20337270_钟海财_实验4

中山大学计算机学院

本科生实验报告

(2021学年春季学期)

课程名称：Artificial Intelligence

教学班级	学号	专业（方向）	姓名
20级软工+网安	20337270	软件工程	钟海财

一、 实验题目

使用博弈树搜索，使用alpha-beta剪枝算法，实现与对手AI进行中国象棋的对弈。

二、 实验内容

1. 算法原理

1.1 博弈树的概念

在博弈过程中, 任何一方都希望自己取得胜利。因此,当某一方当前有多个行动方案可供选择时, 他总是挑选对自己最为有利而对对方最为不利的那个行动方案。 此时,如果我们**站在A方的立场上,则可供A方选择的若干行动方案之间是“或”关系**, 因为主动权操在A方手里,他或者选择这个行动方案, 或者选择另一个行动方案, 完全由A方自己决定。当A方选取任一方案走了一步后,**B方也有若干个可供选择的行动方案, 此时这些行动方案对A方来说它们之间则是“与”关系**,因为这时主动权操在B方手里,这些可供选择的行动方案中的任何一个都可能被B方选中, A方必须应付每一种情况的发生。

这样,如果站在某一方(如A方,即在A要取胜的意义下), 把上述博弈过程用图表示出来, 则得到的是一棵“与或树”。 描述博弈过程的与或树称为博弈树,它有如下特点:

(1) 博弈的初始格局是初始节点。

(2) 在博弈树中, “或”节点和“与”节点是逐层交替出现的。**自己一方扩展的节点之间是“或”关系, 对方扩展的节点之间是“与”关系。双方轮流地扩展节点。**

(3) 所有自己一方获胜的终局都是本原问题, 相应的节点是可解节点；所有使对方获胜的终局都是不可解节点。

1.2 极小极大值分析法

在二人博弈问题中,为了从众多可供选择的行动方案中选出一个对自己最为有利的行动方案, 就需要对当前的情况以及将要发生的情况进行分析,从中选出最优的走步。最常使用的分析方法是极小极大分析法。其基本思想是:

(1) 设博弈的双方中一方为A,另一方为B。然后为其中的一方(例如A)寻找一个最优行动方案。

(2) 为了找到当前的最优行动方案, 需要对各个可能的方案所产生的后果进行比较。具体地说, 就是要考虑每一方案实施后对方可能采取的所有行动, 并计算可能的得分。

(3) 为计算得分,需要根据问题的特性信息定义一个估价函数, 用来估算当前博弈树端节点的得分。此时估算出来的得分称为静态估值。

(4) 当端节点的估值计算出来后, 再推算出父节点的得分, 推算的方法是:**对“或”节点, 选其子节点中一个最大的得分作为父节点的得分,这是为了使自己在可供选择的方案中选一个对自己最有利的方案;对“与”节点, 选其子节点中一个最小的得分作为父节点的得分,这是为了立足于最坏的情况(但是有时候由于对手的评价方式可能与我们不一样,可能会选择其他的节点,所以有时候也会选择将所有"与"节点的子节点的得分之和作为父节点的得分)。这样计算出的父节点的得分称为倒推值。**

(5) 如果一个行动方案能获得较大的倒推值, 则它就是当前最好的行动方案。

1.3 Alpha-beta剪枝算法

在博弈问题中,每一个格局可供选择的行动方案都有很多, 因此会生成十分庞大的博弈树。试图利用完整的博弈树来进行极小极大分析是困难的。**可行的办法是只生成一定深度的博弈树, 然后进行极小极大分析,找出当前最好的行动方案。**在此之后, 再在已选定的分支上扩展一定深度, 再选最好的行动方案。如此进行下去, 直到取得胜败的结果为止。至于每次生成博弈树的深度, 当然是越大越好, 但由于受到计算机存储空间的限制, 只好根据实际情况而定.

