人工智能基础大作业
火柴谜题[1]

姓　　名：程笑天

班　　级：自 62

学　　号：2016011408

[1] https://github.com/greatwallet/Match

目录

第一章　任务描述

火柴谜题是一个经典教育题目，具有很强的趣味性、知识性和测试性。具体问题为：用火柴棍摆放成真值未确定的数字等式，如图 1，希望解谜者能够移动若干根火柴，使得等式成立。

在本项目中，我们对该问题加以以下约束：

- 在解谜过程中，等式长度恒定；不能在空白区域放置火柴棍，也不能移动火柴棍使得某位变成空白位；
- 等式右侧表达式不含任何操作符，且为自然数；
- 不可移动操作符上的火柴棍；
- 操作式不含括号。

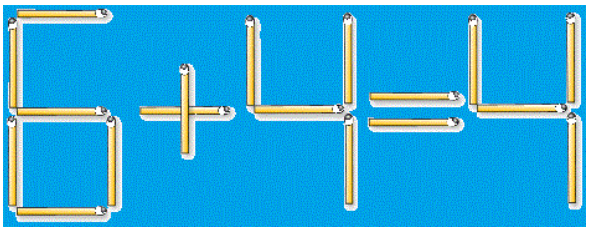


图 1 火柴棍等式

为了实现一个能够提供不同难度谜题的火柴谜题应用程序，除了核心的解谜算法之外，我还需要解决构造谜题、等式真值判定和必要的界面设计等问题。这个项目中，我完成了必要的界面设计，该应用程序具备以下功能（具体操作说明请见第四章）：

- “自定义模式下”，用户可输入有限长度的谜题，并交由程序解谜；
- 在不同的谜题难度下，程序可自主生成相应的谜题；

- 在不同的谜题难度下，程序可自主解答相应的谜题；

  其中，谜题难度受到操作数的位数、操作数的数量、算数操作符的难度（含加、减、乘、整除）和解谜可移动火柴数的影响；
- 用户可设置算法最长运行时间、最大输出答案数量参数；
- 用户可完成对界面进行自定义美化操作。

第二章　问题建模

由于问题涉及到界面表示、数据结构以及搜索方法的实现，因此需要将问题拆解成若干个子问题分别进行建模。对整个火柴谜题而言，子问题可分为界面表示，内存数据结构存储方式、答案搜索算法、谜题生成算法以及等式求值等。

