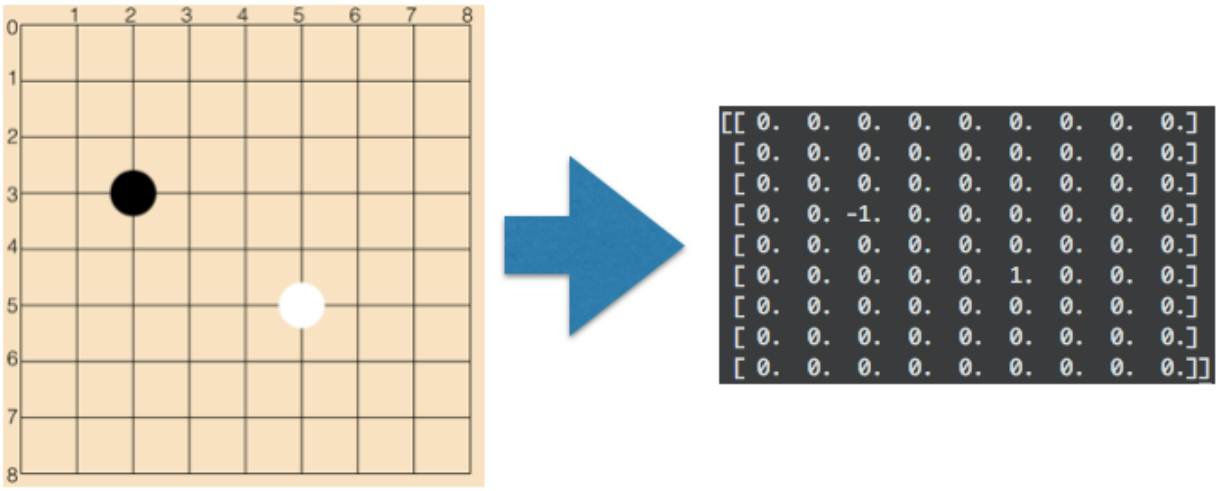
作业说明：

1. 棋盘大小为9*9，作业完成方执白后手，只需走一步，完成相应的要求即可。
2. 示例文件共有7个（train_0.txt, train_00.txt, train_1.txt ~ train_5.txt），其中每个示例文件存放一个围棋残局，存放格式为“行数 列数 颜色”，-1表示黑色，1表示白色，如下图所示的例子，则可存放为“3 2 -1”，“5 5 1”.围棋残局在程序内部以二维矩阵的形式存储，矩阵的元素取-1, 0, 1分别表示黑色，空闲和白色。



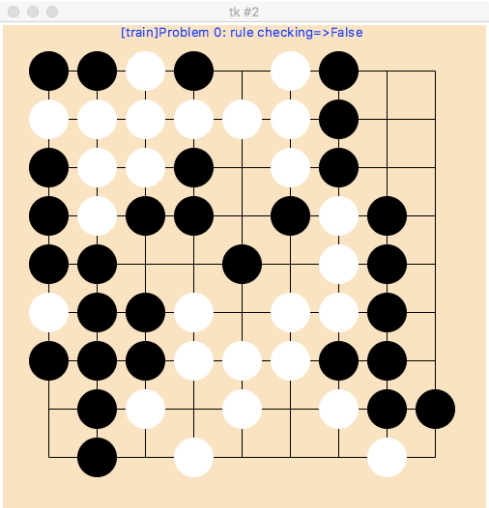
3. train_0.txt和train_00.txt检测程序对围棋规则的判定，train_1.txt ~ train_4.txt的要求是“给出所有可以提子的位置”，train_5.txt的要求是“给出所有可以落子的位置”，围棋的规则见附二。
4. train_0.txt, train_00.txt, train_1.txt ~ train_5.txt是提供给用户自行验证的样例，在作业提交后用来测试成绩的文件为另外的残局文件test_0.txt, test_00.txt, test_1.txt ~ test_5.txt。test_0.txt和test_00.txt检测程序对围棋规则的判定，teat_1.txt ~ test_4.txt的要求是“给出所有可以提子的位置”，train_5.txt的要求是“给出所有可以落子的位置”。

输入输出：

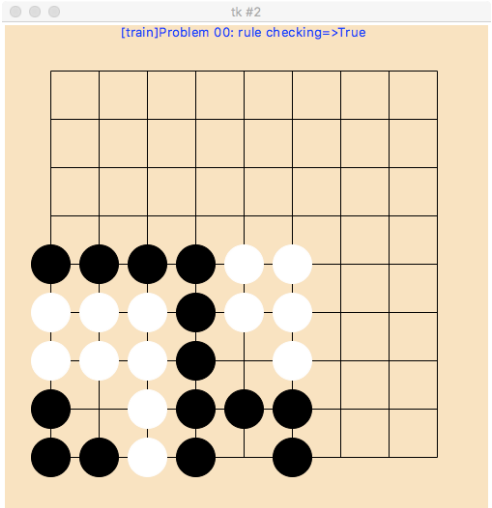
1. 围棋规则1的判定（train/test_0.txt和train/test_00.txt）

