人工智能导论大作业一

填词游戏

Word Puzzle Playground

班级：自 66

姓名：夏卓凡

学号：2016011496

2019 年 4 月 22 日

目录

1 题目描述

1.1 问题说明

填词游戏规则：利用提供的诗词语料库中的诗词，组成特定的“汉字”形状。

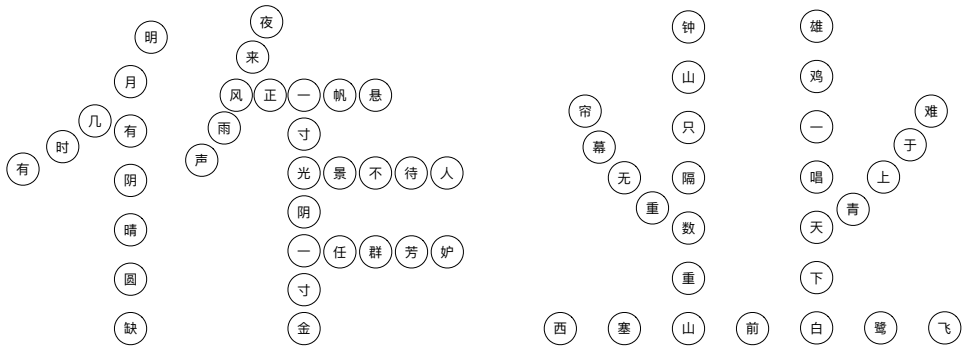

图 1: 填词示例

如上图所示，填词具体要求：

(1) 每一个笔画，用一句诗词填充。

(2) 在笔画交叉处，使用相同的字。

(3) **笔画长短**和**所填充的诗词字数**之间，不要求严格的对应关系。即便是很短的笔画，也可以用一句很长的诗词来填充。不过，请尽可能保证美观。

(4) 交叉点的位置可以根据需要，略微调整，但请尽可能保证美观。

1.2 开发要求

请自己编写一个小软件，能够完成如下功能：

a) **【完成】**利用所提供的诗词语料库的内容，进行填词游戏。本题要求给出“人、工、智、能”这四个字的填词结果。下图提供了一个填词模板，供参考。其中各个笔画的顺序，以序号标出，请不要调整序号。交叉点的位置，在下图中用红点标出。交叉点在诗句中的位置可以按需要作调整，交叉点的数目请不要改变。每一笔画，所对应的诗词字数，没有严格要求，可以自行设置。