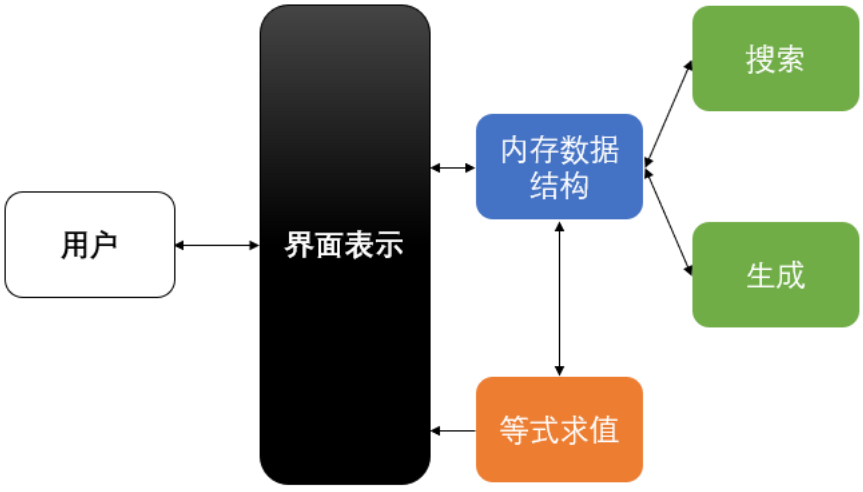


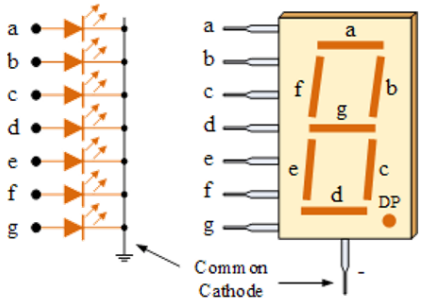
图 2 系统框图

2.1 等式的数据结构存储形式

众所周知，一个数学等式同时具备了操作数和操作符两种元素；在本问题中，我们不仅需要考虑其在内存数据结构的存储方式，更要考虑其如何转换成利用火柴棍表示的界面元素。

在本问题中，我们已经限定不可移动操作符，因此操作符只需设定简单的映射关系即可。

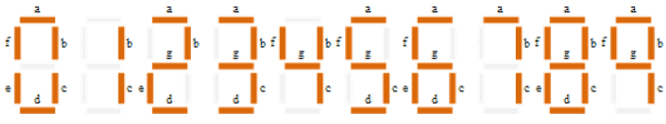
对于操作数，由于火柴棍界面表示形式与七位数码管类似（见图3）。我们不妨将一位操作数设定为共阴极的七位数码管形式，其编码方式与共阴极七位数码管编码方式相同，并利用十六进制整型数进行存储。



(a)

Decimal Digit	Individual Segments Illuminated						
	a	b	c	d	e	f	g
0	×	×	×	×	×	×	
1		×	×				
2	×	×		×	×		×
3	×	×	×	×			×
4		×	×			×	×
5	×		×	×		×	×
6	×		×	×	×	×	×
7	×	×	×				
8	×	×	×	×	×	×	×
9	×	×	×			×	×

(b)



(c)

图 3 七位数码管及其编码 （a）共阴极七位数码管示意图；（b）数字编码表；（c）数字编码显示情况

对于整个表达式，可利用字符串存储，再利用操作数与操作符的映射规则转换成相应的界面元素即可。

2.2 解谜搜索算法

对于火柴棍解谜过程中，若利用人工思路进行分解，则可把解谜过程看作在某位某段上移出一个火柴，再在某位某段上移入一个火柴。不妨将“移入”和“移出”操作视作同等级的基本操作；则可构建图搜索算法：

定义：定义图节点状态为相应的等式及其附加属性；将节点扩展定义为：对等式中任意一个操作数进行任意一段火柴的基本操作（移入或移出）并得到节点序列；该节点序列被称为原节点的后继或子孙。

由以上定义，可以得知若允许移动总火柴数为n，则生成搜索树的深度$d = 2n$。可对图进行任意启发式搜索，理论上可搜索到任意解。以图 4 为例，对于任意一个根节点输入，可通过图搜索算法对节点进行逐一扩展，直至图中每一个节点均遍历完毕。

特别需要定义的是：在解谜搜索过程中，目标节点的性质为：

- 目标节点为叶节点；
- 目标节点符合表达式规范
- 目标节点有意义（即：可判断真值）
- 目标节点真值为真（即：等式成立）

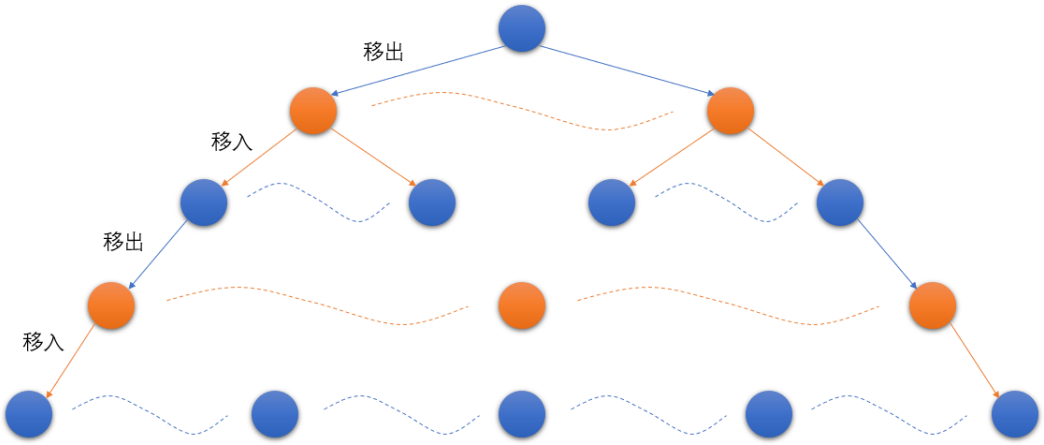


图 4 $n = 2$情况下搜索树，深度为$2n = 4$（定义根节点深度为 0）

2.3 生成搜索算法

本问题中，应用程序需要具备根据不同难度自主生成题目的能力。对于一个题目而言，难度可根据算式长度、操作数大小和操作符难度来定义，具体难度定义可参考第四章。

事实上，生成搜索算法内核即为 2.2 提出的解谜搜索算法，并只需要进行略加修改即可。不同于解谜，程序需要随机生成一个严格成立的等式（作为火柴谜题的谜底），并调用解谜搜索算法，得到的目标节点序列即为题目。

