节点 data 值为 810357426，它与目标的实际距离为 23，即若采用宽度优先搜索则需搜索整个状态树的前 23 层。图 2-1 为两种搜索方法所需时间及其访问节点的数量。

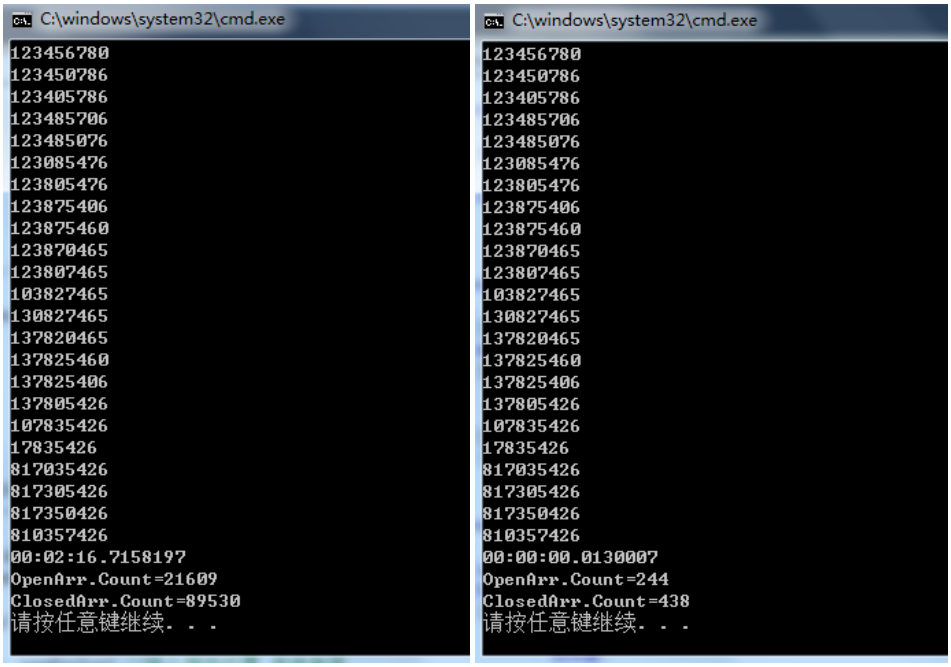

图 2-1　　A*搜索与无信息搜索性能比较

从两种算法对比可以看出，A*算法比无信息搜索访问节点的数量少几百倍，大大提高了搜索速度[11]。

2.4 程序性能分析

采用 vs2010 自带的性能分析，可以测量函数调用计数和用时（图 2-2）。

热路径

基于执行时间的开销最大的调用路径

函数名	已用非独占时间百分比	已用独占时间百分比
ConsoleApplication2.Program.Main(string[])	100.00	1.44
ConsoleApplication2.Program.LoopSearch()	90.60	8.51
ConsoleApplication2.Program.Contain(class System.Collections.ArrayList,int...	76.04	37.39
System.Collections.ArrayList.get_Item(int32)	**19.50**	**19.50**
System.Collections.ArrayList.get_Count()	**19.15**	**19.15**

相关视图: 调用关系树(A) 函数(N)

大部分时间单独工作的函数

占用最多独占应用程序时间的函数

名称	独占时间百分比
ConsoleApplication2.Program.Contain(class System.Collections.ArrayList,int32)	37.39
System.Collections.ArrayList.get_Item(int32)	20.01
System.Collections.ArrayList.get_Count()	19.44
ConsoleApplication2.Program.LoopSearch()	8.51
System.DateTime.get_Now()	6.39

图 2-2　　A*程序性能分析

我们发现程序的大部分时间用在了 Contain 函数上，即检测每个新节点是否已被访问过。由于随着列表长度的加大，所以 Contain 函数的工作量也明显增大。因此我们可以给 Open 表和 Closed 表分别建一个哈希列表，用来标记每个状态的访问情况，这样 Contain 函数可以直接从哈希表中判断节点是否在列表中，省去了每次判断都需从头检测的时间。

第三章　十五数码问题求解及启发函数的优化

3.1　十五数码问题求解

对于 4*4 版本，我们用 16 字节的字符串来标记每个节点的状态。由于此时 4*4 版本的平均步长为 100 多步（3*3 版本平均在 20 步[12]），因此如果采用和 3*3 版本同样的启发函数将很难在短时间内求得解路径，因此我们要寻求更好的启发函数[13-15]。

