人工智能实验报告

实验4　α-β 剪枝

陳日康　信息与计算科学　22336049

一、实验概述

编写一个五子棋博弈程序，要求用 α-β 剪枝算法，实现人机对弈。微信小程序 “欢乐五子棋” 中的残局闯关的前 20 关，任选一关、任选一步、任选搜索树上的一个非叶结点的 15*15 分支的评价函数值 α 值 β 值和剪枝效果进行分析即可。

二、算法原理

α-β 剪枝是一种高效的搜索算法优化技术，主要应用于棋类游戏和其他策略性游戏的人工智能决策制定过程中。这种技术建立在极小化极大算法的基础上，通过两个动态变化的参数—— α 和 β 来实现搜索效率的提升。α 参数代表在搜索过程中，当前玩家（通常称为 Maximizer）可以确保的最低分数，而 β 参数则表示对手（Minimizer）所能接受的最高分数。

在算法执行过程中，如果在探索某个分支时发现即便后续走法采取最佳策略，得分也无法超过已知的 α 或 β 阈值，那么这个分支就可以被剪枝，即终止这一路径的进一步探索。这样的剪枝过程显著减少了搜索树的规模，避免了对那些不会影响最终决策结果的路径的无效搜索，从而节省了计算资源和时间。

α-β 剪枝使得搜索算法能够在更深的层次进行决策分析，尽管它需要在搜索开始前设定合适的 α 和 β 值。这种方法提高了搜索的速度，还增强了找到最优策略的能力，是人工智能领域中一项基础而重要的技术。

三、流程图

gobang.py 流程图：

```
开始
|
|---> 初始化游戏
|       |
|       |---> 棋盘初始化
|       |       |
|       |       |---> 如果提供了chess_file则读取文件
|       |       |       |
|       |       |       |---> 根据文件内容初始化棋盘
|       |       |
|       |       |---> 如果没有提供chess_file则初始化空棋盘
|       |
|       |---> 绘制棋盘
|
|---> 主游戏循环
|       |
|       |---> 玩家点击处理
|       |       |
```

```
|       |       |---> 检查点击位置是否合法
|       |       |       |       |
|       |       |       |       |---> 如果合法，则更新棋盘
|       |       |       |       |       |
|       |       |       |       |       |---> 检查是否有玩家获胜
|       |       |       |       |       |
|       |       |       |       |       |---> 如果有玩家获胜，则结束游戏
|       |       |       |       |       |
|       |       |       |       |       |---> 如果没有玩家获胜，则切换玩家
|       |       |       |       |
|       |       |       |       |---> 如果不合法，则提示错误
|       |       |
|       |       |---> AI回合处理
|       |       |       |
|       |       |       |---> 执行AlphaBetaSearch获取最佳位置
|       |       |       |
|       |       |       |---> 更新棋盘
|       |       |       |
|       |       |       |---> 检查是否有玩家获胜
|       |       |       |
|       |       |       |---> 如果有玩家获胜，则结束游戏
|       |       |       |
|       |       |       |---> 如果没有玩家获胜，则切换玩家
|
结束
```

AlphaBeta.py 流程图：

```
开始
|
|---> 初始化AlphaBetaSearch参数
|
|---> 执行深度优先搜索
|       |
|       |---> 获取可下棋子位置
|       |
|       |---> 对每个可下位置进行评估
|       |       |
|       |       |---> 模拟下棋
|       |       |
|       |       |---> 计算当前局势评分
|       |       |
|       |       |---> 进行递归搜索
|       |       |       |
|       |       |       |---> 更新alpha和beta值
|       |       |
|       |       |---> 回溯并恢复棋盘状态
|
|---> 返回最佳下棋位置和评分
|
结束
```

四、关键代码展示

`load_initial_board` 从指定文件加载初始棋局。`chess_file` 字符串，文件路径。如果提供的文件路径有效，读取文件内容并根据文件中的数字（1或0）设置棋盘。1 代表黑棋，0 代表白棋，其他值则为空。

```
def load_initial_board(self, chess_file):
    if chess_file:
        with open(chess_file, "r") as f:
            for i, line in enumerate(f):
                for j, x in enumerate(line.strip().split(" ")):
                    self.board[i][j] = BLACK if x == "1" else WHITE if x == "0"
else EMPTY
```

