中山大学数据科学与计算机学院
程序设计与数据结构综合实践 II
“五子棋”实验报告

（2018-2019 学年秋季学期）

学　　号：16337113
姓　　名：劳马东
教学班级：教务 2 班
专　　业：超算

1 实验题目

实现一个双人五子棋程序，每一次下棋操作都会输出棋谱 chessboard.txt，0 表示未下子的位置，1 表示先手（即黑棋下的位置），2 表示后手（即白棋下的位置）。chessboard.txt 作为 ai 程序的输入，用来得到 ai 下一步将会下在什么位置，然后下在棋盘对应位置。棋盘大小为 15×15，有禁手规则。

2 实验设计

2.1 游戏主循环

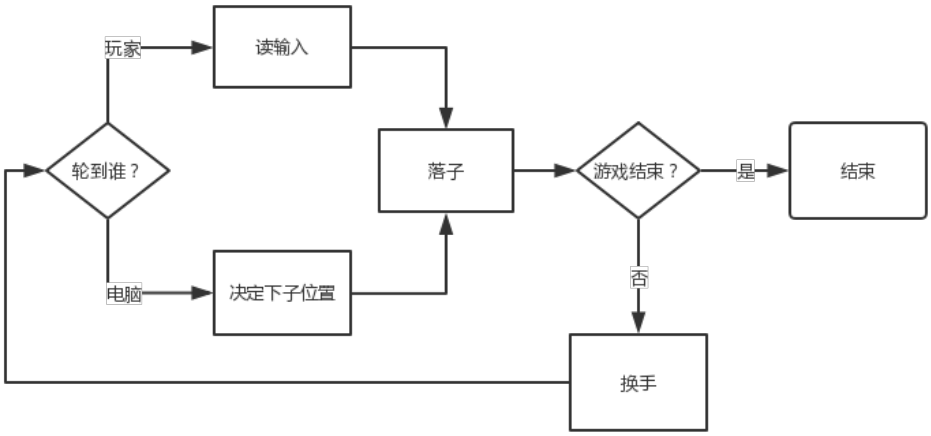

图 1: 游戏主循环

2.2 类设计

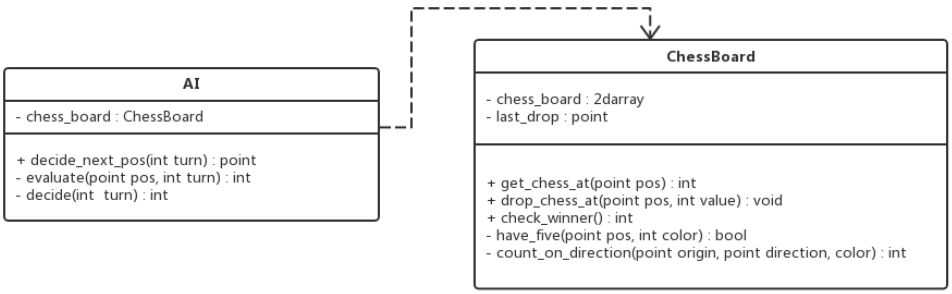

图 2: 主要类关系

1、 ChessBoard 类

(1) 成员变量 *chess_board* 是一个二维数组（初始化为全 0），保存棋盘的落子信息。

(2) *last_drop* 保存上一次落子的位置，用于在输出棋盘时高亮输出落子位置。

(3) *get* 和 *drop* 操作分别获取和设置棋盘特定坐标处的值。

(4) *check_winner* 利用 *have_five* 函数检查是否有五个同色棋子，即是否有一方胜出。

(5) *count_on_direction* 函数统计以 *origin* 为原点，*direction* 向量方向同色的棋子数目。

2、 AI 类

(1) *chess_board* 成员变量是一个 ChessBoard 实例，用于决定落子位置。

(2) *decide_next_pos* 函数利用 *evaluate* 函数计算每个空余位置的评分，并返回对于 *turn* 方来说最优的落子位置。

(3) *evaluate* 函数根据某种原则计算假如在 *pos* 位置落子对 *turn* 方有多大好处。

2.3 关键代码

1、 决定落子位置的估分函数

棋盘的每个位置有三种状态，假设分别为 0（己方落子）,1（空）,2（敌方落子）。那么，当要找最优落子位置时，需检验在空位置落子后以该位置为起点的射线上的五个点的状态。显然，起点的状态是固定的（当前方的棋子颜色），其余四个位置的状态有如下 $3 \times 3 \times 3 \times 3 = 81$ 种 9 类，给每类情况赋予合理的分数，在遍历时累加，就能找出得分最高的落子位置。

预落子	1	2	3	4	取值个数
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	0	2	1
0	0	0	1	*	3
0	0	0	2	*	3
0	0	1	*	*	9
0	0	2	*	*	9
0	1	*	*	*	27
0	2	*	*	*	27

```
__scores = [8 ** 8, 8 ** 7] + [8 ** 7 / 1.5] * 4 + [8 ** 7 / 4.5] * 3 + \
           [8 ** 7 / 9] * 9 + [8 ** 7 / 13] * 9 + \
           [8 ** 7 / 16] * 27 + [8 ** 7 / 15] * 27
```

图 3: 分数设定

```
def _evaluate(self, i, j, who):
    score = 0
    # 对手
    opposite = ChessBoard.get_opposite(who)
    p = point(i, j)
    # 上下左右、左上、右上、左下、右下八个方向
    for direction in ChessBoard.up_half_direction + ChessBoard.down_half_direction:
        # 方向上的四个位置的状态
        lines = [self.__chess_board.get_chess_at(*(p + direction * t)) for t in range(1, 5)]
        # 计算该状态对应的数（状态代表一个四位三进制数）
        power = 1
        number = 0
        for x in reversed(lines):
            if x is None or x == opposite:
                x = 2
            elif x == ChessBoard.EMPTY:
                x = 1
            else:
                x = 0
            number += x * power
            power *= 3
        # 状态对应的分数，预先设定
        score += self.__scores[number]
    return score
```

图 4: 估分函数

3 实验结果

图中 1 表示黑棋（玩家，先手），2 表示白棋（电脑），0 表示未落子，黄色字体表示上一次落子。从图中可以看出白棋在第四列连成 5 个，胜出，程序打印“White win”并退出。

```
   |0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
---------------------------------------------------
0 |1  1  0  0  0  0  0  0  0  0  0  0  0  0  0
1 |0  1  1  0  0  2  0  0  0  0  0  0  0  0  0
2 |0  0  0  2  0  1  0  0  0  0  0  0  0  0  0
3 |0  0  1  0  1  1  2  0  0  0  0  0  0  0  0
4 |0  0  0  2  2  1  2  0  0  0  0  0  0  0  0
5 |0  0  0  0  2  1  1  0  0  0  0  0  0  0  0
6 |0  0  0  2  2  2  1  1  0  0  0  0  0  0  0
7 |0  0  0  0  2  0  0  0  2  0  0  0  0  0  0
8 |0  0  0  0  2  0  0  0  0  0  0  0  0  0  0
9 |0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
10|0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
11|0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
12|0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
13|0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
14|0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
White win!!!
```

图 5: 估分函数