人工智能基础

大作业

七巧板解决方案

学　　号　　2017011589

姓　　名　　吾尔开西

专　　业　　自动化

日　　期　　2019.10.12

目录

任务描述

1、必做任务一

在一个固定的图像库中选择可以拼的图像。

其实这个任务是整个工程的核心，也是基础。在给定的图片中，黑色区域表示摆放图形区域，其余部分为白色。程序的输入是这张图片，需要将七块固定形状和大小比列的板块放到黑色区域中且不重叠，输入图片的黑色区域保证能放下一定大小的板块。程序的输出是这些板块摆放的位置，搜索的路径等。

七巧板即七个板块，包括：两个大三角形，两个小三角形，一个中三角形，一个平行四边形，一个正方形。

2、必做任务二

允许使用者自己定义，或者输入一个可以拼的图像。

这个问题有可选择的余地，为了增强程序的鲁棒性，我选择让使用者在我设计的 ui 界面上定义图像。使用者在 ui 界面上绘制出图形，并给出所定义图形中摆放的板块数。使用者绘制的图形需要保证这些板块能放得下。

3、选做任务一

可以实现任意数量板块的摆放。

必做任务一中只需要实现 7 块板的摆放，该是必做任务一的拓展，增加板块数量，图形的形状也要相应的发生变化，搜索难度增大。

我的做法是将该任务与必做任务二合并，由于必做任务二是用户自己定义图形，自由度较高，可以任意更改各板块类型的数量，所以用户定义的图形能达到 13 巧板，14 巧板等等，这样就可以少做一个用户界面，并且提高用户自定义的自由度。

4、选做任务二

用户给任意一张图像，给出拼接方案。

对于这个任务，需要分三步进行，首先要将用户给出的图片进行二值化处理，得到图片中需要拼接的图形部分，之后将二值图片规则化，将它变成可以用 x 巧板进行拼接的形状，最后进行拼接。

问题建模

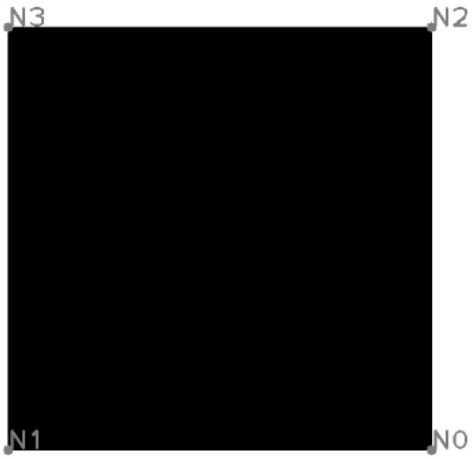
方案一

1、大体思路

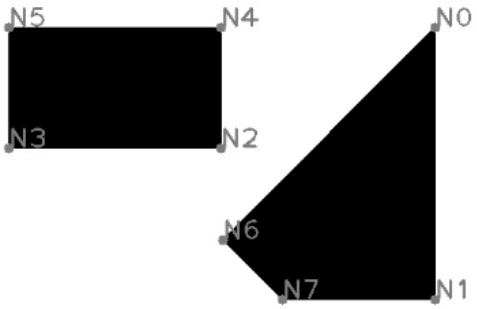
使用 openCV 检测待拼接图形的边缘点，以这些边缘点为顶点来摆放板块，由于板块只能以 45 度角为单位旋转，所以在每个顶点处摆放板块的方式是有限的。在摆放时，计算板

块的摆放位置，并根据一些板块内部点和边缘点处的灰度值确定是否能以某种方式摆放，如果可以，则在搜索图中产生后继节点，节点记录板块的摆放位置。

在拓展某一个节点时，根据该节点记录的板块摆放位置来画出当前图形，即在初始图形上用白色画出各个已摆放的板块，剩下的黑色部分即是当前需要摆放板块的部分。再次用 openCV 检测边缘点，放板块，产生后继节点。



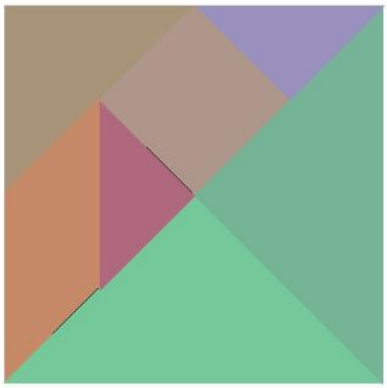
（初始图形的边缘点）



（搜索过程中某节点当前图形的边缘点）

以上过程中，产生的后继节点放入 open 表中，每次只放一个板块，放某板块时可能有多种摆放方式，所以会产生多个后继节点。取下一个节点时从 open 表的最后取，即采用深度优先搜索的方法。

下图中是这种方案求解出来的正方形的摆放结果，用时 343 秒。



2、 优缺点

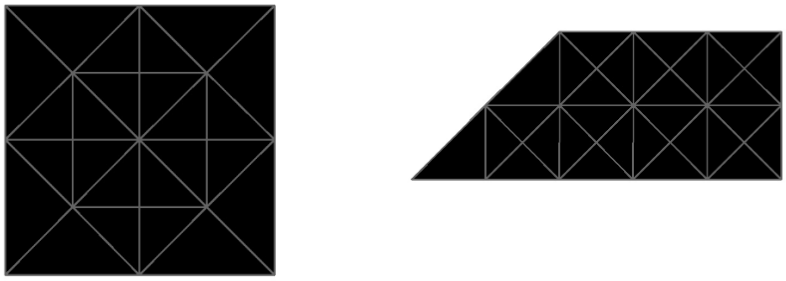
这种方法能保证求出解，但由于 openCV 函数耗时过长，所以程序求解需要较长时间，大概在10^2秒量级。权衡之下，我舍弃了这种方案，没有以它来做 UI 界面，只将有效代码附在 tangram_solve_contour 文件夹中。