3.1.1　启发函数精确度对搜索性能的影响

对于启发函数的选择，可以采用松弛问题的最优解耗散当作启发式。由于该问题的解耗散肯定不低于其松弛问题的解耗散，因此该启发式是可采纳的，即能保证所求解为最优解。对于我们的数字拼图游戏，可以将问题描述如下[16]：

一个棋子可以从 A 格子移动到 B 格子，如果 A、B 相邻且 B 格子为空，可以去掉其中一个或两个限制条件，生成三个松弛问题：

1、一个棋子可以从 A 格子移动到 B 格子，如果 A、B 相邻

2、一个棋子可以从 A 格子移动到 B 格子，如果 B 为空

3、一个棋子可以从 A 格子移动到 B 格子

在上面 3*3 版本的 A*搜索中，我们选取启发函数 $h_2(n)$=所有棋子到其目标位置的距离之和，也称为曼哈顿距离，对应上面第一种情况。另外我们也可以选择启发函数 $h_1(n)$=错位的棋子数，对应上面第三种情况。从上面的描述容易看到，第三种情况是第一种情况的松弛问题，因此 h_2 比 h_1 具有更高的信息度。对于搜索空间中的所有状态均满足 $h_1(n) \leqslant h_2(n)$，显然 h_2 是更好的启发。对于无信息的宽度优先搜索，可以看作是 $h_0(n)=0$ 的 A*搜索。下面我们对这 3 个启发的性能进行对比。

随机生成 50 个有解的 8 数码问题，它们的解长度从 5 到 25，并挑选其中分布较均匀的。分别用上述 3 个启发进行搜索，并记录其访问节点总数，作图如下：

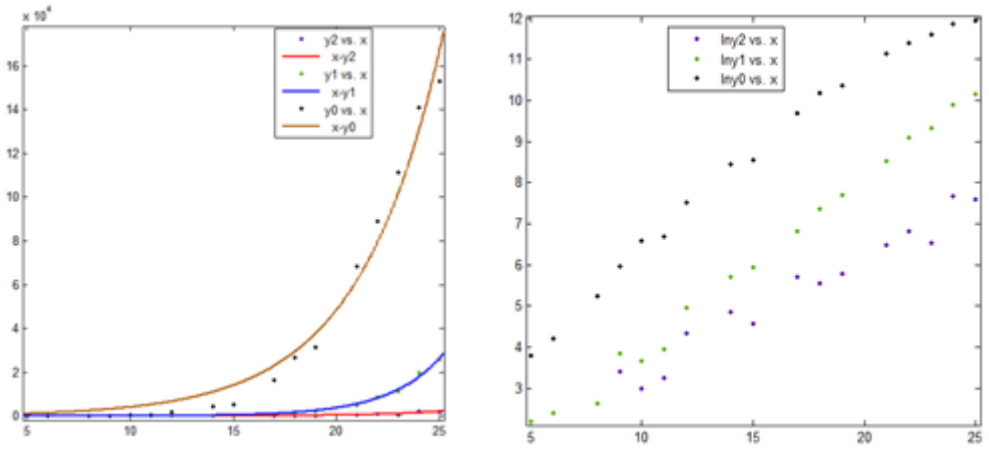

图 3-1　　3 种启发性能对比

其中横坐标为解长度，纵坐标为访问节点总数，右图中的纵坐标采用了对数坐标。从图中可以看出随着解长度的增加，访问节点的数量呈指数增长。在解路径很长时，三种启发表现出的性能差异巨大。

3.1.2　模式数据库

对于 4*4 版本，我们发现再用曼哈顿距离来近似每个棋子的实际移动距离已经不那么精确，因此需要寻找更好的启发。模式数据库的方法[17-19]仍然是从求解松弛问题来获得可采纳的启发，它是求出每个可能的子问题的精确解（如四个棋子和一个空位组成的每个可能局面），然后对搜索中遇到的每个完全状态，通过在数据库里查找出相应子问题的布局都能计算出一个可采纳的启发式。该数据库本身的构造是通过目标状态向后搜索并记录下每个新遇到模式的耗散完成的。

1-2-3-4 的选择是随意的，我们也可以构造 5-6-7-8 或者 2-4-6-8 的数据库。虽然 1-2-3-4 数据库和 5-6-7-8 数据库的子问题看起来没有重叠，但他们得到的启发式相加后就不是可采纳的了，因为对于给定的状态 1-2-3-4 子问题的解和 5-6-7-8 子问题的解几乎肯定有一些重复的移动——不移动 5-6-7-8,1-2-3-4 也不可能移入正确位置，反之亦然。不过我们可以只计入 1-2-3-4 的移动次数。那么很容易看出两个子问题的耗散之和仍然是求解整个问题的耗散下界。这种方法称为无交集的模式数据库[16]。下面我们就用该方法求解 15-puzzle 问题，其生成节点数比使用曼哈顿距离为启发函数时减少了 10000 倍。

