四子棋 AI 实验报告

张锦黔 201604014013

一、问题分析

四子棋游戏：游戏双方持不同颜色的棋子，分先后手依次落子，规则为：在 M 行 N 列的棋盘中，起手每次只能在每一列当前最底部落子，若某一列已经落满，则不能再该列中落子。当某一方在横、纵、斜四个方向中任意一个方向上先使自己的棋子连成四个或以上则取胜，若棋盘落满时双方都没能达成目标则为平局。此问题是经典的棋类博弈问题，使用 alpha-beta 剪枝算法解决。

二、算法设计

2.1 落子策略：极小极大算法

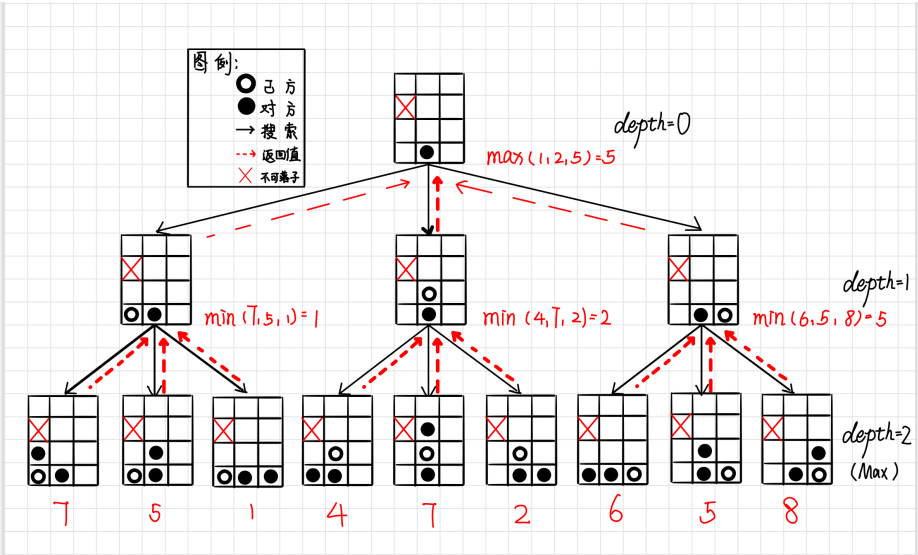
选用博弈论中常见的极小极大（MinMax）算法进行决策选择，具体实现方案为递归式，流程如下：

（1）对于每一个待落子的 M 行 N 列棋盘（搜索深度初始为 0），遍历其 N 个候选落子点（搜索深度+1），得到最终返回值作为其评估值，选择估值最大的位置作为最终落子点；

（2）每一深度的候选落子点同样搜索自己落子后棋盘上的 N 个候选落子点（搜索深度+1），得到 N 个返回值。若此深度属于己方落子，则以下一深度给出的 N 个返回值中的最小值作为本落子点评估值；若属于对方落子，则以 N 个返回值中的最大值作为评估值。

（3）当搜索深度达到预设的最大值，或游戏结束（一方获胜或棋盘已满）时，不再搜索下一深度，按照 2.2 中的方案评估当前整个棋盘得出评价值并返回。

2.2 博弈树示例



以列数为 3、行数为 4 的棋盘为示例，设最大搜索深度为 2，则一个博弈树如上图所示。最终，程序决策委在棋盘右下角落子。

2.3 局面估值方案：模式匹配与启发式算法

在 2.1 中构建了一个 MinMax 博弈树，由于计算资源不是无限的，因此不能搜索整个博弈树到任意深度，而是以某一深度作为上限，这时后续深度的发展情况是未知的，因此需要以已知局面进行启发，计算后续发展情况的近似值，以替代真正的搜索返回值。

本游戏中，采取的估值方案为利用模式匹配识别棋盘上组成的连子数目和后续可利用空间。根据四子棋的游戏规则，并经过多次人机游戏总结经验，对下列情形从高到低赋予不同的分值：

（1）“成四”：一条线上连续 4 个棋子，获胜。

（2）“活三”：一条线上连续 3 个棋子，且两端均为空，最容易发展为“成四”。其难易度随着两端空位到自身所在列的列顶的距离增大而降低。一般认为距离在 0-2 格时有较高价值，3-4 格有一定价值，再高则价值偏低。

