人工智能基础大作业一：火柴棍移动

自75 常成 2017010252

目录

1 任务描述 **3**

2 问题建模 **3**

2.1 状态 3

2.2 状态之间的转换操作 3

2.2.1 移动一根火柴 3

2.2.2 移动两根火柴 5

2.2.3 等式变新等式 7

2.3 代价函数 7

3 算法设计和实现 **7**

3.1 移动一根火柴 7

3.2 移动两根火柴 9

3.3 扩展到更多的长等式 11

4 UI设计和操作 **12**

4.1 软件程序的开发 12

4.2 模式选择 12

4.2.1 题库模式 13

4.2.2 自定义模式 14

4.3 难度及结果数的显示 16

4.4 等式变新等式的提示 17
4.5 题库的扩充及文件操作 17
4.6 UI界面的容错机制 18
4.7 帮助界面 19
4.8 长等式运算 19

5 项目总结 **19**

1 任务描述

火柴数字游戏是一种益智类游戏，可以有效地锻炼玩家的脑力和思维，趣味无穷。本次大作业项目要求的任务如下：

- 允许在固定的等式库(两位数以内的加减乘法)中选择，从而给出答案。
- 允许使用者自己定义，或者输入可以求解的等式。如果无解，回答无解。
- 给出更多的题目和答案。
- 允许移动 2 根火柴棍。
- 给出从等式变为新的等式的题目和难度。

2 问题建模

将移动火柴的过程视为搜索过程，定义如下的特征：

2.1 状态

将火柴等式的当前状态定义为:

$$\boldsymbol{X} = (A, \alpha, B, \beta, C)$$

式中A,B,C代表最多两位数字，α,β 代表运算符号(+,-,×,=)，火柴棍的初始摆放情况即为初始状态，若AαBβC等式成立，则为问题的一个目标状态。

2.2 状态之间的转换操作

2.2.1 移动一根火柴

首先将火柴棍等式中每个数字拆开为独立个位数的形式，定义为$single\boldsymbol{X}$。

例如初始等式状态$\boldsymbol{X} = (16, +, 32, =, 56)$，则$single\boldsymbol{X} = (1, 6, +, 3, 2, =, 5, 6)$。

将状态改变存储为数组形式，对$single\boldsymbol{X}$中的每一位，定义转换操作如下：

$$\boldsymbol{Y} = (\text{初始字符}, \text{添加/减少火柴根数}, \text{结果字符})$$

例如(0,0,6),(0,0,9)表示0不需要添加火柴，经过自身移动一根火柴即可变为数字6或9，而(0,1,8)则表示数字0需要添加一根火柴，才可变为数字8，这样对于0-9以及四个符号，所有可能的转换操作如下表：

0	(0,0,6)	(0,0,9)	(0,1,8)		
1	(1,1,7)				
2	(2,0,3)				
3	(3,0,2)	(3,0,5)	(3,1,9)		
4					
5	(5,0,3)	(5,1,6)	(5,1,9)		
6	(6,0,0)	(6,0,9)	(6,1,8)	(6,-1,5)	
7	(7,-1,-1)				
8	(8,-1,0)	(8,-1,6)	(8,-1,9)		
9	(9,0,0)	(9,0,6)	(9,1,8)	(9,-1,3)	(9,-1,5)
+	(+,0,=)	(+,-1,-)			
-	(-,1,+)	(-,1,=)			
=	(=,0,+)	(=,-1,-)			
×					

这样对于解决移动两根火柴棍的问题，$\boldsymbol{X} = (A, \alpha, B, \beta, C)$存在两种可行的解决方式：

$$(u, 0, v) \qquad u \in single\{A, \alpha, B, \beta, C\}$$

$$((u, -1, v), (m, 1, n)) \quad u, m \in single\{A, \alpha, B, \beta, C\} \quad u! = m$$

分别表示(1)某字符自身移动一根火柴(2)某字符减少一根火柴，另一字符增加一根火柴。

这里的u和m并非数值上一定不相等，仅是不位于等式中的相同位置。例

如等式6+6=6，A，B，C各位数值相等，当u=A且m=B时，u与m虽然数值相等，但是合法的操作，满足了火柴棍从一个数字移动到另一个数字的条件。

两种方式分别形成的新等式中，所有真正成立的等式构成解集，解集中的解都是问题的目标状态。

2.2.2 移动两根火柴

同样地，将状态改变存储为数组形式，对状态$\boldsymbol{X}$中的每一位，定义转换操作如下：

$$\boldsymbol{Y} = (\text{初始字符}, \text{移动次数}, \text{添加/减少火柴根数}, \text{结果字符})$$

例如(0,1,0,6),(0,1,0,9)表示0不需要添加火柴，经过自身移动一根火柴即可变为数字6或9；而(0,0,1,8)则表示数字0需要添加一根火柴，并且不再需要移动，即可变为数字8；(0,1,-1,3),(0,1,-1,5)表示数字0需要减少一根火柴，并且再移动一次，才可得到数字3或5。以此类推，对于0-9以及四个符号，所有可能的转换操作如下表：