输入：残局文件，存放格式为“行数 列数 颜色”，-1表示黑色，1表示白色
输出：True/False（True：该残局否符合规则1，False：该残局不合规）（见【附二】和下图）
要求：无

[train]Problem 0: rule checking:False



[train]Problem 00: rule checking:True



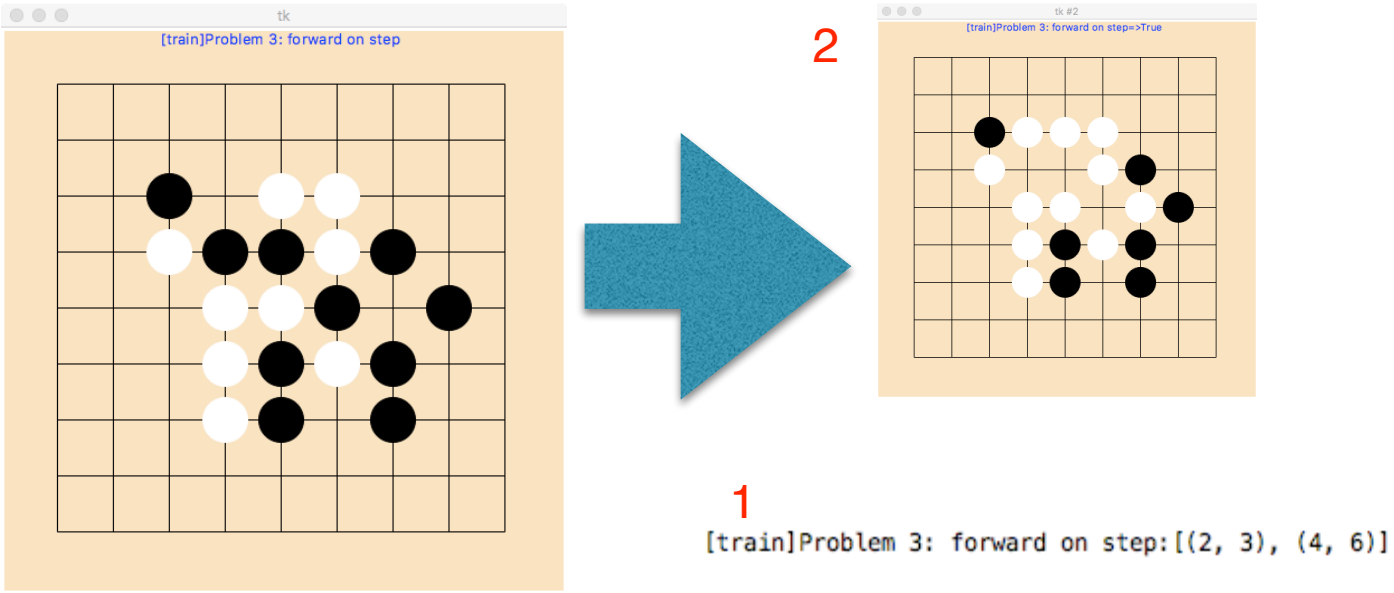
2. 围棋规则2的判定（train/test_1.txt ~ train/test_4.txt）

输入：残局文件，存放格式为“行数 列数 颜色”，-1表示黑色，1表示白色

输出：1. 可以对黑子进行提子的落子位置以(行，列)表示，数字从0开始计数

2. 画出下子并提子后的棋盘结果（所有可行的下子和提子）见附二和下图。

要求：需要进行画图



3. 综合（train/test_5.txt）

输入：残局文件，存放格式为“行数 列数 颜色”，-1表示黑色，1表示白色

输出：所有白子可落子的位置，以(行，列)表示，数字从0开始计数

要求：无

4. 结果保存

上面1～3的所有文本结果，需要保存成文件，格式参考*answer_for_train.txt*。

*注：本项目提供*python2.7*版本的基础代码，但不要求必须用*python*实现，用户可以自行用其他语言实现，只要满足下面评分规则里的要求即可（需要绘制棋盘），文本结果输出参考文件*answer_for_train.txt*。

