A*算法解决8数码问题

小组成员信息：

何奕骁 2020212259　杜嘉骏 2020212257　官子鸣 2020212258

小组成员分工如下：

何奕骁：负责实现图形化界面，撰写实验报告，安排实验进度与各组员的分工。

杜嘉骏：负责路径搜索算法的实现和部分实验报告的撰写。

官子鸣：负责资料的收集，模型的测试和部分实验报告的撰写

一、实验概括

本次实验，我们采用A*算法编程解决了八数码问题。在完成基本实验要求的基础上，我们还借助python的tkinter库实现了图形化界面，为用户提供了随机生成初始化状态，手动设置初始化状态，查看每一步的状态等功能，并有完善的提示与警告弹窗。最后，我们也尝试使用了广度优先算法与遗传算法解决了八数码问题，往图形化界面中加入了切换算法的功能，并与A*算法进行了比较。在实验过程中，我们遇到了很多问题与挫折，但是在我们的不断交流，沟通与坚持下最终解决了问题，取得了令我们比较满意的结果。

二、算法设计和方法说明

在本次实验中我们使用了三种算法解决八数码问题：广度优先搜索算法、A*算法和遗传算法。以下是我们针对三种算法的设计思路和方法说明。

1. 广度优先搜索算法

广度优先算法是最便捷的搜索算法，它是许多搜索算法的原型，比如Dijkstra算法、Prim最小生成树算法均采用了广度优先的思想。

广度优先搜索的主要思想是从起始状态开始，把每一步可以到达的下一个状态都枚举出来。在八数码问题中，就是从起始状态开始把空白方块上、下、左、右移动，把未枚举过的状态考虑进来，获得新的状态，在下一步的移动中，将获得的所有新状态进行所有的移动枚举即可。由于每一步可以到达的状态全部被枚举，所以保证了搜索到的路径必定是最短的。下面是其伪代码：

广度优先搜索算法 PowerShell 复制代码

```
used用来记录已经考虑过的所有状态的set集合
target是记录目标状态
start记录起始状态
left是剩余的还未考虑的状态
while left中还有元素:
    left2用于记录迭代后的新的left
    for state in left:
        将state中的空白方块进行上、下、左、右移动:
            如果获得了新的状态newState，则放入到left2和used中
            如果newState==target，则返回True，找到了路径
    left=left2
return False表示没有找到路径
```

我们的广度优先算法的代码在BreadthAlgorithm.py中，其中我们使用了字符串来记录每个状态，这样方便used进行哈希存储。代码中包含的函数及其功能如下：

judgeISValid（start：list,end：list）->bool：

用来判断输入的start起始状态到end终止状态理论上是否有解的函数，返回值是bool类型，如果是True，则表示有解。

BreadthAlgorithm（start：list，target：list）->list：

这是执行广度优先算法的函数，获得从start到target相应的最短路径，返回值是list，表示路径中的每个状态。

2. A*算法

尽管广度优先算法在路径搜素时已经表现出了很好的性能，它可以保证搜索到的路径时最短路径，但是由于其搜索方向具有盲目性，所以其搜索效率有所折中，而A*算法在广度优先搜索算法的基础上引入了启发函数，优先搜索到目标状态代价最小的状态，使得搜索具有智能性。

A*算法对广度优先算法的改进主要体现在计算两个状态之间的代价的启发函数（或者代价函数）。八数码问题中可选的启发函数很多，我们在实验中选择了三种进行了实现，分别是：

（1）曼哈顿距离：分别获得两个状态A、B的相同数字的坐标，分别为 $(X_A，Y_A)$ 和 $(X_B，Y_B)$，则这个数字在两个状态的曼哈顿距离为 $|X_A-X_B|+|Y_A-Y_B|$，状态的总体曼哈顿距离就是所有的数字的距离之和，$\sum_{k=1}^{8}|X_A(k)-X_B(k)|+|Y_A(k)-Y_B(k)|$。

（2）欧式距离：分别获得两个状态A、B的相同数字的坐标，分别为 $(X_A，Y_A)$ 和 $(X_B，Y_B)$，则这个数字在两个状态的欧式距离为 $(X_A-X_B)^2+(Y_A-Y_B)^2$，状态的总体欧式距离就是所有的数字的距离之和，$\sum_{k=1}^{8}(X_A(k)-X_B(k))^2+(Y_A(k)-Y_B(k))^2$。

（3）切比雪夫距离：分别获得两个状态A、B的相同数字的坐标，分别为 $(X_A，Y_A)$ 和 $(X_B，Y_B)$，则这个数字在两个状态的切比雪夫距离为 $max(|X_A-X_B|,|Y_A-Y_B|)$，状态的总体切比雪夫距离就是所有的数字的距离之和，$\sum_{k=1}^{8}max(|X_A(k)-X_B(k)|+|Y_A(k)-Y_B|(k))$。

