天津大学

《人工智能基础》课程设计报告

基于启发式搜索的 AI 五子棋

学　　号 3022244096

姓　　名 郭睿

学　　院 智能与计算学部

专　　业 计算机科学与技术

年　　级 2022 级

任课教师 廖士中

2024 年 6 月 10 日

一、实验目的

1. 知识掌握

熟悉并掌握博弈树的启发式搜索过程、α－β剪枝算法和评价函数，利用α－β剪枝算法开发一个五子棋人机对弈游戏。

2. 程序实现

掌握状态空间搜索、启发式搜索、max-min 方法和α－β剪枝， 本实验程序使用python 编写，需要使用的第三方库为 pygame，请在运行环境中运行 pip install pygame。代码已上传至 github：https://github.com/Rain718718/- ，并且通过邮箱向廖老师发送了工程文件压缩包。

二、实验摘要

1. 以五子棋人机对弈问题为例，实现 α－β 剪枝算法的求解程序（编程语言不限），要求设计适合五子棋对弈的评估函数。
2. 初始界面需显示 15*15 的空白棋盘，电脑执白棋，人执黑棋，并提供重新开始和悔棋等操作。
3. 设计五子棋程序的数据结构，需具备评估棋势、选择落子和判断胜负等功能。

三、实验内容

(1) 总体设计思路与总体架构

总体设计框架分为三大部分：

① 人机交互界面部分：通过图形用户界面显示棋盘状态和落子次序，打印操作提示文本信息，游戏开始时显示封面，游戏结束时显示胜负信息等。

② 搜索部分:AI 接收代表棋盘状态的矩阵信息后，使用深度受限的启发式搜索，尝试不同位置的落子，并根据这些位置的评估函数选择最优落子位置。

③ 游戏控制部分：通过设置事件监测获取键盘和鼠标的操作信息，利用键盘按键进行开始、悔棋、重新开始等控制行为，通过判断鼠标点击位置最近的网格来获取玩家的落子位置。每次 AI 或玩家落子后判断是否产生输赢。

(2) 核心算法及基本原理：

带 α－β 剪枝的极小极大搜索：

当轮到 AI 落子时，获取当前棋盘状态后，AI 尝试所有可能的落子位置，生成后继状态的节点。采用极小极大搜索形成博弈树，对于所有的 MAX 节点，其 α 值为所有直接后继节点的 β 值的最大值，这意味着在玩家足够聪明并最小化 AI 得分的情况下，AI 能够获得的最大得分。对于所有的 MIN 节点，其 β 值为所有直接后继节点的 α 值的最小值，这意味着在 AI 足够聪明并最大化自己得分的情况下，玩家能限制 AI 获得的最小得分。所有叶子节点都通过效用函数进行评估，用来衡量 AI 的得分。

如果任何 MAX 节点 X 的 α 值不能降低其祖先节点的 β 值，则可以停止对 X 以下分枝的搜索。同样地，如果任何 MIN 节点 Y 的 β 值不能升高其祖先节点的 α 值，则可以停止对 Y 以下分枝的搜索，从而大大减少了搜索状态数。

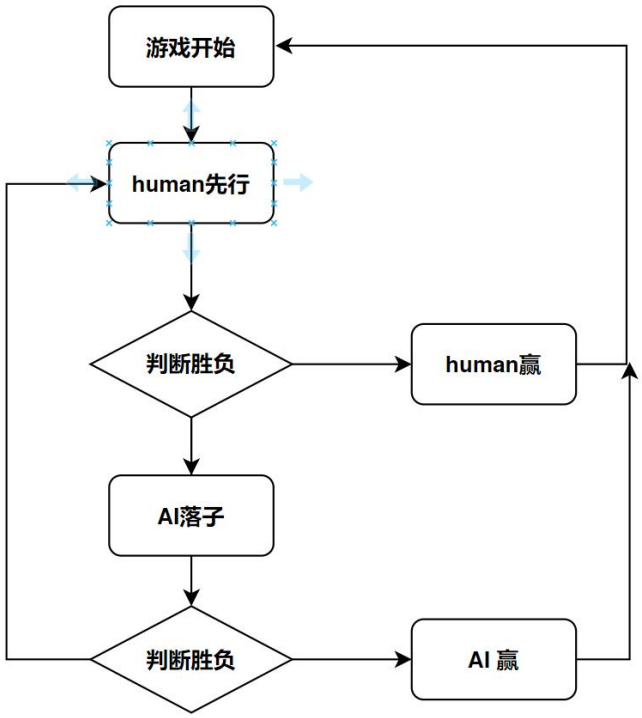
由于是人机对弈游戏，为了考虑玩家的游戏体验，AI 的决策时间不能过长。因此，极小极大搜索不能搜索到叶子节点，而采用深度受限的启发式搜索，对所有节点设置评估函数来代替效用函数评估 AI 的得分，在较短时间内做出具有较大得分期望的决策。