（3）“单三”：一条线上连续 3 个棋子，且有一端为空，有机会发展为“成四”。其价值高低与“活三”同理。

（4）“活二”：一条线上连续 2 个棋子，容易发展为“活三”或“单三”。其价值高低与“活三”同理。

（5）“单二”：一条线上连续 2 个棋子，有机会发展为“单三”。其价值高低与“活三”同理。

（6）“单子”：单独 1 个棋子，且至少一个方向上的相邻位置为空，有机会发展为“单二”或“活二”。其价值高低与“活三”同理。

当考察对象为己方棋子时，得到的估值为“进攻性”的估值，当考察对象为对方棋子时，得到“防守性”的估值。对于每一个需要估值的局面，分别从进攻和防守两个角度取其横、纵、斜四个方向上估值之和，所得结果进行加权（己方为正值，对方为负值）即可得到**兼顾进攻与防守**的综合估值。

估值的准确性依赖于对各个基础模式的赋值，越丰富的模式库和越适当的赋值就能越好地给出局面后续演化到终点时的近似值，实现越好的决策启发。

2.4 优化 1：估值衰减

若在不同深度下，按 2.2 方案计算得到了同样的估值，选取深度更浅的走法总是更为有利的（前提是模式赋值具备足够合理性）。因此应当在 2.2 方案的基

础上使估值随深度衰减，以程序作出更有利的决策。

一个较为简单的衰减算法是线性衰减。记棋局p在 2.2 方案下的原估值为$A(p)$，记$A(p,d)$为同一棋局p在搜索深度为d时得到的估值，则

$$A(p,d)=A(p)\times(1-\lambda\cdot\frac{d}{d_{max}})$$

其中衰减常数$\lambda\in(0,1)$。该公式下，原估值$A(p)$的衰减取决于当前搜索深度与深度上限的比值。

2.5 优化 2：alpha-beta 减枝搜索

在整个博弈树中，一些分支可以证明没有搜索的必要。alpha-beta 减枝给出了以下两种情形：

情形一（alpha 剪枝）：当前深度下为己方落子，会从下一深度选择返回值最大的分支，而上一深度必选择当前深度返回值最小的分支。一旦遇到下一深度的返回值出现了大于上一深度已知最小值的情形，上一深度必然放弃当前分支。

情形二（beta 剪枝）：当前深度下为对方落子，会从下一深度选择返回值最小的分支，而上一深度必选择当前深度返回值最大的分支。一旦遇到下一深度的返回值出现了小于上一深度已知最大值的情形，上一深度必然放弃当前分支。

三、程序设计

3.1 模式匹配、局面评分及优化

（1）所列举的模式及其分值

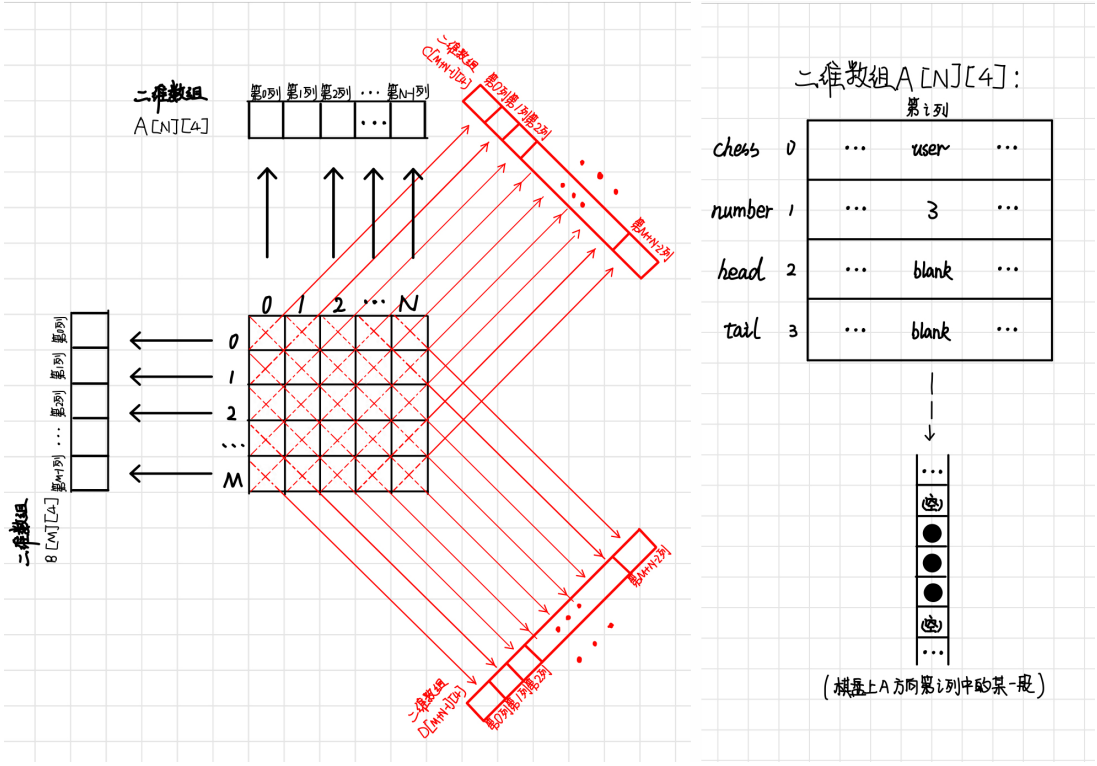
模式	分/个
成四	100000000
活三	100000
单三	10000
活二	100
单二	10
单子	1