对于根节点的生成，我们可根据不同难度的定义，随机生成一个真值为真的等式。接着，我们需要对解谜搜索算法进行以下修改：

- 由于谜面生成需要具备随机性，因此在扩展子节点时，需要对子节点序列进行洗牌，使得搜索路径具备一定的随机性；
- 由于我们只需要生成一个题目，那么算法无需遍历所有图节

点才终止；基于搜索路径具备随机性，可以在搜索到第一个目标节点时即可终止，这样也可保证生成谜面具备随机性；

- 目标节点的定义在生成算法中发生变化，其等式真值不一定为真，只需有意义即可。

2.4 算式求值

在日常生活中，我们所使用的表达式被称为中缀表达式，其特点为非常直观且具备较强可读性。然而由于中缀表达式具备一些约定俗成的运算次序，不容易被计算机所快速处理。波兰数学家 Jan Lukasiewicz 提出后缀表达式，该形式表达式较容易为计算机变成处理。具体算法请见 3.3。

第三章　算法设计和实现

下面将简单介绍各算法的具体实现及一些剪枝技巧。

3.1 解谜搜索算法

首先假设不进行任何剪枝操作。若根节点等式操作数长度为L_{opnd}，则最坏情况下，一个根节点的直接后继数目不大于$7L_{opnd}$；若允许移动火柴数目为n，则搜索树深度为$2n$，则搜索复杂度可以高达$O\left(\left(7L_{opnd}\right)^{2n}\right)$，为 NP 问题。

由以上分析，我们可以得知，搜索树是一个宽度大、深度小的搜

索树，则我最终采用**深度遍历搜索**，以实现搜索运行时间的大幅度缩短。同时我也采取了其他的一些技巧防止程序占用过多内存或者耗费大量的时间：

- 设定时间阈值T，若搜索算法时间超过该阈值，算法终止；
- 设定最大答案数量阈值M，若搜索算法已经搜索到M个目标节点，算法终止。

在调试过程中，我发现算法有可能会陷入“返祖现象”的陷阱之中：所谓“返祖”，即第$l+1$层的移入/移出操作抵消了第l层的移出/移入操作，导致孙子节点与祖父节点的状态完全一致（见**错误!未找到引用源。**）。在这种情况下，我增加了一步相应的剪枝操作，在扩展节点时，若孙子节点与祖父节点完全一致，则将该部分孙子节点舍弃。