3.1.3　生成数据库程序实现

State 类为节点状态类，属性包括：

```
public int data;
public int step;
public int Index;
```

其中 data 为 1,2,3,4 和空格所在位置所组成的一个 5 位 16 进制数；step 记录从起始节点到该节点所走距离；Index 记录父节点的移动信息（为避免下次移动时重复），其值为{-4,-1,1,4}，分别代表上下左右。

主程序成员变量：

```
State[] StateArr = new State[1048576];
ArrayList OpenArr = new ArrayList();
int[] Cnum = { 2, 3, 3, 2, 3, 4, 4, 3, 3, 4, 4, 3, 2, 3, 3, 2 };
int[,] Cpoint = { { 1, 4, 0, 0 }, { 0, 2, 5, 0 }, { 1, 3, 6, 0 }, { 2, 7, 0, 0 }, { 0, 5, 8, 0 },
```

```
{ 1, 4, 6, 9 }, { 2, 5, 7, 10 }, { 3, 6, 11, 0 }, { 4, 9, 12, 0 }, { 5, 8, 10, 13 }, { 6, 9, 11,
14 }, { 7, 10, 15, 0 }, { 8, 13, 0, 0 }, { 9, 12, 14, 0 }, { 10, 13, 15, 0 }, { 11, 14, 0,
0 } };
int data0 = 4671; //0x0123F
```

其中StateArr记录每个状态（共16!/11!=524160）的信息，数组大小设为1048576=16^5（所有5位16进制数），是为了将StateArr当作哈希表使用，节点的data值即索引。搜索过程跟宽度优先搜索相同，通过目标状态向后搜索并记录下每个新遇到的节点及所走路程。OpenArr记录待扩展的节点，判断节点是否被访问过可以根据StateArr[data]是否为空来判断。

所有524160种状态搜索完之后，将StateArr数组中每个节点的data值和step值写入txt文件，成为1234子问题数据库，用同样的方法可以得到5678、9ABC、DEF0共4个模式数据库。在求得DEF0（0代表空格）数据库时，我们发现若仅有3个数字，所求的实际移动步数同曼哈顿距离相同，因此子问题数据库的下限为必须有4个数字。

3.1.4 求解十五数码问题程序实现

根据上面求得的模式数据库，我们用4个数据库中的step值相加求得每个完全状态的启发值，并利用A*搜索求得解路径。搜索程序同3*3版本相同，仅节点数据表示方法和求启发值有所改变。

程序实现如下：

State类

```
public String data;
public int fvalue;
public State Father;
```

为方便计算，每个节点的data值我们采用16个数字的位置来记录，如节点状态为“123456789ABCDEF0”则data值为“0123456789ABCDEF”。

主程序成员变量及方法说明如下：

```
ArrayList OpenArr = new ArrayList();
ArrayList ClosedArr = new ArrayList();
int[,] modeDB = new int[4, 1048576];
int[] Cnum;
int[,] Cpoint;
String goal = "0123456789ABCDEF";
```

```
public void readDB()//将txt文件读入modeDB数组
public int getfvalue(String pdata)//每个数据库中step值相加求得启发值
public String DataToPosition(String data)//将状态节点值转换为用位置来表示
private bool IsAim(State Node)
private void SetPath(State Node)
private int Contain(ArrayList Arr, String data)
private int Sortfvalue(ArrayList Arr, int fvalue)
private String ChangeString(char[] datac, int p, int u)//将字符串p、u位置字符对调
public int CharToNum(char c)//将字符转换为相应数字
public char NumToChar(int n)//将数字转换为相应字符
private void LoopSearch()
```

3.2 机器学习方法优化启发函数

启发函数h(n)是用来估计从节点n到目标节点的解耗散的。智能体如何才能构造这样的一个函数？一个方案是设计一个很容易找到最优解的松弛问题，在上面的分析中我们都是这样选择启发函数的。另一个方案就是从经验里学习[20]，例如，我们可以求解大量的问题。每个问题的最优解都提供了可学习的h(n)的实例。每个实例都包括解路径上的一个状态和从这个状态到达解的耗散。通过这些例子，我们可以用一个归纳学习算法来构造能够预测其他状态解耗散的函数，这个函数就是我们找的启发函数。