方案二

1、 大体思路

将图形分解成 16 个小三角形，以这些小三角形为基本单位，来进行板块摆放时的判断、记录等。

分解小三角形前要计算出图片黑色区域的面积，通过面积求出小三角形的直角边长。分解小三角形的方式有两种，一种是小三角形斜边横向，另一种是斜边斜向，如图所示。



（斜边横向与斜边斜向分解方式）

程序运行时会尝试两种分解方式，并根据生成的节点决定选择哪一种。

上面所说的 16 个小三角形是指面积意义上的 16 个小三角形，程序中得到的小三角形其实更多，因为不同的小三角形之间会有部分重叠。此外，面积三角形个数是由各个板块的数目决定的，七巧板对应的便是 16 个小三角形。

得到图形的基本单位——小三角形的相关信息后，之后就不会涉及到图像处理了。利用小三角形的信息可以判断某板块可放置的位置，比如正方形板块可以放在两个斜边重合的小三角形上，平行四边形板块可以放在直角边重合且顶点满足一定位置条件的两个小三角形上。在某些小三角形上放置板块后，需要将这些小三角形以及有重叠部分的小三角形标记为“occupied”，以便后继节点查找摆放位置。

搜索时仍使用深度优先搜索，每次摆放一个新板块，可能有多种摆放位置，每种摆放位置产生一个新节点，相当于本节点的后继节点，将这些新节点放入 open 表中，下次取节点时从 open 表最后一个位置取。

下面是这种方法得到的正方形的解，用时 0.117s。

2、优缺点

这种方法虽然不能保证求出解，但只在初始化阶段涉及图像处理，所以求解速度非常快，大概在10^{-1}秒量级，对全部 20 张图片都能在这个量级的时间内求解完成。而且这种方案对必做任务二和选做任务都更加适用，尤其是选做任务二，所以我决定在 UI 算法中使用这种方案。

算法设计和实现

由于 UI 算法中使用方案二求解，所以本小节主讲方案二的算法设计。讲述过程中并不着眼于具体代码，注重算法思路的讲解，代码结构请参考 readme 文件和代码注释。

1、初始化

（1） 图形节点

首先使用 cv2 的 goodFeaturesToTrack 函数获取图形的边缘点，再求出内部点。通过面积和小三角形个数可以得到小三角形的直角边长度，求内部点时，利用这个基本长度来得到边缘点周围的内部点，剔除重复的点。用第一轮得到的内部点和边缘点一起得到新一波内部点，同样要剔除重复的点。经过几轮这样的循环后，就能得到全部的图形节点，图形节点之间距离是基本长度——小三角形的直角边长度，或基本长度的$\sqrt{2}$倍，具体与分解小三角形的方式有关（见问题建模部分，有斜边横向与斜边斜向两种方式）。

图形节点存放在 Pic_feature 自定义数据结构中，该结构不止存放了节点的坐标，还存了节点的邻居节点的序号以及相关小三角形的序号。

由于有两种分解小三角形的方式（斜边横向与斜边斜向），所以有两种求解图形节点的方式，程序中需要

（2） 基本三角形

在得到全部图形节点后，就能利用这些节点来求解基本三角形了。具体方法是：遍历图形节点，找出以这些节点为直角顶点的所有小三角形，遍历时要避免重复定义，还要做好记录，在图形节点中标记好以 90 度角相邻的小三角形（该小三角形的直角顶点是该图形节点），

以及以 45 度角相邻的小三角形。

基本三角形的数据结构是 Small_tri 自定义结构，存放了其直角顶点和两个 45 度角顶点的序号，提高空间利用率，还存放了 occupied 标志位，标识该三角形是否被占用，方便搜索时摆放新板块。

（3） 其他

其他初始化步骤包括初始化 open 表、记录图形。值得一提的是，与图形相关的数据结构，包括图形节点、基本三角形等，保存在 Graph 自定义结构中，在“pic_process.py”文件里；而与搜索相关的数据结构，包括搜索图节点 Node_open_list，搜索类 Solver 等，在“game_class.py”文件里。

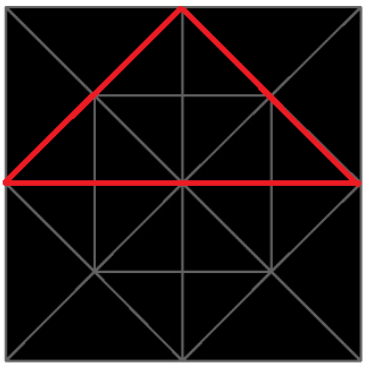
2、 摆放板块

搜索过程中，需要产生后继节点，即在当前图形中摆放新板块，采用一次摆放一个板块的方式产生新节点。

摆放板块时，遍历所有基本三角形，检查该基本三角形是否有条件摆放该板块，不同板块的摆放方法不同，最简单的是小三角形板块，只需检查该基本三角形是否已被占用，如果没有，那就可以摆放，否则不可。