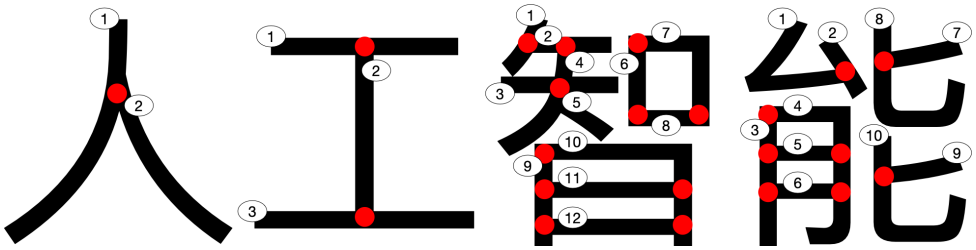

图 2: 填词模板

每个字的笔顺，以及每一笔的类型，如下图所示（供参考）：

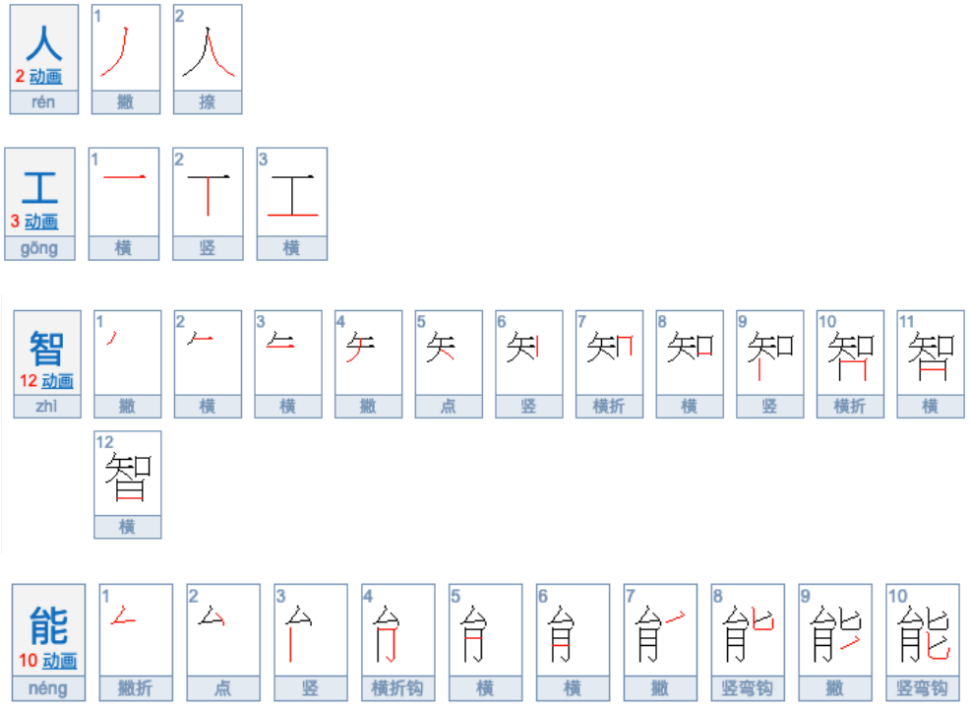

图 3: 参考笔画

[注 1] 软件的输出结果要以某种清晰易懂的方式，进行展示：

(1) 需要给出每一笔画代表的诗词

(2) 需要标出交叉点位置的字。下面给出了一种简单的输出方式，可以参考，不过可以不局限于此，可以自行设计清晰美观的输出方式。

“人”：
笔画 1：明月几时有
笔画 2：月有阴晴圆缺
交叉点：月

[注 2] 如果软件无法给出完整结果（或无解），可以只给出其中一部分结果（视情况酌情减分），对于无解的区域，进行留空并标注出来，比如：

“人”：
笔画 1：明月几时有
笔画 2：[无解]
交叉点：月

b) **【完成】**允许用户交互，对结果进行灵活调整。

(1) 用户可以指定其中某几句的诗词，可以指定交叉点的位置。例如，要求“人”字第一笔为代表的诗词为“举头望明月”；交叉点的字为“头”。

(2) 如果用户对生成结果中的某一句诗词不满意，可以要求换一句满足交叉点要求的诗词。例如，软件对“人”字第一笔给出的结果原本为“低头思故乡”，交叉点为“头”。那么用户可以要

求软件换一句诗词，保持交叉点不变。那么软件能够给出另一个可行结果，比如“举头望明月”。