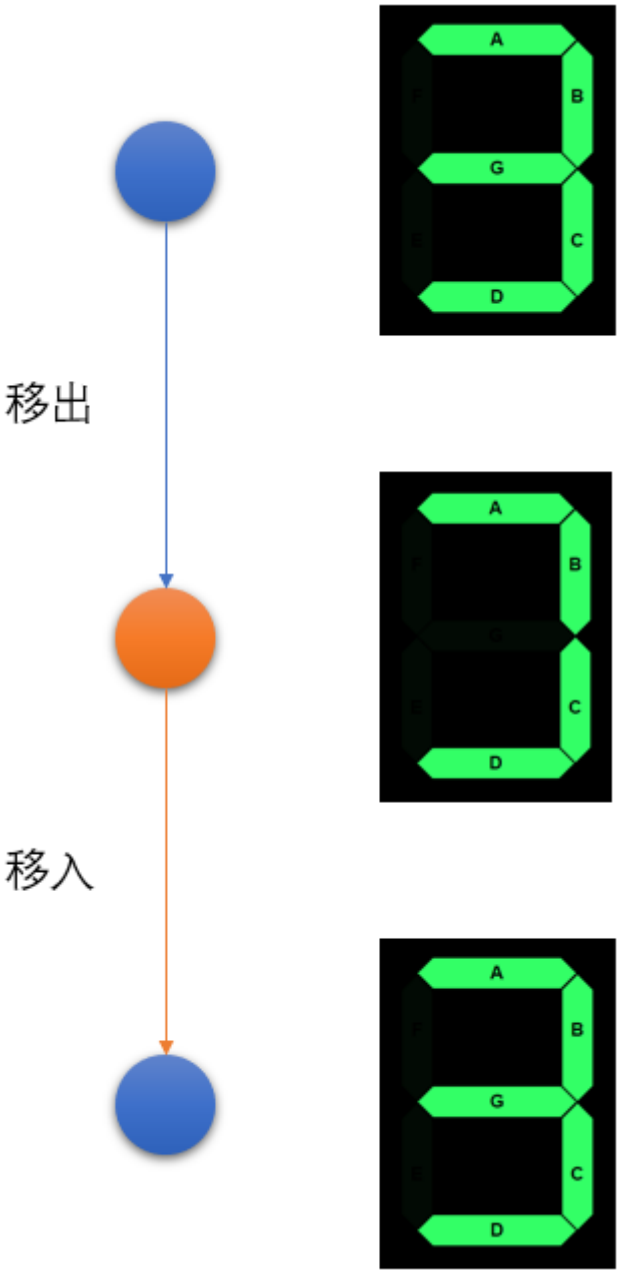


图 5“返祖现象”：孙子节点与祖父节点状态一致，导致重复搜索。

3.2 生成搜索算法

通过 2.3 的分析，我们可知生成搜索算法可由 3.1 的深度搜索算

法改写而成；由于生成算法仅需要一个目标节点，则深度搜索算法的优势性在这里将更大，即算法将更快地终止。

因此生成搜索算法为：

(1) 根据输入的难度随机生成一个有意义的等式；
(2) 构造根节点，进行深度搜索；
(3) 若搜索到一个目标节点，搜索终止；

另外不得不提的是，在调试过程中，我发现了当$n = 2$时，生成搜索算法生成的结果与根节点在部分情况下并非“严格差值为 2”的，而是“严格差值为 1”的。对于“严格差值为n”，我有以下定义

