人工智能基原理项目 1

火柴棍移动游戏

自 16 刘赫 2021010417

1 任务描述

用火柴棍可以摆成一个数字等式，希望移动一根火柴使得等式成立。

1. (必做)允许在一个固定的等式库（两位数以内的加减乘法）中选择，从而给出答案。
2. (必做)允许使用者自己定义，或者输入一个可以求解的等式。如果无解，回答无解。
3. (必做)允许移动 2 根火柴棍。
4. (选做)给出从等式变为新的等式的题目和难度。

2 问题建模

2.1 模型概述

任何一个两位数加减乘法即可表示为一个状态，起始状态为用户输入的等式或者系统自动生成的等式，目标状态为成立且与原等式不相同的等式且步数满足要求。状态转换函数为改变其中任意一个数字或符号，及其带来的火柴棍总移动次数变化及多出或缺少的火柴棍。

从起始状态到目标状态，找到一条长度等于用户给定步数的路径。然后利用广度遍历搜索算法进行搜索即可，因此我们需要定义两个状态间的距离，由于每次操作只改变其中一个数字或符号，因此只需要定义数字与数字、符号与符号之间的距离即可。

2.2 数字的表示

我们以数码管的形式来表示 0-9 这 10 个数字，每个数字最多由 7 根火柴构成，我们将这 7 个位置分别编上 1-7 号。因此对于 0-9 的每个数字可以由一个 7 位的 0/1 序列来表示：第 i 位为 0 代表该位置上没有火柴；第 i 位为 1 代表该位置上有火柴。

2.3 数字和符号之间的距离

我们需要用到两个参数，不妨将距离记作 (dis1, dis2).

第一个参数 dis1 是从原数字到目标数字的剩余火柴数。反之，如果火柴数目不够的话，这一参数的取值为负。第二个参数 dis2 是去掉多余（或补足缺少）的火柴之后还需要移动的最少步数。具体计算方法是目标数字存在的为 1 但原数字为 0 的位置的数目和目标数字存在的为 0 但原数字为 1 的位置的数目

所以从数字 3 到数字 2 的距离可表示为 (0, 1). 同理，我们可以定义两个符号之间的距离。

```
def opdis(op1, op2):
    opdismatrix = [[[0, 0], [1, 0], [0, 2]],
                   [[-1, 0], [0, 0], [-1, 1]],
                   [[0, 2], [1, 1], [0, 0]]]
    return opdismatrix[oper.index(op1)][oper.index(op2)]
2 用法
def dis(num1, num2):
    dis1 = count1 = count2 = 0
    for i in range(7):
        if numstr[num1][i] == '1' and numstr[num2][i] == '0':
            dis1 += 1; count1 += 1
        elif numstr[num1][i] == '0' and numstr[num2][i] == '1':
            dis1 -= 1; count2 += 1
    return [dis1, min(count1, count2)]
```

3 算法设计和实现

3.1 答案搜索算法

用 6 个数字、1 个符号加上未决定的火柴数、已移动的火柴数共 9 个参数来表示状态，初始状态为用户输入或随机生成的 6 个数字和符号，未决定的火柴数、已移动的火柴数初始均为 0。未决定的火柴数，可以看做是改变完某个数字时多出来（或者缺少的）火柴，已移动火柴数代表从初始状态到现在状态总共移动了多少根。每当状态改变时，未决定的火柴数和已移动的火柴数都要根据数字之间的距离进行更新。

目标状态应满足的条件为：未决定的火柴数等于 0 且以移动的火柴数等于设定根数（用 step 指代）。在搜索上可以采用宽度优先（BFS）的算法，所有状态可以看成是一个深度为 7 的树，第一层为初始状态、先依次遍历各个数字再遍历符号。

