实验报告

一、叶子节点设置

由于井字棋的状态空间较小，我选择用叶子节点表示游戏终局，共有三种情况，玩家胜，电脑胜，平局，在构建博弈树时分别将这三种情况对应的 value 设为-1,1 和 0，用于“标记”该叶子节点，后面在剪枝函数中会对 value 值进行更新以实现“尽早获胜”功能。

二、实现方法

在 determine_move 函数以外，我额外设计了三个函数，它们的作用分别为寻找可落子点位（find_possible_move）、构建博弈树（init_node）和 α - β 剪枝（alpha_beta_search），并相应的构造了节点类 Node。

下面进行分别介绍：

1. 节点类 Node 用于存储每一个节点的所有子节点，当前节点对应的棋盘，当前棋盘对应的上一步棋，以及如果为叶子节点则会计算的 value 值。

2. find_possible_move 的作用是返回当前棋盘所有空位。

3. init_node 首先判断即将构建的节点是否为叶子节点，是则直接返回有初始 value 值的 Node 类变量；接着构建所有的子节点，最终将所有参数都传入 Node 的构造函数。

4. 核心剪枝实现方法在 alpha_beta_search 函数中，在这里首先判断当前处于最大层还是最小层，接着对于最大层则更新 α 值（相对的，最小层更新 β 值），并在 α ≥ β 时进行剪枝，最终返回最大（最小）收益值与最佳落子位置 best_move；此外，对于叶子节点，用 depth 变量更新 value 值以实现“尽早获胜”，即确定可以取胜的棋局不会进行额外拖延。

三、结果

调用 determine_move 函数时，会针对当前棋盘构建一颗博弈树，然后进行 α - β 剪枝搜索，得到 next_move。

目前代码已经可以使电脑做出最佳选择，实现“不败”，并且决策速度很快。

四、可能的小改进

目前电脑先手时的第一步棋往往会偏慢，而棋盘可以分为三个部分，角位、边位、中央，对于局面无威胁的情况，可以选择从这三个部分入手，从而避免重复搜索。