0	(0,1,0,6)	(0,1,0,9)	(0,0,1,8)	(0,1,-1,2)	(0,1,-1,3)	(0,1,-1,5)		
1	(1,0,1,7)	(1,0,2,4)						
2	(2,1,0,3)	(2,2,0,5)	(2,1,1,0)	(2,1,1,9)	(2,1,1,6)	(2,0,2,8)		
3	(3,1,0,2)	(3,1,0,5)	(3,0,1,9)	(3,1,1,0)	(3,1,1,6)	(3,0,2,8)	(3,1,-1,4)	(3,0,-2,7)
4	(4,1,1,3)	(4,1,1,5)	(4,0,2,9)	(4,0,-2,1)	(4,1,-1,7)			
5	(5,1,0,3)	(5,2,0,2)	(5,0,1,9)	(5,0,1,6)	(5,1,1,0)	(5,0,2,8)	(5,1,-1,4)	
6	(6,1,0,0)	(6,1,0,9)	(6,0,1,8)	(6,0,-1,5)	(6,1,-1,2)	(6,1,-1,3)		
7	(7,,0,-1,1)	(7,1,1,4)	(7,0,2,3)					
8	(8,0,-1,0)	(8,0,-2,2)	(8,0,-2,3)	(8,0,-2,5)	(8,0,-1,6)	(8,0,-1,9)		
9	(9,1,0,0)	(9,1,0,6)	(9,0,1,8)	(9,0,-1,3)	(9,0,-2,4)	(9,0,-1,5)	(9,1,-1,2)	
+	(+,1,0,=)	(+,2,0,×)	(+,0,1,-)					
-	(-,0,1,+)	(-,0,1,=)	(-,1,1,×)					
=	(=,1,0,+)	(=,2,0,×)	(=,0,-1,-)					
×	(×,2,0,=)	(×,2,0,+)	(×,1,-1,-)					

这样对于解决移动两根火柴棍的问题，$\boldsymbol{X} = (A, \alpha, B, \beta, C)$存在以下可行的

解决方式：

$$(u,2,0,v) \qquad u \in single\{A,\alpha,B,\beta,C\}$$

$$((u,0,-2,v),(m,0,2,n)) \quad u,m \in single\{A,\alpha,B,\beta,C\} \quad u! = m$$

$$((u,0,1,v),(m,1,-1,n)) \quad u,m \in single\{A,\alpha,B,\beta,C\} \quad u! = m$$

$$((u,0,-1,v),(m,1,1,n)) \quad u,m \in single\{A,\alpha,B,\beta,C\} \quad u! = m$$

$$((u,1,0,v),(m,1,0,n)) \quad u,m \in single\{A,\alpha,B,\beta,C\} \quad u! = m$$

$$((u,0,-1,v),(m,0,-1,n),(e,0,2,f) \quad u,m,e \in single\{A,\alpha,B,\beta,C\} \quad u! = m \quad u! = e \quad m! = e$$

$$((u,0,1,v),(m,0,1,n),(e,0,-2,f) \quad u,m,e \in single\{A,\alpha,B,\beta,C\} \quad u! = m \quad u! = e \quad m! = e$$

$$((u,0,-1,v),(m,0,1,n),(e,0,1,f),(g,0,-1,h) \quad u,m,e,g \in single\{A,\alpha,B,\beta,C\} \quad (互不同)$$

分别表示：(1)某字符自身移动两次火柴;(2)某一个字符加两根，另一个字符减两根;(3)某一个字符加一根，另一个字符减一根再自身移动一次;(4)某一个字符减一根，另一个字符加一根再自身移动一次;(5)两个字符分别自身移动一根火柴;(6)两个字符分别减少一根火柴，第三个字符增加两根火柴;(7)两个字符分别增加一根火柴，第三个字符减少两根火柴;(8)两个字符分别增加一根火柴，另两个字符分别减少一根火柴。