评估函数的设计：

找出与落子位置相连的本方棋子数量，根据形成的棋型来评价方案的期望得分。不同棋型的评估值如下：

① 单子，5 分：即周围没有相连的本方棋子。
② 死二，10 分：一端被封死的二连子棋型。
③ 活二，20 分：没有被封死端的二连子棋型。
④ 死三，20 分：一端被封死的三连子棋型。
⑤ 活三，45 分：没有被封死端的三连子棋型。
⑥ 死四，60 分：一端被封死的四连子棋型。
⑦ 活四，120 分：没有被封死端的四连子棋型。
⑧ 成五，300 分：五连子棋型。

节点的评估函数为该节点落子位置在横向、竖向和两个斜向方向上形成的四种棋型的评估值。为了避免 AI 陷入固定模式，以 0.8 的概率选择最大的两个评估值之和作为返回的评估值，以 0.15 的概率选择最大的三个评估值之和，以 0.05 的概率选择所有四个评估值之和。



(3) 模块设计：本实验的具体模块设计

① 主循环模块：通过计数器记录每一步轮到哪一方落子，并且在轮到玩家落子时设置键盘和鼠标监测来探知玩家的操作（如落子、悔棋等）。在 AI 或玩家落子后，将新的棋盘局面呈现在屏幕上。在游戏开始时调用封面显示，在游戏结束时调用结束显示。

② 评估模块：对每个节点，通过检测落子位置在横、竖、两个斜向方向上所能形成的四种棋型，将其中最大的两个评估值相加作为评估函数，评价 AI 的得分期望。同时，有一定几率将最大的三或四个评估值相加作为评估函数，以实现对同一局面作出不同的决策，避免被玩家使用特定棋路连续击败。

③ 搜索模块：使用深度受限的带 α－β 剪枝的极小极大搜索确定 AI 的最佳落子位置。

④ 记录模块：在每次 AI 或玩家落子后，更新对局步数，并将该步所下棋子的位置和颜色送入栈中记录，同时更新反映棋盘状态的矩阵。悔棋操作通过将栈顶两步落子位置弹出并将棋盘对应位置更新为无棋子实现。

⑤ 显示模块：显示游戏封面、背景、结束对话框等。同时在游戏过程中以高频率刷新棋盘状态的显示，在无操作时肉眼无法察觉到画面闪烁，而在悔棋或落子操作时可以明显看出棋盘变化。

(4) 其他创新内容或优化算法

① 落子位置优化： 由于极小极大搜索深度受限，得到的结果只能是当前得分期望最大的落子选择，没有战略性的长远眼光。因此，对于离已落棋子很远的棋盘位置，显然不符合得分期望最大这一要求。如果每次落子都搜索整个棋盘，会造成时间上的大量浪费。

可以缩小尝试的落子位置范围，对于每个节点，选择落子位置只在其对应棋盘状态中的已有棋子外切矩形的附近（宽度 1~2 格）和内部范围。对于离外切矩形很远的位置，落子后所形成的棋型大部分都是单子，因此外切矩形附近和内部的落子位置评估通常大于这些单子。通过将落子范围限制在已有棋子的外切矩形附近和内部，可以明显缩短 AI 的决策时间。

同时存在一种特殊情况，当棋盘的正中央很空时，优先落子到棋盘的正中央，因为那里更不容易出界，从而有更多的连子机会。

② 搜索方式实现创新： 在实现深度受限的带 α－β 剪枝的极小极大搜索时，可以采用递归实现方式，但是实际运行时，搜索耗时较长，玩家落子后 AI 需要的思考时间较长，对弈体验不佳。