定义：若 A 节点与 B 节点“严格差值为n”，则二者所有操作数十六进制异或之和等于n。

若两个节点“严格差值为 1”，则说明两个节点无需经过移动 2 个火柴棍，而只需移动 1 个火柴棍也能达到同样的效果。具体案例请见图 6

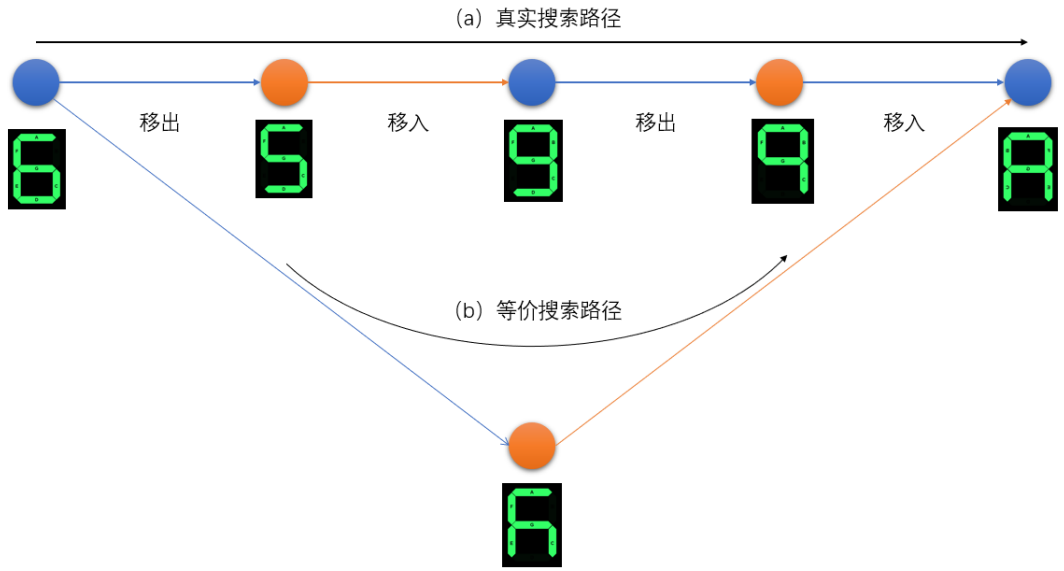


图 6“严格差值为 1”：真实搜索结果与根节点严格差值仅为 1，说明可通过深度更小的搜索路径抵达目标节点，这样的结果不符合我们的预期。

因此，为了防止结果出现严格差值与允许移动火柴数不相等的情况（事实上这种情况相当的多），我们应当在搜索至目标节点时进行判断并滤除不合格目标节点。

3.3 算式求值

算式求值算法如下：

(1) 判定表达式是否可运算
(2) 设定运算符优先级表；
(3) 准备两个空栈：运算数栈&运算符栈；
(4) 从左到右扫描表达式，对其中字符逐一做相应处理；

(i) 若当前运算符优先级低于栈顶运算符，将运算符与运算数压栈；

(ii) 否则，弹出栈顶两个运算数、栈顶运算符进行运算；

(iii) 重复以上直至运算符栈空

(5) 返回运算数栈顶

第四章 UI 设计和使用说明

我已经将本项目中所有文件上传至 https://github.com/greatwallet/Match。请感兴趣的朋友查看。

4.1 文件构成与编译环境

本项目在 Visual Studio 2015 中 C# .NET Framework 4.6.1 下进行开发。需要加载

- CSkin 16.1.14.3 版本下 CSkin 4.0 中 CSkin.dll，通过官网 http://www.cskin.net/下载；
- MaterialSkin (版本 0.2.2)，通过 NuGet 包管理器下载。

若需要重新编译，请打开 Match.sln 文件，并在 Visual Studio 中清理并重新生成解决方案。在 Match/bin/Debug/ 下或 Match/bin/Release/下找到 Match.exe，并运行。

注意：若您需要移动 Match.exe，请一并移动 CSkin.dll 与 MaterialSkin.dll 并保证三个文件在同一目录下。

更多详细内容请参考 https://github.com/greatwallet/Match/blob/master/README.md。对于各个文件的作用与功能，请查看 https://github.com/greatwallet/Match/blob/master/Match/FILE_INSTRUCTIONS.md。