特别地，若允许增加数字，比如A=3，通过增加两根火柴变为A=13或A=31，则问题还有如下两种解决方案：

$$((u,0,-2,v),m \to n) \quad u \in single\{A,\alpha,B,\beta,C\} \quad m \in single\{A,B,C\} \quad u! = m$$

$$((u,0,-1,v),(e,0,-1,f),m \to n) \quad u,e \in single\{A,\alpha,B,\beta,C\} \quad m \in single\{A,B,C\} \quad (互不同)$$

$$其中 n = m + 10 \quad or \quad n = m * 10 + 1$$

例如等式12+8=7即可通过此种操作方式变为12+5=17，在这里我们认为是合法的操作。

以上通过任何一种方式形成的新等式，通过判断真正成立后，都将被纳入解集，解集中的每个解都是问题的一个目标状态。

2.2.3 等式变新等式

对于等式变新等式问题，转换操作可以看成是上述两种情况的并集，即认为通过移动一根火柴或者两根火柴形成的正确等式都在问题的解集里，也均是目标状态。

2.3 代价函数

对于三种问题模式，代价函数定义为找到所有目标状态(可行解)时，在转换操作中生成的新等式的总数目。

3 算法设计和实现

对于整个问题来讲，要求得到所有的可行解，因此必须遍历所有可能的情况，并且目标状态从最初就不是固定的，因此没有必要设定固定的启发函数。

3.1 移动一根火柴

对移动一根火柴的问题，采用广度遍历的方法，由简到繁，首先判断字符自身移动一根火柴是否可以得到成立的新等式，再判断两个字符之间移动的情况。将每个字符的转换操作存储为字典形式，便于检索。移动一根火柴的问题的伪代码如Algorithm 1表示：

代码中获取i全部位置的函数为*get_pos*，是利用列表切片获取的，每次定位

Algorithm 1 移动一根火柴

Input: A+B=C,state:$\boldsymbol{X}$
Output: all correct anwsers

```
for single X中的每个字符i do
    for i在字典中对应的每个(*,0,*)型的可行操作 do
        获取i在single X中的全部位置
        for 每个位置的i do
            改变i的值
            if 形成的新等式成立 then
                将等式加入解集
            end if
        end for
    end for
    for 该字符在字典中对应的每个(*,1,*)型的可行操作j do
        choice_1 ← choice_1 + j
    end for
    for 该字符在字典中对应的每个(*,-1,*)型的可行操作k do
        choice_2 ← choice_2 + k
    end for
end for
for choice_1中的每一步操作 do
    获取该操作待改变字符在single X的全部位置pos_1
    for choice_2中的每一步操作 do
        获取该操作待改变字符在single X的全部位置pos_2
        for pos_1中的每个位置u do
            for pos_2中的每个位置v do
                if u!=v then
                    将u和v位置上的字符改变为操作后的对应值
                end if
                if 形成的新等式成立 then
                    将等式加入解集
                end if
            end for
        end for
    end for
end for
```

之后，都将列表切片从该位置截断，取该位置之后的值组成新的列表，对新的列表再次定位，直到找到所有的i。通过上述算法即可遍历得到所有的解。

3.2 移动两根火柴

对于移动两根火柴的问题，需要考虑的情况种类更多，但算法思路与移动一根火柴的类似，涉及仅一个字符自身移动的与两个字符相互配合移动的与上述无异，以下面这种方法为例简述三个字符配合移动的情况(见Algorithm 2)：

$$((u,0,1,v),(m,0,1,n),(e,0,-2,f)\quad u,m,e\in single\{A,\alpha,B,\beta,C\}\quad u!=m\quad u!=e\quad m!=e$$

由于三个字符有两个都是要增加一根火柴，这属于同一转化操作列表，为了避免操作搭配的重复，只有当y与z的位置满足index(y)<=index(z)时，才进行转化为新等式的操作，这样每个for循环里，执行的三种操作的搭配是唯一的，注意到此时=不能去掉，因为即使是同一操作数组，也可能是由不同位置上的相同字符执行的，这与之前数学表达式中(!=)的含义也是相符的。

当single$\boldsymbol{X}$存在相同字符时，该算法虽不存在操作搭配上的重复，但会存在位置搭配上的重复(当index(y)=index(z)时，pos_2和pos_3会出现相等的情况，若此时pos_2和pos_3的列表长度大于1，会出现重复)，这部分重复属于少数，在程序最终予以去除即可。

当允许增加额外的数字，两个字符分别减少一根火柴，生成额外的一位数字1，同样相当于三个字符相互配合，仅是第三个数字变为n=m+10或者n=m*10+1，算法实现与上述相同。

当涉及到四个字符相互配合移动两根火柴时，是最为复杂的情况，在这里

Algorithm 2 三个字符相配合，移动两根火柴

```
for single𝑿中的每个字符i do
    for i在字典中对应的每个(*,0,-2,*)型的可行操作j do
        choice_1 ← choice_1 + j
    end for
    for i在字典中对应的每个(*,0,1,*)型的可行操作k do
        choice_2 ← choice_2 + k
    end for
end for
for choice_1中的每一步操作x do
    获取x待改变字符在single𝑿的全部位置pos_1
    for choice_2中的每一步操作y do
        获取y待改变字符在single𝑿的全部位置pos_2
        for choice_2中的每一步操作z do
            if index(y)<=index(z) then
                获取z待改变字符在single𝑿的全部位置pos_3
                for pos_1中的每个位置u do
                    for pos_2中的每个位置v do
                        for pos_3中的每个位置w do
                            if u!=v and v!=w and u!=w then
                                将u,v,w位置上的字符改变为操作后的对应值
                            end if
                            if 形成的新等式成立 then
                                将等式加入解集
                            end if
                        end for
                    end for
                end for
            end if
        end for
    end for
end for
```