在引入启发函数的基础上，我们就可以改进广度优先搜索算法，优先搜索到目标状态启发函数值较小的状态，这样就可以使得搜索具有方向感，使得搜索的效率提升。虽然搜索的效率有所提升，但是由

于搜索的方向有所倾向，并且二维坐标的距离并不能完全代表八数码问题的代价，所以搜索的路径不一定是最短的。

以下是A*算法的伪代码：

A*算法的伪代码 PowerShell 复制代码

```
used用来记录已经考虑过的所有状态的dict字典
target是记录目标状态
start记录起始状态
left是一个最小优先队列，存储剩余的还未考虑的状态
while left队列中还有元素：
    令state=（left的队首元素），是代价函数最小的状态：
        将state中的空白方块进行上、下、左、右移动：
            如果获得了新的状态newState：
                如果newState在used中：
                    如果newState到target的代价比上一次遇到的时候代价变小了：
                        则重新考虑该状态，将newState和代价函数值作为键值放到used
中
                        将newState放到left队尾
                如果newState不在used中：
                    考虑该状态，将newState和代价函数值作为键值放到used中
                    将newState放到left队尾
                    如果newState==target，则返回True，找到了路径
return False表示没有找到路径
```

A*算法的具体代码实现放在了SearchPath.py中，我们使用了queue库中的PriorityQueue数据结构作为最小优先队列，其中存放代价函数值和状态组成的元组，这样就可以按照代价函数值最小进行排序；和广度优先算法的实现一样，我们在A*算法中也是用字符串来存储状态。下面是代码中的函数及其功能：

函数说明：

judgeISValid（start：list，target：list）->bool：

用来判断输入的start起始状态到end终止状态理论上是否有解的函数，返回值是bool类型，如果True，则表示有解。

costMANHATUN（start：str，target：str）->float：

计算两个状态之间的曼哈顿距离。

costEUCLIDEAN（start：str，target：str）->float：

计算两个状态之间的欧氏距离。

costCHEBYSHEV（start：str，target：str）->float：

计算两个状态之间的切比雪夫距离。

moveTOnextState（nowState：str，nowZeroP：int，zeroOFFSET：int）–>str：

参数的意义是：此时空白方块的下标是nowZeroP，此次移动后的位置是nowZeroP+zeroOFFSET，函数的作用就是获得移动后的状态的字符串，在具体实现中我们使用的是字符串拼接的方法。

judgePlaceValid（x:int，y:int）–>bool：

判断x和y坐标在3*3的网格中，坐标是否合法，如果合法，则返回True。

AStarAlgorithm（state：list，end：list，costFUNCTION：int=1）–>list：

执行A*算法的函数，state是起始状态，end是终止状态，costFUNCTION用于指定使用的距离类型（如果是1，则使用曼哈顿距离；如果是2，则使用欧氏距离；如果是3，则使用切比雪夫距离），默认使用的距离是曼哈顿距离。

3. 遗传算法

解决路径搜索问题的算法还有很多，比如D*、LPA*、D* lite等算法，但是这些算法都是在环境改变的场景中才会体现算法的效果，在八数码问题中没有很好的体现，所以我们选择了另一种经典的启发式搜索算法——遗传算法应用到八数码问题的求解中。

遗传算法的主要思想就是模拟大自然的优胜劣汰、遗传变异等进化方式，以此不断筛选出符合要求的结果。在八数码问题中，我们的设计思路是：

（1）将对空白方块的上、下、左、右移动操作分别使用整数0、1、2、3来表示，那么从起始状态经过一系列操作得到终止状态的操作就可以表示成一个整数序列。

（2）我们将上述的八数码问题的整数序列定义为遗传算法中的基因，然后模拟基因的遗传、变异等逐渐筛选出能够达到目标状态的操作序列。

（3）在大自然的遗传中，遵循优胜劣汰的原则，即如果基因符合要求，则尽可能多的遗传下去，否则，被淘汰掉。因此我们就需要定义一个评估基因对要求的适应度的函数（适应度有一个最大值，如果达到最大值，就说明找到了目标基因），然后使用轮盘选择的方式，按照概率大小选择遗传的对象。下面是我们定义的评估基因对要求的适应度的函数伪代码，其中我们使用字符串表示八数码：

