实验二:井字棋与对抗搜索

一、与 AI 交互的过程

①第一阶段：将原先 3×3 棋盘扩展为 4×4 棋盘

用户提示词：

请在已有代码的基础上做出修改，将原本 3×3 的棋盘扩大至 4×4 棋盘。

Copilot：

我已经成功将代码从 3x3 棋盘修改为 4x4 棋盘。主要修改包括:

棋盘初始化：将 board 从 3x3 改为 4x4。

打印函数：调整 print_board 以显示 4 列。

胜利检查：修改 check_winner 以检查 4 连线（行、列、对角线）。

满格检查：更新 is_board_full 为 16 格。

AI 算法：将 minimax 和 get_best_move 中的循环范围从 3 改为 4。

用户输入：更新提示信息为 0-3 范围。

代码现在支持 4x4 井字棋，AI 使用 Alpha-Beta 剪枝算法（通过 Minimax 实现）进行最佳移动。游戏规则为 4 连线获胜，16 格满为平局。

②第二阶段：引入 Alpha_Beta 剪枝算法，以优化搜索树的遍历效率

用户提示词：

由于 4*4 的井字棋计算时过于复杂，运行长时间没有结果。现在请你在原代码基础上引入 Alpha_Beta 剪枝算法，以优化搜索树的遍历效率。

Copilot：

我已经成功在原代码基础上引入了 Alpha-Beta 剪枝算法。主要修改包括:

修改 minimax 函数签名：添加了 alpha 和 beta 参数，用于剪枝判断。

在 maximizing 分支：更新 alpha = max(alpha, best_score)，当 alpha >= beta 时提前终止搜索。

在 minimizing 分支：更新 beta = min(beta, best_score)，当 alpha >= beta 时提前终止搜索。

更新 get_best_move 调用：传递初始的-float('inf')和 float('inf')作为 alpha 和 beta。

Alpha-Beta 剪枝通过消除不可能影响最终决策的分支，大幅优化了搜索树的遍历效

率。对于 4x4 井字棋（16 个位置），这将显著减少计算时间，使 AI 能在合理时间内做出最佳移动。代码现在应该运行得更快了。

③第三阶段：增加最大搜索深度限制，实现深度受限的 Alpha-Beta 剪枝

用户提示词：

单靠引入 alpha_beta 剪枝算法还是不能有效改善计算量大运行时间长的问题，现在请你在原代码的基础上，为算法增加最大搜索深度限制，实现深度受限的 Alpha-Beta 剪枝，以优化搜索树的遍历效率。

Copilot：

我已经成功在原代码基础上实现了深度受限的 Alpha-Beta 剪枝算法。

添加评估函数：evaluate_line(line)：评估一行/列/对角线，基于连线数量给分（4 连线=100 分，3 连线=10 分，2 连线=1 分）。evaluate(board)：综合评估整个棋盘的行、列和对角线。

修改 minimax 函数：添加 depth 和 max_depth 参数。当 depth >= max_depth 时，返回 evaluate(board)而不是继续搜索。

更新 get_best_move：添加 max_depth 参数，传递给 minimax。

在 main 函数中：设置 max_depth = 4，限制搜索深度为 4 层。在 AI 回合中传递 max_depth 给 get_best_move。

通过限制搜索深度，算法在达到指定深度时使用启发式评估代替完全搜索，大幅减少计算量。对于 4x4 井字棋，深度限制为 4 应该能在几秒内完成 AI 移动，同时保持较好的游戏质量。你可以根据需要调整 max_depth 值来平衡速度和准确性。

④第四阶段：平衡井字棋先手优势，人为引入 Pie 规则

用户提示词：

由于井字棋存在先手优势，我们现在人为加入 pie 规则，Pie 规则说明如下……请按照以上要求修改代码。

Copilot：

我已经成功在代码中实现了 Pie 规则。主要修改包括：

添加随机模块：导入 random 用于随机选择先手。

随机选择先手：player_first = random.choice([True, False])根据结果告知用户谁先手，

并设置初始 next_player。

第一步处理：如果玩家先手，让玩家输入落子位置。如果 AI 先手，调 get_best_move 决定第一子并落子。

交换机制：仅当 AI 先手（玩家是后手）时，询问玩家是否交换。如果交换：第一子符号从'O'改为'X'，设置 next_player = 'X'（玩家下第二步）。如果不交换：next_player = 'X'（玩家下第二步）。