c) **【完成】**【选做】提供输入接口，输入某种格式的汉字形状信息（要求简便、易操作），不局限于“人工智能”这四个字，可以得到输入的任意汉字的填词结果。

d) **【未完成】**【选做】对于一个给定的填字结果，可以转变为一个请人玩的游戏：1. 没有任何提示信息（最难级别）2. 提示每一个交叉点的字。3. 提示每一个交叉点的字和每一句诗词的第一个字。4. 提示每一句诗词的前一半的字。

2 算法设计

2.1 问题分析与建模

事实上，填字游戏作为搜索问题来看，应当具有以下几个特征。

搜索空间 搜索空间为给定词库的规模，对于要填入笔画的每一句诗词，其长度为一确定值。而对词库的分析发现，各个长度诗句数目分别为 5：6074，6：1400，7：5552，8：330，9：173，10：91，11：41，12：22，13：8，14：3，15：3，16：2，17：1，18：1，20：2。那么例如一个 7 画和 5 画的汉字“人”来说，其搜索空间大概在 1×10^6 到 1×10^7 之间。可见，在一个汉字字中连接区域进行搜索，该问题的规模，也即搜索空间的大小，与连接区域中句子的个数是指数关系。

产生解 产生解的过程即是不断确定一个汉字中的一个连通区域中每个句子应该填入哪些诗句的过程。产生的可行解都是最优解，即完成汉字连通区域填充，就可以完成问题的求解。而产生一个可行解的过程较为复杂，需要满足汉字结构的诸多约束，算法的核心就在于确定一种总能找到解的过程。