评估基因环境适应度的函数伪代码 Python 复制代码

```python
def EnvironmentalFitness(nowState:str,targetState:str)->int:
    """
    用于求两个状态之间的重合度的函数
    """
    Fitness=0
    for nowStateATp,targetStateATp in zip(nowState,targetState):
        if nowStateATp==targetATp:
            Fitness+=100-10*int(nowStateATp)
            return Fitness

def SumEnvironmentalFitness(gene,target)->int:
    """
    用于求一个基因的环境适应度的函数
    """
    endState=按照gene的操作步骤获得的最终状态
    thisGenesEnvironmentalFitness=EnvironmentalFitness(endState,target)
    return thisGenesEnvironmentalFitness
```

（4）在基因的变异中，我们主要选择了两种变异方式：交叉变异和替换。其中交叉变异的主要方式就是按照交叉变异的概率选择两个基因，然后交换随机数量的相同位置的基因粒；替换变异的主要方式随机选择基因，随机选择要改变的基因粒位置，并且将其随机改变成0、1、2、3中的操作。

以上就是我们针对遗传算法的设计方案，在具体实现中，我们设计了一个种群类，其中存放许多基因，代表一个种群，可以对遗传、变异进行整体操作，以下是我们的代码设计和功能：

Python 复制代码

```python
class Creatures:
    def __init__(self,start,target,Genelength,GeneSize,loop)->None
    def inherit(self,FitSum)->None
    def mutations(self)->None
    def crossMutaion(self)->None
    def SumEnvironmentalFitness(self)->int
    def searchPath(self)->list
```

def __init__(self,start,target,Genelength,GeneSize,loop)->None：

功能：初始化对象中的变量。初始化start为起始状态，target为目标状态，Genelength为每一个基因的长度，即基因所包含的操作个数，GeneSize为基因的个数，即种群的大小，loop是种群进化的迭代次数。

def inherit(self,FitSum)->None：

功能：模拟种群的遗传。针对适应度使用轮盘赌随机选择遗传的基因，如果适应度高，则以高概率遗传。代码中我们使用了基因适应度的前缀和，定义为： $Presum[k] = \sum_{i=0}^{k} GeneFitness[i]$ ，这样就可以使用二分查找算法，伪代码如下：

```
Python | 复制代码

Presum是基因的前缀和
genes是上一代的基因
LastPopulation是对genes的深拷贝
for genePlace in range(len(genes)):
    RandomINT=random.randint(0,Presum[-1])# 模拟在轮盘中随机选择一个整数
    Place=从Presum中找到第一个大于等于RandomINT的前缀和的位置(可以采用二分查找算法)
    将genes[genePlace]的基因改变成LastPopulation[Place]
```