这个归纳学习可以用上一章提到的机器学习算法来实现，当然，在我们提供了一些问题的有效特征时算法的实现将会变得简单。例如我们可以使用多个不同特征的线性叠加来构造启发函数：

$$h(n) = c_1 x_1(n) + c_2 x_2(n) \tag{3-1}$$

上面的式子与直线拟合类似，因此可以用最小二乘法拟合直线来求得参数c1、c2。我们选择的特征越好，则最后各点偏离直线的程度就越小。

对于我们的问题，可以选择特征一为“曼哈顿距离”。特征二为“颠倒棋子的数量”，即相邻的棋子要满足目标顺序必须交换位置的情况，选择这样的特征是由于我们意识到了相邻棋子换位的难度，此时需要的移动远不止两次，因为各棋子必须相互绕来绕去。这与上面讲到的模式数据库中移动第四个棋子构成一排时所需的移动大于曼哈顿距离类似。

第四章　　软件设计与实现

4.1　需求分析

4.1.1　游戏如何吸引玩家

对于我们设计的这款软件属于游戏软件，首先我们需要分析下游戏应该在哪些方面吸引玩家：

1、挑战与成就

挑战在竞技类游戏中体现得淋漓尽致，现今多数游戏都支持网络对战，玩家在一起切磋技艺或同心协力完成同一目标，成就系统记录玩家在游戏过程中取得的成绩并给予一定的奖励和称号，这是吸引玩家的一个重要地方。

2、交流

游戏中交流包括多种形式，如聊天、交友等，这样可以使玩家认识越来越多的朋友，其便捷程度丝毫不亚于 QQ 等即时聊天系统，能够和朋友一起聊天游戏是最能留住玩家的一点。

3、满足审美需求

每个游戏者都希望能够看到精美的游戏画面，这也是人们满足审美需求的心理，因此现在制作的游戏都追求精致的游戏画面，给人以视觉冲击效果的技能释放等。交互界面和游戏画面的设计关乎整个游戏的成败，即使你的游戏非常有意思，如果没有做好外观的话也不会吸引玩家去玩。

4、消遣休闲

玩家可以在工作之余，找到一个虚拟的世外空间，无拘无束，游戏过程中，经常获得各种各样的奖励，比起迷茫地闲逛、观看枯燥无味的电视节目，更让人兴奋。

4.1.2　推盘游戏需求分析

将上面的游戏设计理念应用到我们的推盘游戏，从界面和功能两方面分别对软件进行设计：

界面设计方面应尽可能的简洁实用，且能够吸引玩家的眼球。

1、该游戏属于休闲益智类游戏，在游戏界面上应该让玩家能充分享受休闲的乐趣。

2、游戏的主要元素为数字推盘，我们使用 button 控件来表示，其外观可以进行一定的设计使得更加美观。

3、为满足人们的审美需求，推盘游戏不应该只有单调的数字版本，应该支持图片形式。这样玩家就可以选择自己喜欢的图片进行拼图。

4、窗体设计方面，应该支持不同的皮肤选择，另外可以设计不规则窗体等，更

能吸引玩家的眼球。

功能设计方面应尽可能的方便用户操作，并且尽可能的增加游戏的可玩性。
1、由于该游戏为拼图游戏，应该提供玩家能随时看到完整图片的功能。
2、另外应该提供保存游戏和装载游戏的功能，以便玩家可以在一段时间后继续完成未完的游戏。
3、为增加游戏可玩性，应该有不同的难度选择，以及不同的外观设计。我们这里选择了 3*3 和 4*4 两个版本的数字拼图游戏。在外观设计上，选择了数字和图片两种形式，其中对于图片的选择支持用户自定义的形式。
4、为激发玩家的兴趣，应该设立“N-puzzle 英雄榜”，统计每次玩家胜利所需的时间以及移动的步数，记录其中的前几名，在每次玩家破纪录时应该给予提示信息，从而激发玩家不断进行挑战。
5、该软件的核心功能为能够自动给出从任一状态到目标节点的解路径，我们可以采用搜索分析的方式和 solve one step 的方式编写用户界面
6、游戏的主要操作为滑块的上下左右移动，为了尽可能的满足不同用户的需求，在程序设计时，我们采用了键盘操作、鼠标操作、菜单操作 3 中不同方式。
7、在游戏过程中，程序应能够提供悔棋功能和自动打乱功能，在程序开始时应有进度条的提示。
8、游戏软件应有帮助文档，以方便玩家更快上手以及了解该游戏的玩法。