Alpha-beta剪枝的本质就是一种**基于极小化极大算法的改进方法。** 在人机博弈中，双方回合制地进行走棋，己方考虑当自己在所有可行的走法中作出某一特定选择后，对方可能会采取的走法，从而选择最有利于自己的走法。 这种对弈过程就构成了一颗博弈树，双方在博弈树中不断搜索，选择对自己最为有利的子节点走棋。 在搜索的过程中，将取极大值的一方称为max，取极小值的一方称为min。 max总是会选择价值最大的子节点走棋，而min则相反。 这就是极小化极大算法的核心思想。 极小化极大算法最大的缺点就是会造成数据冗余，而这种冗余有两种情况：①极大值冗余；②极小值冗余。 相对应地，alpha剪枝用来解决极大值冗余问题，beta剪枝则用来解决极小值冗余问题，这就构成了完整的Alpha-beta剪枝算法。

基本思想：根据倒推值的计算方法，或中取大，与中取小，在扩展和计算过程中，能剪掉不必要的分枝，提高效率。

定义：

α值：有或后继的节点(极大节点)，取当前子节点中的最大倒推值为其下界，称为α值。节点倒推值>=α；

β值：有与后继的节点(极小节点)，取当前子节点中的最小倒推值为其上界，称为β值。节点倒推值<=β；

α-β 剪枝：

（1） β剪枝：节点x的α值不能降低其父节点的β值，x以下的分支可停止搜索，且x的倒推值为α；

（2） α 剪枝：节点x的β值不能升高其父节点的α值，x以下的分支可停止搜索，且x的倒推值为β ；

2. 伪代码

```
function alphabeta(node, depth, α, β, Player)
```

```
    //达到最深搜索深度或胜负已分
    if  depth = 0 or node is a terminal node
        return the heuristic value of node
    if  Player = MaxPlayer // 极大节点
        for each child of node // 子节点是极小节点
            α := max(α, alphabeta(child, depth-1, α, β, not(Player) ))
            if β ≤ α
            // 该极大节点的值>=α>=β，该极大节点后面的搜索到的值肯定会大于β，因此不会被其上层的极小节点所选用了。
            // 对于根节点，β为正无穷
                 break //beta剪枝
        return α
    else // 极小节点
        for each child of node //子节点是极大节点
            β := min(β, alphabeta(child, depth-1, α, β, not(Player) )) // 极小节点
            if β ≤ α // 该极大节点的值<=β<=α，该极小节点后面的搜索到的值肯定会小于α，因此不会被其上层的极大节点所选用了。
            //对于根节点，α为负无穷
                break //alpha剪枝
        return β
```