我们并不直接求解，观察当执行(u,0,-1,v),(m,0,1,n)操作后，原待求等式变为了一个新的待求等式，这时我们再调用移动一根火柴的算法函数，一定可以找到另一组(e,0,1,f),(g,0,-1,h)，从而得到解集。这种方法虽然会与前几种方法寻找的解集存在较多重复，但在代码书写上带来极大的便利，对于重复我们在程序最终予以去除即可。

值得一提的是，并不是所有的待求等式都可以通过连续两次调用移动一根火柴的算法函数来解决，比如数字4和7增加一根火柴将不会变成任何数，而增加两根火柴便会变为9和3，若增加一根后再移动一次可变为5和4，这样若首先直接调用移动一根火柴的函数，待求等式的状态并不会发生改变。类似这种情况的存在，使得我们必须像上文一样分多种情况来解决问题。

对于等式变新等式，只需首先获取移动一根火柴的全部解，再获取移动两根火柴的全部解，即可构成全部解集。算法函数即为之前两种。

3.3 扩展到更多的长等式

以上讨论的均是$A\alpha B\beta C$的形式，而以上两种算法对所有等式都具有普适性，因此可以将简单的等式扩展到很长的等式(比如A+B×C-D=E的形式)，解决算法和上述相同，唯一不同的是判断形成的新等式是否成立的方法，对于简单的等式直接计算判断即可，而对于复杂的长等式，则需要利用正则表达式。

在这里参考https://www.cnblogs.com/Faker006/articles/7211577.html

利用python语言实现数学公式的计算，使用re.complie()编译正则表达式，匹配待计算的加法、减法和乘法等式，再重新定义MUl、Add、Sub方法，利用括号的添加即可实现字符串类型的数学公式计算，由于等式可能有多个等号，因此我们利用列表切片，将等式用等号分隔开，分别计算各部分的值并判断是

否相等。(注:这部分只有正则表达式部分参考链接原文，其余判断等式是否成立均为自己实现)

4 UI设计和操作

注：由于扩展模式即长等式处理部分为最后添加，故起初介绍的程序界面与真实相比略有不同(少了一个扩展模式按钮)。

4.1 软件程序的开发

本软件采用Python 3.6语言开发，UI界面是基于PYQT5框架编写的，生成的exe文件在dist文件夹中，由pyinstaller打包生成，文件夹中已经包含有程序运行依赖的python包和图片资源，双击main.exe即可开始火柴游戏之旅。

4.2 模式选择

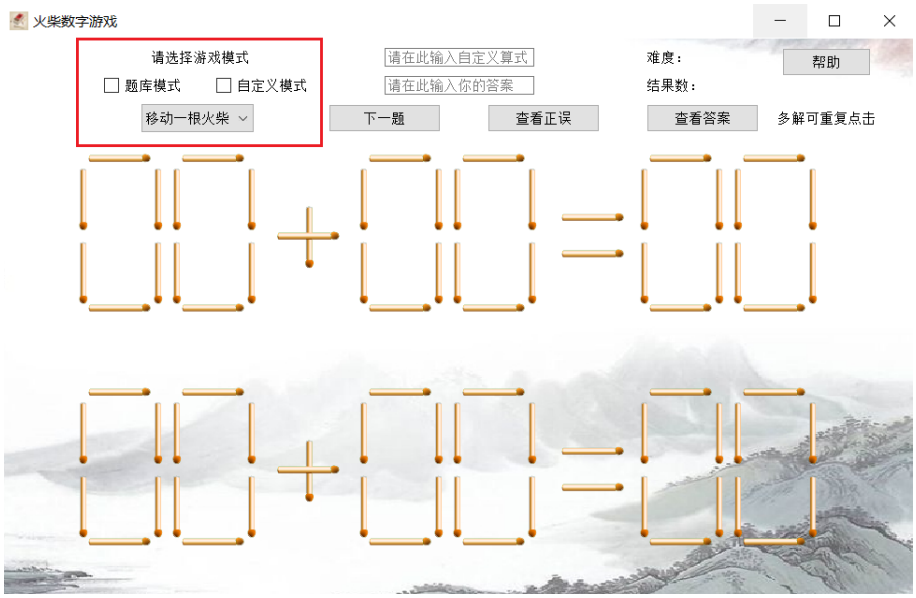

图 1: 模式选择

如图 1，本游戏采用山水风景图作为背景，以真实火柴摆放数字，既可以采取题库模式，也可以采用自定义模式，共提供三种游戏方案：移动一根火柴、

移动两根火柴、等式变新等式。玩家首先根据自身需要，选择唯一的游戏模式。如果未选择或多选择，系统将会弹出对话框予以提醒。

说明：本系统对于移动两根火柴的定义是，必须移动两根火柴解决问题，如果某个移动两根火柴的方案，移动一根火柴即可完成，此解将不在移动两根火柴的解集中，而应该在移动一根火柴中查看。