4.2 主界面

应用程序分为主界面与设置界面，主界面负责进行生成火柴谜题与解题；而设置界面用于设置各项参数。下面介绍主界面下的两种模式：自定义模式与自主生成模式。

4.2.1 自定义模式

自定义模式下，用户可以自己输入一个有意义的等式，并请求程序解答。具体操作方式见下：

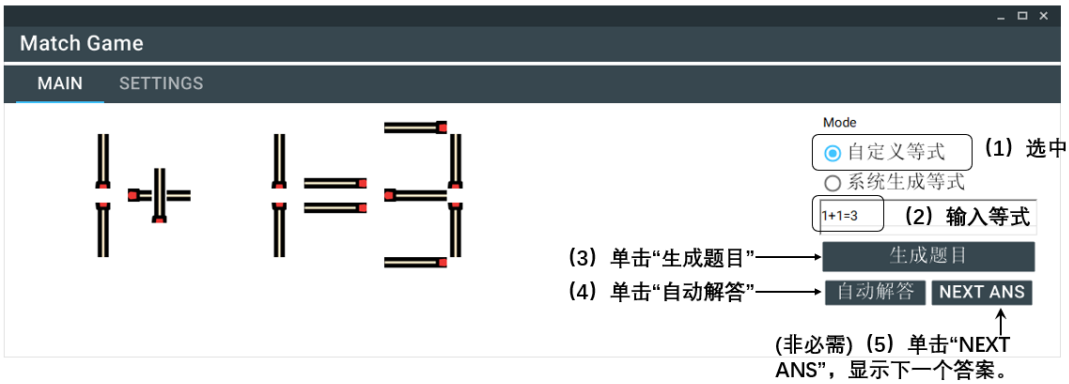


图 7 主界面自定义模式操作示意

结果如下所示

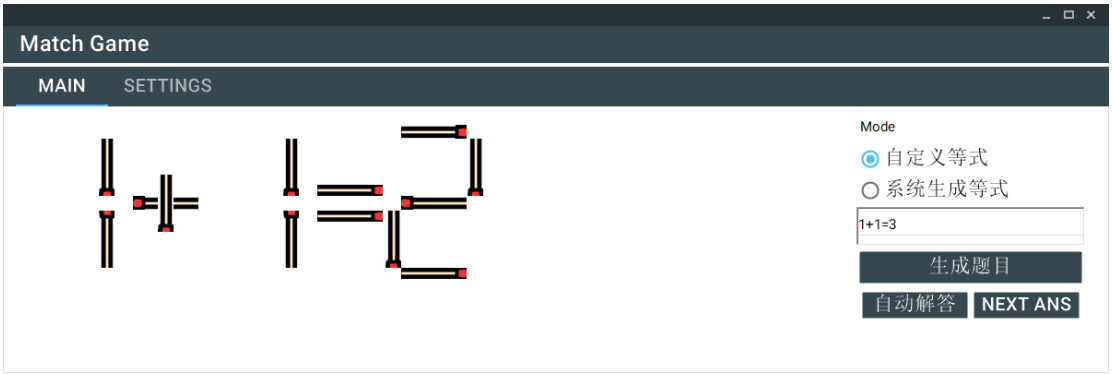


图 8 主界面自定义模式操作结果

4.2.2 自主生成模式

自主生成模式下，用户可命令程序按照一定难度自动生成题目，并予以解答。其中难度分为“EASY”，“MEDIUM”和“HARD”模式；这三者的区别在于：

- EASY：操作符集合{+，-}，运算数为个位数，运算数总共不超过 3 个；
- MEDIUM：操作符集合{+，-，*}，运算数为个位数，运算数总共不超过 4 个；
- HARD：操作符集合{+，-，*，/}，运算数为两位数，运算数总共不超过 5 个。