以最复杂的大三角形板块为例，讲解摆放过程：大三角形板块面积上占四个基本三角形，实际占 6 个基本三角形，可能影响 10 个基本三角形的 occupied 状态。遍历所有基本三角形，对每个基本三角形 tri，把它视作大三角形板块的直角三角形，即 tri 的直角顶点便是大三角形板块的直角顶点。找到与 tri 斜边重合的三角形，以及大三角形板块的 45 度角顶点对应的两个基本三角形，检查这些三角形是否被占用，如果没有，那就可以在这个位置摆放大三角形板块，于是生成新的搜索节点。

以上的寻找过程通过基本三角形顶点的各种邻居关系来完成，并不需要多重遍历，速度较快。



除了判断某位置能否摆放特定的板块，还要在摆放板块后将涉及到的基本三角形 occupied 状态设为 True，这些被波及的基本三角形不止包括所摆放板块处的基本三角形。

3、 搜索算法

2 中讲解了摆放板块的方法，其实就是搜索时生成新节点的方法。

算法采用深度优先搜索，搜索时，最新的搜索节点在 open 表末端，即摆放板块更多的搜索节点在 open 表末端，所以表示节点走过的路程的 g(n)函数就没有必要考虑，而对于启

发信息 h(n)函数，难以找到一个合适的计算方法，如果该函数不合适，不仅额外消耗计算时间，而且可能将搜索方向带偏。此外，该方案求解速度很快，所以没有使用 A*算法。

根节点是图形初始化状态，没有摆放任何板块。进入循环，首先摆放大三角形板块，得到根节点的后继节点，放入 open 表，再取 open 表最后一个节点，摆放第二个大三角形板块，如果该问题只要求摆放一个大三角形板块，而该节点中已经摆放了大三角形板块，那就顺次摆放其他板块，以此类推，直到所有板块都摆放完毕。

抽象地来看，摆放某种板块相当于以某种方式拓展后继节点，即在搜索图中前进一步，从 open 表的最后取元素相当于先拓展深度（已摆放板块的数量），即深度优先。

虽然每次拓展新节点时只向某一个方向走（即只摆放一种板块），但由于这种方法能得到摆放某板块时所有的可能位置，所以即使不拓展其他后继节点，也能找到解，并且速度更快。

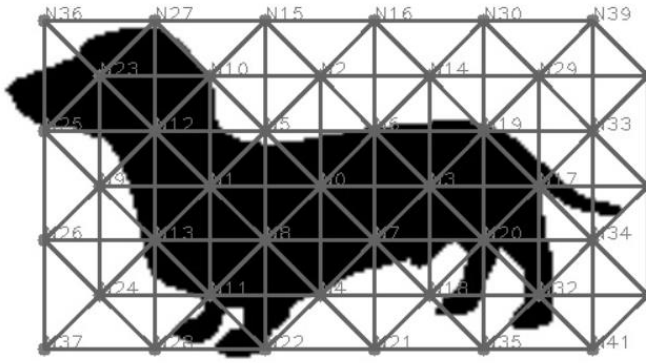
4、 板块数目变化

选做任务一要求板块数目变化时仍能求解，即从七巧板变为 13 巧板、14 巧板等。解决此问题，算法不变，只需要拓展一下程序的接口即可。使基本三角形数目可变，拓展新节点时摆放板块的数目可变，判断目标节点时板块数量可变，都是程序接口上的改变，改变后搜索效率也比较高。

5、 任意图形拼接

选做任务二要求程序能够拼接任意输入图片，如任务描述中所言，首先要将用户输入的图片二值化得到要求拼接的图案，灰度阀值由用户决定。

得到图案后需要将图案规范化，因为图案可能有曲线，无法拼接。规范化的方法与必做任务的解决方案有密切联系。首先求出图案的面积，将其分割成 26 个小三角形，求出小三角形的直角边，作为基础长度。利用这个基础长度，同 1.（1）的方法，将整个图片分割成若干小三角形，如下图所示。



将图案的面积分成 26 个小三角形是为了尽可能准确地拟合图形原来的形状，如果基础长度过长，拟合效果会很差。

得到这些基本三角形后，计算每个三角形覆盖的黑色区域面积，如果面积比例大于某个阀值（如 80%），就保留这个三角形，否则剔除。此外，还要考虑另一种基本三角形的布置方式（斜边斜向），根据面积比较两种方式优劣，选择覆盖面积更大的一种。

在剔除不满足面积条件的基本三角形后，可能出现基本三角形有一半被覆盖的情况，这种图形是无法拼接的，所以在剔除之后还要检查是否有这种情况出现，并复原，用最后剩下的基本三角形得到规则化图形