`click` 处理鼠标点击事件，执行落子操作。`x, y` 鼠标点击的坐标。首先检查游戏是否已经结束，若结束则返回 `False`。然后将坐标转换为棋盘的行列索引。如果点击的位置已被占用，则提示错误并返回。将当前玩家的棋子放在棋盘上，调用 `draw_chess` 绘制棋子。切换当前玩家，检查是否有玩家胜利，如果胜利则更新游戏状态并显示胜利者信息。

```
def click(self, x, y):
    if self.end: return False
    i, j = y // SIDE, x // SIDE
    if self.board[i][j] != EMPTY:
        pygame.display.set_caption("落子错误")
        return False
    self.board[i][j] = BLACK if self.black else WHITE
    self.draw_chess(self.board[i][j], i, j)
    self.black = not self.black

    if self.check_win():
        self.end = True
        pygame.display.set_caption("胜利者: " + ("黑" if self.board[i][j] == BLACK
else "白"))
    return True
```

`check_win` 检查棋盘上是否有玩家获胜。遍历整个棋盘，对于每一个位置调用 `check_chess` 函数判断该位置是否可以构成五子连线。如果找到胜利的连线，返回 `True`，否则返回 `False`。

```
def check_win(self):
    for i in range(ROWS):
        for j in range(ROWS):
            if self.check_chess(i, j):
                return True
    return False
```

`check_chess` 检查指定位置的棋子是否可以形成五子连线。`i, j` 棋盘的行列索引。如果该位置为空，直接返回 `None`。否则，获取该位置棋子的颜色。对每个方向进行检查，使用循环找出所有连续的相同颜色的棋子。如果找到的棋子数量达到 5 个，返回这些棋子的坐标。

```
def check_chess(self, i, j):
    if self.board[i][j] == EMPTY: return None
    color = self.board[i][j]
    for dire in DIRE:
        chess = []
        x, y = i, j
        while 0 <= x < ROWS and 0 <= y < ROWS and self.board[x][y] == color:
            chess.append((x, y))
            x, y = x + dire[0], y + dire[1]
        if len(chess) >= 5:
            return chess
    return None
```

`AI_player` 控制 AI 玩家进行下一步棋。检查当前是否为黑棋的回合，若不是则直接返回。使用 α-β 剪枝算法计算最佳落子位置，然后调用 `click` 方法进行落子。

```
def AI_player(self):
    if not self.black: return
    x, y, alpha = AlphaBetaSearch(self.board, EMPTY, BLACK, WHITE, self.black)
    self.click(y * SIDE, x * SIDE)
```

`evaluate` 估价函数，对整个棋盘进行评分，或者检查某方是否获胜。

```
    def evaluate(self, board, turn, checkWin=False):  # 估价函数
        self.reset()
        mine = turn  # turn 1代表黑子下，0代表白子下
        opponent = abs(1 - turn)
        for y in range(self.len):
            for x in range(self.len):
                if board[y][x] == mine:
                    self.evaluatePoint(board, x, y, mine, opponent)
                elif board[y][x] == opponent:
                    self.evaluatePoint(board, x, y, opponent, mine)
        mine_count = self.count[turn]
        opponent_count = self.count[abs(turn - 1)]
        if checkWin:
            return mine_count[FIVE] > 0
        else:
            mscore, oscore = self.getScore(mine_count, opponent_count)
            return (mscore - oscore)
```

函数 `search` 实现了 α-β 剪枝算法来搜索最佳落子点。先获取所有可落子位置。遍历每个可落子位置，模拟落子，并递归调用 `search` 进行下一步搜索。根据评分更新 `alpha` 和 `beta`，进行剪枝。返回最优的落子点及其得分。