操作步骤与结果见下：

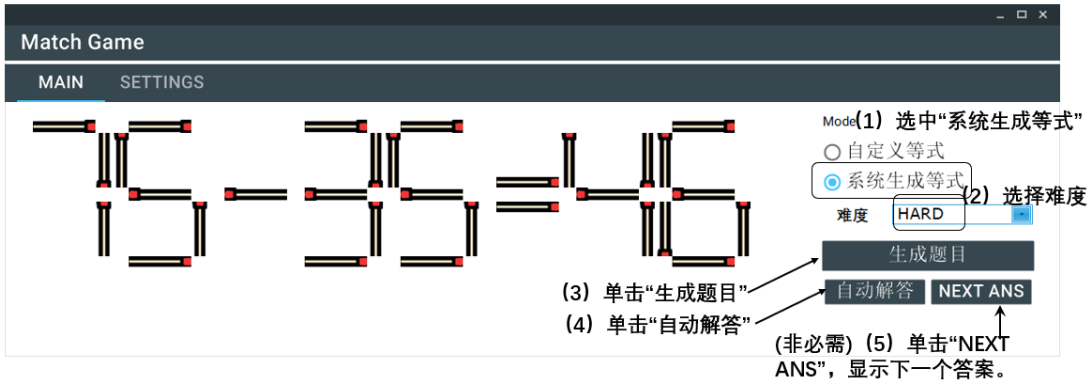


图 9 主界面系统生成模式操作示意

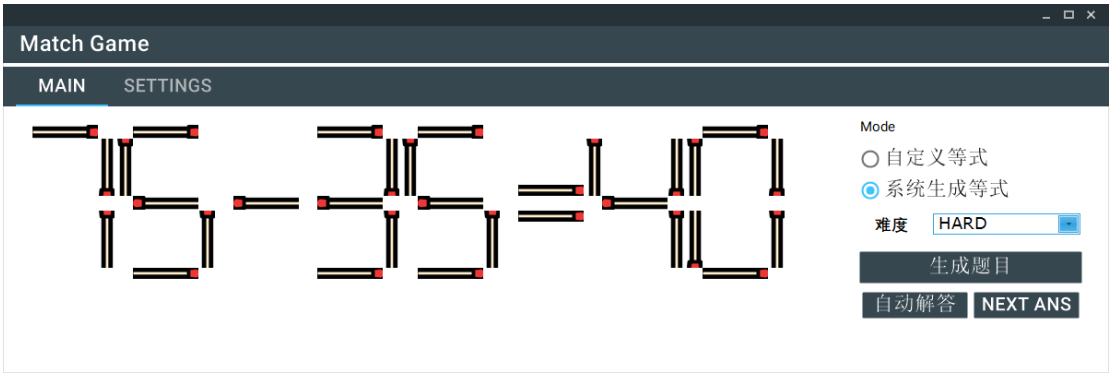


图 10 主界面系统生成模式结果

4.3 设置界面

在设置界面中，用户可自行设置相关参数。下面我将逐一介绍各参数的作用。

图 11 设置界面

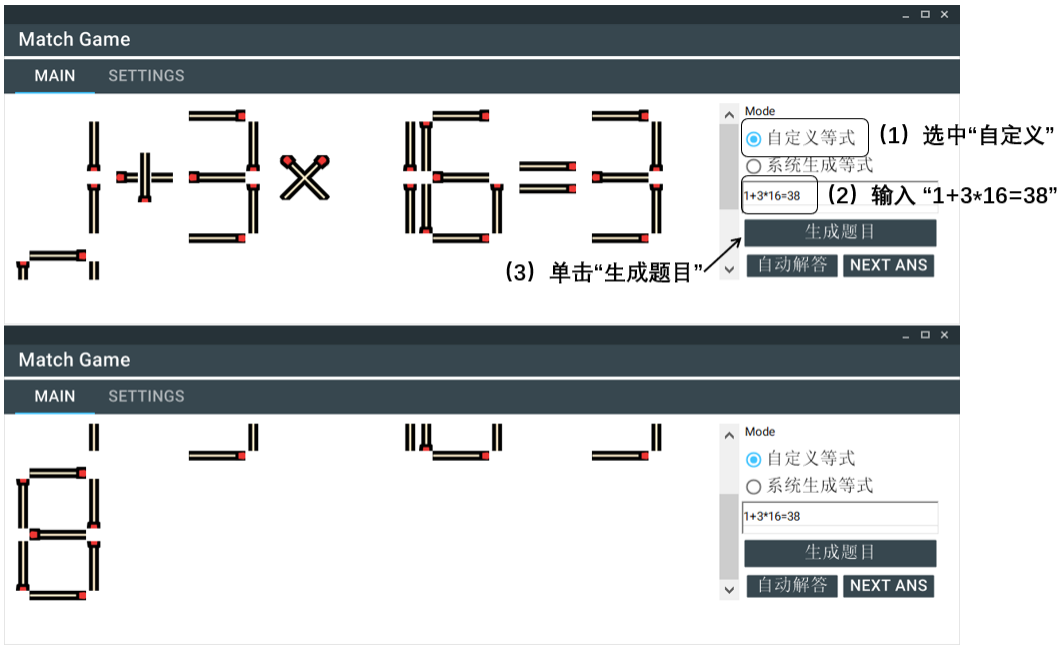
- 个性化：用户可改变界面主题与颜色；
- 火柴数目：即允许移动的火柴数目n；
- 搜索答案最大数量，即搜索算法能够给出的最多答案数目，若选择∞，则搜索算法将遍历所有节点。
- 允许算法最长搜索时间：若超出该阈值，搜索算法终止且宣告失败。最小值 1 秒，最大值 20 秒。

