Python 程序设计实验报告

五子棋（阶段三）

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 30 日

目录

1 选题描述：

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。

本次阶段三我的主要工作是利用阶段二设计的评估函数结合**min-max算法**和**alpha-beta剪枝优化**实现更加智能的AI.

2 设计方案：

2.1 Minimax 算法思路

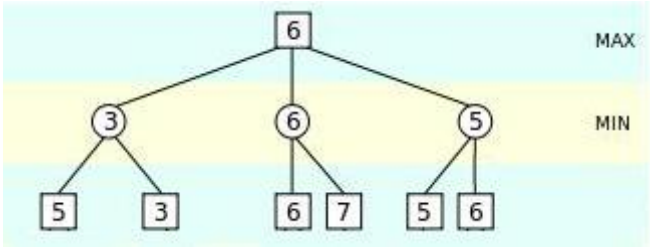
回顾阶段二我的AI实现：AI 先获取当前所有可以下的位置（就是棋盘上的空格），然后每次在其中一个位置下子，根据棋型评估函数获取一个分数，所有位置都下过一遍后，从中获取评分最高的位置。

相当于AI只思考了当前局面的最优解，但当前的最优解不一定是对AI最有利的位置，就像一些经验丰富的人类玩家一样，他们往往会考虑到后面几步的情况，然后综合进行判断。**Min-max算法**就是基于这样的思想实现的。

Min-max算法常用于棋类等由两方较量的游戏和程序。该算法是一个零总和算法，即一方要在可选的选项中选择将其优势最大化的选择，另一方则选择令对手优势最小化的一个。

对于五子棋而言，我们不妨假设AI走的是max层，即轮到AI下棋时，总是选择对AI最有利的位置，玩家走的是min层，即轮到玩家下棋时，总是选择评分最低（对AI最不利）的位置。

如下图所示，假设一个两层的博弈树，最上面一层是树的根节点，这里MAX表示会选取下一层子节点中评分最高的。第二层的MIN表示会选取下一层子节点中评分最低的。第三层是叶子节点，只需要计算评分。



极大极小值搜索是一个深度优先的算法，当第二层第一个节点的子节点都计算好评分后，因为这层是MIN层，会选取子节点中最低的评分作为这个节点的评分，就是 3。依次类推，第二层第二个节点评分为 6，第三个节点为 5。当

第二层节点都获取到评分后，因为第一层是MAX层，会选取子节点中最高的评分作为这个节点的评分，就是第二层第二个节点的评分 6，这个节点所代表的下棋位置对于AI来说就是最有利的。最后算法返回评分 6 和第二层第二个节点的下棋位置。

2.2 alpha-beta 剪枝优化

极大极小值搜索算法的缺点就是当博弈树的层数变大时，需要搜索的节点数目会指数级增长。比如上面每一层的节点为 50 时，六层博弈树的节点就是 50 的 6 次方，运算时间会非常漫长。

在上面的例子中，我们会计算所有叶子节点的评分，但这个不是必要的。

Alpha-Beta剪枝就是用来将搜索树中不需要搜索的分支裁剪掉，以提高运算速度。基本的原理是：

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，根节点因为是MAX层，所以 β值 初始化为正无穷大(+∞)。

初始化节点的 α 值，如果是MAX层，初始化α值为负无穷大(-∞)，这样子节点的评分肯定比这个值大。如果是MIN层，初始化α值为正无穷大(+∞)，这样子节点的评分肯定比这个值小。

2.3 启发式评估

影响alpha beta剪枝效率的关键，是要让评分高的位置更早的被搜索到，这样可以更快的进行剪枝。

通过启发式评估后，我们可以先预估子节点的评分，在生成博弈树时，通过调用子节点的前后顺序，就可以更快的进行剪枝。

要实现这一点，就需要对每一个可以下的位置进行评分的预估，让预估分高的位置排在前面。目前采用的预估评分方法是对于一个空的位置，分别下白棋或黑棋，获取这个点四个方向能够形成的棋型，然后打分。