4.2 游戏设计与实现

4.2.1 功能模块及其接口

根据上面的需求分析，游戏的功能模块设计如下：
1、游戏操作模块

该模块由主窗体程序负责完成，创建新游戏时包括产生一个随机数组Arr（各位数字都不相同）以及各变量的初始化，包括判断游戏是否胜利（bool win）、游戏size（3*3或4*4）、是否为图片（bool picture）以及图片名称（String bmpname）等。在游戏过程中，提供保存和装载游戏的功能，我们选择txt文件进行保存，在建立文件对话框时设定好初始路径InitialDirectory和过滤器Filter。保存数据依次为游戏难度、是否为图片、图片所在路径名以及Arr数组数据。在打开文件时需要调用OptionChange函数，它完成各button控件的刷新和图片的剪裁等，相当于新游戏的初始化。另外滑块移动、悔棋、顺序打乱等操作也由主窗体程序负责，这些操作都是对Arr数组进行的，它记录了当前数字的排列顺序。然后再对button控件的信息进行更改就可以将这些操作显示在界面上。
2、游戏胜利模块

该模块功能为在游戏胜利时给出 Congratulations 的信息。输入数据为该盘所用时间 Span 和所走步数 step，在模块加载时需要从记录的文件中读出统计信息，并和 Span、step 进行比较，若此次游戏能够排上“N-puzzle 英雄榜”，则程序给出“新的记录”提示信息，并将此次游戏时间和步数写入记录文件。

3、选项模块（Options）

该模块提供了让用户重新选择游戏难度和游戏外观（图片还是数字）的功能。输入数据为当前游戏难度跟当前图片的名称，若是数字形式则图片名称为空。这样在 Option 窗体装载时，可以显示当前用户的游戏选择情况，如要修改，则可选择其他单选按钮。程序提供了图片玩家自定义的功能，因此在自定义按钮选中时需要新建一个文件夹对话框。若玩家修改了选项，则需调用主窗体中的 OptionChange 函数完成修改，该模块的输出数据为修改后的游戏 size、是否为图片及图片名称（含路径）。

4、完整图片预览模块（Preview solution）

该模块使玩家能够在游戏过程中对图片进行预览，其输入数据为当前游戏 size 和图片的名称。对于数字形式我们预先保存了 PrtSc 好的两张图片。

5、自动求解模块

该模块已在上两章中讨论了具体实现，输入数据为当前游戏 size 和 Arr 数组。输出包括当前节点值的具体解路径、搜索时间、访问节点数量等，并且需要提供能在主程序中自动显示求解过程的功能。