def mutations(self)->None：

功能：模拟基因中的替换变异。随机选择要变异的基因，随机选择要变异的基因粒，将其随机变成0、1、2、3中的任意一种操作即可。

def crossMutaion(self)->None：

功能：模拟基因中的交叉变异。随机选择要交叉变异的两个基因，随机选择要交叉交换的基因粒，然后交换即可。

def SumEnvironmentalFitness(self)->int：

功能：用于求一个基因的环境适应度的函数。上面已经介绍过。

def searchPath(self)->list：

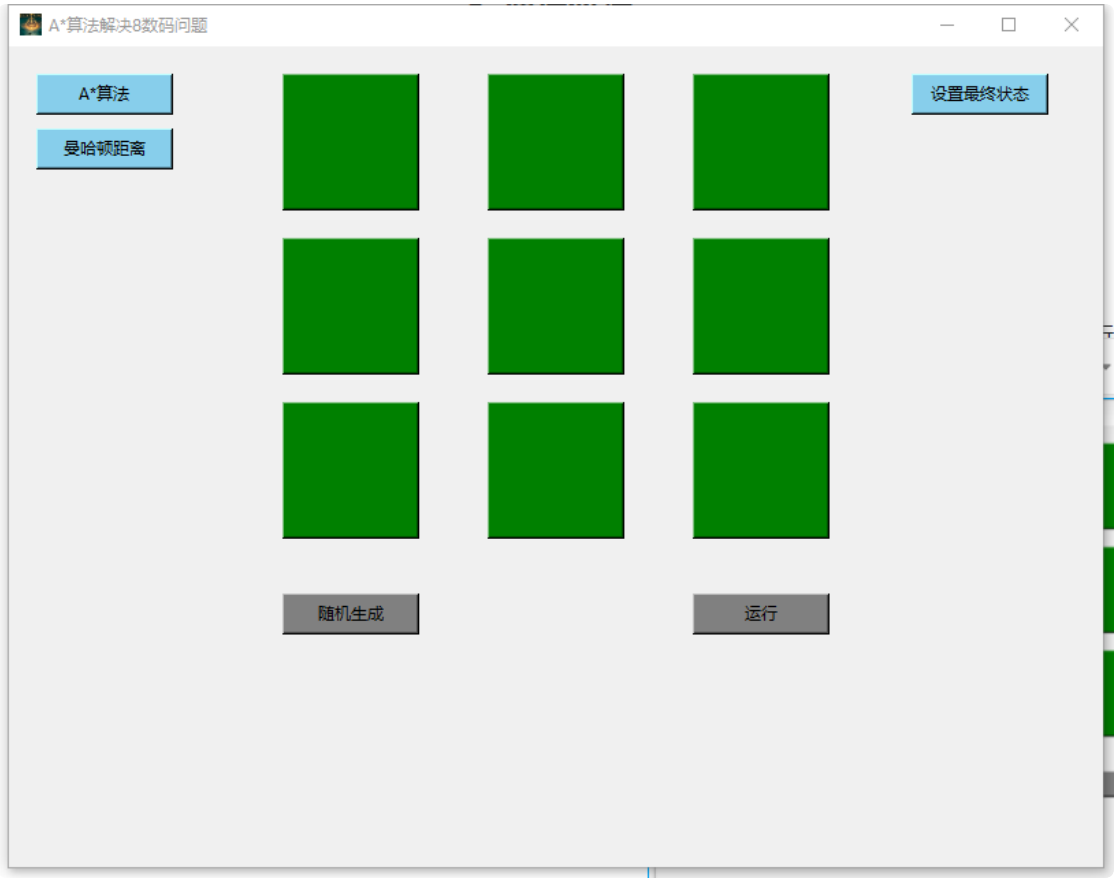
功能：控制种群遗传、变异过程的主函数，如果找到了目标基因，则返回目标基因。其伪代码如下：

```python
for EvolutionLoop in range(Loop):
    计算每一个基因的适应度
    如果适应度中有最大值:
        则返回目标基因
    如果没有:
        inherit()进行遗传
        crossMutaion()进行交叉变异
        mutaions()进行替换变异
如果没有找到目标基因，则返回空列表代表未找到
```

三、图形化界面说明

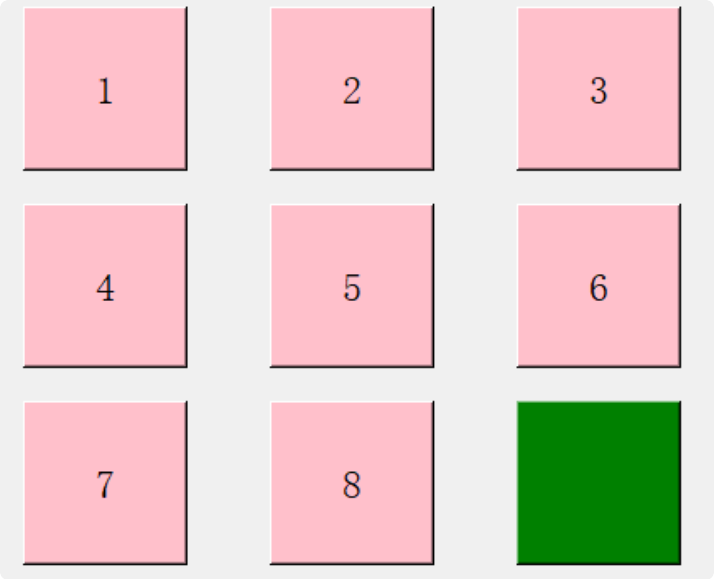
1. 图形化界面简介与操作说明

本次实验的图形化界面是使用python自带的tkinter库实现的，运行ShowPage.py，即可展示出图形化界面，如下所示：

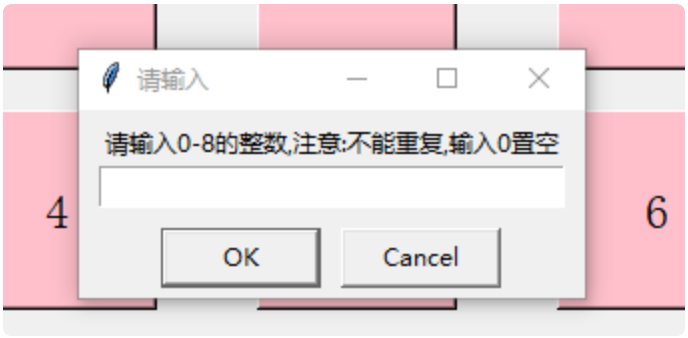


其中，左上角的蓝色按钮是切换算法的按钮，点击该按钮可以改变使用的算法。切换顺序：A*算法，广度优先算法，遗传算法；它下面的按钮为切换A*算法启发函数的按钮，点击该按钮可以改变A*算法使用的启发函数。切换顺序：曼哈顿距离，欧氏距离，切比雪夫距离。