评价解 由于可行解即是最优解，无需对解进行评价。

继续对填字游戏本身的问题进行分析，其约束为交叉点上的字必须相同。也就是对于相连接的两个笔画 1 和 2 而言，如果笔画 1 所填内容确定了，那么笔画 2 所填内容必须在笔画 1 连接点处具有相同的字；但如果笔画 2 尚未确定，而填充笔画 1 的时候就不存在这样的约束。这就是说，在搜索的过程中，约束是不断变化的，需要动态地考虑这些变化。完成填充之后，所有约束都将起作用。所以，对于给定的汉字结构，包括每个笔画长度以及连接点的位置，只要这些确定下来，初始值的选取将不再重要。所以可以按照笔顺逐次进行搜索。

将整个问题建模为一个图搜索问题序列，就是**将每个笔画对应可能的句子看作节点**，而**将连接点的约束看成边**。这张图由上万个节点和边构成，我们要做的搜索，就是找到一条连通的路径，将这些待搜索的节点通过约束构成的边连接起来。完成这些一个连通区域即完成了一次图搜索，而一个汉字根据结构可以分成一个或多个这样的连通区域，依次进行搜索即可。

2.2 求解算法的设计

基础的求解算法基于深度优先搜索构建，逐步确定每个句子，找到满足当前约束的笔画后立刻搜索下一画。设定一个连通区域为一个“连接”，程序中为“stroke”，记为 S；一个句子为“verse”，记为 v；一个连接点“anchor”记为 a。其中一个句子对应一个给定的长度 L，一个连接点由一个四

元数组构成，$a[0]$ 为起始笔画的下标，$a[1]$ 为连接点所在起始笔画中的下标，$a[2]$ 为终止笔画的下标，$a[3]$ 为连接点所在终止笔画中的下标。其意图为**连接点将 $a[0]$ 的 $a[1]$ 位置与 $a[2]$ 的 $a[3]$ 位置连接起来**。设定两个栈“candidate”和“proposal”，用于存储搜索的状态，分别代表未搜索的笔画（“候选”）和已经搜索的笔画（“提案”）。算法运行只需要不断从“candidate”（C）中取出元素，根据当前填词情况确定有效约束，再遍历相应词库，找到合适的诗词后放入“proposal”（P）即可，。另外还应考虑这一层搜索找不到的情况下需要回溯，需要设定一个集合表示已经搜索过的诗句，记为 U，算法表示如下。

Algorithm 1 基本搜索算法

set $P = \{\}$, $C = \{N-1, \ldots, 0\}$
repeat
  set i =pop(C), constraints=$\{a|a[2] = i$ and $a[0] \in P\}$, L_i
  find $j \in$ database$[L_i]$ s.t. $v[a[0]][a[1]] = v[a[2]][a[3]] = v_j[a[3]]$ $\forall a \in$ constraints
  if $\exists j$ **then**
    push(P,i), add(U_i,j)
  else
    set l =pop(P), push(C,i), push(C,l), clear(U_i)
  end if
until $C \neq \{\}$

实际操作中，受限于实际输入的字形，在某些极端条件下求解过程十分漫长，耗时数十秒才能给出结果；所以在使用的时候，加上了最大搜索步数限制。在搜索步数限制的情况下，得到结果之后就可以将结果收集并呈现了。实际程序中还加入了随机初值功能，用来增加求解的多样性，但同样会遇到一些极端的条件导致耗时较长。

Algorithm 2 附加约束的搜索算法

set $P = \{\}$, $C = \{N-1, \ldots, 0\}$
repeat
  set i =pop(C), constraints=$\{a|a[2] = i$ and $a[0] \in P\}$, L_i
  if v_i is fixed **then**
    find $j \in$ database$[L_i]$ s.t.database$[L_i][j]= v_i$
  else
    find $j \in$ database$[L_i]$ s.t. $v[a[0]][a[1]] = v[a[2]][a[3]] = v_j[a[3]]$ or $\exists a[4], a[0] = i$ $\forall a \in$ constraints
  end if
  if $\exists j$ **then**
    push(P,i), add(U_i,j)
  else
    set l =pop(P), push(C,i), push(C,l), clear(U_i)
  end if
until $C \neq \{\}$