剪枝：当我们探索到某个节点时若发现其已移动的火柴数目＞ step 时，可以对其进行剪枝，这使得可以较快解决 1 根、两根的移动问题。先判定是否剪枝再遍历的部分代码如下：

```
while len(list1) != 0:
    if abs(list1[0][7]) + list1[0][8] > step:
        list1.pop(0); continue
    for j in range(10):
        temp = list1[0].copy()
        temp[i] = j
        distant = numdismatrix[10 * list1[0][i] + j]
        temp[8] += min(abs(temp[7]), abs(distant[0])) + distant[1]
        temp[7] += distant[0]
        list2.append(temp)
        if list2[-1][7] == 0 and list2[-1][8] == step:
            if judge(list2[-1]):
                result.append(list2[-1])
    list1.pop(0)
```

3.2 出题算法

生成题目分为两种，首先是不成立等式，先通过随机数生成一个等式作为初始状态，状态转换函数为改变某个数字或符号，目标状态同样应满足：未决定的火柴数等于 0 且以移动的火柴数等于 step，然后根据 3.2 所述的流程利用广度优先探索目标状态，由于目标状态可能不止一个，当探索到满足目标状态条件时应当及时停止。

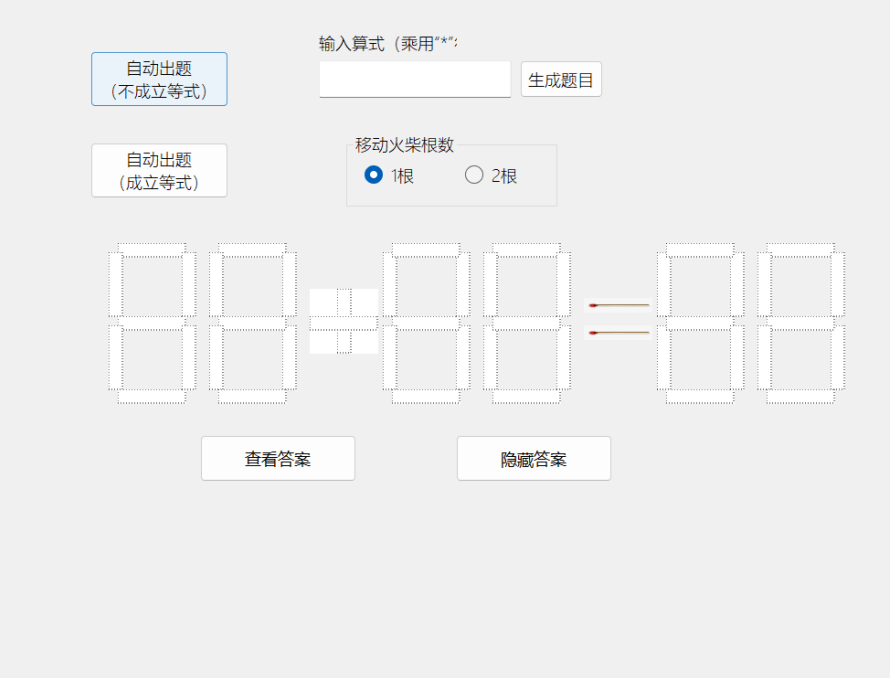
然后是成立等式，并判断进一步变成另一个等式的难度。通过随机数生成一个等式即可，又求解所需步数越多，求解方式越少，该题越难。故经测试得该等式求解的难度定义为： diff = step_min - 0.1* res_num 其中 step_min 表示解决该等式所需的最小步数，res_num 表示最小步数下，该等式求解方式的个数。具体难度计算代码如下：

```
if mode == 2:
    for i in range(1,2):
        res = solve.SolveProblem(i,self.problem)
        if len(res)!=0: break
    self.radioButtonList[i-1].setChecked(True)
    QMessageBox.about(self, "提示", "已成功随机生成成立等式\n该题目至少移动"+repr(i)+
                      "根火柴\n系统评估本题难度为"+repr(i-0.1*len(res)));
```