可以在限制深度较浅时将递归实现的方式改为循环嵌套的方式，如下：

```python
# alpha-beta剪枝搜索
def ai_go():
    global x_min, x_max, y_min, y_max, color_flag, matrix
    time_start = time.time()
    evaluate_matrix = [
        [0 for i in range(SIZE + 2)] for j in range(SIZE + 2)
    ]  # 结点估值矩阵
    if step != 0:
        if step == 1:
            # 第一步下的位置
            if matrix[(SIZE + 1) // 2][(SIZE + 1) // 2] == 0:
                rx, ry = (SIZE + 1) // 2, (SIZE + 1) // 2
            else:
                if (
                    matrix[(SIZE + 1) // 2][(SIZE + 1) // 2] != 0
                    and matrix[(SIZE + 1) // 2 + 1][(SIZE + 1) // 2 + 1] == 0
                ):
                    rx, ry = (SIZE + 1) // 2 + 1, (SIZE + 1) // 2 + 1
        else:
            min_tx1, min_ty1, max_tx1, max_ty1 = range_legal(x_min, y_min, x_max, y_max)
            evaluate_matrix = [
                [0 for i in range(SIZE + 2)] for j in range(SIZE + 2)
            ]  # 第一层的估值矩阵
            Max = -100000
            rx, ry = 0, 0
```

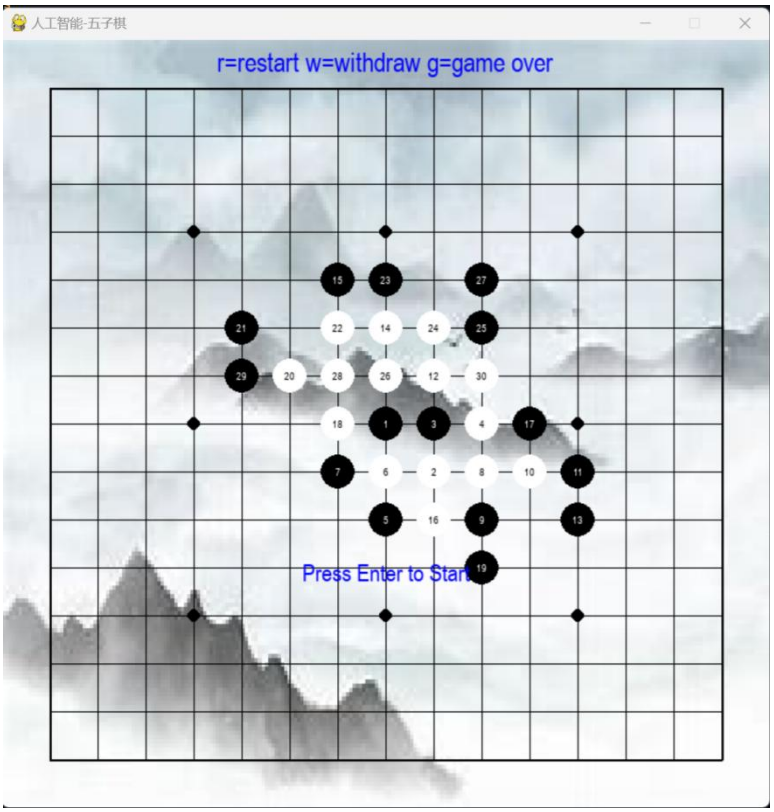
```
            for i in range(min_tx1, max_tx1 + 1):
                for j in range(min_ty1, max_ty1 + 1):
                    cut_flag = 0  # 剪枝标记
                    evaluate_matrix2 = [
                        [0 for i in range(SIZE + 2)] for j in range(SIZE + 2)
                    ]

                    if matrix[i][j] == 0:
                        matrix[i][j] = color_flag
                        min_tx2, min_ty2, max_tx2, max_ty2 = range_legal(
                            min_tx1, min_ty1, max_tx1, max_ty1
                        )
                        [list_h, list_v, list_s, list_b] = get_list(i, j, color_flag)
                        eva1 = evaluate(list_h, list_v, list_s, list_b)

                        for ii in range(min_tx2, max_tx2 + 1):
                            for jj in range(min_ty2, max_ty2 + 1):

                                if matrix[ii][jj] == 0:
                                    matrix[ii][jj] = -color_flag
                                    [list_h, list_v, list_s, list_b] = get_list(
                                        ii, jj, -color_flag
                                    )
                                    eva2 = -evaluate(list_h, list_v, list_s, list_b)

                                    evaluate_matrix2[ii][jj] = eva2 + eva1
                                    matrix[ii][jj] = 0
                                    # 剪枝
                                    if evaluate_matrix2[ii][jj] < Max:
                                        evaluate_matrix[i][j] = evaluate_matrix2[ii][jj]
                                        cut_flag = 1
                                        break
                            if cut_flag:
                                break

                        if cut_flag == 0:
                            Min = 100000
                            for ii in range(min_tx2, max_tx2 + 1):
                                for jj in range(min_ty2, max_ty2 + 1):
                                    if (
                                        evaluate_matrix2[ii][jj] < Min
                                        and matrix[ii][jj] == 0
                                    ):
                                        Min = evaluate_matrix2[ii][jj]

                            evaluate_matrix[i][j] = Min

                            if Max < Min:
                                Max = Min
                                rx, ry = i, j

                        matrix[i][j] = 0

        time_end = time.time()
        print("Time cost:", round(time_end - time_start, 4), "s")
        add_chess(rx, ry, color_flag)
```

