中山大学计算机学院

人工智能

本科生实验报告

（2022 学年春季学期）

课程名称：Artificial Intelligence

教学班级	信息与计算科学班	专业（方向）	信息与计算科学
学号	21311359	姓名	何凯迪

一、 实验题目

编写一个中国象棋博弈程序，要求用 alpha-beta 剪枝算法，可以实现人机对弈。可以基于提供的包括棋局评估方法、下棋界面的程序补充完成 alpha-beta 剪枝的深度优先 Minimax 算法；另外，可以从界面优化、增加新的功能、参考已有文献实现其他评估函数、优化剪枝策略使搜索深度提高等方面进行改进，可作为实验的创新点得到相应加分。

二、 实验内容

1. 算法原理

MiniMax 策略：

MiniMax 算法的基本思想是，假设当前节点表示某一玩家的回合，而在该节点之后的所有子节点均为对手的回合。那么，该玩家应该选择在子节点中价值最大的那个节点，而对手会选择价值最小的子节点。

MiniMax 算法会搜索整个博弈树，从根节点开始，先搜索到所有可能的一步子节点，再以同样的方式继续搜索下去。在每个叶子节点

处，需要估算该节点的价值。对于一个玩家来说，其胜利的节点价值为正数，而失败的节点价值为负数。对于对手来说，则相反。在搜索完整个博弈树后，选择根节点的最大值作为当前玩家的最优策略。

Alpha-beta 剪枝的深度优先 Minimax 算法：

Alpha-beta 剪枝是一种优化深度优先搜索（DFS）Minimax 算法的技术，可以减少搜索树的节点数，从而提高算法的效率。

Alpha-beta 剪枝的基本原理是在 Minimax 算法的搜索过程中，及时地剪掉那些不会影响最终结果的子树，从而减少搜索的节点数，提高算法的效率。具体地，Alpha-beta 剪枝会维护两个值，alpha 和 beta，分别代表当前搜索过程中 Max 节点已知的最大值和 Min 节点已知的最小值。在搜索过程中，如果当前节点的价值已经超出了 alpha-beta 区间，则可以直接剪掉当前子树，因为对于父节点来说，这个子树的价值已经不会对最终结果产生影响了。

2. 伪代码

```
# 定义游戏状态评估函数
def evaluate(state):
    # 根据游戏状态计算评估值
    return evaluation

# 定义深度优先 Minimax 算法
def minimax(state, depth, alpha, beta, maximizingPlayer):
    # 判断是否达到搜索深度或者游戏结束
    if depth == 0 or game_over(state):
        return evaluate(state)
    # 判断当前是极大节点还是极小节点
    if maximizingPlayer:
        max_eval = float('-inf')
        # 遍历所有可能的动作
```

```
        for action in possible_actions(state):
            # 执行动作，计算新状态
            new_state = execute_action(state, action)
            # 递归搜索下一层
            eval = minimax(new_state, depth-1, alpha, beta, False)
            # 更新最大评估值和 alpha 值
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            # alpha 剪枝
            if beta <= alpha:
                break
        return max_eval
    else:
        min_eval = float('inf')
        # 遍历所有可能的动作
        for action in possible_actions(state):
            # 执行动作，计算新状态
            new_state = execute_action(state, action)
            # 递归搜索下一层
            eval = minimax(new_state, depth-1, alpha, beta, True)
            # 更新最小评估值和 beta 值
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            # beta 剪枝
            if beta <= alpha:
                break
        return min_eval
# 调用深度优先 Minimax 算法
best_eval = minimax(initial_state, depth, float('-inf'), float('inf'), True)
```

3. 关键代码展示（带注释）

get_next_step:

```
        def get_next_step(self, chessboard: ChessBoard):
        self.alpha_beta(1,float('-inf'),float('inf'),chessboard)
        return self.old_pos+self.new_pos
```

alpha_beta:

```
    def alpha_beta(self, depth, a, b, chessboard: ChessBoard):
        # 当达到最大搜索深度，返回当前棋盘的估值
        if depth >= self.max_depth:
```

```
            return self.evaluate_class.evaluate(chessboard)

        # 获取当前棋盘上所有棋子
        chess_list = chessboard.get_chess()
        # 对于每一个棋子，尝试每一个可能的落子位置
        for cs in chess_list:
            # 判断当前棋子是己方还是对方
            team_match = cs.team == self.team if depth % 2 == 1 else cs.team != self.team
            if team_match:
                # 获取当前棋子可以落子的位置
                next_moves = chessboard.get_put_down_position(cs)
                # 对于每个可能的落子位置，尝试落子并计算估值
                for new_x, new_y in next_moves:
                    last_x, last_y = cs.row, cs.col
                    # 落子
                    origin_chess = chessboard.chessboard_map[new_x][new_y]
                    chessboard.chessboard_map[new_x][new_y] =
chessboard.chessboard_map[last_x][last_y]
                    chessboard.chessboard_map[new_x][new_y].update_position(new_x,
new_y)
                    chessboard.chessboard_map[last_x][last_y] = None
                    # 递归计算估值
                    temp = self.alpha_beta(depth + 1, a, b, chessboard)
                    # 恢复棋盘状态
                    chessboard.chessboard_map[last_x][last_y] =
chessboard.chessboard_map[new_x][new_y]
                    chessboard.chessboard_map[last_x][last_y].update_position(last_x,
last_y)
                    chessboard.chessboard_map[new_x][new_y] = origin_chess
                    # 在第一层时，记录最佳落子点
                    if depth == 1 and (temp > a or not self.old_pos):
                        self.old_pos = [cs.row, cs.col]
                        self.new_pos = [new_x, new_y]
                    # 更新 alpha 和 beta 值
                    a = max(a, temp) if depth % 2 == 1 else a
                    b = min(b, temp) if depth % 2 == 0 else b
                    # alpha-beta 剪枝
                    if b <= a:
                        return a if depth % 2 == 1 else b
        # 返回 alpha 或 beta 的值
        return a if depth % 2 == 1 else b
```

4. 创新点&优化（如果有）

加了背景音乐

```
#bgm
pygame.mixer.init()
pygame.mixer.music.load('D:\python 文件\\alpha-beta-AIchess\\bgm.mp3')
pygame.mixer.music.play(-1,0)
```

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

前四次对弈棋局情况：


pygame window
楚河 汉界
悔棋
红方走


pygame window
楚河 汉界
悔棋
红方走

与微信小程序“天天象棋”进行对弈，小程序段执红棋（先走）。首先进行 2 级对弈，结果为平局。与 5 级对弈，结局为小程序方胜。

测试过程中，我发现还是有部分无用的棋步，可能是因为探索的深度不够。但目前大概能够应对象棋初学者

四、 参考资料

https://blog.csdn.net/qq_41854911/article/details/122697088?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216827556501680021502744O%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=168275565016800215027440&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-122697088-null-null.142^v86^insert_down1,239^v2^insert_chatgpt&utm_term=pygame&spm=1018.2226.3001.4187