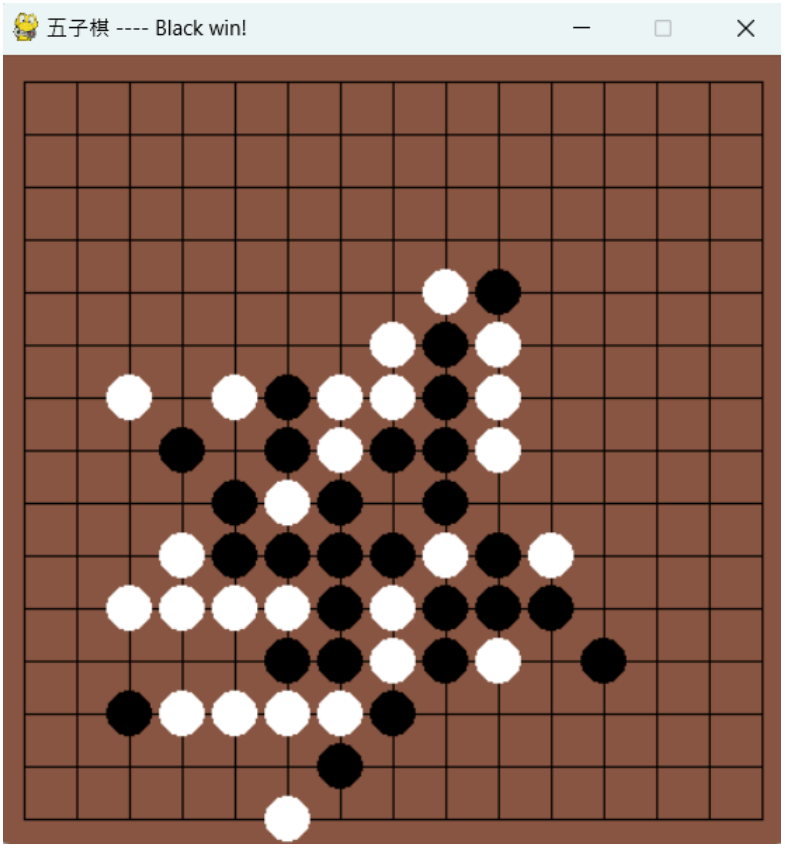
```
def search(board, Ai, alpha, beta, if_max, turn, depth, limit):  # 深度优先搜索
    global sum1
    sum1 += 1
    moves = Ai.get_move(board)
    max_score = -1000000  #
    min_score = 1000000  #
    alpha_tmp = alpha
    beta_tmp = beta
    x_ans = -1
    y_ans = -1
    for move in moves:
        x, y = move[1], move[2]
        board[x][y] = turn
        if turn == 0:
            if Ai.check_win(board, x, y):
                x_ans = x
                y_ans = y
                min_score = -1000000  #
        else:
            if Ai.check_win(board, x, y):
                x_ans = x
                y_ans = y
                max_score = 1000000  #
        if depth == limit:
            sum1 += 1
            score = Ai.evaluate(board, turn)
            if turn % 2 == 0:
                score = -score
        else:
            x_tmp, y_tmp, score = search(board, Ai, alpha_tmp, beta_tmp, 1 -
if_max, 1 - turn, depth + 1, limit)
        board[x][y] = -1
        if if_max == 1:
            if score > max_score:
                x_ans = x
                y_ans = y
                max_score = score
            if max_score > beta_tmp or max_score == beta_tmp:
                break
            if max_score > alpha_tmp:
                alpha_tmp = max_score
        else:
            if score < min_score:
                x_ans = x
                y_ans = y
                min_score = score
            if min_score < alpha_tmp or min_score == alpha_tmp:
                break
            if min_score < beta_tmp:
                beta_tmp = min_score
    if if_max == 1:
        return (x_ans, y_ans, max_score)
    else:
        return (x_ans, y_ans, min_score)
```

函数 `AlphaBetaSearch` 这是搜索的入口函数，重构棋盘后调用 `search` 函数进行搜索，并返回最佳落子点。

```
def AlphaBetaSearch(board1, EMPTY, BLACK, WHITE, black):  # alpha beta剪枝搜索
    board = rebuild(board1)
    Ai = Chess_AI(len(board))
    if_max = 1
    turn = 0
    alpha = -10000000000
    beta = 100000000000
    if black:
        turn = 1
    limit = 2  # 搜索深度
    x, y, score = search(board, Ai, alpha, beta, if_max, turn, 1, limit)
    global sum1
    print("------------------------")
    print(sum1)
    return (x, y, score)
```

五、实验结果分析

第 11 关：



下棋的位置顺序与分数：

```
------------------------
784
```

```
8 8 -90
------------------------
1747
9 5 -96
------------------------
2677
7 5 -1600
------------------------
4063
10 8 -12
------------------------
6581
6 5 -1600
------------------------
10454
10 6 -1598
------------------------
12286
11 5 -8
------------------------
15653
8 4 -1608
------------------------
17376
7 3 -16
------------------------
20707
10 9 -1616
------------------------
22540
9 4 -2002
------------------------
24496
9 6 -2006
------------------------
25589
11 6 -114
------------------------
26629
4 9 -2020
------------------------
29902
12 7 -1626
------------------------
31352
11 8 -1630
------------------------
32558
13 6 -34
------------------------
33493
9 9 -2012
------------------------
36223
12 2 -2002
------------------------
```

```
37268
10 10 -126
------------------------
39091
11 11 1000000
```

六、参考资料

1. https://blog.csdn.net/sheziqiong/article/details/128191179?ops_request_misc=&request_id=&biz_id=102&utm_term=alpha%20beta%E5%89%AA%E6%9E%9D%20%E4%BA%94%E5%AD%90%E6%A3%8B&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-128191179.142