4.2.1 题库模式

当选择题库模式时，点击下一题，系统由选择的游戏方案找到对应的内置题库随机给出问题，如图 2 所示

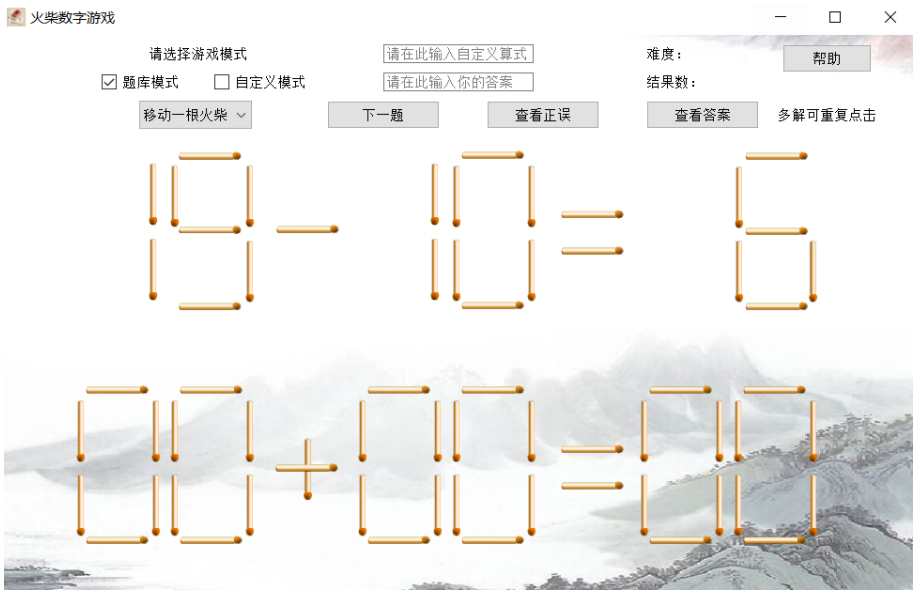

图 2: 题库模式：移动一根火柴

这时玩家可以选择跳过此题，只需点击下一题，系统便又会随机选取题目，如果碰巧与上一题相同，请再次点击刷新。当选到自己感兴趣的题目，玩家可以在请在此输入你的答案区域给出自己的解答，并点击查看正误判断解答是否正确，如图 3。

之后玩家可以查看答案(也可以自己不作解答直接查看)，页面下方的等式便

图 3: 玩家给出解答

会显示正确答案，如果出现多解，只需重复点击查看答案按钮，显示的答案便会持续更新，直到没有更多的解，再循环到第一个解显示，如图 4 5 6。

图 4: 第一个解

4.2.2 自定义模式

若玩家选择自定义模式，需要在请在此输入自定义算式区域给出自己定义

图 5: 第二个解

图 6: 没有更多的解

的题目，在输入栏输入符合要求的等式，满足

$$A \pm B = C \quad A \times B = C \quad A = B \pm C \quad A = B \times C \quad A = B = C$$