针对另一个问题，即增加约束条件，在给定句子或给定连接点位置的字的情况下，需要对这个框架做一下变换。即将“anchor”增加一元，用于表示附加约束。设定 $a[4] = c$，其中 c 为附加约束

对应的字在转换后的表示。事实上，给定句子和给定连接点的字在搜索的约束上没有区别，只需要稍作处理即可，将给定的句子的字固定到连接点上。上面给出附加约束的搜索算法中就提到，对于给定的句子，直接搜索下一句即可，对于给定的连接点的字，在搜索前一笔画的时候需要考虑与之相关的连接点约束。至于连接点位置的变化，事实上基本搜索算法也可以处理。

3 软件开发与使用

3.1 软件的开发

本项目使用 Visual Studio 2017 和 Blend for Visual Studio 2017 开发，使用 C# 语言的 WPF 框架进行编写，UI 使用第三方库 MaterialDesignInXAML 进行构建，.NET Framework 版本为 4.7.2。在文件方面，对原有的 `THUOCL_poem.txt` 进行整理，并得到一张查找表，用于将文字转换为 16 位无符号整数进行表示，这一部分使用 Python 进行预处理。

3.2 软件设计架构

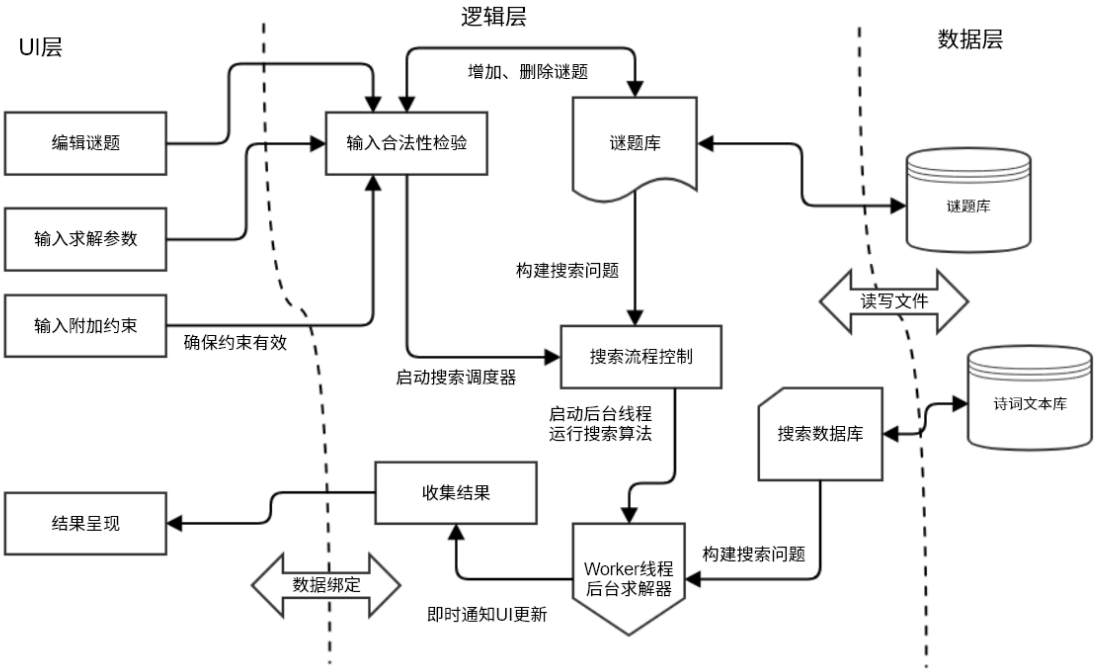

图 4: 软件设计架构图

受益于 WPF 的数据绑定机制，我们可以使用较少的代码构建出相对复杂的 UI 来展示数据；将更多的代码放在运行逻辑和具体算法的实现上。UI 层的功能主要是对用户提供接口，使用户可以编辑谜题，输入附加约束以及查看问题的求解结果。逻辑层的主要功能是调取数据，完成编辑谜题库的更新以及控制搜索流程。其中运行搜索算法本身单独使用 `BackgroundWorker` 做成工作线程，进行异步调用，使得 UI 界面能即时更新结果，消除卡顿。谜题库的存储涉及到文件的读写，我定义了一组谜题的表示方法，用文本文件的形式进行序列化存储，其中存储每个汉字的连通区域，每个连通区域中的连接点以及笔画长度，对应还需要保存一组在 UI 容器坐标系中的坐标，用于可视化。UI 与后台的数据绑定的具体细节不再赘述，虽然实现起来比较复杂，但不是本次大作业的核心问题。

