中山大学计算机学院
人工智能实验报告
本科生实验报告

课程名称	Artificial Intelligence	学年学期	2022 学年春季学期
教学班级	计科三班	专业	计算机科学与技术
学号	21311592	姓名	刘佳瑜

1 实验目的

编写一个中国象棋博弈程序，要求用 alpha-beta 剪枝算法，可以实现人机对弈

2 实验内容

2.1 实验原理

2.1.1 alpha-beta 剪枝算法

alpha-beta 剪枝算法是一种用于搜索树的优化算法，其基本思想是在搜索过程中，通过比较当前节点的上下界值，来判断当前节点是否需要进一步搜索其子节点，从而剪枝一些无用的节点，在搜索过程中，每个节点都有一个最小值和最大值，表示在当前节点下一步的最好情况和最差情况，对于最大化玩家，它会选择能够使得最大值最大化的节点，而对于最小化玩家，它会选择能够使得最小值最小化的节点。当搜索过程中出现某个节点的值超出了其父节点的上下界值时，该节点的子节点就可以直接被剪枝，因为它们不可能被选择。这种算法可以大幅度减少搜索的节点数，从而提高算法效率。

2.1.2 中国象棋

a.中国象棋游戏规则

1. 棋子走法

将：只能在九宫格内移动，可以直走和斜走，但不能和对方的将直接对决

士：只能在己方的九宫格内移动，走斜线，每次只能走一格

象：走田字形，只能在己方的半边棋盘内移动，不能过河

马：走日字形，如果该位置有己方棋子，则不能移动

车：可以沿着横线或竖线走动，可以走任意格数，但不能斜着走

炮：移动方式和车相似，但吃子时必须隔开一个棋子才能够吃掉对方棋子

兵：只能向前走，不能后退，过河后可以左右走

2. 吃子规则

除了炮和马以外，所有的棋子吃子的方式和移动方式一样。如果一方的棋子吃掉了对方的棋子，被吃掉的棋子就必须离开棋盘，不能再参与比赛

3. 胜负规则

将军：指将的位置被对方棋子攻击，但无法躲避的状态

将死：将死是指一方的将被对方攻击，而对方无法躲避将军的攻击

和棋：如果一方无法将对方将军，而又无法避免自己被将军，则为和棋

b. 中国象棋策略的 alpha-beta 剪枝算法应用

在博弈搜索树中，每个叶子节点的价值代表当前局面的优劣程度，对当前局面进行评价来表示局面的优劣，我通过评价函数来得到对优劣程度的判断，在设计评价函数时，我考虑的因素有：

1. 棋子的固定子力值：棋子本身的价值越大，其具有的固定子力值亦越大，具有更大固定子力值之和的一方占有更大的优势，设计如下：

将	士	象	马	车	炮	卒
10000	250	250	300	300	300	80

2. 棋子的位置价值：象棋局势评价与棋子位置亦有巨大联系

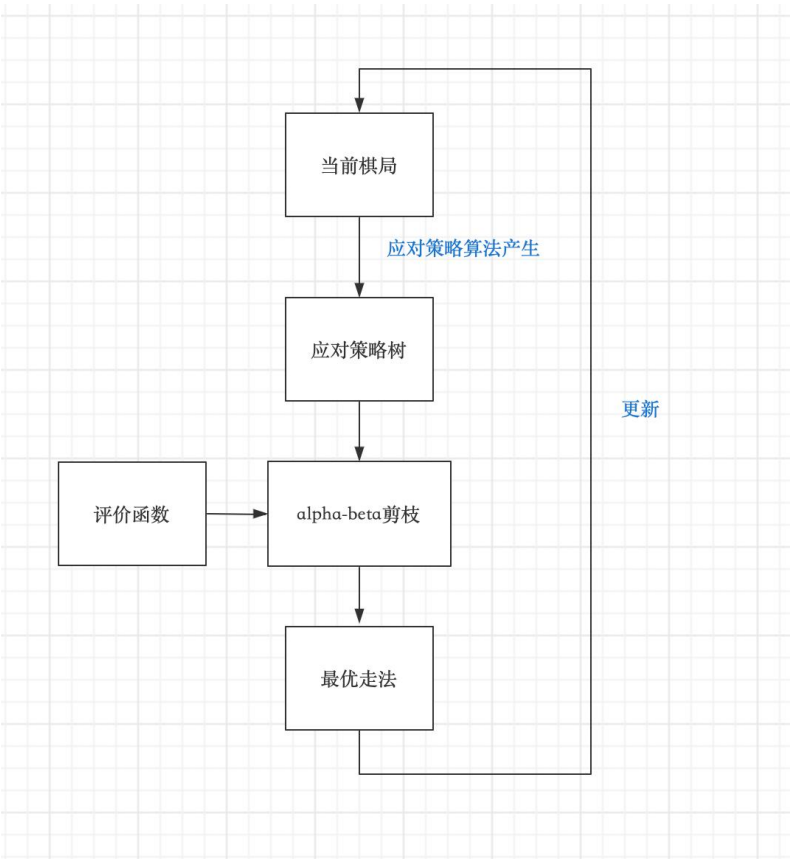
3. 棋子灵活度评估值：活度高的棋子，例如车，可以在战场上快速发挥作用，评价值较高，而灵活度较差的棋子则评分较低，设计如下：

