浙江大学

翻转棋实验报告

课程名称：人工智能

姓　　名：苑子琦

学　　院：计算机科学与技术学院

系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：3170105371

指导教师：王东辉

2020 年　　4 月　　26 日

一、 实验目的和要求

1. 使用『最小最大搜索』、『Alpha-Beta 剪枝搜索』或『蒙特卡洛树搜索算法』实现 黑白棋程序 miniAlphaGo for Reversi（三种算法择一即可）。
2. 使用 Python 语言。
3. 算法部分需要自己实现，不要使用现成的包、工具或者接口。
4. 提交程序报告,请在本地编辑并命名为『程序报告.docx』后，上传到左侧文件列表中。

二、 实验内容和原理

1. 实现 Alpha-Beta 剪枝搜索。

   整体代码结构如下图所示，可以看到启发函数、最大搜索、最小搜索和排序函数。因为搜索不到所有的搜索空间，所以要在搜索到一定层数的时候停止搜索，通过启发函数计算当前棋盘的分数。排序函数是在获得当前棋盘可以走动的合法落子点集合后，对合法点进行一个排序，在时间不足的情况下可以首先搜索比较可能的点，得到更优化的点。

```
def get_move(self, board):
    """
    根据当前棋盘状态获取最佳落子位置
    :param board: 棋盘
    :return: action 最佳落子位置, e.g. 'A1'
    """
    if self.color == 'X':
        player_name = '黑棋'
    else:
        player_name = '白棋'
    print("请等一会，对方 {}-{} 正在思考中...".format(player_name, self.color))

    virtual_board = deepcopy(board)
    self.start = time.clock()
    action, value = self.max_value(virtual_board, -65, 65, 0)

    return action

def heuristic(self, virtual_board, flexibility):...

def max_value(self, virtual_board, alpha, beta, depth):...

def min_value(self, virtual_board, alpha, beta, depth):...

def list_sort(self, virtual_board, legal_list):...
```

2. 限定深度。

   由于要防止超时，在每次经过 max_value()函数的时候，都要执行一次时间检测函数和深度检测函数。如果时间超过预警时间 53 秒，就直接返回启发函数的值。如果深度到达 6 层，且离搜索到最后还有很大的深度，就直接进入启发函数。

```
s_sum = virtual_board.count('X') + virtual_board.count('O')
if time.clock() - self.start >= 53 or (depth >= 6 and s_sum < 57):
    return None, self.heuristic(virtual_board, len(legal_list))
```

3. 排序函数设计。

```
def list_sort(self, virtual_board, legal_list):
    for i in range(len(legal_list)):
        for j in range(len(legal_list) - i - 1):
            x1, y1 = virtual_board.board_num(legal_list[j])
            x2, y2 = virtual_board.board_num(legal_list[j + 1])
            if self.second_weight[x1][y1] < self.second_weight[x2][y2]:
                tmp = legal_list[j]
                legal_list[j] = legal_list[j + 1]
                legal_list[j + 1] = tmp
```

4. 启发式函数策略。

   启发式函数主要采用 3 种策略进行组合。第一种策略是静态矩阵赋值，第二种是稳定子，第三种是自由度。

   本实验中，我使用的静态矩阵如下所示。

```
self.second_weight = [[100, 2, 9, 9, 9, 9, 2, 100],
                      [2, 0, 3, 3, 3, 3, 0, 2],
                      [9, 3, 5, 5, 5, 5, 3, 9],
                      [9, 3, 5, 5, 5, 5, 3, 9],
                      [9, 3, 5, 5, 5, 5, 3, 9],
                      [9, 3, 5, 5, 5, 5, 3, 9],
                      [2, 0, 3, 3, 3, 3, 0, 2],
                      [100, 2, 9, 9, 9, 9, 2, 100]]
```

   本实验中，我使用的稳定子赋值如下所示。

```
self.stable_score = 10
self.corner_score = 100
self.s_corner_score = 40
self.s_line_score = 20
```