游戏循环调整：使用 next_player 控制轮次，确保第一步后正确切换玩家。循环中根据 current_player 决定是玩家输入还是 AI 计算。

现在游戏支持随机先手和 Pie 规则的交换机制，确保公平性。玩家在 AI 先手的情况下有机会通过交换获得先手优势。代码逻辑完整，能够正确处理所有情况。

⑤第五阶段：多轮调试与改错，成功实现 Pie 规则

二、核心代码的理解与注释

（1）Alpha-Beta 剪枝的边界传递逻辑

Alpha-Beta 剪枝是对 Minimax 算法的优化，通过维护两个边界值 alpha 和 beta 来剪掉不可能影响最终决策的分支。

alpha：当前 Maximizing 玩家（AI）所能保证的最好分数（下限），初始为 -∞

beta：当前 Minimizing 玩家（玩家）所能保证的最坏分数（上限），初始为 +∞

在搜索过程中，当 alpha >= beta 时，说明当前分支已经不会对最终决策产生更好影响，可以立即剪枝，停止搜索该节点的其余子节点。

传递规则：

Max 节点（AI 回合）：更新 alpha=max(alpha, child_score)，并将新的 alpha 传递给子节点作为其 alpha，子节点的 beta 保持不变。

Min 节点（玩家回合）：更新 beta=min(beta, child_score)，并将新的 beta 传递给子节点作为其 beta，子节点的 alpha 保持不变。

代码对应片段（已添加注释）：

```
if is_maximizing:     # AI 的回合，最大化自己的得分
    best_score = -float('inf')
    for i in range(4):
        for j in range(4):
            if board[i][j] == ' ':
```

```
                    board[i][j] = 'O'
                    # 递归调用，轮到玩家，转递当前的 alpha 和 beta 值，并增加深度
                    score = minimax(board, False, alpha, beta, depth + 1, max_depth)
                    board[i][j] = ' '
                    best_score = max(best_score, score)
                    alpha = max(alpha, best_score)  # 更新 alpha 值
                    if alpha >= beta:
                     #如果 alpha 不小于 beta，说明 Min 节点会避免该分支，直接剪掉
                        break
            else:
                continue
            break
    return best_score
else:    # 玩家回合，最小化 AI 的得分
    best_score = float('inf')
    for i in range(4):
        for j in range(4):
            if board[i][j] == ' ':
                board[i][j] = 'X'
                score = minimax(board, True, alpha, beta, depth + 1, max_depth)
                board[i][j] = ' '
                best_score = min(best_score, score) # 更新 beta 值
                beta = min(beta, best_score)
                if alpha >= beta:
                 #如果 alpha 不小于 beta，说明 Max 节点会避免该分支，直接剪掉
                    break
        else:
            continue
        break
    return best_score
```

（2）深度受限模块的代码说明

由于 4×4 棋盘状态空间巨大，完整搜索会指数爆炸。因此我们设置了最大搜索深度 max_depth。当递归深度 depth>=max_depth 时，不再继续展开子节点，而是调用启发式评估函数 evaluate(board)返回一个估计分数。

代码对应片段（已添加注释）：

```
# 深度限制检查：如果达到最大搜索深度，返回当前棋盘的评估分数
if depth >= max_depth:
    return evaluate(board)  #不继续递归，直接评估当前棋盘状态
# 否则继续进行 Minimax 搜索
```

三、启发式函数的设计

启发式函数用于在非终局状态下评估当前棋盘对 AI 的优势程度。程序设计了两个层次的函数：evaluate_line 评估单条线（行、列、对角线），evaluate 汇总所有线得到总分。

（1）单线评估函数 evaluate_line(line)

输入：一条线上的 4 个格子。

输出：该线对 AI 的分数贡献（正分表示 AI 有利，负分表示玩家有利）。

评分规则（启发式权重）：

情况	分数	解释
四个 O	+100	AI 已获胜（最高奖励）
四个 X	-100	玩家已获胜（最高惩罚）
三个 O 且无 X	+10	非常接近获胜，威胁大
三个 X 且无 O	-10	玩家接近获胜，需紧急防守
两个 O 且无 X	+1	有一定潜力
两个 X 且无 O	-1	玩家有一定潜力
其他情况	0	混合或没有连续

（2）全局评估函数 evaluate(board)

将棋盘上所有行、列、两条主对角线共 10 条线的分数相加，得到总评分。

$$\text{score}(\boldsymbol{board}) = \sum_{\text{每条线 } \boldsymbol{L}} \text{evaluate\_line}(\boldsymbol{L})$$