4 界面设计及操作指南

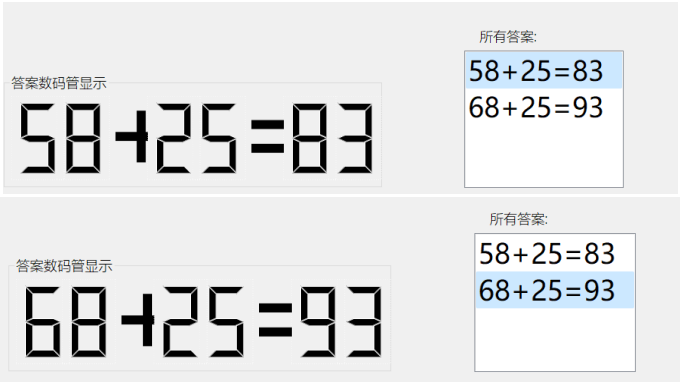
4.1 输入表达式求解

界面如图，用户首先设置移动火柴的数目，然后将表达式输入到方框内。

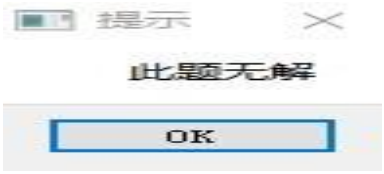


在生成题目之后，系统会自动以火柴棍的方式显示在下方区域，用户可以点击“查看答案”按键进行求解，求解完成后会弹出提示框，告诉用户所有答案个数以及搜索用时。关闭提示框后，在界面右下角“所有答案”的列表中，玩家可以查看该题在指定移动根数下的全部答案。

右侧列表为该题的全部答案，选中其中某一项可以在左侧查看更加直观的数码管显示



如果用户输入的题目无解，系统也会给用户提示：

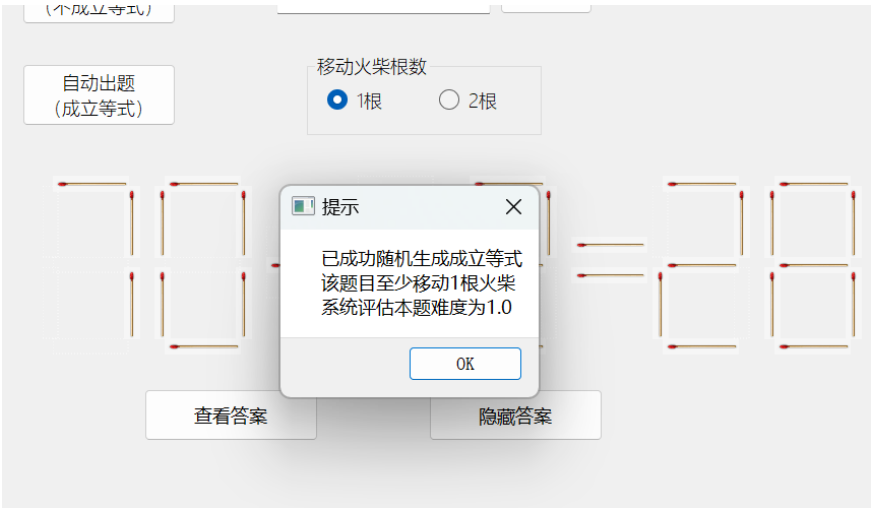


4.2 系统自动出题

用户首先选择移动的火柴数目，然后点击“系统自动出题（不成立等式）”按钮，即可在下方以火柴棍的形式展示出一道系统出的新题目，其余与 4.1 相同。

4.3 从等式变成新的等式

用户可点击“系统自动出题（成立等式）”按钮随机生成成立等式，系统会告诉用户完成该题所需的最少移动次数与本题预估难度，同样用户可点击“查看答案”对该题进行求解。



5 实验总结

此次作业是第一次接触 Python 和 Pyqt5 以及扩展工具 QT designer，学习的过程中着实遇到了一些困难。但总体学下来也是收获颇多。这次项目也复习了课上学的搜索方法以及剪枝等相关知识，也学习了如何对于具体问题通过状态搜索的知识进行建模并系统地解决。总之，这次项目让我再知识和技术水平上都收获很多，也让我有动力去继续尝试新的项目。