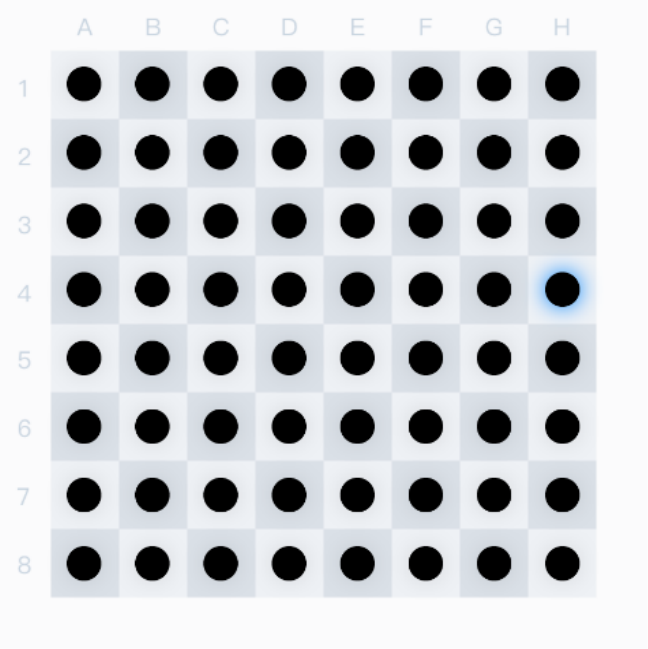
本实验中，我还使用了自由度和棋子数量差作为第三个评判标准。

在启发式函数中，首先进行稳定子的赋值，分数保存在变量 value 里面。此后，如果棋盘上的棋子总数少于 46 个，则 value 首先加上 2 倍的自由度，然后再根据棋子数量决定是否要加上棋子数量差。如果总棋子数多于 46 个，则不再顾及自由度和棋子数量差，直接根据静态矩阵决定每个位置的分数。

```
if s_sum < 46:
    value += 2 * flexibility
    if s_sum < 30:
        value += s_diff
else:
    for i in range(8):
        for j in range(8):
            if board[i][j] == self.color:
                value += self.second_weight[i][j]
            elif board[i][j] == self.op_color:
                value -= self.second_weight[i][j]
```