之一的形式，其中A,B,C均是不超过两位数的整数，×请用叉号代替，然后点击下一题，页面便会显示您刚才输入的题目，这里以移动两根火柴为例，如图7。

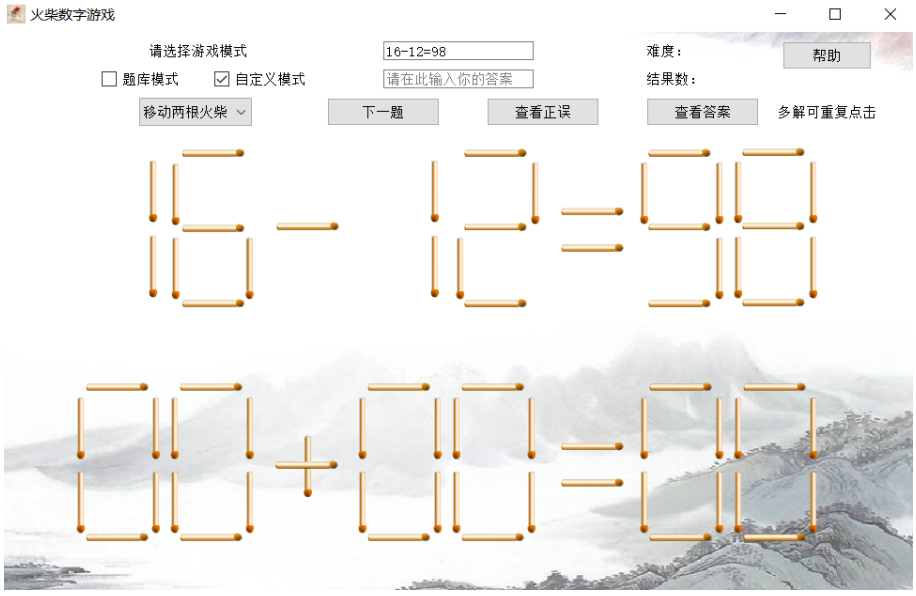

图 7: 自定义模式：移动两根火柴

同样地，这里玩家也可以给出自己的解答，或者直接查看答案，如果无解，系统会显示无解，答案显示本题有四个解，如图 4.2.2：

4.3 难度及结果数的显示

观察上面截图显示答案时，系统会给出对应问题的解决难度及总结果数，结果数即解集中解的总数，关于难度的评测，本系统遵循如下算法：

$$hard_{par} = ln(\frac{\text{操作过程中需判断是否成立的新等式总数}}{\text{结果数}})$$

此数字大致在1-7之间，显示值越高，表示此题越难解决。

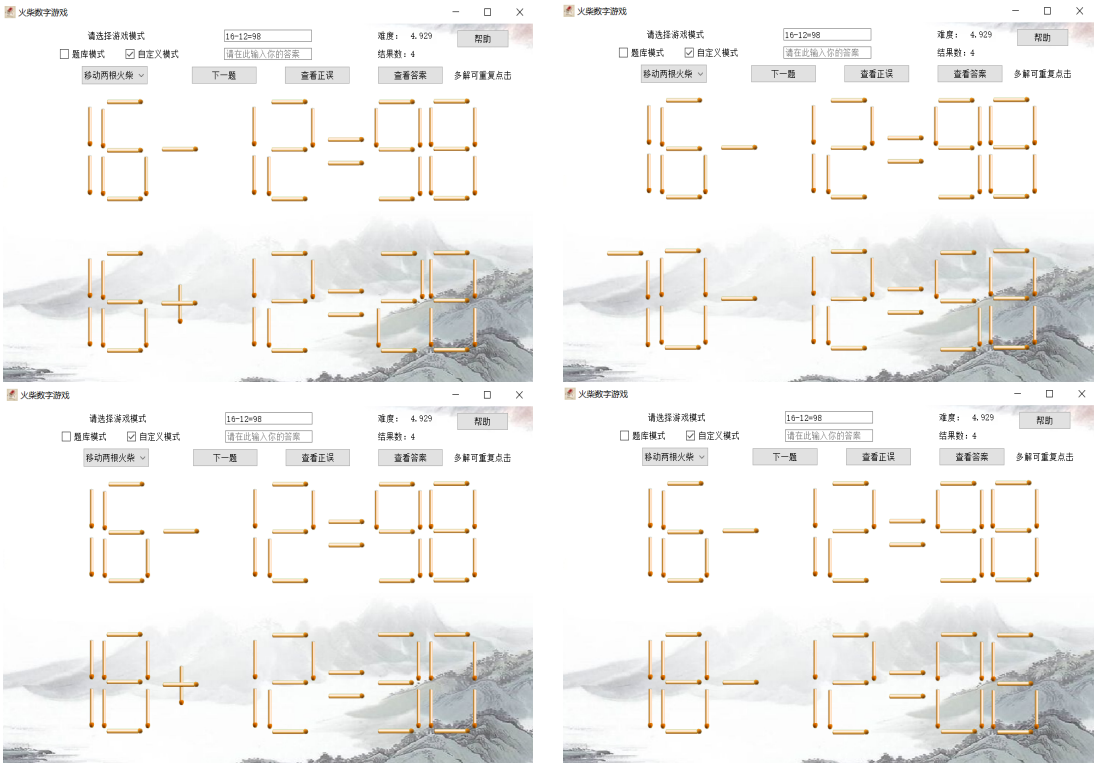

图 8: 16-12=98移动两根火柴的四个解

比如对于移动一根火柴9-5=8问题，在操作过程中，得到了32个带评测的新等式，经判断有2个等式是成立的，即本问题难度为：$ln\frac{32}{2} = 2.772$。

4.4 等式变新等式的提示

当选择等式变新等式模式，查看答案时每个解出现时，系统会提示玩家需要移动几根火柴可以得到该等式,如图 9。

4.5 题库的扩充及文件操作

考虑到系统本原题库数量的局限性，我们可以利用玩家的智慧来扩充我们的题库。当玩家采用自定义模式，若输入的等式可以求解，且不是原题库中的等式，我们就将这个等式写入对应的游戏方案的文件中，即data_1.txt(一根火柴),data_2.txt(两根火柴),data_3.txt(等式转换)三个文件。文件起始不存在，在