中间三行三列的方块即代表九个方格，我们要实现的目标是移动空白方块(绿色)，使得方格达到最终状态，默认最终状态如下：



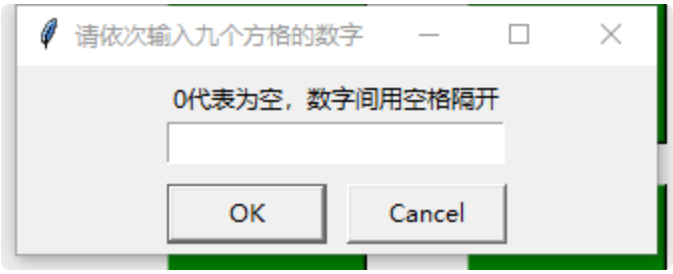
我们可以指定每个方块的初始化状态，点击方块，如下：



如果输入的非0–8的整数，则会报错提醒；如果输入了重复的数字，则会忽略这次输入。

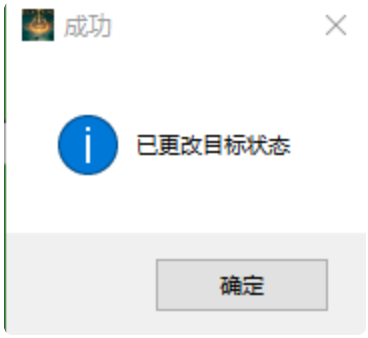
也可以不用自己输入每个方块的初始化状态，点击左下方的灰色随机生成按钮，则会随机的初始化九个方块的状态。

