**自我评价与收获**：在本次工科创课程中，我主要负责的是图像处理以及算法设计与实现这一部分。因为我之前就对opencv函数库有所接触，所以图像处理这一部分并没有对我们造成很大问题。在算法方面，我设计了一种基于树的迷宫遍历算法，可以根据“节点”（在软件介绍部分详细解释）迅速地遍历迷宫，运行情况良好。在整个课程中，我认为我的贡献还是比较大的。通过本次课程地学习，我也收获了许多，收获了如何与组员齐心协力完成一个一个分划出来的小目标，收获了对opencv函数库的更加熟练的使用能力，以及收获了对树这种数据结构的更深入的理解。

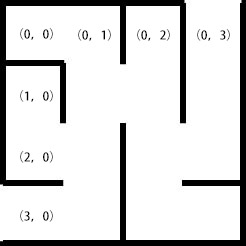
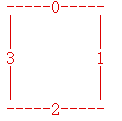
**软件介绍：**

我使用了python语言，利用python中的opencv接口实现该程序，程序介绍如下：

首先实现了两种数据结构：

* Node（）

因为大部分的迷宫中格子形状都是矩形，所以对每一个迷宫，我们都可以将其视为一个一个矩形组成的。每一个Node数据结构就代表着一个矩形，在其中储存的是每一个矩形的基本信息。Node中包含着以下几种矩形的基本信息：



1. self.pos: 蕴含着每一个矩形的位置，如上图所示，在左下角的节点位置为（3，0）.
2. self.state: 蕴含着矩形的各边信息，state是一个4个元素的数组，其中下标所对应的边如右图所示。其中如果对应下标处的边为黑边，那么数组中该元素即为1.例如左下角顶点（3，0），其state = [ 1, 0, 0, 1 ].
3. self.\_l\_son, self.\_r\_son: 指向该节点的下两个可以经过的节点。例如图中的（2，1）节点的下两个可以经过得到节点即为（2，0）和（1，1）.
4. self.to\_end: 其值为真或者假，其中记录的是从迷宫起点通向终点的路径中是否存在该节点。
5. self.len: state中值为1的元素的个数。
6. self.rnode: 判断该节点是否为关键节点。关键节点的意思是如果在该节点小车需要判断是否变向，那么该节点即为关键节点。图中的（3，1），（2，1），（2，0），（1，1），（0，1），（1，2），（2，2），（3，2），（2，3）为关键节点。