评分规则：

1. 程序可运行，若运行过程中出错，视为作业失败。

2. 在后台的测试样例上全部运行正确。（作业结束后会公布样例）

附一：Python入门

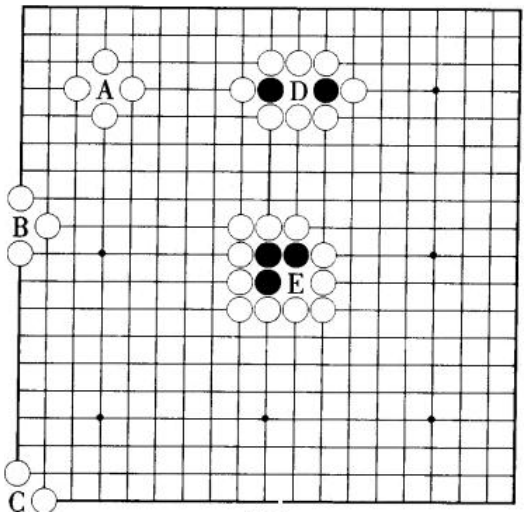
1. 入门资料：简明python教程（http://www.kuqin.com/abyteofpython_cn/）1~10章

2. 运行示例程序需要首先安装Numpy（Python的数值运算库）和Tkinter（Python的UI库），示例程序说明见【附三】。

附二：围棋规则：

1. 无气状态的棋子不能在棋盘上存在，除非规则2.

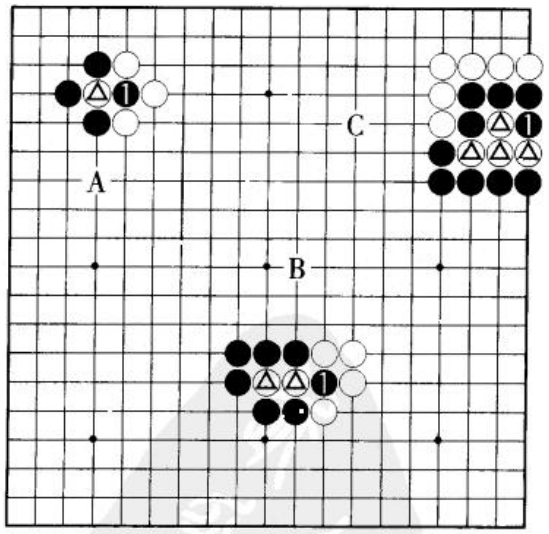
解释：一个棋子在棋盘上，与它直线紧邻的空点是这个棋子的“气”。 棋子直线紧邻的点，如果有同色棋子存在，则它们便相互连接成一个不可分割的整体。它们的气也应一并计算。 棋子直线紧邻的点上，如果有异色棋子存在，这口气就不复存在。如所有的气均为对方所占据，便呈无气状态。把无气之子提出盘外的手段叫“提子”。



例子：下图中A,B,C,D,E位置对于黑棋都是无气状态，因此黑棋不能再上述五个位置下子。

2. 下子后，双方棋子都呈无气状态，应立即提取对方无气之子。

解释：若是在下子之后，自己虽然没气，却也使对方的棋子处于没气状态，即可以提取对方的子，那是允许下子的。



例子：下图三个例子中，黑1下子后，黑1本身处于无气状态，但同时会使相应的白棋处于无气状态，这种情况下黑1允许下子，且可以吃掉处于无气状态的白棋。

附三：

示例程序go_test.py中116行开始是程序的主逻辑，按顺序读入文件，并输出结果。用户可以选择在这个框架下补完go_judege, is_alive，user_step_eat和user_setp_possible四个函数来完成作业或者自行根据输入输出描述写作程序。

go_judge函数：接受一个保存了当前棋盘的go_arr数组，返回一个bool值，bool表示输入的棋盘是否符合【附二】中的规则1。go_judge的思想是根据一颗棋子，通过搜索的方式（is_alive函数的功能）遍历所有与棋子相连的同色棋子，判断整段相连的棋子是否有气，并通过给所有的棋子添加状态变量的方式避免重复搜索。 go_judge中的代码是完整的，用户主要补充is_alive即可。

is_alive函数接受整个棋盘状态变量，棋盘数组，搜索开始的棋子位置，棋子颜色；输出与开始棋子相连的未搜索棋子（同色棋子，is_alive最后一个参数指定待搜索棋子块的颜色）的死活状态，这个操作可以通过深度/广度优先搜索完成。

user_step_eat函数接受棋盘数组，返回白方可以提取黑方子的位置ans和所有ans的位置上放置了白子后的棋盘状态（即提去了黑子的结果）。

user_setp_possible函数接受棋盘数组，返回白方能在棋盘上落子的所有位置。

用户还需要自行书写保存文本结果的代码。