alpha-beta 算法实验报告

2023200440

2024 年 10 月 10 日

1 实验目的

本次实验通过对 MinMax 和 - 剪枝算法的学习，基于 basecode 实现井字棋的 $\alpha-\beta$ 剪枝算法，加强对对抗搜索算法的理解和掌握，理解 $\alpha-\beta$ 剪枝在零和博弈中的作用和效果。

2 实验方法

2.1 实验环境

环境：windows11
解释器：python3.12
编辑器：pycharm

2.2 实验步骤

1. basecode 代码修改
(1) 决定先手时，判断条件有误（if opt == ”X” or ”x”）：始终为真，HUMAN 先手

```
if opt == "X" or opt == "x":
    next_move = HUMAN
elif opt == "O" or opt == "o":
    next_move = COMPUTER
```

(2) 修改打印顺序：棋盘打印在每次循环开始处，机器落子后不能看到落子结果，改为 HUMAN 和 COMPUTER 下棋后打印棋盘，去掉循环结束后打印棋盘 (没必要)

2. determine_move 实现第一步下棋在 determine_move 实现，便于返回 next_move(- 剪枝返回值是评估函数值，即得分)。实现思路如下:

① 给定 α, β 初值 (负无穷和正无穷)

② 遍历第一步落子位置，尝试落子

③ 调用 $\alpha - \beta$ 函数获得当前落子的得分

④ 撤销落子，更新 next_move 和 maxvalue

⑤ 返回 next_move

3. alpha_beta 算法实现思路如下:

① 定义 MAX=True(先手)，MIN=False(后手)

② 判断当前下棋选手 player(MAX-先手，MN-后手)

③ 判断是否结束 (胜负已知或平局)

④ 落子，下一位选手下棋

⑤ 撤销落子，更新 α

⑥ 判断是否需要剪枝

⑦ 返回 α 或 β 值

3 代码细节

1. determine_move

```
def determine_move(board):
"""
    决定电脑(玩家O)的下一步棋(使用Alpha-beta剪枝优化
    搜索效率)
    Args:
```

```
        board (list):井字棋盘
    Returns:
        next_move(int): 电脑(玩家O) 下一步棋的位置
"""
    # alpha, beta初值分别设为负无穷和正无穷
    alpha = float('-inf')
    beta = float('inf')
    next_move = None
    maxvalue = float('-inf')
    for slot in SLOTS:
        if board[slot] == Open_token:   # 找到空白格
            board[slot] = O_token      # 落子
            value = alpha_beta(board, alpha, beta, MIN, 0)
            board[slot] = Open_token   # 撤销落子
            alpha = max(alpha, value)
            if maxvalue < value:   # 找到赢面更大的slot
                maxvalue = value   # 更新maxvalue和nextmove
                next_move = slot
    return next_move
```

---

2. alpha_beta 算法实现

```
def alpha_beta(board, alpha, beta, player, depth):
"""
    alphabeta剪枝算法
    depth: 当递归深度很大时可以用于控制深度
    player: 当前下棋选手
"""
    score = winner(board)
    if score != 0 or not legal_move_left(board):  # 胜负知晓或者平局返回
        return score
    # 轮到先手,找赢面最大的
    if player:
```

```
        for slot in SLOTS:
            if board[slot] == Open_token:
                board[slot] = O_token
                value = alpha_beta(board, alpha, beta, MIN, depth + 1)
                board[slot] = Open_token
                alpha = max(alpha, value)
                if beta <= alpha:
                    break      # alpha剪枝
        return alpha
    # 轮到后手，找使先手赢面最小的
    else:
    //省略代码同第一种情况......
                board[slot] = X_token
                value = alpha_beta(board, alpha, beta, MAX, depth + 1)
                ......
                beta = min(beta, value)
            ......
```

4 实验结果

4.1 结果展示

实验结果如下：

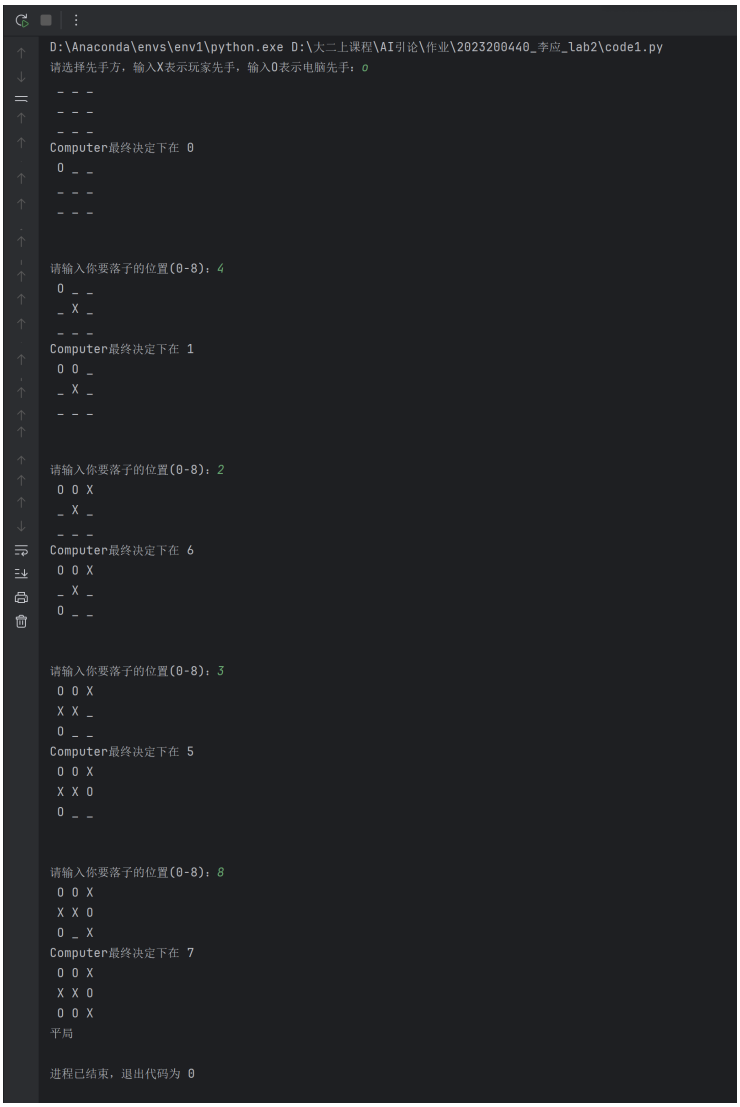

图 1: 实验结果截图

5 参考文献

参考以下博客:
https://blog.csdn.net/m0_51662391/article/details/129246041
https://zhuanlan.zhihu.com/p/65108398