为了拼接出规则化图形，还无法直接使用必做任务的接口，因为不知道每种板块能摆放

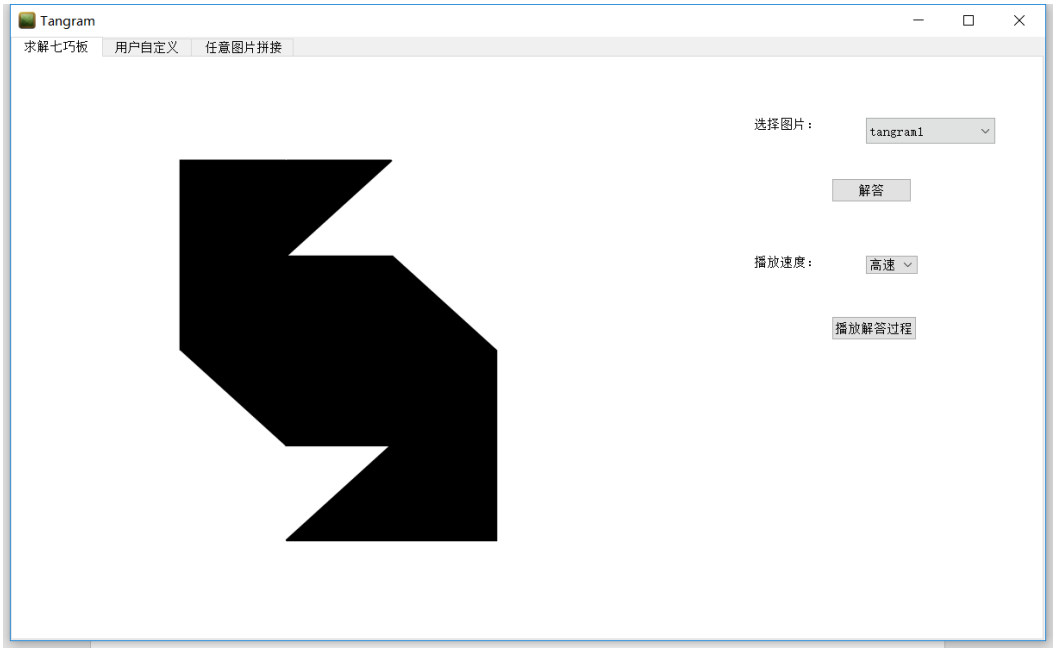
几个。我的方法是试错，已知规则化图形的基本三角形个数，先假设大三角形板块、中三角形板块、平行四边形板块和正方形板块都能放两个，其余的基本三角形用于放小三角形板块。在规则化图形上依次摆放这些板块，如果有一个放不下，就减少该板块数量，增加小三角形板块数量。这样做虽然比较直接，但能保证规则化图形可解。

UI 设计和使用说明

UI 界面使用 pyqt 设计，虽然不方便使用拖放组件的方式设计，但纯编程的设计自由度更高，让我得以实现像用户自定义图形这样的复杂操作。UI 的最高层是一个 QMainWindow 组件，里面有一个标签页，包含三个标签，分别是：“求解七巧板”，代表必做任务一的界面；“用户自定义”，代表必做任务二的界面；“任意图片拼接”，代表选做任务二。

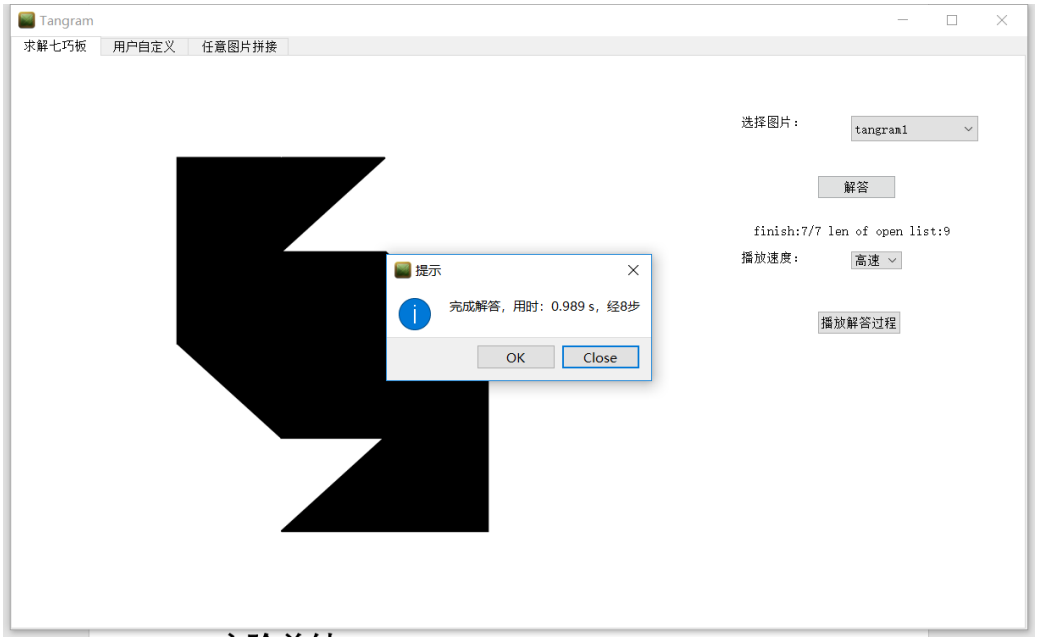
“用户自定义”界面允许用户输入任意板块数量的图形，所以选做任务一的 13 巧板、14 巧板等要求包含在“用户自定义”界面中。