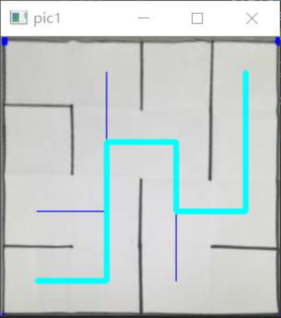
在节点中，我还定义了两种方法：

1. figure\_state: 判断该节点的self.state信息。
2. Is\_a\_rnode: 判断该节点是否为关键节点。

* Tree（）

正如tree的字面意思，该数据结构中保存的即为一颗遍历完整个迷宫的由关键节点组成的树。这棵树中有一个头节点（self.head），代表着迷宫的起始位置的节点。

在这棵树中，我也定义了多个方法，介绍如下：

1. search\_Node: 这个方法的功能即为从self.head开始遍历迷宫，搜索每一个关键节点。该方法的运行截图如下：

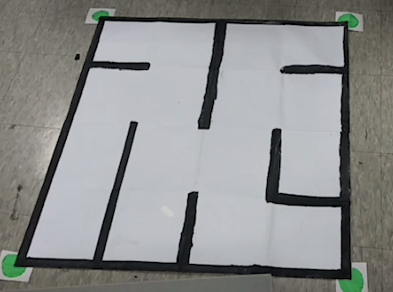
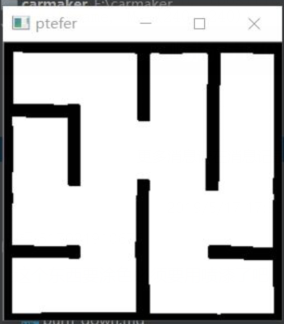
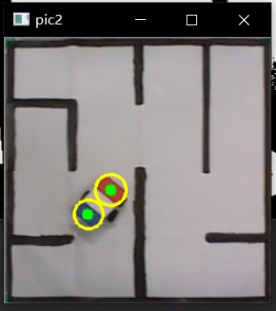
其中左下为迷宫起点，右上为迷宫终点。该方法的实现思路即为对每一个节点，判断其是否为关键节点，如果是关键节点，那么遍历该节点除了进入该节点的边的其他边，判断是否可以由这条边进入下一个关键节点，如果是，那么将该节点的\_l\_son或者\_r\_son指向下一个关键节点，再对下一个关键节点调用该函数。这个函数的实现算法实际上就是一个dfs算法。

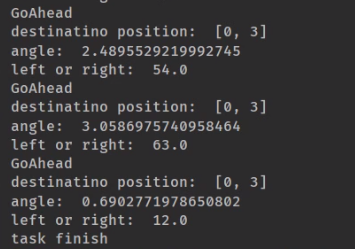
1. findPathToEnd: 该方法在利用search\_Node方法获取了整张地图的关键节点之后调用，并且它实际上是一个包装函数，实际实现功能的函数为pfindPathToEnd。该方法利用pfindPathToEnd返回的一条可以从起点走到终点的路径，在对应的frame中画出该路径。如上图，浅蓝色粗路径即为findPathToEnd画出的路径。

工具函数：

1. pfindPathToEnd: 本函数由findPathToEnd调用，其功能是在有节点信息的基础上找到一条可以走完迷宫的路径。本函数遍历tree中的所有关键节点，是一个递归函数，终止条件是找到了迷宫的终点，返回的上一个调用就知道终点的信息，进而将本次调用中的节点的to\_end信息修改为真。
2. drawline： 一个工具函数用来在frame中绘制遍历节点的路径。如上图中被部分覆盖的细蓝色线条。

下面是主函数的运行 ：

1. 初始化
   1. 初始化蓝牙，连接上小车。
   2. 从摄像头中获取图片，将获取的frame转化为灰度图。
   3. 利用滑动条调整mask值，并用其将frame转化为二值化图片。
2. 调用GetTheMaze()函数获取地图信息。
   1. 识别图片利用的是图片四角的绿色方格，根据cv2.findCountours()获得的轮廓信息判断地图的方位。
3. 将获得的地图转化成含有240\*240个像素点的二值化图片，对这张图片进行两次腐蚀，去掉所得图片中黑色部分的少量白色像素点。以下为所获得的地图：
4. 不断重复过程2，3，获取地图的较好位置直至用户按‘y’确定。
5. 以迷宫的起点构造上述树形结构，运行search\_Node找到关键节点信息，运行findPathToEnd找到迷宫的正确路径。图片为介绍树时使用的图片。
6. 调用GetCarPosition获取小车的位置，并且在相应frame中将小车中的红色与蓝色中心标注出来。
7. 判断小车与下一个关键节点相差的距离，以及小车前进方向是否指向下一个目标节点，利用蓝牙发送对应指令控制小车运动。
   1. angel为小车红色中心，蓝色中心和下一目标节点所夹得夹角。
   2. destination position为目标节点的位置。
   3. left or right为判断小车左转右转的依据，数值为正则需要左转，为负则需要右转。



1. 当小车的中心与关键节点的中心接近到一定距离，调整关键节点到下一个关键节点。
2. 在小车运动的过程中同时不断获取迷宫的信息。
3. 重复过程5-9直至小车运行到终点。输出“task finish”.结束程序。
4. 如果用户中途按下了‘q’键，则程序立即终止。

以下为程序运行总截图：