4.4 使用案例

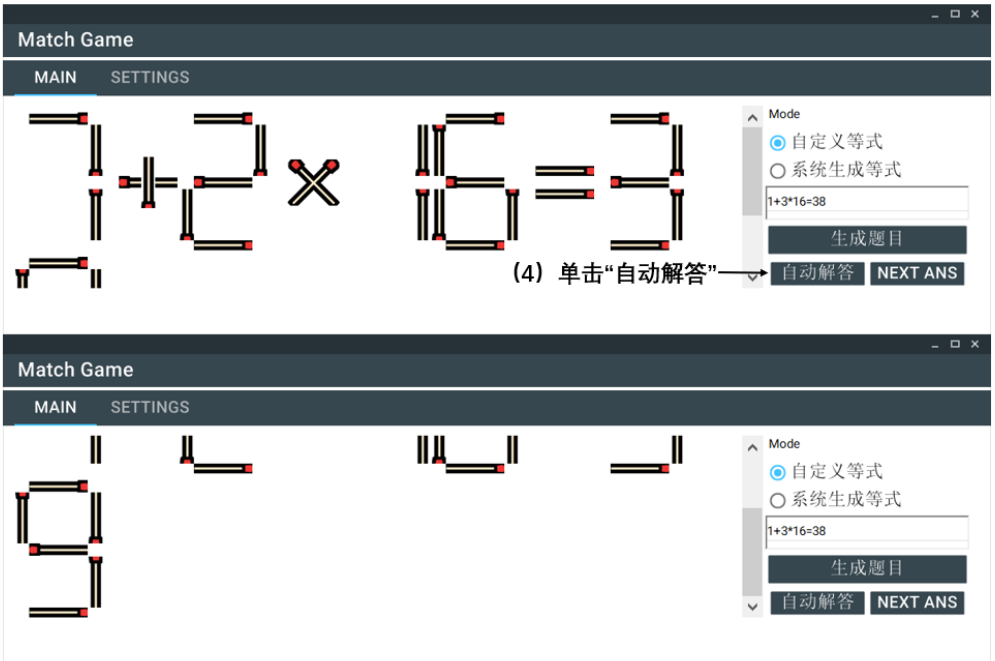
下面将展示一个使用案例：

(1) 设置火柴数目为 2，搜索答案最大数量为 2；

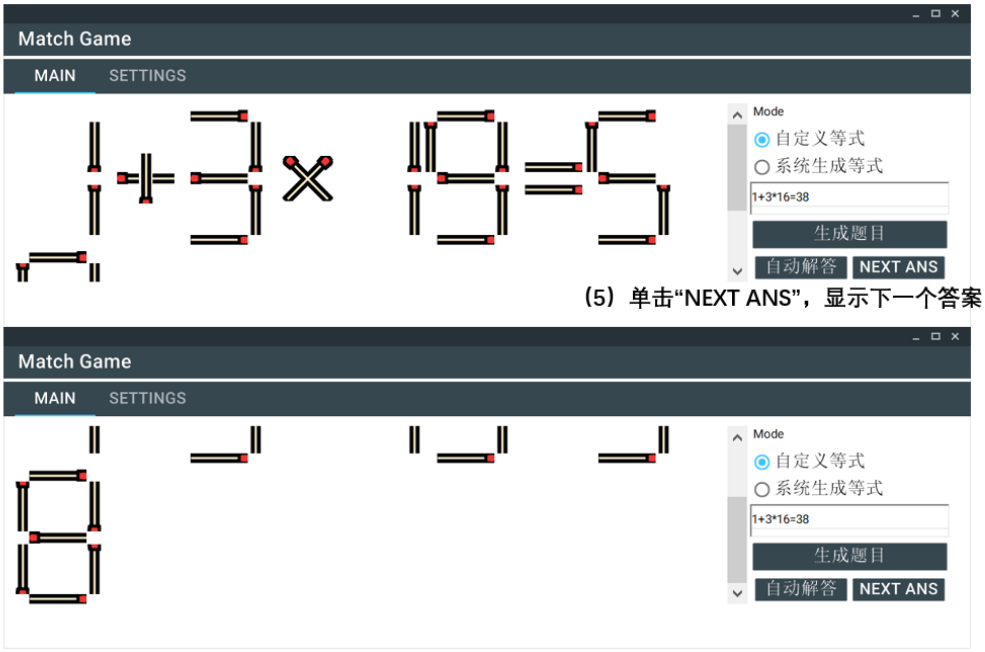
(2) 在自定义模式下输入“1+3*16=38”，图框中出现题目；



(3) 单击自动解答，此时约需等待 6 至 7 秒时间，出现第一个答案“7+2*16=39”



(4) 单击“NEXT ANS”显示下一个答案，显示“1+3*19=58”



（5）单击“NEXT ANS”，显示下一个答案。

以上即为案例展示。

第五章　实验总结

本项目以火柴谜题为背景，利用深度遍历搜索和部分剪枝技巧完成了实验。项目整体使用了 C#语言完成了搜索算法和界面设计。用户可在不同难度和模式下生成和求解谜题，同时还具备了其他参数自定义功能。不过仍有部分功能未完成，譬如用户和程序交互式部分。我计划在下一阶段完成该部分内容，并完善算法，减小空间消耗与搜索运行时长，使其更加完备，成为一个兼具娱乐性和教育性的产品。