1、“求解七巧板”界面

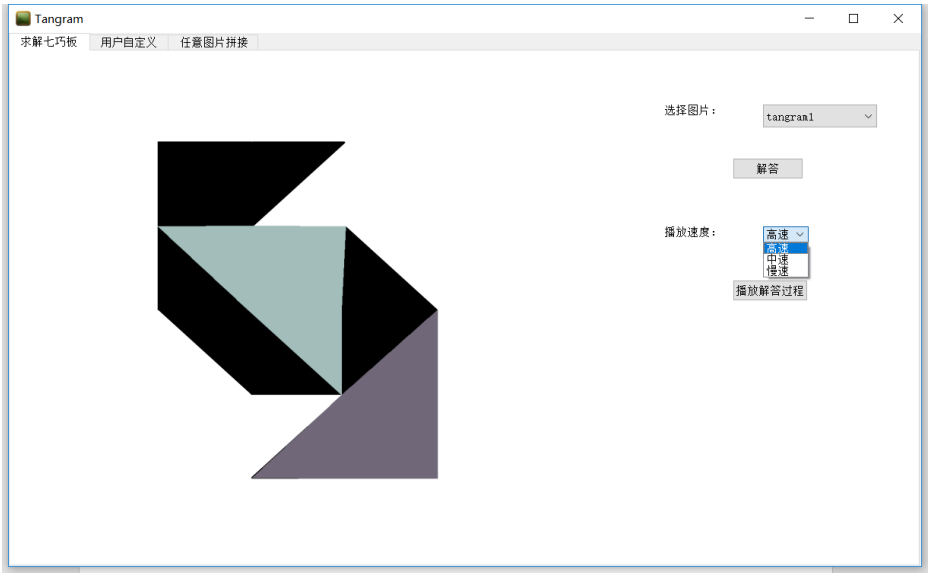


该界面实现了必做任务一的要求，用户可以用第一个下拉框选择要拼接的图片（一共二十个），原始图片会显示在左边的图示区域。

选择好图片后，按解答按钮开始解算，解答按钮下方会有提示信息。解答完成后弹出提示框，显示解答时间和搜索步数。关闭提示框后在图形区域显示求解结果。



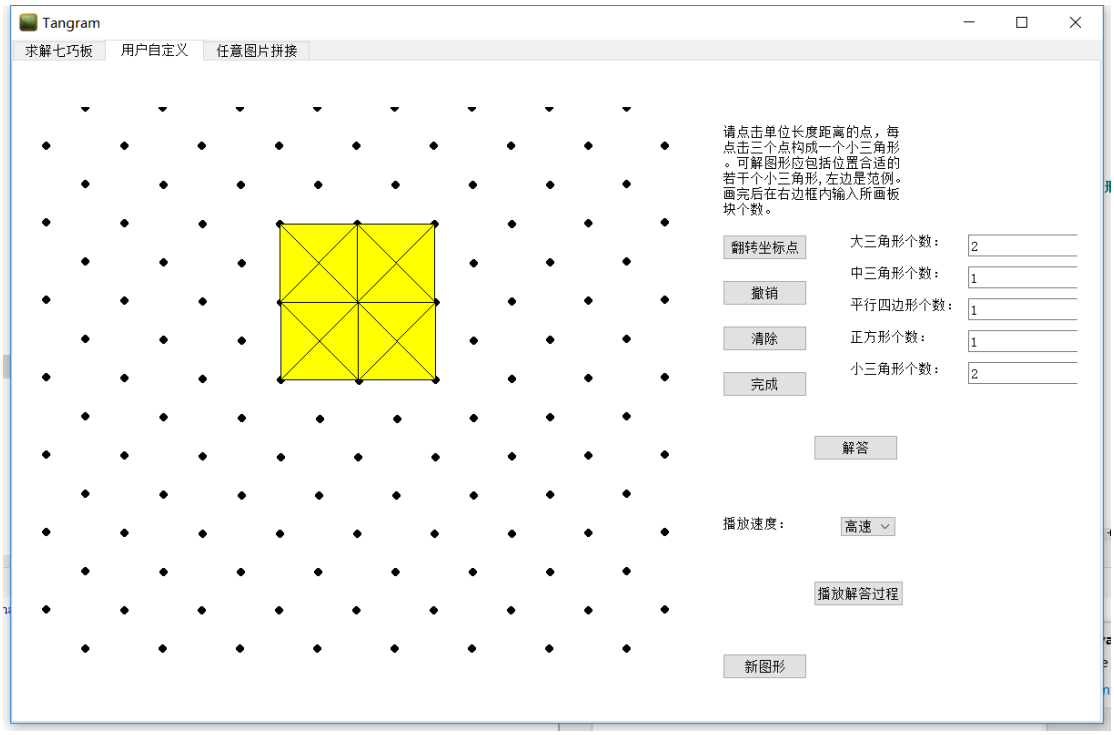
解答完成后，用户可以选择播放速度并播放解答过程。在播放过程中，用户可以重新选择播放速度并按“播放解答过程”按钮来改变速度。用户也可以用选择图片下拉框切换到其他图片，重新开始上述流程。