3.3 功能实现与使用说明

3.3.1 样例问题——“人工智能”的求解

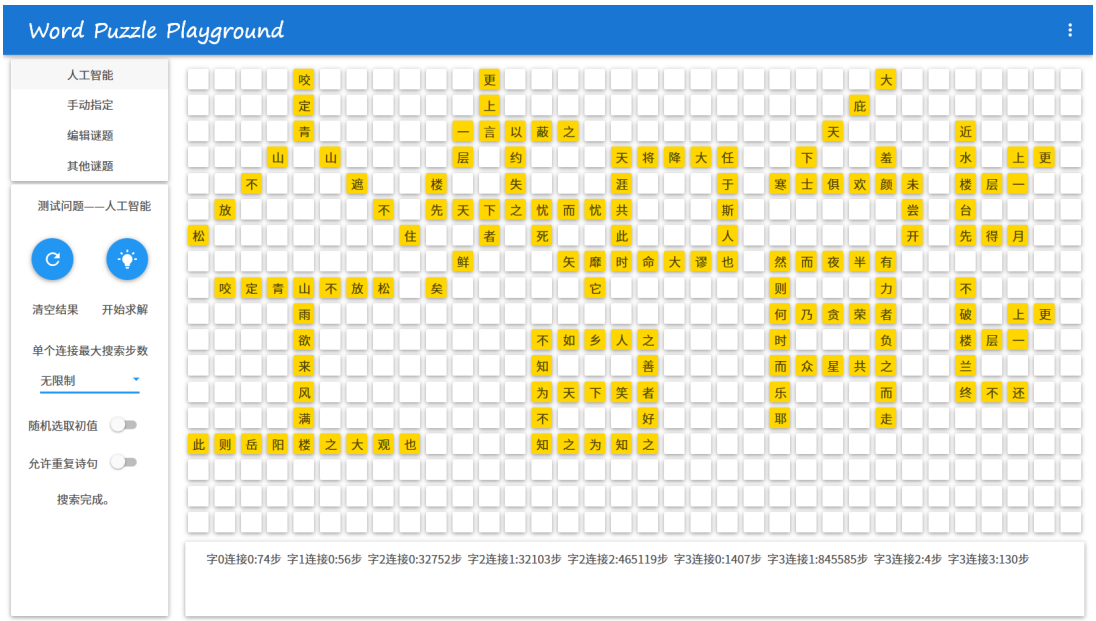

图 5: 样例问题的求解 1

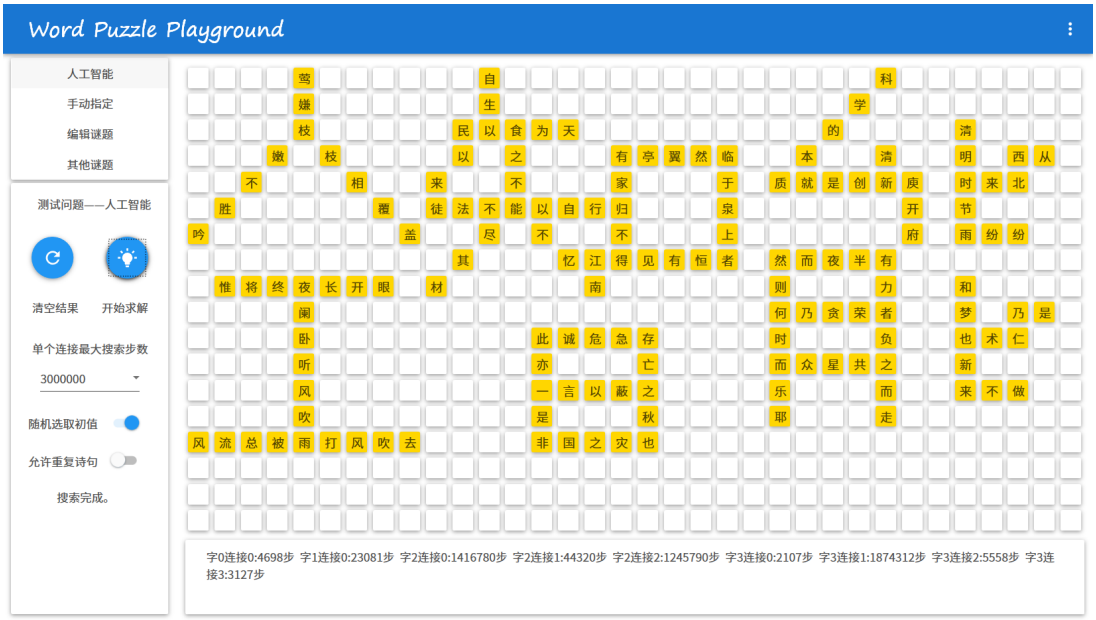

图 6: 样例问题的求解 2

以上两图给出了测试样例问题的方法，即在“人工智能”页中点击开始搜索即可。由于随机初值可能导致搜索时间过长，可以通过限定搜索步数，搜索不同的结果。下方的卡片上显示了每个连接所用的搜索步数。经过测试发现“智”字的下半部分和“能”字的左下部分满足条件的结果较少，搜索比较困难，一般来说会在这两个地方出现步数超限的情况。

3.3.2 附加约束的求解

图 7: 附加约束的求解 1

图 8: 附加约束的求解 2

以上两图给出了附加约束情形下的两种结果，由于涉及到交叉点变化，无法用可视化的方法进行表达，在左侧卡片上输出文字结果。情形 1 为指定其中一个笔画的句子，情形 2 为指定连接点位置和交叉点内容。情形 1 指定第 0 句为“拔剑四顾心茫然”，在此基础上搜索，第 1 句的结果为“四十而不惑”。情形 2 指定第 0 句的第 1 个字和第 1 句的第 0 个字相同，并且为“头”，在此约束下搜索结果为“举头已觉千山绿”和“头上金爵钗”。**特别要注意，如果同时指定交叉点内容和诗句内容但二者不兼容的时候，以诗句内容为准。**