3. 关键代码展示（带注释）

首先在main.py中修改主函数，将接口接到我们的AI对象my_ai:

```
import ChessAI as He
import MyAI as Me
...
    # 创建AI对象
    game.computer_team = 'b'  # 电脑先手改为r,电脑后手改为b
    game.user_team = 'b' if game.computer_team == 'r' else 'r'  # 根据电脑先手/后手，改变我方的先后手
    ai = He.ChessAI(game.computer_team)
    my_ai = Me.ChessAI(game.user_team)

    # 主循环
    step = 0    # 步数
    while True:
        # AI行动
        if not game.show_win and not game.show_draw and game.AI_mode and game.get_player() == ai.team:
            print(ai.team, '电脑方动： 第', step, '步')
            if game.back_button.is_repeated():
                print("重复走子，电脑方获胜...")
                game.set_win(game.get_player())
            else:
                # AI预测下一步
                cur_row, cur_col, nxt_row, nxt_col = ai.get_next_step(chessboard)
                # 选择棋子
                ClickBox(screen, cur_row, cur_col)
                # 下棋子
                chessboard.move_chess(nxt_row, nxt_col)
                # 清理「点击对象」
                ClickBox.clean()
                # 检测落子后，是否产生了"将军"功能
                if chessboard.judge_attack_general(game.get_player()):
```

```
                    print("将军....")
                # 检测对方是否可以挽救棋局，如果能挽救，就显示"将军"，否则显示"胜利
                    if chessboard.judge_win(game.get_player()):
                        print("电脑方获胜...")
                        game.set_win(game.get_player())
                    else:
                        # 如果攻击到对方，则标记显示"将军"效果
                        game.set_attack(True)
                else:
                    if chessboard.judge_win(game.get_player()):
                        print("电脑方获胜...")
                        game.set_win(game.get_player())
                    game.set_attack(False)
                if chessboard.judge_draw():
                    print("和棋...")
                    game.set_draw()
                # 落子之后，交换走棋方
            game.back_button.add_history(chessboard.get_chessboard_str_map())
            game.exchange()
        else:
            print(my_ai.team, '我方动： 第', step, '步')
            if game.back_button.is_repeated():
                print("重复走子，我方获胜...")
                game.set_win(game.get_player())
            else:
                # AI预测下一步
                cur_row, cur_col, nxt_row, nxt_col = my_ai.get_next_step(chessboard)
                # 选择棋子
                # 如果是我方后手，需要将位置映射到b黑色方，对称的(x1,y1)(x2,y2)->
x1+x2=9,y1+y2=8
                ClickBox(screen, cur_row, cur_col)
                # 下棋子
                chessboard.move_chess(nxt_row, nxt_col)
                # 清理「点击对象」
                ClickBox.clean()
                # 检测落子后，是否产生了"将军"功能
                if chessboard.judge_attack_general(game.get_player()):
                    print("将军....")
                    # 检测对方是否可以挽救棋局，如果能挽救，就显示"将军"，否则显示"胜利
                    if chessboard.judge_win(game.get_player()):
                        print("我方获胜...")
                        game.set_win(game.get_player())
                    else:
                        # 如果攻击到对方，则标记显示"将军"效果
                        game.set_attack(True)
                else:
                    if chessboard.judge_win(game.get_player()):
                        print("我方获胜...")
                        game.set_win(game.get_player())
                        game.set_attack(False)
                if chessboard.judge_draw():
                    print("和棋...")
                    game.set_draw()
                # 落子之后，交换走棋方
                game.back_button.add_history(chessboard.get_chessboard_str_map())
                game.exchange()
        step += 1
        # 显示游戏背景
        screen.blit(background_img, (0, 0))
        screen.blit(background_img, (0, 270))
        screen.blit(background_img, (0, 540))
```

```
        chessboard.show_chessboard_and_chess()

        # 标记点击的棋子
        ClickBox.show()

        # 显示可以落子的位置图片
        Dot.show_all()

        # 显示游戏相关信息
        game.show()

        # 显示screen这个相框的内容（此时在这个相框中的内容像照片、文字等会显示出来）
        pygame.display.update()

        # FPS（每秒钟显示画面的次数）
        clock.tick(60)  # 通过一定的延时，实现1秒钟能够循环60次
        if game.show_win:
            show_game(screen, background_img, chessboard, game, clock)
            break
    while 1:
        show_game(screen, background_img, chessboard, game, clock)
```

然后在ChessBoard.py的ChessBoard类中新增两个函数：

一个（move_chess2）用于移动棋子得到儿子节点；

另一个（back）用于从儿子节点返回到父亲节点。

```
    def move_chess2(self, old_row, old_col, new_row, new_col):
    # 直接通过出末位置移动棋子得到一个儿子节点(保存在当前位置)
        # 移动位置
        self.chessboard_map[new_row][new_col] = self.chessboard_map[old_row]
[old_col]
        # 修改棋子的属性
        self.chessboard_map[new_row][new_col].update_position(new_row, new_col)
        # 清楚之前位置为None
        self.chessboard_map[old_row][old_col] = None

    def back(self, old_row, old_col, new_row, new_col, father_chess):   # 返回到父亲
节点
        # 移动位置
        self.chessboard_map[new_row][new_col] = self.chessboard_map[old_row]
[old_col]
        # 修改棋子的属性
        self.chessboard_map[new_row][new_col].update_position(new_row, new_col)
        # 恢复原节点的值
        self.chessboard_map[old_row][old_col] = father_chess
        if self.chessboard_map[old_row][old_col]:
            # 修改棋子的属性
            self.chessboard_map[old_row][old_col].update_position(old_row, old_col)
```