4.2.2 各窗体具体实现

1、主窗体（Form1.cs）

根据上述功能模块的划分，将程序菜单设计如下：

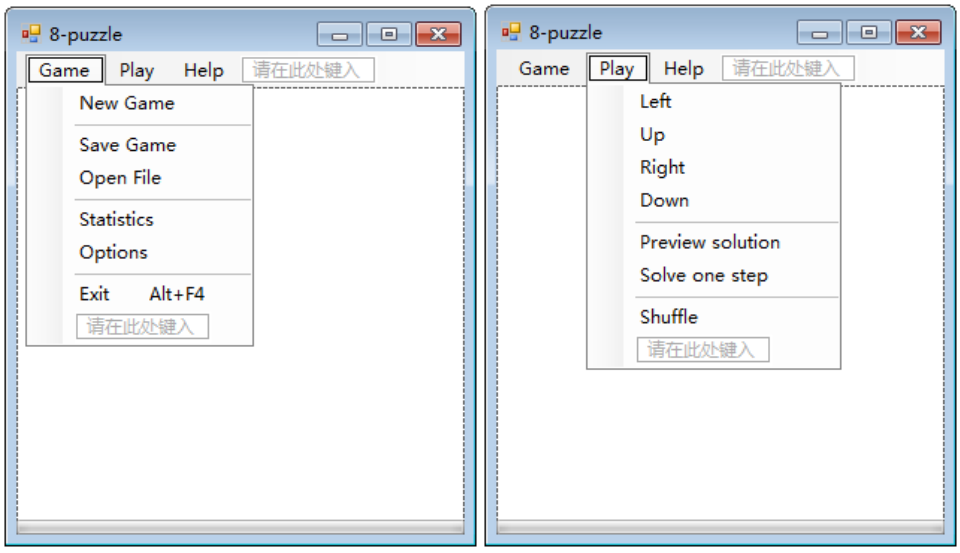

图 4-1　　主窗体菜单

以及程序运行后新添加的 button 控件

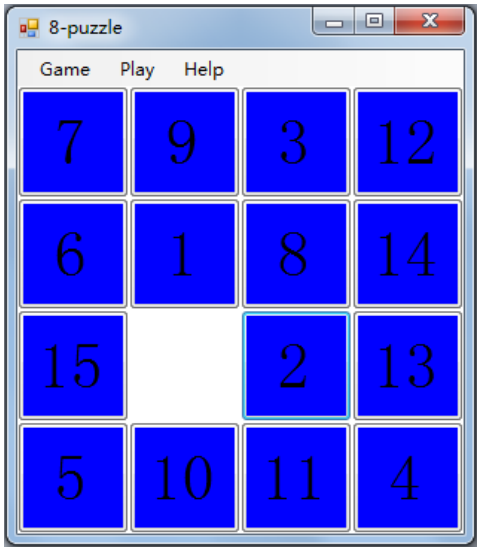

图 4-2　　主窗体添加的 button 控件

2、统计信息窗体（Statistics.cs）

统计信息窗体只需在加载时读入记录文件并显示即可，功能包括重置信息，即将所有信息清零，并写入文件。

图 4-3　　Statistics 窗体

窗体控件包括：

button1——“关闭”

button2——“重置”

listBox1——包含两个 Item{“3*3”, “4*4”}

groupBox1——“最佳时间”

groupBox2——“最佳步数”

label1-label8——显示统计信息

3、选项窗体（Option.cs）

预设了 3 幅图片供选择（StarCraft II.jpg/Frost Wyrm.jpg/constellation.jpg）。图片支持用户自定义功能，游戏 size 用户自定义功能为带扩展功能。

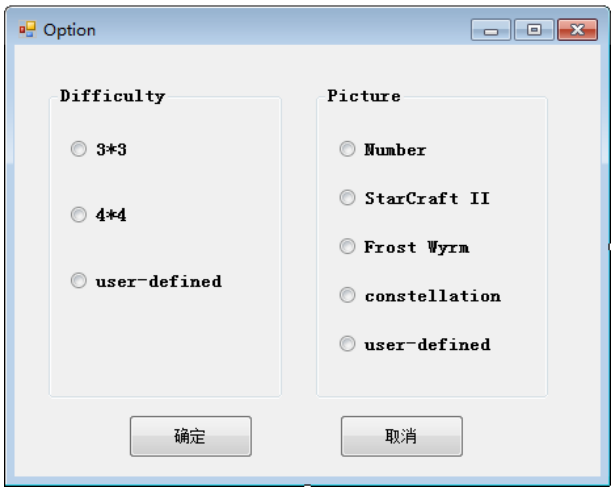

图 4-4　　Option 窗体

窗体控件包括：

button1——“确定”

button2——“取消”

groupBox1——“Difficulty”

groupBox2——“Picture”

radioButton1-radioButton8——各项单选按钮

4、游戏胜利窗体（Win.cs）

图 4-5　　Win 窗体

窗体控件包括：

button1——“退出”

button2——“再来一局”

label1——“Congratulations! You win!”

label2——“难度：”

label3——“时间：”

label4——“步数：”

label5-label8——记录该局各项信息

5、Solve one step 窗体

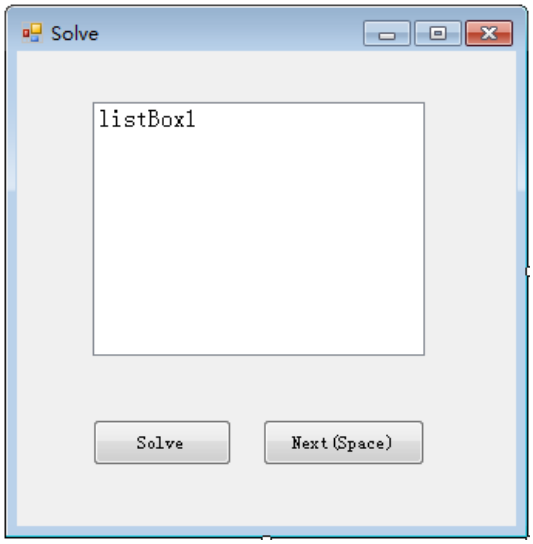

图 4-6　　Solve 窗体

窗体控件包括：

button1——“Solve”

button2——“Next(Space)”

listBox1——显示求解信息

4.3　主窗体程序代码实现

1、成员变量

记录游戏模式的主要有游戏size、是否为图片、图片名称等。

```
private int n = 3;
private int size = 9;
private bool picture = false;
private String bmpname = null;
private int[] Arr = new int[16]; //记录数字排列顺序
private Button[] btn = new Button[16];//button控件数组，当作推盘
private Bitmap bmp0 = null;//bmp0存储图片
private Bitmap[] bmpsize = new Bitmap[16];//该数组存储剪裁后的图片
private int X, Y;//记录窗体大小，在改变窗体大小时使用
private bool win = false;
```

```
private DateTime start;
private TimeSpan span = new TimeSpan();//记录游戏所用时间
private int step = 0;//记录游戏所走步数
```