该标签页的实现在 ui_basic.py 文件中，我将解答按钮、解答过程播放按钮和速度选择框封装成一个类，供三个标签页重复使用。

2、“用户自定义”界面

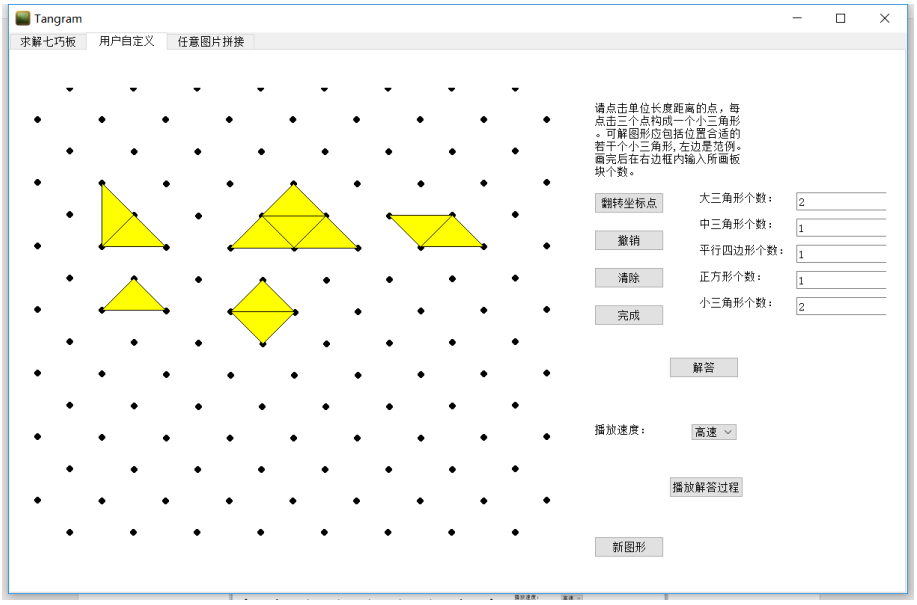
该界面实现了必做任务二的要求，可供用户自己定义图形。

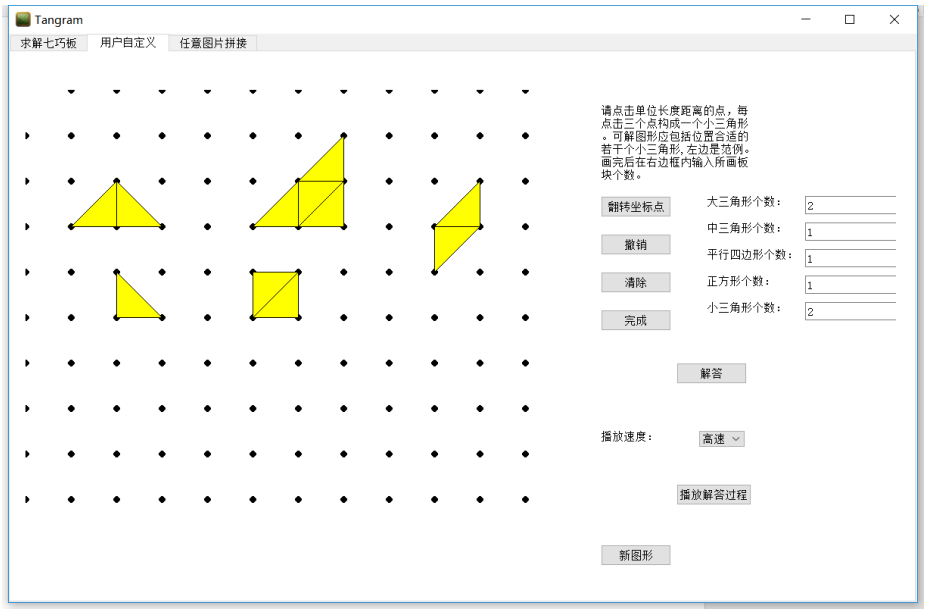


如图所示，用户需要点击单位长度距离的点，每点击三个点构成一个小三角形。可解图形应包括位置合适的若干个小三角形,左边是范例，画完后需要在右边框内输入拼接该图形需要的板块数量。

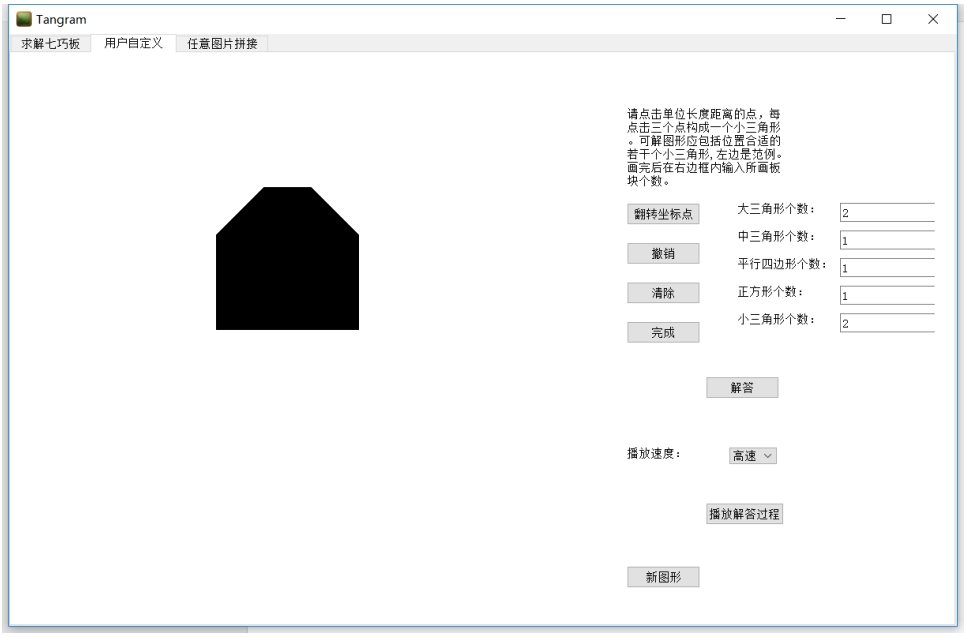
用户在画图过程中可以撤销，清除，也可以翻转坐标点，在基本三角形摆放方式为斜边横向和斜边斜向之间进行翻转。