三、 实验结果分析

对阵测试用例的结果为全胜，且领先棋子数在 40 到 64 不等。



棋局胜负：黑棋赢

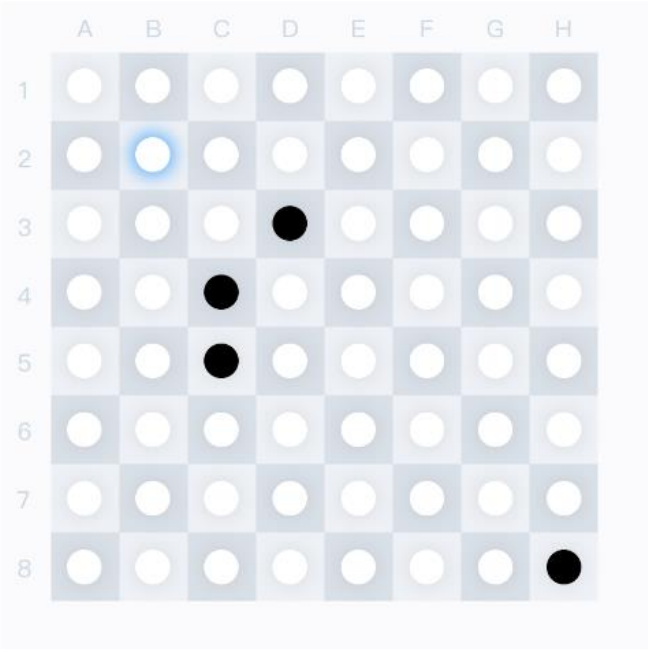
先后手：黑棋先手

棋局难度：初级

当前棋子：黑棋

当前坐标：H4

64 / 64



棋局胜负：白棋赢

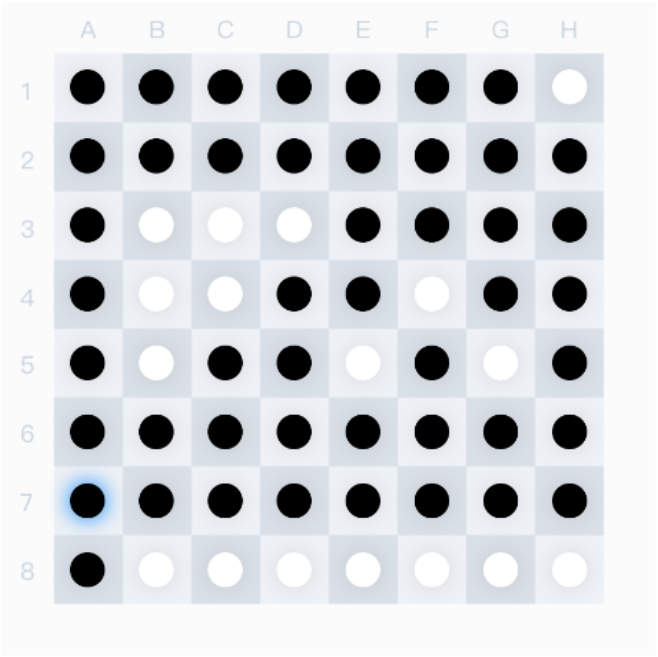
先后手：白棋后手

棋局难度：高级

当前棋子：白棋

当前坐标：B2

64 / 64



棋局胜负：黑棋赢

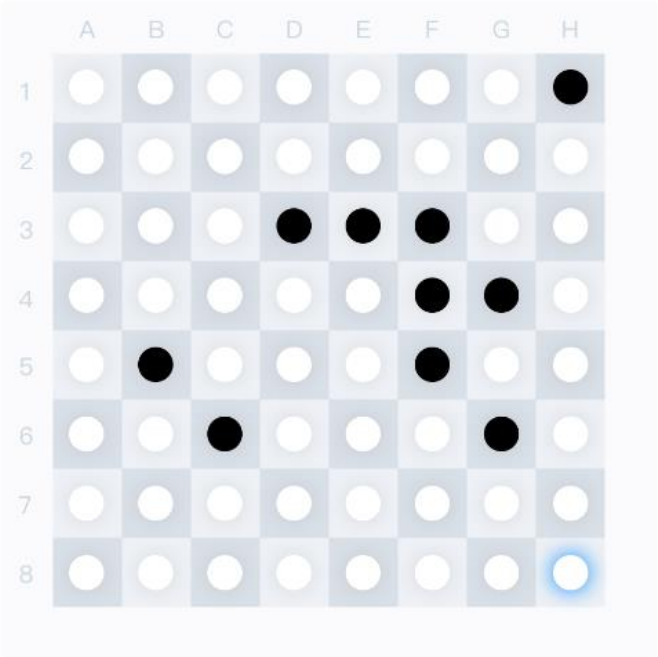
先后手：黑棋先手

棋局难度：高级

当前棋子：黑棋

当前坐标：A7

64 / 64



棋局胜负：白棋赢

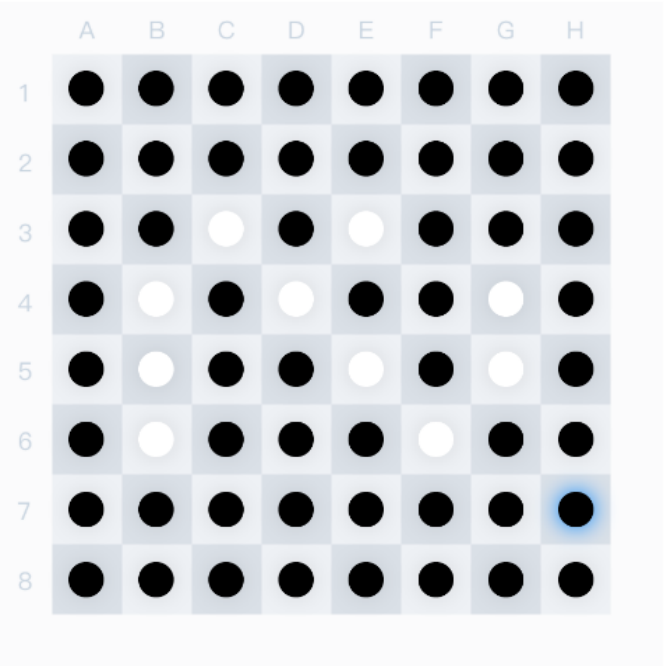
先后手：白棋后手

棋局难度：中级

当前棋子：白棋

当前坐标：H8

64 / 64



棋局胜负：黑棋赢

先后手：黑棋先手

棋局难度：中级

当前棋子：黑棋

当前坐标：H7

64 / 64