3.3.3 编辑谜题功能

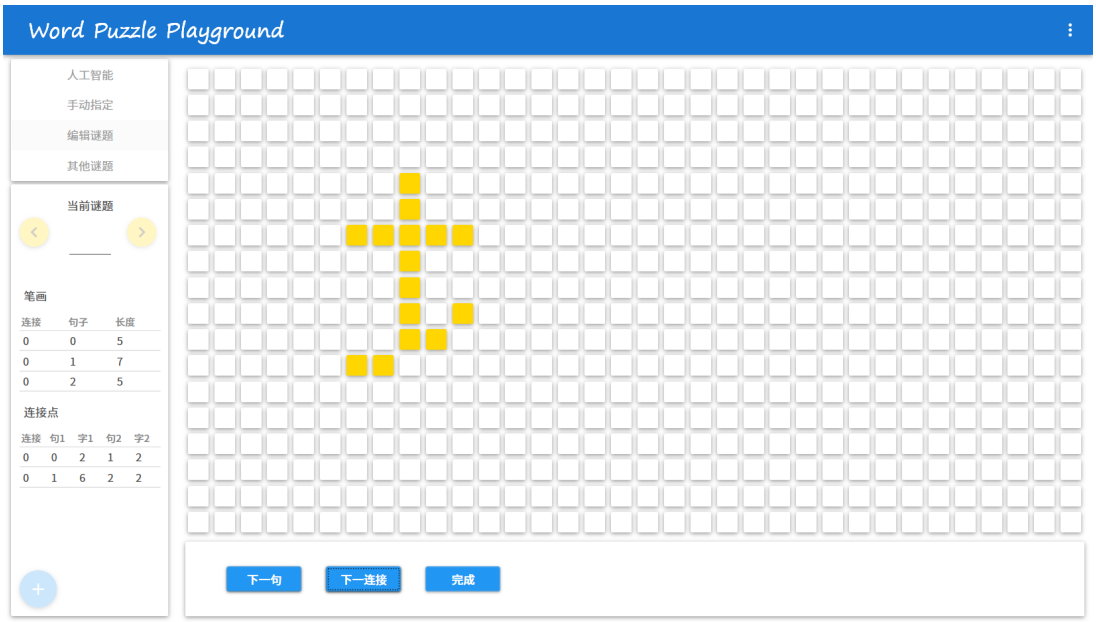

图 9: 编辑谜题 1

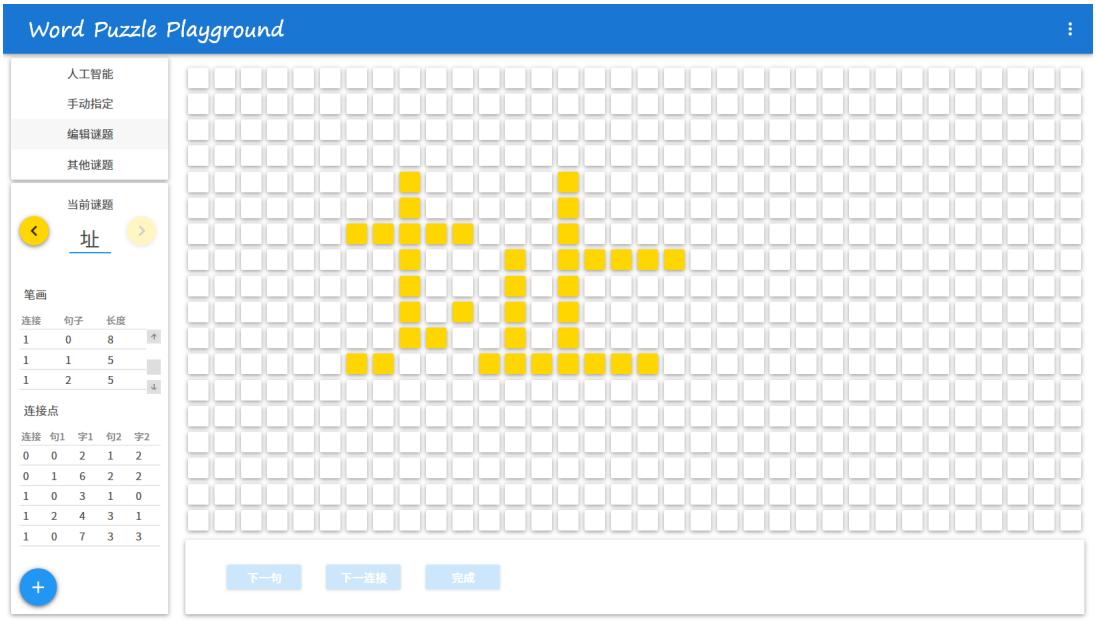

图 10: 编辑谜题 2

上面两幅图展示了如何使用编辑谜题输入汉字形状。首先，将待输入的汉字分解成数个连通区域，每个连通区域依次输入，软件默认每次点击网格输入的是同一笔画，在输入完一个笔画后点击“下一句”即可输入下一笔画。同一个连通区域中不同笔画的交叉点自动识别为连接点，软件禁止了同一句和自己连接，但没有处理不同连通区域直接共用一个点的情况，**请输入不同连通区域时不要有交叉点**，否则会造成未定义的结果。同样请输入时注意笔画长度，软件不会检查笔画长度是否在诗词库中有对应诗句存在，**输入不存在的长度同样会导致未定义的结果**。在输入完一个连通区域之后，点击“下一连接”输入下一个连通区域；输入完一个汉字之后，在“当前谜题”下方的文本框

内输入这个汉字，给谜题命名。左侧卡片的表格展示了当前谜题的汉字结构信息，包括笔画的长度和连接点信息。下方的“+”按钮弹出“添加”和“删除”谜题的提示，可以对谜题库进行编辑，但不能删除“人工智能”这 4 个谜题，否则测试样例无法使用。