将	士	象	马	车	炮	卒
0	1	1	12	6	6	15

4. 棋子威胁与保护评估值：当棋子处于威胁之中时，应当降低评价值；当棋子处于保护之中时，应当增加评价值

2.2 实验流程图

评估与搜索的流程图如下所示：



2.3 关键代码展示

1.alpha-beta 剪枝算法函数

```
def alpha_beta(self, depth, alpha, beta):
        #判断当前玩家
who = (self.max_depth - depth) % 2
#判断游戏是否结束
        if self.is_game_over(who):
            return cc.min_val
        #判断是否到达指定深度
        if depth == 1:
            #print(self.evaluate(who))
            return self.evaluate(who)

        #利用应对策略得到所有可能的走法
        move_list = self.board.generate_move(who)
        # 利用历史表得到分数
        for i in range(len(move_list)):
            move_list[i].score =
self.history_table.get_history_score(who, move_list[i])
        #排序
move_list.sort()
        best_step = move_list[0]
        score_list = []
        for step in move_list:
            temp = self.move_to(step)
            score = -self.alpha_beta(depth - 1, -beta, -alpha)
score_list.append(score)
            self.undo_move(step, temp)
            #进行剪枝
            if score > alpha:
```

```
                alpha = score
                if depth == self.max_depth:
                    self.best_move = step
                best_step = step
            if alpha >= beta:
                best_step = step
                break
        #更新历史表
        if best_step.from_x != -1:
            self.history_table.add_history_score(who, best_step, depth)
        return alpha
```

2.当前局面评估代码

```
    for x in range(9):
        for y in range(10):
            num_attacked = relation_list[x][y].num_attacked
            num_guarded = relation_list[x][y].num_guarded
            now_chess = self.board.board[x][y]
            type = now_chess.chess_type
            now = now_chess.belong
            unit_val = cc.base_val[now_chess.chess_type] >> 3
            sum_attack = 0  # 被攻击总子力
            sum_guard = 0
            min_attack = 999  # 最小的攻击者
            max_attack = 0  # 最大的攻击者
            max_guard = 0
            flag = 999  # 有没有比这个子的子力小的
            if type == cc.kong:
```

```
                continue
            # 统计攻击方的子力
            for i in range(num_attacked):
                temp = cc.base_val[relation_list[x][y].attacked[i]]
                flag = min(flag, min(temp, cc.base_val[type]))
                min_attack = min(min_attack, temp)
                max_attack = max(max_attack, temp)
                sum_attack += temp
            # 统计防守方的子力
            for i in range(num_guarded):
                temp = cc.base_val[relation_list[x][y].guarded[i]]
                max_guard = max(max_guard, temp)
                sum_guard += temp
            if num_attacked == 0:
                relation_val[now] += 5 * relation_list[x][y].num_guarded
            else:
                muti_val = 5 if who != now else 1
                if num_guarded == 0:  # 如果没有保护
                    relation_val[now] -= muti_val * unit_val
                else:  # 如果有保护
                    if flag != 999:  # 存在攻击者子力小于被攻击者子力,对方将愿意换子
                        relation_val[now] -= muti_val * unit_val
                        relation_val[1 - now] -= muti_val * (flag >> 3)
                    #如是二换一且最小子力小于被攻击者与保护者子力之和，则对方可能以一子换两子
                    elif num_guarded == 1 and num_attacked > 1 and min_attack <
cc.base_val[type] + sum_guard:
                        relation_val[now] -= muti_val * unit_val
                        relation_val[now] -= muti_val * (sum_guard >> 3)
                        relation_val[1 - now] -= muti_val * (flag >> 3)
```

```
            #如是三换二并且攻击者子力较小的二者之和小于被攻击者子力与保护者子力之和，则对方可能以两子换三子
                elif num_guarded == 2 and num_attacked == 3 and sum_attack - max_attack < cc.base_val[type] + sum_guard:
                    relation_val[now] -= muti_val * unit_val
                    relation_val[now] -= muti_val * (sum_guard >> 3)
                    relation_val[1 - now] -= muti_val * ((sum_attack - max_attack) >> 3)
            #如果是 n 换 n，攻击方与保护方数量相同并且攻击者子力小于被攻击者子力与保护者子力之和再减去保护者中最大子力，则对方可能以 n 子换 n 子
                elif num_guarded == num_attacked and sum_attack < cc.base_val[now_chess.chess_type] + sum_guard - max_guard:
                    relation_val[now] -= muti_val * unit_val
                    relation_val[now] -= muti_val * ((sum_guard - max_guard) >> 3)
                    relation_val[1 - now] -= sum_attack >> 3
```