2、方法及说明

1）public void GetInitialFormSize()——获得客户区的大小，赋给X、Y。

2）private void Shuffle()

该函数为打乱顺序，具体做法是将目标状态随机的向前操作若干次，从而达到打乱的效果。每次随机生成0至3的随机数，分别代表上下左右移动。首先我们要找到数组中0（代表空格）的位置，然后判断其是否在边缘，能否完成移动操作，不能完成则跳过此次循环。其中空格和数字的交换由ChangArr函数完成。由于New Game菜单也是调用该函数，因此可以把游戏初始化的操作也加入其中，包括win设为false、开始时间和步数清零、进度条的显示等。

3）private void LeftMove()

private void UpMove()

private void RightMove()

private void DownMove()

以上函数为移动操作，同打乱程序类似，需要判断空格是否在边界，能否完成相应操作。如执行了移动操作则 step++。

4）private bool CheckWin()——判断当前Arr数组是否与目标数组相同。

5）private void Changebtn(int[] Arr)

该函数根据Arr数组改变button控件的信息。分数字形式和图片形式两种情况，若为图片则改变button的背景图片，若为数字则改变button的text值，并将空格button的visible属性设为false。当Arr数组为目标时，则新建Win窗体，并计算此次游戏所用时间和所走步数，并将空格button的属性还原，展现完整的图片。

6）private void ChangeArr(int[] TempArr, int p, int q)——将数组元素Arr[p]和Arr[q]换位。

7）private int GetPosition(int[] Arr)——获得空格的位置。

8）public void OptionChange(int newn,int newsize,bool newpicture,String newbmpname)

该函数为选项改变时调用的函数，参数包括newsize、newpicture和newbmpname。如果为图片形式则需将newbmpname路径的图片拷入bmp0位图文件，并根据newsize大小进行剪裁，并将剪裁后的每个bmp位图取成100*100大小赋给bmpsize数组。如果游戏size改变则需改变客户区（ClientSize）的大小为n*100。

Button 数组的初始化包括 Name、Size、Location 等属性的赋值，如果为图片则

需对 BackgroundImage 属性赋值，如果为数字形式则需对 Text 属性赋值。另外给每个 button 添加一个 Click 事件：

```
btn[i].Click += new EventHandler(Btn_Click);
```

3、事件及说明

1）private void Form1_Load(object sender, EventArgs e)

在窗体装载时，需要初始化数字形式的 3*3 版本游戏。同 OptionChange 函数中数字形式的初始化相同。

2）void Btn_Click(Object sender, EventArgs e)

该事件为每个 button 控件添加的 Click 事件，目的是让玩家使用鼠标操作可以移动滑块。该事件触发时首先需要判断周围是否有空格按钮，若有则调用 ChangeArr 和 Changebtn 函数执行移动操作。

3）private void Form1_KeyDown(object sender, KeyEventArgs e)

同Btn_Click事件相同，目的是让玩家通过键盘操作也可以移动滑块。我们设定WSAD四个按键分别为数字的上下左右移动，分别调用LeftMove()、UpMove()、RightMove()、DownMove()四个函数。

第五章　　总结

本文首先介绍了状态空间的搜索策略，并用 A*搜索对 N-puzzle 问题进行了求解，在对 15 数码求解的过程中分析了如何对启发函数进行优化。最后通过对 N-puzzle 游戏软件的实现，学习了 C#编程，并且亲身体验到了软件工程的一些知识。下面对各章作如下总结：

不同搜索策略各自的特点。其中宽度优先搜索的优点是算法简单，完备且是最优的，但缺点是空间时间复杂度往往很大。深度优先搜索的存储需求很小，但是不能够保证最优解，也是不完备的。启发式搜索中最著名的为 A*搜索，在选取可采纳的启发式时其解是最优的也是完备的。并且随着越来越精确的启发函数，该算法的搜索性能将不断提升。它的变种均是为了权衡空间复杂度和时间复杂度所设计的，比如采用哈希列表则牺牲了空间复杂度而提升了搜索速度；与深度优先搜索结合则降低了空间复杂度而放弃了解的最优性和完备性，因而其时间复杂度不能够得到保障。