最后修改MyAI.py中我们的AI类ClassAI：

新增两个函数：

get_sons函数用来得到己方当前所有走法；

alpha_beta通过alpha/beta剪枝的博弈树来找到最佳走法。

```
max_depth = 3

class ChessAI(object):
    def __init__(self, computer_team):
        self.team = computer_team
        self.my_team = computer_team    # 用于保存我方的队伍r/b
        self.his_team = 'b' if computer_team == 'r' else 'r'
        # 根据我方棋子(r/b)判断对手棋子(b/r)
        self.evaluate_class = Evaluate(self.team)
        self.best_move = (-1, -1, -1, -1)  # 最佳走法

    def get_sons(self, chessboard: ChessBoard):
        # 得到己方当前局面所有的可以移动的棋子的前后位置
        all_chess = chessboard.get_chess()
        all_sons = []
        for chess in all_chess:
            if chess.team == self.team:     # my_team
                for it in chessboard.get_put_down_position(chess):
                    all_sons.append((chess.row, chess.col, it[0], it[1]))
        return all_sons

    def alpha_beta(self, depth, alpha, beta, player, chessboard: ChessBoard):
        # alpha_beta剪枝，以局面值为评估值
        # 搜索到了对应深度或者没有可移动棋子
        if depth == 0:
            return self.evaluate_class.evaluate(chessboard)
        if player:  # 我方，alpha剪枝,或中取大
            self.team = self.my_team  # 我方，alpha剪枝,或中取大
            son_list = self.get_sons(chessboard)  # 儿子节点
            v = float("-inf")  # 负无穷
            for move in son_list:  # 遍历所有子节点，这里子节点是移动棋子步
                old_row, old_col, new_row, new_col = move
                father = chessboard.chessboard_map[new_row][new_col]
                # 保存被覆盖的棋子（如果有的话）,即父节点
                chessboard.move_chess2(old_row, old_col, new_row, new_col)
                # 产生一个儿子节点
                score = self.alpha_beta(depth - 1, alpha, beta, not player,
chessboard)
                # 对儿子节点进行剪枝并得到递推值
                if v < score:  # 取最大的score，我方或中取大
                    if depth == max_depth:
                        self.best_move = move
                    v = score
                if v > alpha:
                    alpha = v
                chessboard.back(new_row, new_col, old_row, old_col, father)
                # 返回父亲节点
                if alpha >= beta:  # 进行alpha剪枝
                    break
            return v  # 返回递推值
        else:   # 对手，beta剪枝,与中取小
            self.team = self.his_team  # 对手，beta剪枝,与中取小
            move_list = self.get_sons(chessboard)  # 儿子节点
            v = float("inf")  # 正无穷
            for move in move_list:
                old_row, old_col, new_row, new_col = move
                father = chessboard.chessboard_map[new_row][new_col]
                # 保存被覆盖的棋子（如果有的话）,即父节点
                chessboard.move_chess2(old_row, old_col, new_row, new_col)
                # 产生一个儿子节点
```

```
                score = self.alpha_beta(depth - 1, alpha, beta, not player,
chessboard)
                # 对儿子节点进行剪枝并得到递推值
                if v > score:  # 取最小的score，对方与中取小，和
                    if depth == max_depth:  # 达到最大深度
                        self.best_move = move
                    v = score
                chessboard.back(new_row, new_col, old_row, old_col, father)
                # 返回父亲节点
                if alpha >= beta:  # 进行beta剪枝
                    break
            return v    # 返回递推值

    def get_next_step(self, chessboard: ChessBoard):
        team = self.team
        max_val = float("inf")  # 正无穷
        min_val = float("-inf")  # 负无穷
        value = self.alpha_beta(max_depth, min_val, max_val, True, chessboard)
        # 开始我方先走，True
        print('该步移动的评估值为：', value)
        self.team = team  # 避免更改了队伍信息
        return self.best_move
```

