Python 程序设计实验报告

五子棋（阶段二）

院系：人工智能学院

姓名：张运吉

学号：211300063

班级：21 级人工智能学院 AI2 班

邮箱：211300063@smail.nju.edu.cn

时间：2022 年 5 月 1 日

目录

1 实验目的：

五子棋是一种风靡世界的棋类游戏，在世界各处五子棋的表现可能各有不同，但其规则和内核都大致相同。五子棋简单而富有趣味，双方按顺序分别在棋盘上布子，先将 5 个相同棋子横向、纵向或对角线方向连接的一方获胜。本项目最终目标是完成一个可与人类博弈的五子棋程序。阶段二的目标是实现人机对战。

2 实验内容：

阶段一：

1. 学习五子棋规则，自定义棋盘大小（15 x 15 或更大）。

2. 完成可支持两位玩家轮流下棋并使用命令行交互的五子棋程序（可使用坐标描述落子位置）。

3. 在上述基础上添加图形交互界面，支持鼠标点击。

阶段二：

1. 使用搜索策略（如min-max算法、alpha-beta剪枝优化等）实现五子棋对弈智能体。

2. 程序可与玩家进行博弈。

阶段三：

尝试使用机器学习或深度学习方法训练模型来实现更强的五子棋对弈程序，如使用卷积神经网络、强化学习算法等。

3 人机对战：

主要介绍电脑模拟人类下棋的思路以及相关算法。

为了让电脑能够模拟人类下棋，需要电脑能够评估一个棋局的局势，也就是需要一个评估函数来对当前局势给出得分，然后电脑遍历所有空的位置并给这个位置评分，最后选择评分最高的位置就是电脑落子的位置。评估函数决定了电脑的“智商”，评估函数越合理，电脑就越聪明。我在设计的时候实现过两种评估函数，一种是基于同色棋子的数量，另一种则是基于棋形的种类的个数。

3.1 评估函数 1：

对于一个空的位置，判断其四个方向上的赢法数组的情况，五个连续的位置视为一个组。

计算每个组的分数，做累加，就得到总体得分。

当没有棋子时，该位置得到 7 分。

当有一个己方棋子时，该位置得到 35 分。

当有两个己方棋子时，该位置得到 800 分。

当有三个己方棋子时，该位置得到 15000 分。

当有四个己方棋子时，该位置得到 800000 分。

当有一个对方棋子时，该位置得到 15 分。

当有两个对方棋子时，该位置得到 400 分。

当有三个对方棋子时，该位置得到 8000 分。

当有四个对方棋子时，该位置得到 100000 分。

当双方棋子都在棋盘存在时，该位置得 0 分。

在此基础上，遍历整个棋盘，找到整个棋盘上评分最高的位置。

部分代码示例如下：

```
    def pos_score(self, pos):
        cur_score = 0    # 空位的初始得分初始化为0
        x = pos[0]
        y = pos[1]
        black_num = 0
        white_num = 0
        for i in range(5):          # 水平方向
            for j in range(5):
                if [x - j + i, y] in self.black_info:      # self.black_info是一个二维矩阵
，里面存的是黑棋的坐标
                    black_num += 1
                if [x - j + i, y] in self.white_info:
                    white_num += 1
            cur_score = cur_score + self.get_score(black_num, white_num)
            white_num = 0
            black_num = 0
        for i in range(5):          # 竖直方向
            for j in range(5):
                if [x, y - j + i] in self.black_info:
                    black_num += 1
                if [x, y - j + i] in self.white_info:
                    white_num += 1
            cur_score = cur_score + self.get_score(black_num, white_num)
            white_num = 0
            black_num = 0
        for i in range(5):    # 右斜(/)方向
            for j in range(5):
                if [x + j - i, y - j + i] in self.black_info:
                    black_num += 1
                if [x + j - i, y - j + i] in self.white_info:
                    white_num += 1
            cur_score = cur_score + self.get_score(black_num, white_num)
            white_num = 0
            black_num = 0
        for i in range(5):    # 左斜(\)方向
            for j in range(5):
                if [x - j + i, y - j + i] in self.black_info:
                    black_num += 1
                if [x - j + i, y - j + i] in self.white_info:
```

```
                    white_num += 1
            cur_score = cur_score + self.get_score(black_num, white_num)
            white_num = 0
            black_num = 0
        return cur_score
```

经过测试，用这个评估函数做出的机器人还是比较聪明的，如果稍不留心就会败北，但这个评估函数的缺点是只是对棋子个数进行判断，没有对棋型判断.