右上角的蓝色按钮是切换最终状态(目标状态)的按钮，点击该按钮，如下所示：

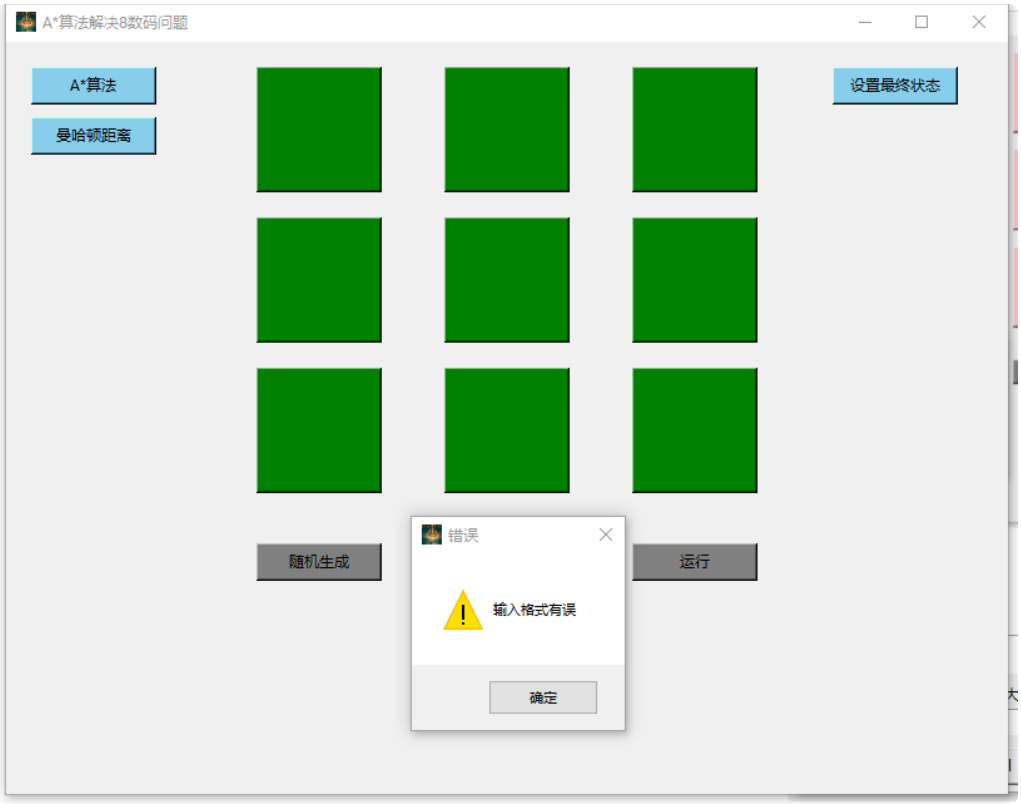


我们可以按照从上往下，从左往右的顺序依次输入九个方格中的数字，输入0表示为空，数字间用空格隔开，若输入的格式不规范，则会有相应的错误提示：

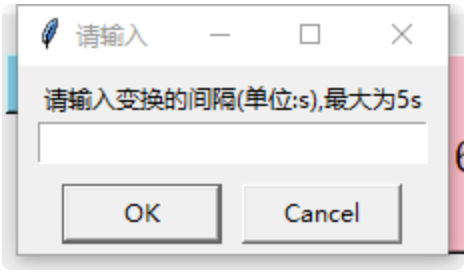
若输入正确无误，则会更改最终状态，并提示修改最终状态成功：



我们可以点击右下方的运行按钮运行代码，但是若初始化的状态不符合要求，就会有报错提示，如下图：

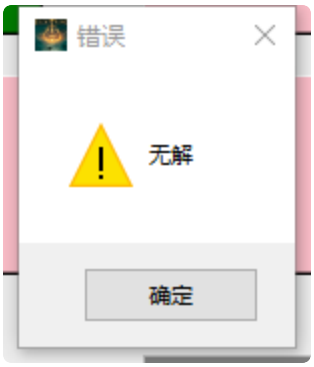


若初始化格式正确，点击运行按钮，则会弹出如下窗口：

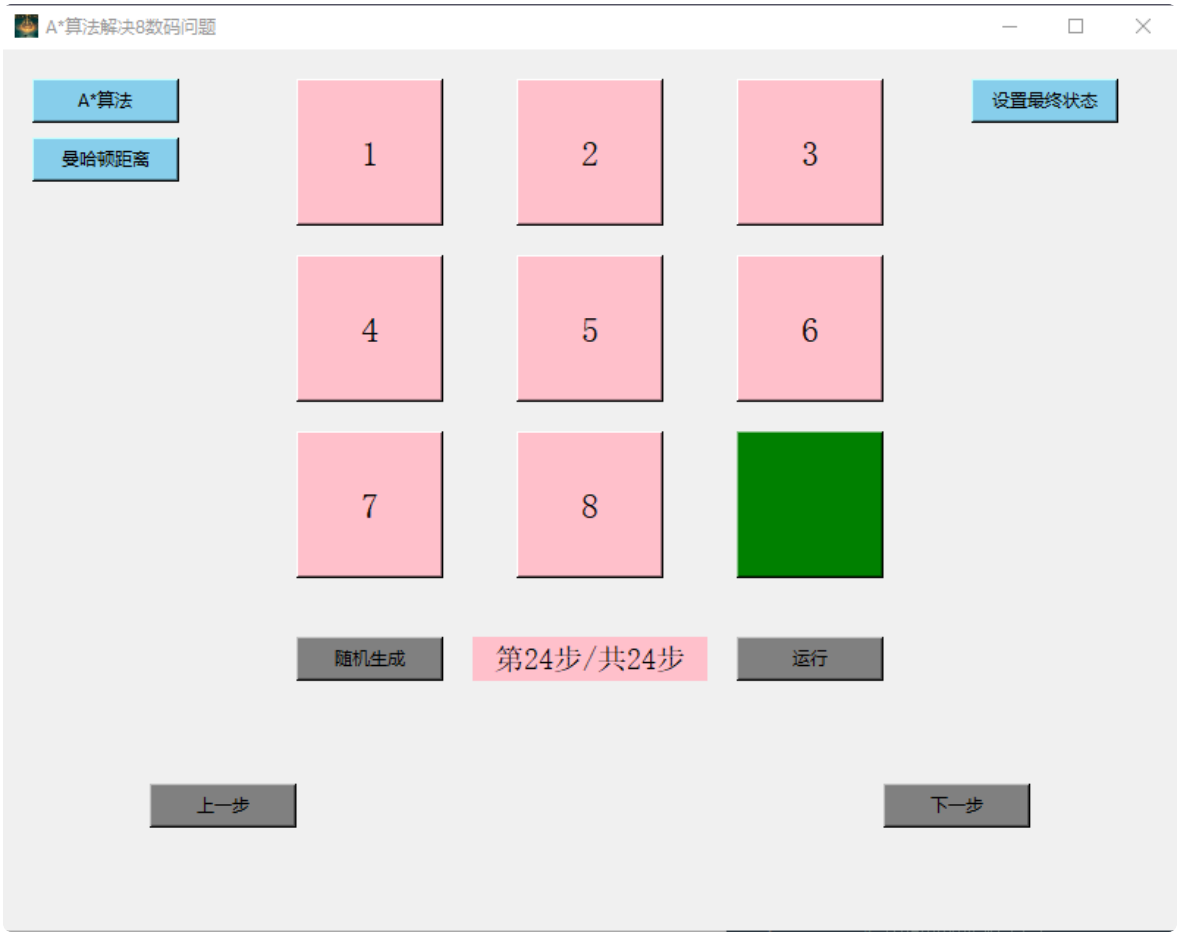


这里的变换间隔指的是每个状态所停留的秒数，若有解，输入该秒数后九个方格则会从初始化状态一步一步变化到最终状态，其中在每一步停留的时间就是输入的变换间隔。

若无解，则会报错：



若有解，九个方格则会一步一步的变换状态，最终，会弹出成功提醒框，点击确定后，如下图所示：



可以看到，页面的左右下角多了上一步与下一步按钮，点击相应按钮，就可以跳转到上一步或下一步的状态。但是，若更改了九个方格，则上一步与下一步两个按钮与正下方的步数统计信息将会隐藏。