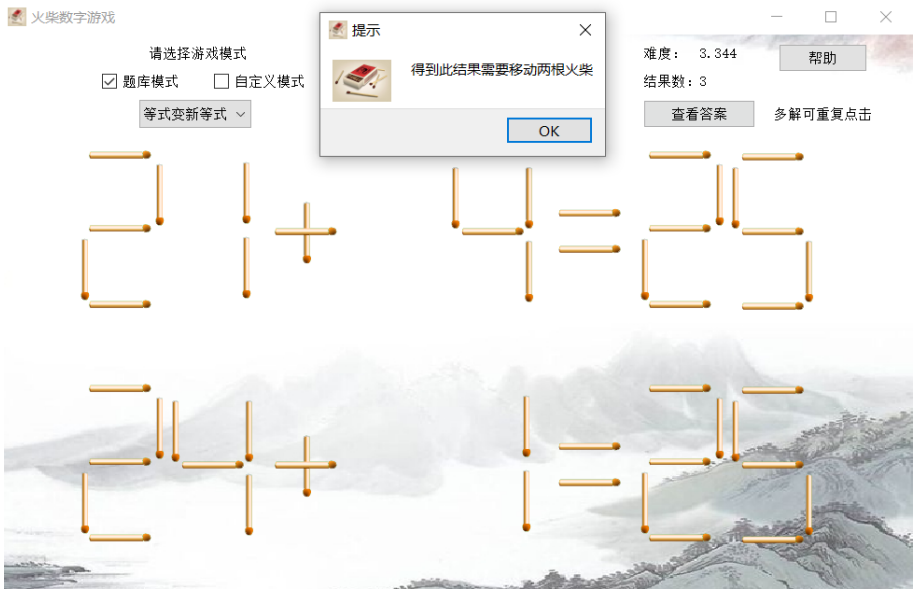

图 9: 题库模式：等式变新等式

玩家自定义输入后，会自动创建在exe所在文件夹，并随玩家的输入扩充题库。

这样在之后的玩家采取题库模式游戏时，系统除读取自带题库外，也会读取对应文件中的扩展题库，这样就利用不同玩家的智慧扩展了系统题库，使题目更加丰富，游戏更加有趣。

4.6 UI界面的容错机制

最终发布的应用程序，应具有高度的鲁棒性。在UI界面的设计中，我兼顾了容错机制的考虑:

1.当玩家未选取或多选取游戏模式时，给出警告；

2.当自定义模式输入算式不合法时，系统警告输入符合要求的等式，这里采用了try-except异常处理机制，当出现列表或数组越界时，程序不会崩溃，而是转到except，弹出QMessageBox，增强了程序的稳定性。