3.2 评估函数 2：

意识到评估函数 1 的缺点后，我上网搜索了并实现了另一种评估函数。

评估函数 2 是对整个棋盘上形成的棋型判断，弥补了评估函数 1 的缺点。

具体做法是：

(1). 棋盘上有 15 条水平线，15 条竖直线，不考虑长度小于 5 的斜线，有 21 条从左上到右下的斜线，21 条从左下到右上的斜线。获取这些线，把每一条线上的数据转化成字符串存在一个数组中。

(2). 使用字符串自带的统计子串个数的方法获取每一条线上形成的棋形个数，并分别存在两个字典中。

(3). 根据棋盘上黑棋和白棋的棋型统计信息，按照一定规则进行评分（因为想不出来要怎么评分，所以参考了网上的评分规则，但我也修改了其中一些不合理的部分，比如评分不是特别准确的问题）。假设形成该棋局的最后一步是黑棋下的，则最后的评分是（黑棋得分 - 白棋得分），在相同棋型相同个数的情况下，白棋会占优，因为下一步是白棋下。比如黑棋有个冲四，白棋有个冲四，显然白棋占优，因为下一步白棋就能成连五。

按照下面的规则依次匹配，下面设的评分值是可以优化调整的：

前面 9 条为必杀情况，会直接返回评分，

黑棋连 5，评分为 10000，

白棋连 5，评分为 -10000，

黑棋两个冲四可以当成一个活四，

白棋有活四，评分为 -9050，

白棋有冲四，评分为 -9040，

黑棋有活四，评分为 9030，

黑棋有冲四和活三，评分为 9020，

黑棋没有冲四，且白棋有活三，评分为 9010，

黑棋有 2 个活三，且白棋没有活三或眠三，评分为 9000，

下面针对黑棋或白棋的活三，眠三，活二，眠二的个数依次增加分数，评分为（黑棋得分 - 白棋得分） 。

部分代码示例：

```
def get_line1(self, board):
        """获得水平直线(一共15条)"""
        linelst = []

        for i in range(self.chess_len):
            line = ''
            for j in range(self.chess_len):
                line += str(board[i][j])
            linelst.append(line)

        return linelst
def analy_line(self, line, white_type_num, black_type_num):
        """统计每一条线上形成的棋形"""
        white_type_num['LIVE_FIVE'] += line.count('11111')
        black_type_num['LIVE_FIVE'] += line.count('00000')
        white_type_num['LIVE_FOUR'] += line.count('311113')
        black_type_num['LIVE_FOUR'] += line.count('300003')
        white_type_num['CHONG_FOUR'] = white_type_num['CHONG_FOUR'] +
line.count('13111') + line.count('11131') + line.count('11311')+ line.count('311110')
+ line.count('011113')
        black_type_num['CHONG_FOUR'] = black_type_num['CHONG_FOUR'] +
line.count('03000') + line.count('00030')+ line.count('00300') + line.count('300001')
+ line.count('100003')
        white_type_num['LIVE_THREE'] = white_type_num['LIVE_THREE'] +
line.count('311133') + line.count('331113') + line.count('313113') +
line.count('311313')
        black_type_num['LIVE_THREE'] = black_type_num['LIVE_THREE'] +
line.count('300033') + line.count('330003') + line.count('303003') +
line.count('300303')
        white_type_num['SLEEP_THREE'] = white_type_num['SLEEP_THREE']   +
line.count('0311130') + line.count('013113') + line.count('313110') +
line.count('011313') + line.count('311310')
        black_type_num['SLEEP_THREE'] = black_type_num['SLEEP_THREE'] +
line.count('1300031') + line.count('103003') + line.count('303001') +
line.count('100303') + line.count('300301')
        white_type_num['LIVE_TWO'] = white_type_num['LIVE_TWO'] +
line.count('3113') + line.count('31313') + line.count('313313')
        black_type_num['LIVE_TWO'] = black_type_num['LIVE_TWO'] +
line.count('3003') + line.count('30303') + line.count('303303')
        white_type_num['SLEEP_TWO'] = white_type_num['SLEEP_TWO'] +
line.count('0113') + line.count('3110')+ line.count('01313') + line.count('31310')
        black_type_num['SLEEP_TWO'] = black_type_num['SLEEP_TWO'] +
line.count('1003') + line.count('3001') + line.count('10303') + line.count('30301')
```

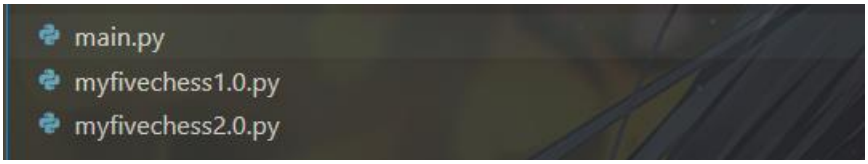
3.3 min-max 算法、alpha-beta 剪枝优化

min-max算法：在评估函数 2 中，获得了玩家 和 AI 的当前优势值，将这两个数字做差，返回 max-min 的值，这就是该种走法的分数，值越大对AI更有利，值越小对玩家更有利。

alpha-beta剪枝优化：假设AI走的是max层，玩家走的是min层，初始化α值为负无穷大(-∞)，β值初始化为正无穷大(+∞)，当一个 min 层节点的 α值 ≤β值时，剪掉该节点的所有未搜索子节点，当一个 max 层节点的 α值 ≥β值时 ，剪掉该节点的所有未搜索子节点。其中 α 值是该层节点当前最有利的评分， β 值是父节点当前的 α 值。

4. 代码运行：

一共有三个文件，main.py是一阶段的实现双人对战的五子棋，myfivechess1.0.py是用评估函数 1 实现的人机对战程序，myfivechss2.0 是用评估函数 2 结合min-max算法、alpha-beta剪枝实现的人机对战程序。可以分别运行这三个文件。



运行效果图：