2. 界面实现说明

图形化页面类概述

图形化页面是通过show类实现的，show类如下所示：

show类 Python 复制代码

```python
class show:
    # 页面的初始化算法
    def __init__(self) -> None
    # 切换算法的方法
    def changeAlgorithm(self)
    # 切换启发函数的方法
    def changeCostFunction(self)
    # 设置目标状态的方法
    def setTarget(self)
    # 为方格手动选择数字的方法
    def changeText(self, event)
    # 随机初始化
    def randomCreat(self)
    # 运行算法的方法
    def run(self)
    # 更新能否进行下一步，上一步以及步数
    def updateCangoback(self)
    # 返回上一步的方法
    def goback(self)
    # 去下一步的方法
    def gonext(self)
```

关键方法说明

__init__(self) -> None：

这是图形化页面的初始化方法，初始化各个按钮以及九个方格的状态，并绑定相应的事件；

changeAlgorithm(self)：

这是切换算法的方法，该方法与左上角切换算法的按钮绑定，点击该按钮调用这个方法，修改类的成员变量algorithmType的数值(0,1,2时分别为A*算法，广度优先算法与遗传算法)与按钮的文字与颜色；

changeCostFunction(self)：

这是切换A*算法启发函数的方法，该方法与左上角切换启发函数的按钮绑定，点击该按钮调用这个方法，修改类的成员变量costFunctionType的数值(1,2,3时分别为曼哈顿距离，欧氏距离与切比雪夫距离)与按钮的文字与颜色；

setTarget(self)：

这是设置目标状态的方法，与界面右上角设置最终状态的按钮绑定，若输入非法，弹窗警告；否则更改成员变量target(存有目标状态信息)并弹窗提醒更改目标状态成功。

changeText(self, event)：

这是手动修改九个方格数值时调用的方法，若输入合法，则更改类的成员变量textList(储存有九个方格对应数字的信息)中相应的信息，进而改变九方格的数值；

randomCreat(self)：

这是随机生成按钮所绑定的方法，点击该按钮，随机生成九个方格的数值信息，并更新到textList中，进而更新到页面上；

run(self)：

这是运行算法的方法，伪代码如下所示：

run方法 PowerShell 复制代码

```
(1)if 输入格式有误
      弹出错误弹窗并return
(2)弹出输入变换间隔弹窗。若输入非法，则弹窗提醒并return
(3)if algorithmType == 0
      showTextList = A*算法提供的接口方法(传入textList、target与
costFunctionType)
   else if algorithmType == 1
      showTextList = 广度优先遍历算法提供的接口方法(传入textList与target)
   else
       showTextList = 遗传算法提供的接口方法(传入textList与target)
   /*showTextList为二维列表，列表的每一行都存有每一步方格的状态*/
(4)if showTextList为空列表
      弹窗提醒该初始化状态无解并return
(5)根据变换间隔，动态展示九方格的变化过程
(6)显示上一步、下一步按钮以及步数统计信息
```