3.3.4 自定义谜题的求解

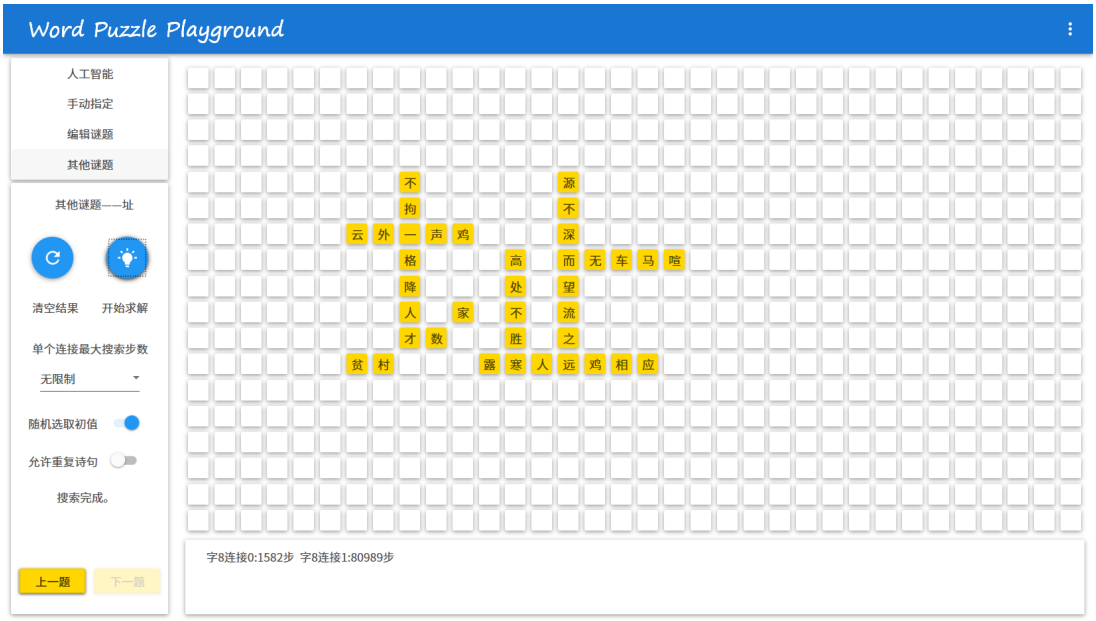

图 11: 自定义谜题求解

这一页面下可以对自己输入的汉字谜题进行求解，也可以求解“人工智能”的单个字，使用方法与测试样例页面类似。

4 总结与反思

算法实践 本次大作业给了我一次解决之前从未解决过的问题的机会，我也是第一次有机会安静下来仔细思考填字游戏这一问题的解决方案。从开始构思到算法 work 大概花了将近两周的时间，探索问题、尝试想法的快乐确实是其他大作业所不能带来的。当然，由于其中有一周期中考试，本次大作业的时间非常紧，我甚至没有考虑优化算法，或者寻找更多的求解思路。所以目前这个软件的搜索性能较差，也是无法否认的事实。尽管如此，我还是想到了一些可能的思路，算作是 Future Work 吧。

可能的改进思路 之前与同学交流思路的时候提到对于字符串搜索，可以建立从字到句的索引，虽然看上去规模庞大，但确实可以提高效率，这是一种空间换时间的办法。关于约束的排序，事实上也可以引入启发函数，对交叉点上出现概率高的字优先考虑填入，或许可以加快速度。目前的算法还是过于直接，很遗憾没有时间能实践这些改进思路。

选做功能的舍弃 我认为本次大作业的选做功能实际上有些问题，在构思的时候，选做 1 实际上等同于必做，因为如果不设定一个交互式输入问题的方式，在当前的 UI 为主软件架构下无法完成人工智能这四个字的显示。但选做 2 似乎又和搜索算法没什么关系，我一开始也想过做这个，代码里

也保留了网格坐标到实际诗句的反射，但后来考完期中考试只剩一周，遂作罢。或许不在大三选这门课，才能真正有时间享受学习人工智能的乐趣吧。

A 可执行文件使用方法

1. 需求 `Windows 10`，`.NET Framework >= 4.7.2`。
2. 安装`./fonts` 文件夹下的字体。
3. 执行`./bin` 文件夹下的 `WordPuzzle.exe`。