优点：计算简单，快速。鼓励 AI 形成长连，同时防御玩家的长连。对空位较多的开局也能给出合理估值。

局限性：未考虑棋子之间的位置关系，但结合深度限制和 Alpha-Beta 剪枝，足以在 4×4 井字棋中做出明智决策。

四、UI 界面设计展示

（1）界面交互说明

本程序采用**高质量的命令行交互界面**，玩家通过终端输入指令与 AI 进行对弈。

①启动程序：运行 python alpha_beta.py，终端显示 4×4 网格带边框棋盘，并随机决定先手（玩家执 X，AI 执 O）。

②第一步落子：

若玩家先手则提示输入行号和列号（0-3），落子后棋盘更新；

若 AI 先手则 AI 自动落子并显示棋盘；

③交换阶段：

若 AI 先手则玩家被询问是否交换（输入 y 或 n）。若交换，第一子变为玩家棋子，AI 继续下第二步；若不交换，玩家下第二步。

若玩家先手则 AI 自动决定是否交换，并输出结果（“AI 选择交换”或“AI 选择不交换”）。

④正常对弈：双方轮流落子，每次玩家输入行列坐标，AI 思考后自动落子。每次落子后刷新棋盘，并显示胜负或平局结果。

⑤错误处理：输入非数字、越界或重复位置时，程序会提示重新输入。

(2) 运行效果截图

```
(base) PS D:\Zhejiang University\AI Experiments\Lab2\debug(1)> python alpha_beta.py
AI先手！AI 执 'O'，你执 'X'。
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
AI 正在思考第一步...
-----------------
| O |   |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
第一子已落下，你是后手，是否交换？(y/n): y
已交换！第一子归你，AI继续下第二步。
AI 正在思考...
-----------------
| X | O |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
请输入行号 (0, 1, 2, 3): 3
请输入列号 (0, 1, 2, 3): 3
-----------------
| X | O |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   | X |
-----------------
```

```
(base) PS D:\Zhejiang University\AI Experiments\Lab2\debug(1)> python alpha_beta.py
请输入行号 (0, 1, 2, 3): 3
请输入列号 (0, 1, 2, 3): 3
-----------------
| X | O |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   | X |
-----------------
AI 正在思考...
-----------------
| X | O |   | O |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   | X |
-----------------
请输入行号 (0, 1, 2, 3): 3
请输入列号 (0, 1, 2, 3): 0
-----------------
| X | O |   | O |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
| X |   |   | X |
-----------------
AI 正在思考...
-----------------
| X | O |   | O |
-----------------
| O |   |   |   |
-----------------
|   |   |   |   |
-----------------
| X |   |   | X |
-----------------
```

```
(base) PS D:\Zhejiang University\AI Experiments\Lab2\debug(1)> python alpha_beta.py
请输入行号 (0, 1, 2, 3): 2
请输入列号 (0, 1, 2, 3): 2
-----------------
| X | O |   | O |
-----------------
| O |   |   |   |
-----------------
|   |   | X |   |
-----------------
| X |   |   | X |
-----------------
AI 正在思考...
-----------------
| X | O |   | O |
-----------------
| O | O |   |   |
-----------------
|   |   | X |   |
-----------------
| X |   |   | X |
-----------------
请输入行号 (0, 1, 2, 3): 3
请输入列号 (0, 1, 2, 3): 6
输入无效，请输入 0-3 之间的数字！
请输入行号 (0, 1, 2, 3): 0
请输入列号 (0, 1, 2, 3): 2
-----------------
| X | O | X | O |
-----------------
| O | O |   |   |
-----------------
|   |   | X |   |
-----------------
| X |   |   | X |
-----------------
AI 正在思考...
-----------------
| X | O | X | O |
-----------------
| O | O | O |   |
-----------------
|   |   | X |   |
-----------------
```

```
(base) PS D:\Zhejiang University\AI Experiments\Lab2\debug(1)> python alpha_beta.py
| X |   |   | X |
----------------
AI 正在思考...
----------------
| X | O | X | O |
----------------
| O | O | O |   |
----------------
|   |   | X |   |
----------------
| X |   |   | X |
----------------
请输入行号 (0, 1, 2, 3): 3
请输入列号 (0, 1, 2, 3): 2
----------------
| X | O | X | O |
----------------
| O | O | O |   |
----------------
|   |   | X |   |
----------------
| X |   | X | X |
----------------
AI 正在思考...
----------------
| X | O | X | O |
----------------
| O | O | O | O |
----------------
|   |   | X |   |
----------------
| X |   | X | X |
----------------
很遗憾，AI 赢了！
```