（2）优化的打分算法

一个很容易想到的打分方法是分横、纵、左斜、右斜四个方向分别遍历棋盘，识别出所有模式并将所有分值按评分规则汇总。为了加快程序运行的速度，我设

计了一个方法，用四个二维数组分别记录局面分值，仅需对棋盘遍历一次即可完成四个方向分值的汇总。

该快速汇总方法的原理如下图所示：



从棋盘左上角开始遍历，每到一个格子，得到该处的棋子值，棋子值有 :空、用户、机器、边界（含不可落子点）四种值。根据该值更新代表横竖斜四个方向的四个数组数据。

对于上述四个二维数组中的任意一个，其第一个维度代表列的编号，第二个维度记录四个量：当前棋子值、当前棋子连续次数、前一棋子值、后一棋子值。

每次走到新的格子，会根据该格坐标找到数组中对应的列编号，将新格子的值与数组中的当前棋子值对比，若相同，则棋子连续次数自增 1；若不同，则将新值记录进“后一棋子值”，便得到了上一个连续棋子串的种类、数目和头尾空间情况信息，从而进行一次打分并记录。

之后重新初始化数组中的该列，将“前一棋子”值记录为原“当前棋子”的值，“当前棋子”记录新格中棋子值，连续次数置为 1，“后一棋子”根据下一格传来的新值刷新。

函数还会根据棋子坐标进行一些判断，来处理不可落子和棋盘边缘的情况。

3.3 类的声明及主要成员

situation 类（局面类），包含一个记录棋盘的二维数组 map[ ][ ]（非 const），

记录棋盘实时信息；一些变量如整型值 M、N、nx、ny、depth 等，辅助记录棋盘的边界、不可落子点、当前搜索深度等信息。

3.4 估值函数

返回一个 double 值作为当前局面的估值，具体算法参照 3.1.(2)中所述执行。

3.5 递归函数

用递归的方法建立博弈树。主要流程如下：

（1）落子：

被调用时得到传入参数 Y（上层指定的落子点的 y 坐标）和 bestbro（上层已知最优候选值），根据 Y 找到 map[ ][ ]中的对应空位进行落子，并使 depth 自增 1。

（2）判断：

对当前局面进行判断，若游戏结束（一方获胜或平局）或深度 depth 达到上限，执行（3），否则执行（4）。

（3）评分：

运行估值函数，得出当前局面估值并记录进准备作为返回值的变量中。

（4）模拟：

从 0 到（N-1）依次作为 Y 参数调用递归函数实现下一层搜索，每一次调用之前刷新当前最优候选值作为传给下一层的 bestbro 参数。完成所有模拟后将最优候选值记录进准备作为返回值的变量中。

（5）回滚：

将第（1）步在棋盘内的落子移除，并使 depth 自减 1。

（6）返回最优候选值

3.6 最终决策函数

上述递归函数实际上是从 depth=1 层开始构建博弈树，而博弈树的根节点（depth=0）由此函数建立并完成收集所有后代节点返回值、做出最终决策的工作。其最终返回给 getPoint 函数一个值作为其 y 坐标值，而 x 值由 getPoint 函数中调用 top 数组得到。

3.7 其它辅助函数

包含判断游戏结束、具体模式评分等。

四、实验结果及分析

4.1 实验结果及分析

改变最大深度的值，通过模拟棋局的结果数据发现，在对手相同的情况下，最大搜索深度越小，胜率越低；增大最大搜索深度，程序落子时间变长，但胜率不断提高，能战胜难度更高的对手。当最大搜索深度取 6 时，本 AI 达到的棋力水平大约在 TestCases 中 55~60 之间。