updateCangoback(self)：

这是更新页面能否进行上一步下一步以及显示步数统计信息的方法，若九方格被修改，则调用该方法，隐藏上述按键与信息；

goback(self)与gonext(next)：

这是与上一步、下一步两按钮分别绑定的方法，点击按钮，调用对应方法，通过读取showTextList中的信息，更新textList并最终更新界面上所展示的九方格。

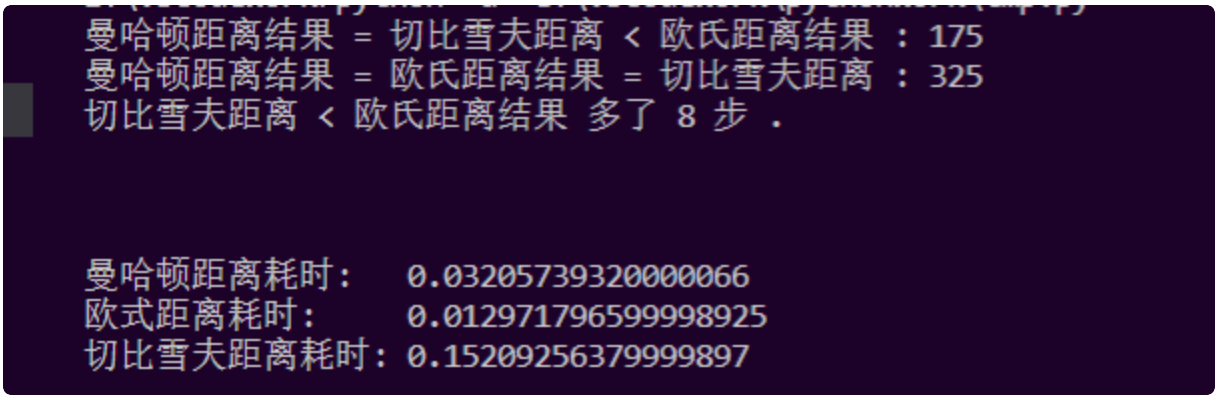
四、实验结果分析

本次实验中我们学到了启发式搜索的思想，同时实现了三种算法来解决八数码问题，但是，为了更加深入理解启发式搜索的优点和不同启发方式的不同效果，我们进行了相关的性能测试（相关代码在Efficiency.py文件中，测试结果存放在result.txt中），得到了如下分析结果。

1、A*算法不同启发函数的搜索效率和搜索路径的长度比较：

（1）测试方法和结果：

我们随机生成500个有解的八数码问题，然后分别使用曼哈顿距离、欧氏距离和切比雪夫距离解决这些问题，获得不同启发函数搜索的路径长度，同时使用timeit库来计算总体需要的时间，求出解决一个问题所需的平均时间。测试结果如下：



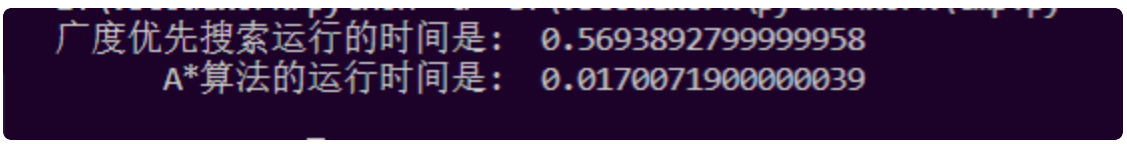
（2）结果分析：

可以看到，使用曼哈顿距离和切比雪夫距离搜索得到的路径长度总是最短的，所以使用曼哈顿距离和切比雪夫距离的启发函数效果最好；虽然欧氏距离得到的长度可能不是最短的，但是欧式距离搜索的平均时间最短，效率最高；并且欧式距离搜索的路径长度比最短的路径长度只多了8步，并不是很影响搜索的结果。所以综上，我们认为使用欧氏距离的启发函数效果最好。

2、比较A*算法和广度优先算法的搜索效率：

（1）测试方法和结果：

我们随机生成10个测试样例，用欧氏距离启发A*算法和广度优先算法解决，使用timeit计时，获得平均解决时间，如下：



（2）结果的分析：

可以看到，A*算法的搜索速度是广度优先算法的50倍以上，可以看到A*算法引入启发函数使得搜索更加智能，大大提高了搜索的效率。

3、A*算法、遗传算法启发方式的比较：

在实验中，我们发现遗传算法的搜索效率非常低，耗时非常长，根本原因是遗传算法是一种通用的算法框架，并没有针对具体的问题进行算法优化，并且遗传算法模拟的是大自然中的遗传变异，虽然遵循“适者生存”的启发原则，但是在搜索过程中仍具有很大程度的随机性，并没有明显的启发方向，十分盲目；而相比于A*算法，其启发函数给出了一个十分强有力的启发方向，使得搜索更加智能。

但是遗传算法进行全局搜索的方式可以为我们提供更多的可行解和启发，加深我们对原问题的认知，更适合于求解一些NP难问题或者更复杂的问题。

五、实验总结与思考

通过这次实验，我们实现了使用A*算法解决八数码问题，并且将A*算法使用不同启发函数的效果进行了对比分析，而后我们还创新性的添加了其他算法来解决八数码问题。我们在实现这些算法的过程中加深了我们对于机器智能课上所学的理解，通过实现算法与不同算法的对于与分析，我们对这些算法有了更深层次的理解，提升了自己通过编程解决实际生活问题的能力。

另外，在完成可视化页面的过程中，通过将不同的功能进行划分与实现，我们提升了前端与客户端开发的能力，提升了对项目模块分离与划分的能力。在小组分工，实现不同功能的过程中我们也提升了协作开发的能力与交流沟通的能力。

总之，通过这次实验，我们学到很多知识，有很大的进步。