从而大大节省了 AI 思考时间，同时 AI 的决策能力并没有出现明显下降，依然能够保持较高水平。

③ 引入随机化因素避免 AI 被同一棋路击败：由于 AI 的评估函数确定时，对于同一局面会得出相同的落子位置，存在某种棋路使得玩家只要使用这种棋路 AI 就必败。因此，在评估函数中引入随机化因素。除了选取四个方向上的棋型中分数最高的两个相加之外，还有一定几率选取分数最高的三个或四个相加。这

样，AI 会在一些情况下选择略劣于严格取分数最高的两个相加评估出的最优落子位置的位置上落子，避免被玩家使用同一棋路连续击败。

四、实验结果



Time cost: 0.0 s
Time cost: 0.0459 s
Time cost: 0.0561 s
Time cost: 0.0498 s
Time cost: 0.0572 s
Time cost: 0.0487 s
Time cost: 0.0574 s
Time cost: 0.0869 s
Time cost: 0.0708 s
Time cost: 0.0454 s
Time cost: 0.0695 s
Time cost: 0.0586 s
Time cost: 0.0501 s
Time cost: 0.0511 s
Time cost: 0.0517 s

实验结论： 使用深度受限的带 α－β 剪枝的极小极大搜索设计的五子棋 AI 具有较高的人机对弈水平和能力，并且搜索用时短，决策速度快。引入随机

化评估函数选取机制后，AI 能够在同一局面下做出不同决策，避免被玩家使用某些棋路连续击败，更加具备人类思维的特点。

五、实验总结

(1) 实验中存在的问题及解决方案
问题 1：原始程序效率低下
解决方案：原始程序搜索整个棋局范围，但分析发现，若下的棋子距离对手过远，不会是一招好棋。于是我们将 AI 落子范围限定在已有棋子的外切矩形周边和内部，大大减少了搜索范围。同时，结合下棋经验，将 AI 的第一步落在棋局的中央或其附近，这样整体更有利。

问题 2：悔棋时已经放上的棋子无法撤销显示
解决方案：通过高频率刷新游戏界面，在棋盘无变化时维持用户视觉感受，在棋盘变化时只需修改棋盘状态矩阵即可显示新的局面。

问题 3：当限制搜索深度时，使用递归方式实现带 α－β 剪枝的极小极大搜索较慢，玩家落子后 AI 思考时间较长
解决方案：在确定限制深度后，改用循环嵌套的方式替代递归实现。虽然不易修改限制深度，但换来了更快的 AI 思考时间。较浅的搜索深度已足以对抗人脑决策，实现人机博弈。

问题 4：AI 出招套路固定，针对同一局面的落子点相同。如果对弈者反复尝试，可能容易找出战胜 AI 的方法，并且之后利用这个套路可以一直战胜 AI。
解决方案：修改 AI 对棋局的评估函数，在相同局面下，AI 有一定概率采取其他评估函数。这种概率较小，类似于突变。这样 AI 大体上按照同一种评估方法，以保证整体评估方式的一致性，同时以小概率采取其他评估方式，使得棋招更具多变性。

(2) 后续改进方向
① 关于五子棋的棋型还存在更多复杂的种类（如眠三等），通过引入更多棋型，可以进一步提高 AI 决策的最优程度。
② 加大搜索深度，使 AI 具有更长远的眼光，能够识破玩家的一些埋伏性棋路，从而进一步提升人机对弈的难度。
③ 使用深度学习的方法训练 AI，使 AI 通过不断训练获得更好的效果。甚至可以用深度学习训练的 AI 与剪枝算法的 AI 对抗，帮助调整剪枝算法的评估函数和各种棋局类型的估分。
④ 通过多线程方式实现 MTD(F) 或 PVS，加快 AI 搜索速度，加大搜索深度，获得更好的决策效果。