3.查看答案时，当点击出全部解后，系统提示没有更多的解，这一部分是由界面类的全局变量控制的。

4.7 帮助界面

玩家在游戏过程中如有疑问，请点击帮助按钮，即可弹出帮助界面如图 10

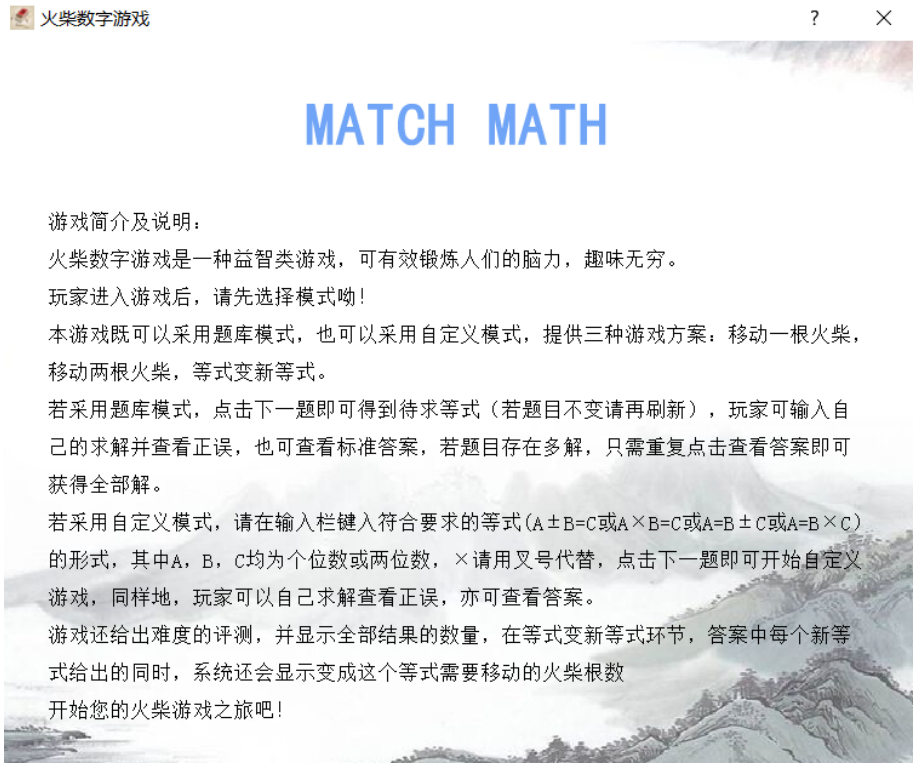

图 10: 帮助界面

4.8 长等式运算

在项目的最后，我将算法扩展到较长的数学等式，玩家可以点击扩展模式按钮，如图 11

得到界面如图 12

玩家同样可以选择游戏模式，点击查看答案即可在下方得到符合要求的全部结果，受限于界面大小无法用火柴棍显示，游戏体验有所下降。

5 项目总结

本次大作业完成过程中曾经出现如下问题：

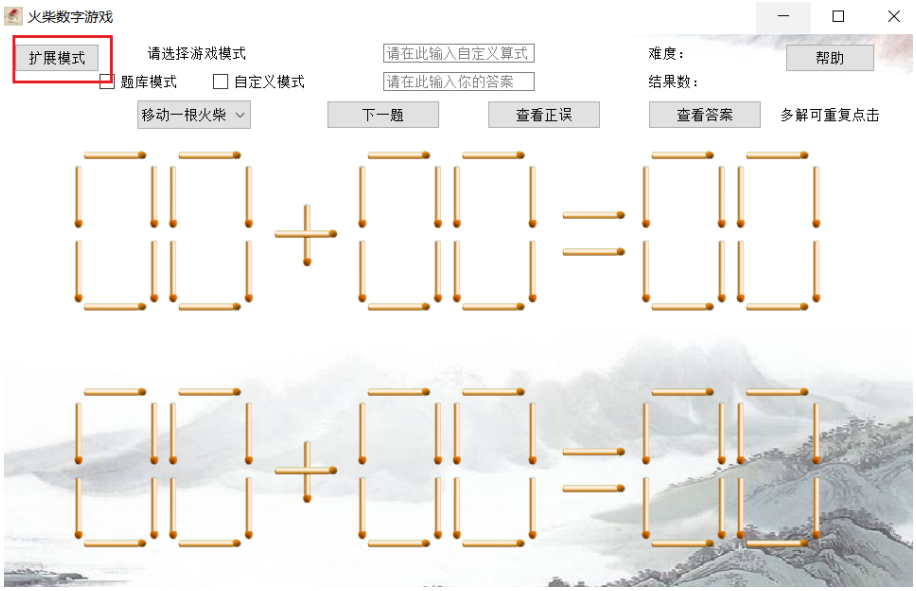

图 11: 扩展模式的添加

图 12: 长等式界面

1. 0显示的bug:如果某个数字经改变后为0，且出现在十位上，在界面中起初无法显示，后来我通过比较起始等式与最终等式的位数，加入判断语句予以解决；同理在挤出数字1的操作中，如果存在某些数字变为0(比如8+8=7移动两根火柴可以变为01+6=7)，也需要比较位数解决显示问题。

2. 起初移动两根火柴的解包含移动一根火柴的解，需要将这部分去除；同时注意去除所有搜索过程的重复

3. PYUIC在将UI文件生成py文件时，务必要指定清楚ui文件，有一次在生成的时候，我误指定了main文件，导致我的主文件瞬间变空，幸亏我提前有备份，损失很小，不然耗时一天的工作就白费了。也提醒我今后在项目开发时养成及时备份的习惯。

4. pyinstaller打包成exe文件，起初我在没有python和QT环境的计算机测试时，总是显示Falied to execute main.exe,原因是程序无法读取PYQT5的库，参考https://www.cnblogs.com/wisir/p/11515211.html

在import PYQT5前加入：

```
import sys, os
if hasattr(sys, 'frozen'):
    os.environ['PATH'] = sys._MEIPASS + ";" + os.environ['PATH']
```

问题得以解决，在纯净的计算机上可以运行。

总结：

本次大作业让我更加熟悉python语言，更加熟练地使用pycharm软件和PYQT5框架，对QT信号槽的消息处理机制的理解也进一步加深，还学习了正则表达式和文件处理的相关知识。

本次大作业还让我加深了对搜索的理解，但是题目要寻找出全部解，设计启发函数比较困难，对于火柴棍等式状态也很难定义深度，因此在本次大作业设计中我主要采取的是广度遍历的迭代方法和少量的递归，对于其他的算法运用较少，希望在之后的作业项目中得以应用，不断磨练自己的编程和算法水平。