目录

[一、 作业要求 2](#_Toc127632990)

[二、环境介绍 2](#_Toc127632991)

[二、 实现过程 2](#_Toc127632992)

[三、 重要优化 3](#_Toc127632993)

[四、 核心算法 3](#_Toc127632994)

[五、 测试过程 3](#_Toc127632995)

1. 作业要求

根据当前棋盘状态计算返回下一步落子的接口next，实现五子棋AI，同学需要实现这个next函数，我们将next函数的AI嵌入到我们事先做好的对弈框架，随机选择N个同学的AI进行对弈，最后根据对弈胜率给分。

要求：

1. python语言实现

2. 每步落子计算时间不超过10秒，超时扣分，累积超时太多自动判负

二、环境介绍

操作系统(Operating System)

——Microsoft Windows 10

Python版本

——Python 3.6.4

测试工具

——PyCharm 终端下命令行测试

1. 实现过程

**评估方法介绍：**

查阅资料可知五子棋有7种有效棋型（连五，活四，冲四，活三，眠三，活二，眠二），我们可以创建黑棋和白棋两个数组，记录棋盘上黑棋和白棋分别形成的所有棋型的个数，然后按照一定的规则进行评分。

这里用的方法是，对整个棋盘进行遍历，对于每一个白棋或黑棋，以它为中心，记录符合的棋型个数。

**具体实现方式如下：**

1. 遍历棋盘上的每个点，如果是黑棋或白旗，则对这个点所在四个方向形成的四条线分别进行评估。四个方向即水平，竖直，两个斜线( \ , / ），四个方向依次按照从左到右， 从上到下，从左上到右下，从左下到右上来检测。
2. 对于具体的一条线，如下图，已选取点为中心，取该方向上前面四个点，后面四个点，组成一个长度为9的数组。然后找下和中心点相连的同色棋子有几个，比如下图，相连的白色棋子有3个，根据相连棋子的个数再分别进行判断，最后得出这行属于上面说的哪一种棋型。具体判断可以看代码中的 analysisLine 函数。
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。

**AI介绍：**

确定评估函数后，轮到AI下棋时，就要针对当前的棋局，找到一个最有利的位置来下。AI会尝试在每个空点下棋，形成一个新的棋局，然后用评估函数来获取这个棋局时的评分。从中选取评分最高的位置。

AI 获取最有利位置的逻辑：

* 遍历棋盘上的每一个空点：

在这个空点下棋，获取新的棋局的评分

如果是更高的评分，则保存该位置

将这个位置恢复为空点

* 获得最高评分的位置

1. 重要优化
2. 采用深度学习的最大最小值**MaxMin算法**来提高胜率。

极大极小值搜索算法就是在树的每一层搜索时，根据下面的策略：

* AI下棋的层，称为MAX层，这一层 AI 会选取子节点中评分最高的位置
* 玩家下棋的层，称为MIN层，这一层玩家会选取子节点中评分最低的位置

1. 采用**Alpha-Beta剪枝**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。

* 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(+∞)。

1. 运用**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，我们可以先预估部分节点的评分，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。

1. 核心算法
2. MaxMin算法及Alpha-Beta剪枝函数：find（）、maxmin（）
3. 评估函数：getScore（），对黑棋和白棋进行评分。
4. 启发式评估算法函数：evaluatePointScore（），获取己方或对方棋子时形成的棋局的评分，然后将评分较高的位置加入单独的列表中。
5. 测试过程
6. 将编写好的代码文件发进测试框架中ais文件下，并命名为20174958.py。
7. 将runChess.py代码中的对弈双方改成20174958和20200430（baseline.py）。
8. 运行runChess.py。
9. 得到测试结果，运行中满足每步落子不超过10s要求。