两种坐标状态下都提供了七巧板绘制范例，用户也可以清除范例，自己绘制，下图是我绘制的在两种坐标状态下五种基本板块的形状。这种方法的缺陷是，在同一张图里，一个板块只能以 90 度角为单位来旋转。





绘制完成后，用户点击“完成”按钮，将所绘三角形变成黑白图形，此时会检查用户输入的板块数目是否与所画图形相符，如果不符，会给出提示。

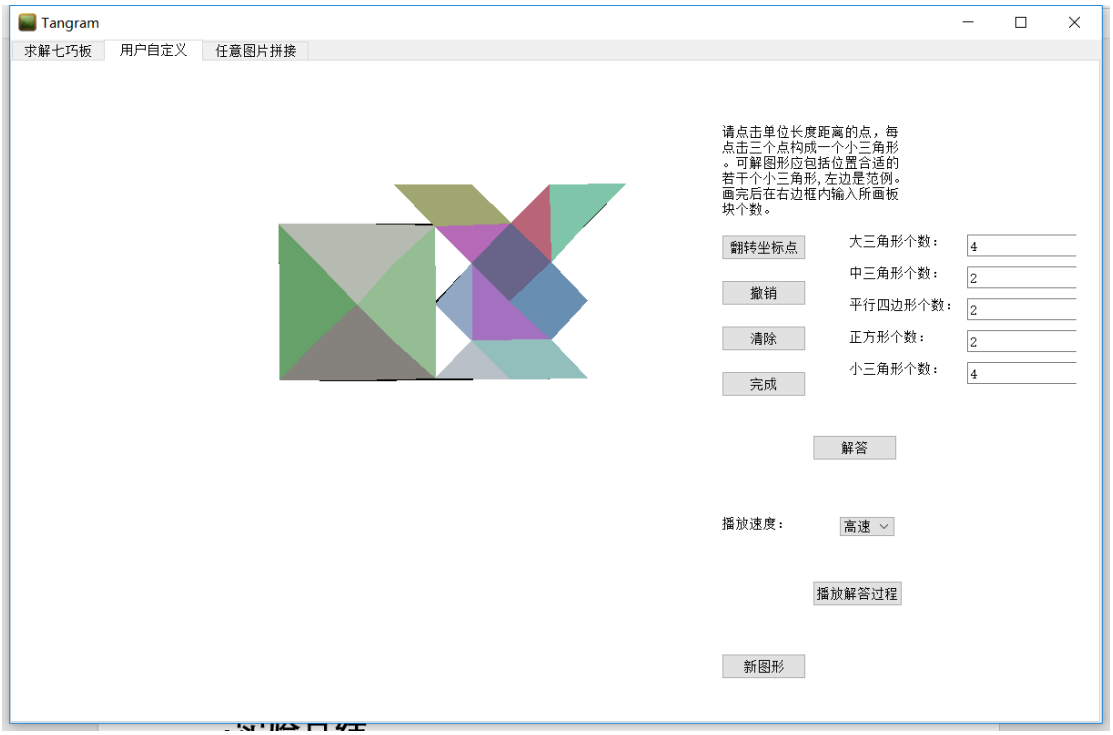


得到黑白图形后，可以解答并播放解答过程，这里的 UI 组件复用了“求解七巧板”界面的相关组件，算法与必做任务的算法相同。

用户可以点击“新图形”按钮重新开始上述流程。

3、选做任务一

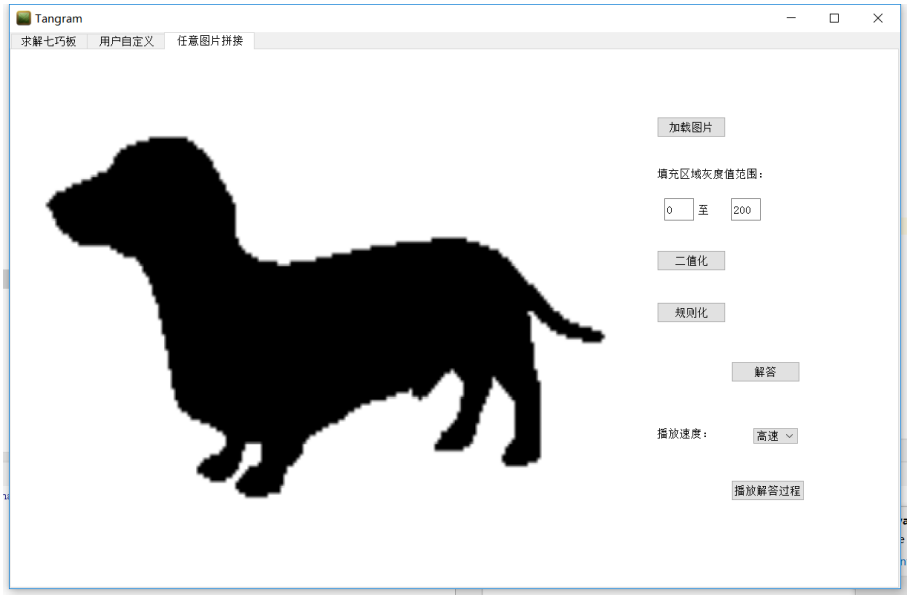
选做任务一要求能够求解 13 巧板，14 巧板，15 巧板等，由于“用户自定义”界面允许用户绘制任务数量的板块，所以该界面已实现了这个任务，便没有新增多余的标签页来体现。下图中是我定义的 14 巧板的解答结果：