第二章通过对 8 数码问题的求解，我们对无信息搜索与启发式搜索进行了比较。并且选择了不同的启发函数重新求解问题，我们发现随着解路径的变长程序访问的节点数量呈指数增加。对于不同精确度的启发函数，在解路径很长时性能差异明显。

在求解 15 数码问题时，若仍采取求解 8 数码时的启发函数将很难在短时间内找到解，因此必须对启发函数进行优化。模式数据库的想法是将问题分步求解，子问题的复杂度比整体小得多，而程序搜索时间是与解长度成指数增长的，因此模式数据库的方法将大大增加搜索速度。而该方法的缺点是数据库的生成本身也需要编码、计算，并且对于背景信息很少的未知领域，将很难得到这样的模式数据库。最后我们介绍了机器学习的方法，该方法克服了模式数据库的缺点，并且可以通过不断地学习数据，对启发函数进行调整，从而总结出问题的特征。

参考文献

[1] George F. Luger. 人工智能——复杂问题求解的结构与策略[M]. 机械工业出版社, 2006.

[2] Pearl J. Heuristics: Intelligent Search Strategies for Computer Problem Solving.[M] Addison-Wesley Pub, 1984.

[3] Hart P E, Nilsson N J, Raphael B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths[J]. IEEE Transactions on Systems Science and Cybernetics SSC4 4 (2): 100–107.

[4] Stuart Russell. Efficient memory-bounded search methods[C]. Proceeding ECAI-92.

[5] Stern R, Kulberis T, Felner A, Holte R. Search space reduction using swamp hierarchies[C]. Proceedings of the National Conference on Artificial Intelligence 1, 2010, 185-190.

[6] Felner A, Zahavi U, Holte R, Schaeffer J, Sturtevant N, Zhang Z. Inconsistent heuristics in theory and practice[C]. Artificial Intelligence 175 (9-10), 2011, 1570-1603.

[7] Barnett V. Comparative Statistical Inference[M]. John Wiley, New York, 1982.

[8] Berger J O. Statistical Decision Theory and Bayesian Analysis[M]. Springer-Verlag, New York, 1985.

[9] Press S J. Bayesian Statistics: Principles, Models, and Applications[M]. John Wiley, New York, 1989.

[10] Bishop C M. Neural Networks for Pattern Recognition[M]. Clarendon Press, Oxford, 1995.

[11] 詹志辉，胡晓敏，张军. 通过八数码问题比较搜索算法性能[J]. 计算机工程与设计,2007,28(11):2505-2508.

[12] Alexander Reinefeld. Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA*[C]. In IJCAI, 1993.

[13] Ratner D, Warmuth M. Finding a shortest solution for the N*N extension of the 15-puzzle is intractable[M]. AAAI-86 Proceedings.

[14] Cullberson J C, Schaeffer J. Efficiently Searching the 15-Puzzle[M]. The University of Alberta, Technical Report, 1994.

[15] 温安国，李松年. N 数码问题直接求解及优化研究[J]. Computer Application and Software, 2010, 27(5):266-268.

[16] Russell S, Norvig P. 人工智能——一种现代方法[M]. 人民邮电出版社, 2004.

[17] Felner A, Korf R E, Meshulam R, Holte R. Compressed pattern databases[J]. Journal of Artificial Intelligence Research, 2007, 30:213-247.

[18] Feiner A, Korf R E, Hanan S. Additive pattern database heuristics[J]. Journal of Artificial Intelligence Research, 2004, 22:279-318.

[19] Korf R E, Felner A. Disjoint pattern database heuristics[J] Artificial Intelligence, 2002, 134(1-2):9-22.

[20] Mitchell T M. Machine Learning and Data Mining[J] Communications of the ACM, 1999, 42(11):31-36.

致　　谢

本论文的完成得益于很多人的帮助、关心和支持。

首先感谢我的指导老师张坤龙老师，在论文写作期间，张老师细心指导我论文的写作内容和算法思想，并为我挑选了参考书目和英文文献。由于我是双学位的学生，并没有学习过 C#，在编程过程中遇到了很多困难，都是张老师帮助我一一解决的。张老师认真严谨的教学态度也深深的影响了我，让我终身受益。

最后衷心感谢我的同学和室友，是他们对我的关心、支持和帮助才让我顺利的完成了本科毕业论文。离别的时刻即将到来，在此我祝愿所有 07 级物理 2 班的同学今后一切顺利，事业有成。