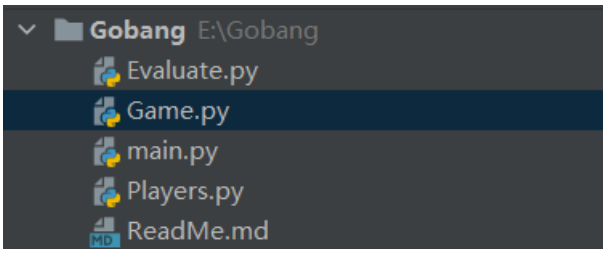
2.4 部分相关代码

```
    def alpha_beta(self, board, turn, depth, alpha, beta):
        # 如果深度小于等于0，直接返回
        if depth <= 0:
            score = self.min_max(board, turn)
            return score
        score = self.min_max(board, turn)
        # 产生必胜情况，直接返回
        if abs(score) >= 9999 and depth < self.max_depth:
            return score
        moves = self.genmove(board)
        bestmove = None
```

```
            for score, row, col in moves:
                board[row][col] = turn
                nturn = (turn + 1) % 2
                score = - self.alpha_beta(board, nturn, depth - 1, -beta, -alpha)
                board[row][col] = 3
                if score > alpha:
                    alpha = score
                    bestmove = (row, col)
                    if alpha >= beta:
                        break
            # 保存得分最好的走法
            if depth == self.maxdepth and bestmove:
                self.move = bestmove
            return alpha

        def get_bestmove(self, board, turn, depth=4):
            self.max_depth = depth
            self.move = None
            score = self.alpha_beta(board, turn, depth=4, alpha=-0x7fffffff,
beta=0x7fffffff)
            if abs(score) > 8000:
                self.maxdepth = depth
                score = self.alpha_beta(board, turn, depth=4, alpha=-0x7fffffff,
beta=0x7fffffff)
            return self.move[0], self.move[1]
```

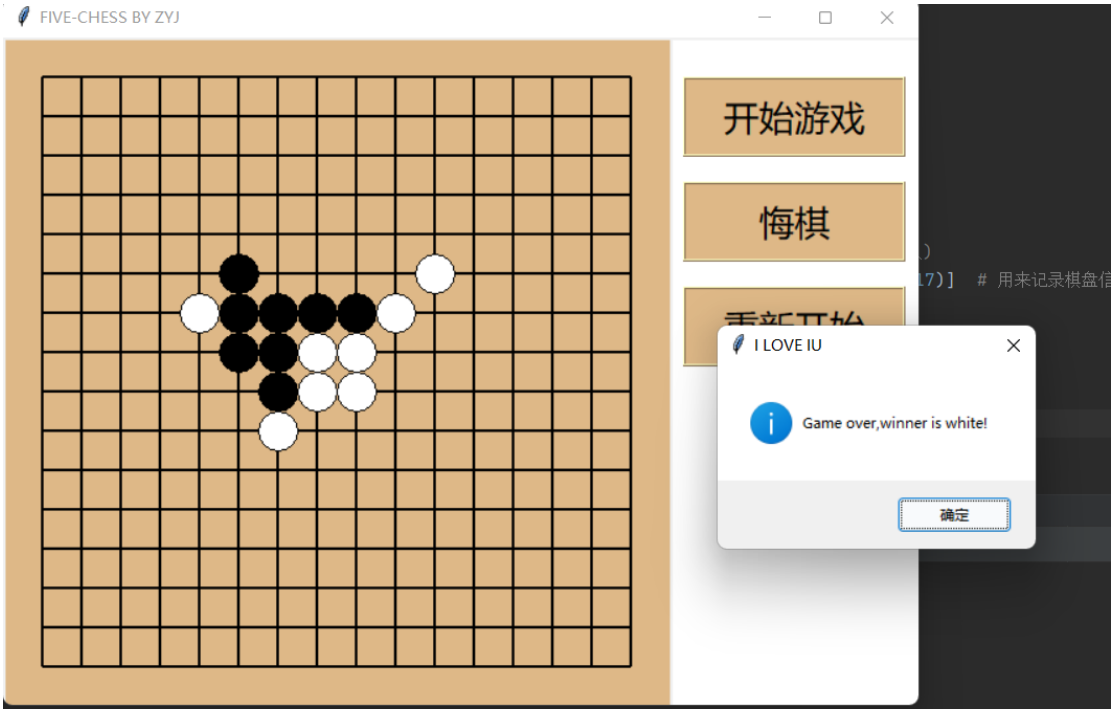
3 代码模块的功能划分与描述：



整个项目一共分成四个文件，Evaluate.py里面定义了一个Evaluate类，封装了二阶段设计的评估函数；Players.py里面定义了Human类和AI类，封装了玩家下棋和AI下棋的一些操作；Game.py里面定义了Game类，实现了整个五子棋的游戏逻辑；main.py是提供的最终接口，在里面有一些常量，可以通过修改这些常量来进行一些个性化的操作（比如定义棋盘大小等）。

4. 实现效果：

直接运行main.py文件就可实现人机对弈。



经过测试，发现AI可以打败一些新手或经验不足的玩家，但不能战胜一些经验丰富的玩家。但相比二阶段的AI，胜率明显提高。