3. 人机交互设计逻辑

```
    def PutdownPieces(self, t, x, y):
      selectfilter=list(filter(lambda cm: cm.x == x and cm.y == y and cm.player == MainGame.player1Color,MainGame.piecesList))
      if len(selectfilter):
         MainGame.piecesSelected = selectfilter[0]
         return

      if MainGame.piecesSelected :
         #print("1111")
```

```
            arr = pieces.listPiecestoArr(MainGame.piecesList)
            if MainGame.piecesSelected.canmove(arr, x, y):
                self.PiecesMove(MainGame.piecesSelected, x, y)
                MainGame.Putdownflag = MainGame.player2Color
        else:
            fi = filter(lambda p: p.x == x and p.y == y, MainGame.piecesList)
            listfi = list(fi)
            if len(listfi) != 0:
                MainGame.piecesSelected = listfi[0]


    def PiecesMove(self,pieces,  x , y):
        for item in  MainGame.piecesList:
            if item.x ==x and item.y == y:
                MainGame.piecesList.remove(item)
        pieces.x = x
        pieces.y = y
        print("move to " +str(x) +" "+str(y))
        return True


    def Computerplay(self):


        if MainGame.Putdownflag == MainGame.player2Color:
            print("轮到电脑了")


            computermove = computer.getPlayInfo(MainGame.piecesList,
self.from_x, self.from_y, self.to_x, self.to_y, self.mgInit)
            if computer==None:
                return
            piecemove = None
```

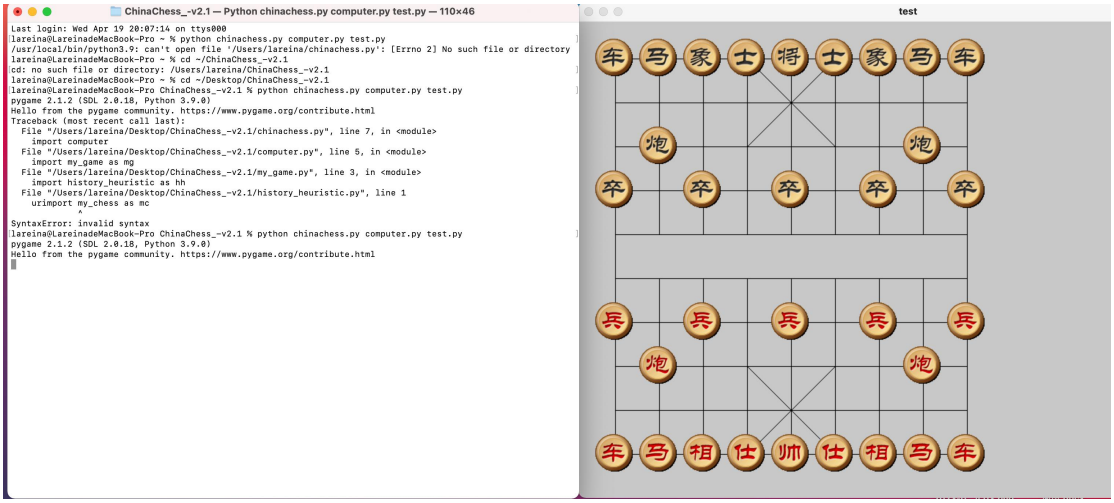
```
            for item in MainGame.piecesList:
                if item.x == computermove[0] and item.y == computermove[1]:
                    piecemove= item


            self.PiecesMove(piecemove, computermove[2], computermove[3])
            MainGame.Putdownflag = MainGame.player1Color


    #判断游戏胜利
    def VictoryOrDefeat(self):
        txt =""
        result = [MainGame.player1Color,MainGame.player2Color]
        for item in MainGame.piecesList:
            if type(item) ==pieces.King:
                if item.player == MainGame.player1Color:
                    result.remove(MainGame.player1Color)
                if item.player == MainGame.player2Color:
                    result.remove(MainGame.player2Color)


        if len(result)==0:
            return
        if result[0] == MainGame.player1Color :
            txt = "失败！"
        else:
            txt = "胜利！"
        MainGame.window.blit(self.getTextSuface("%s" % txt),
(constants.SCREEN_WIDTH - 100, 200))
        MainGame.Putdownflag = constants.overColor
```

3 实验结果展示



4 实验参考资料

[1] 埃里克·马瑟斯著，袁国忠译,《 Python 编程：从入门到实践》，人民邮电出版社，2016-07
[2] 马克· 卢茨著， 秦鹤译， 《 python 学习手册》 ， 机械工业出版社， 2018-10
[3] https://blog.csdn.net/weixin_43398590/article/details/106321557
[4] https://en.wikipedia.org/wiki/Xiangqi