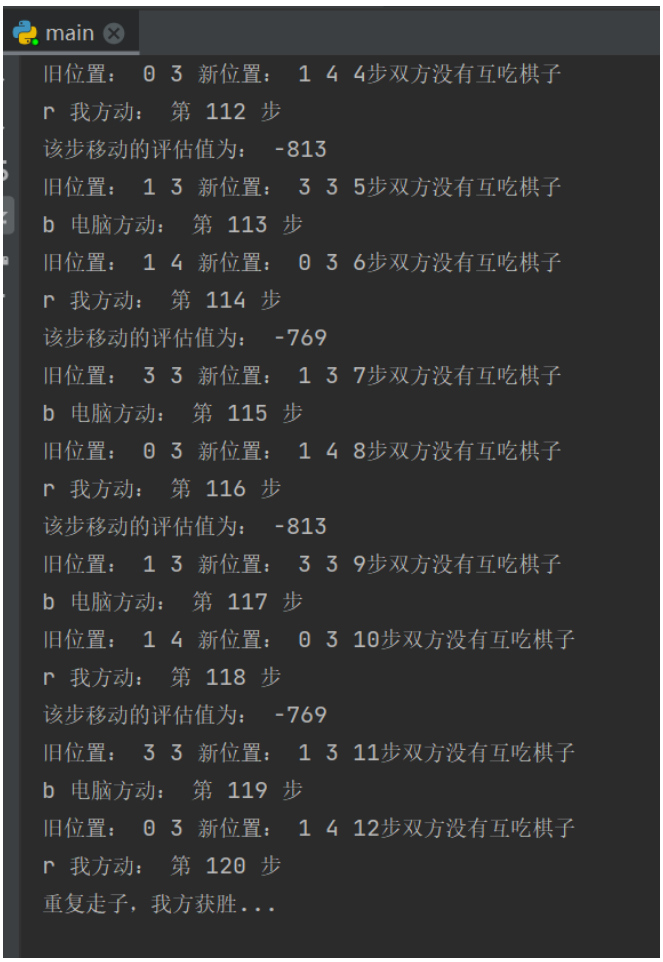
4. 创新点&优化（如果有）

无。

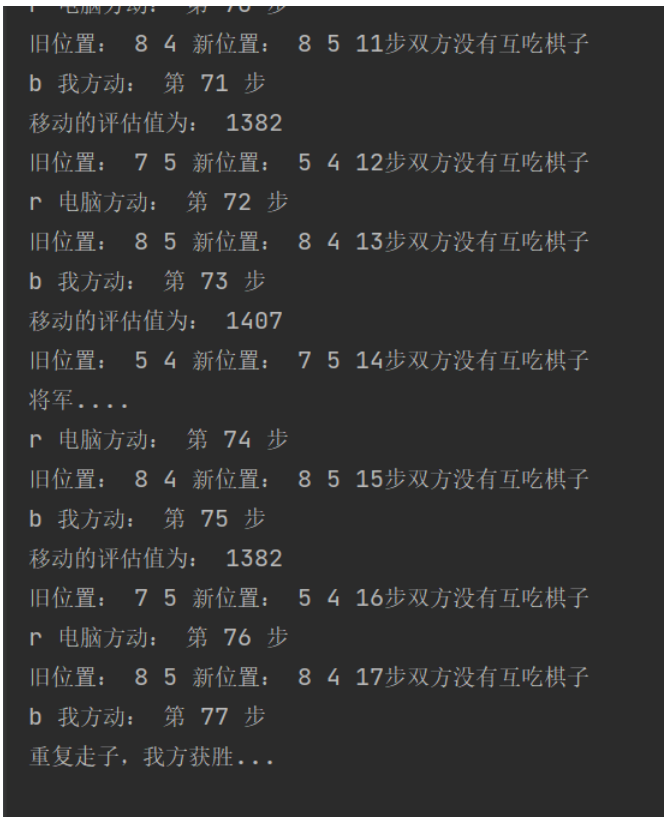
三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

1.1 我方先手：120步获胜（最大搜索深度为3时）



1.2 敌方先手：77步获胜（最大搜索深度为3时）



两次都是因为重复走子获胜。

2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

评测指标：

运行快慢及是否获胜。

分析：

搜索深度	是否胜利（先手，后手）	运行快慢（平均运行时间/s）
1	（否，否）	快（0.001）
2	（否，否）	快（0.1）
3	（是，是）	快（2）
4	（是，是）	非常慢（40）

可见，最大搜索深度为1，2时，不能胜过对手AI；当最大搜索深度>=3时，能胜过对手AI，但随着深度+1，平均运行时间呈指数级增长，特别是深度为4时，运行时间特别久。

所以我选择3作为最大深度。

四、 思考题

本次实验无思考题。

五、 参考资料

1. 实验文档：07_博弈树搜索.pdf
2. 课本：《人工智能》(第三版) 清华大学出版社
3. 课件：03_搜索技术.ppt