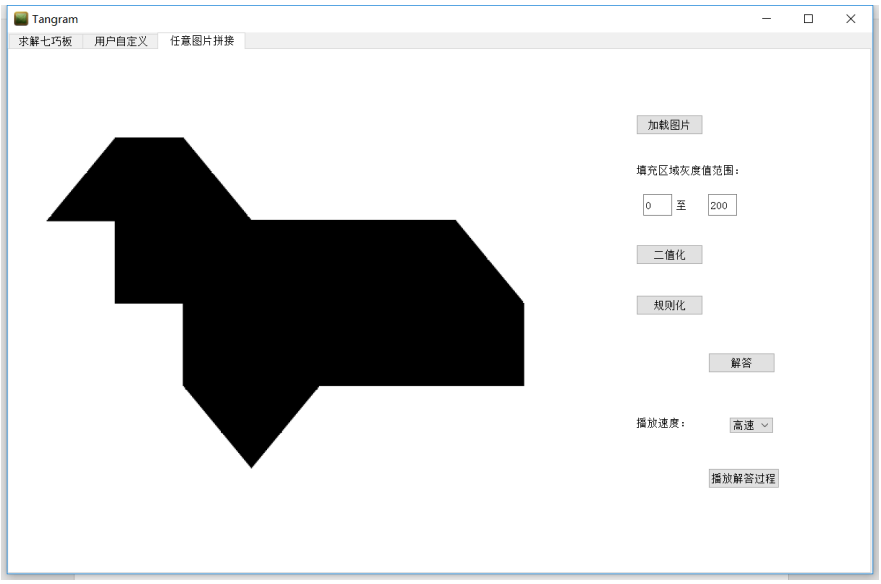


4、“任意图形拼接”界面

该界面要求用户按“加载图片”按钮选择一张本地图片，加载的图片会转化成灰度图，显示在左边的图形显示区域。第二步是图形二值化，用户输入所需拼接图形的灰度值范围，按二值化按钮进行二值化，该过程可重复调整。

二值化成功后，用户按规则化按钮进行图形的规则化，结果显示在图形区域。





规则化后，用户便可以“解答”，“播放解答过程”等，与前两个标签页相同。重新加载图片可以重新开始上述流程。

实验总结

本次大作业我尝试了两种方案，第一种方案基于 openCV 的轮廓点检测功能，普适性强，但搜索时间过长；第二种方案基于图形分割，将图形分割成若干个基本三角形，以这些基本三角形为单位来进行搜索，不能覆盖所有求解范围，但搜索速度很快。我利用第二种方案完成了两个必做任务和两个选做任务，并设计了 UI 界面。其中，选做任务一的实现与必做任务二相结合，给了用户更大的自由度。

搜索时使用深度优先搜索，由于方案二搜索速度快，每次寻找新节点时能考虑某板块可摆放的所有位置，搜索步数少，所以没有使用 A*算法，以免得不偿失。

虽然这次大作业花了我很长时间，尝试了两种方案，但体会到了独立完成一个完整项目的乐趣，收获了成就感，不仅提高了我的编程能力，加深了我对搜索算法的理解，还提高了

我解决问题的能力以及抗压能力，收获满满，也算是付出得到了回报。

代码 README 文件

文件架构

- tangram_solve 文件夹中是方案二的 python 代码和 ui 界面的设计代码；方案二是 ui 可执行程序所使用的方案
- tangrams*olve*contour 文件夹中是方案一的 python 代码，可运行 game_class.py 查看效果
- tangram_app 文件夹中是打包的可执行程序，运行“七巧板解答.exe”。该文件夹内的 tangrams 文件夹是必做任务一的图形文件，不能被改动，否则 exe 文件无法运行。

代码架构

game_class.py

搜索算法核心代码，包括以下自定义类：

- Node*open*list：存储在 open 表中的搜索节点，保存搜索节点的父节点序号，全部的基本三角形信息，各个板块占据的基本三角形序号等。
- Solver：搜索所用的类，保存初始图像的 Graph 对象，open 表，求解轨迹等，有 solve_dfs2 深度优先搜索方法及相关的一系列生成后继节点所用函数。

pic_process.py

图像处理部分的核心代码，包括以下自定义类：

- Graph:存储输入图像，以及输入图像的图形节点，基本小三角形等。定义有获取图形节点、基本小三角形的相关方法。
- Pic feature: 图形节点类，存储图形节点的坐标，以及 8 个邻居图形节点，相关基本三角形，其中相关基本三角形包括以该点为 90 度角顶点的基本三角形和以该点为 45 度角顶点的基本三角形。
- Small tri：基本三角形类，存储图形中的基本三角形的三个顶点序号。

除了这些类之外还有距离、面积测量，图形可视化等函数。

ui.py

UI 界面的顶层 Widget

ui_basic.py

“求解七巧板”标签页的 ui 设计

ui_userDefine.py

“用户自定义”标签页的 ui 设计

ui_general.py

“任意图形拼接”标签页的 ui 设计

编译运行环境

python 3.6, pyqt 5.13.1, opencv-python 3.4.3.18, numpy 1.15.2, Pyinstaller 3.5