中山大学计算机学院人工智能本科生实验报告

2022学年春季学期

课程名称：Artificial Intelligence

教学班级	人工智能（陈川）	专业（方向）	计算机科学与技术人工智能与大数据
学号	20337025	姓名	崔璨明

一、实验题目

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。可以基于提供的包括棋局评估方法、下棋界面的程序补充完成alpha-beta剪枝的深度优先Minimax算法；另外，可以从 **界面优化、增加新的功能、参考已有文献实现其他评估函数、优化剪枝策略使搜索深度提高** 等方面进行改进，可作为实验的创新点得到相应加分。

二、实验内容

1、算法原理

MiniMax策略

MiniMax搜索策略是双发博弈问题中常用的搜索策略，设博弈双方中一方为MAX，一方为MIN，设计相关的算法为其中的一方寻找一个最优的行动方案。为了选取最优的行动步骤，需要对下一步的状态进行评分，此时需要设计一个**评估函数**，评估函数的好坏将会影响MiniMAX策略的性能。

设我们在博弈的过程中可以得到一棵博弈树，每一层轮流表示MAX方和MIN方的决策，博弈树的叶子结点为最终状态的得分（在限制深度内）。对于MAX决策的层，MAX结点将会选取其子结点中得分最大的结点作为自己的决策；对于MIN决策的层，MIN结点将会选取其子结点中得分最小的结点作为自己的决策，依次由下往上反推，得到最终的决策。

Alpha-beta剪枝的深度优先Minimax算法

alpha-beta剪枝对Minimax算法进行了优化。在MIN、MAX不断的倒推过程中若满足某些条件，则可以去掉一些不必要的搜索分支以提高算法效率，例如一个MAX结点在搜索过程中得到的值大于其父结点（MIN结点）当前的值时，它的搜索便可以终止，因为双方都是选择相对于同一评估函数的最优策略，因此不会选择这一分支，如此便可以剪掉一部分的搜索分支。

设MAX层的下界为alpha，MIN层的上界为beta，alpha-beta剪枝规则描述如下：

- alpha剪枝。若任一MIN值层结点的beta值不大于它任一前驱MAX值层结点的alpha值，即alpha(前驱层) >= beta(后继层)，则可终止该MIN层中这个MIN结点以下的搜索过程。这个MIN结点最终的倒推值就确定为这个beta值。
- beta剪枝。若任意MAX层结点的alpha值不小于它任一前驱MIN层结点的beta值，即alpha(后继层) >= beta(前驱层)，则可以终止该MAX层中这个MAX结点以下的搜索过程，这个MAX结点最终倒推值就确定为这个alpha值。

2、算法伪代码

Alpha-beta剪枝的深度优先Minimax算法伪代码如下：

```
input:搜索起点state,博弈树限制深度MAX_depth,起始的alpha和beta值a,b,当前的博弈者
node
output:做出得分最大的决策

function alpha-beta(state,depth,a,b,node):
        if(depth==MAX_dapth) or state==last_state//深度限制已到或为最终状态
                return heuristic(state) //返回评估函数的得分
        if node == MAX:
                for next_state in next(state)://对于每个当前状态可以到达的状态
                        temp=max(a,alpha-
beta(next_state,depth+1,a,b,~node))
                        if b<=temp: break//不需要扩展了，剪枝
                        if(temp>a and depth==1):
                                a=temp
                                save(command)//保存决策
                return a
        else:
                for next_state in next(state)://对于每个当前状态可以到达的状态
                        b=min(b,alpha-beta(next_state,depth+1,a,b,~node))
                        if b<=a: break//不需要扩展了，剪枝

                return b
```

3、关键代码展示

在实验给出的程序中补充完成 *alpha_beta()* 函数和 *get_next_step* 函数，代码如下：

```
def get_next_step(self, chessboard: ChessBoard):
        '''
        该函数应当返回四个值:
        1 要操作棋子的横坐标
        2 要操作棋子的纵坐标
        3 落子的横坐标
        4 落子的纵坐标
        '''
        self.alpha_beta(1,-999999,999999,chessboard)
```

```
        return self.old_pos+self.new_pos
#raise NotImplementedError("Cannot determin next step!! Implement function
ChessAI::get_next_step !!")
```

```
def alpha_beta(self, depth, a, b, chessboard: ChessBoard):
        if depth>=self.max_depth:
                return self.evaluate_class.evaluate(chessboard) '''到达深度
限制则返回评估值'''
        else:
                chess_list=chessboard.get_chess() #获取所有棋子对象
                for cs in chess_list:
                        #该层为max
                        if depth%2==1 and cs.team ==self.team:
                                next=chessboard.get_put_down_position(cs)
'''获取当前棋子可以走的列表'''
                                for new_x,new_y in next:
                                        last_x=cs.row
                                        last_y=cs.col
                                        '''保存下一步位置上的棋'''

origin_chess=chessboard.chessboard_map[new_x][new_y]
                                        '''走到下一步'''
                                        chessboard.chessboard_map[new_x]
[new_y]=chessboard.chessboard_map[last_x][last_y]
                                        '''更新图片'''
                                        chessboard.chessboard_map[new_x]
[new_y].update_position(new_x,new_y)
                                        '''原来的位置置为空'''
                                        chessboard.chessboard_map[last_x]
[last_y]=None
                                        '''深度优先搜索'''

temp=self.alpha_beta(depth+1,a,b,chessboard)
                                        '''复原当前棋局'''
                                        chessboard.chessboard_map[last_x]
[last_y]=chessboard.chessboard_map[new_x][new_y]
                                        chessboard.chessboard_map[last_x]
[last_y].update_position(last_x,last_y)
                                        chessboard.chessboard_map[new_x]
[new_y]=origin_chess
                                        '''1、得分大于当前值或者还没赋值
                                        2、如果是第一层,则可以设置要移动的坐
标'''
                                        if(temp>a or not self.old_pos) and
depth==1:
                                                self.old_pos=
[cs.row,cs.col]
                                                self.new_pos=[new_x,new_y]
                                        a=max(a,temp)
                                        if b<=a:'''剪枝'''
```

```
                                        return a
                        '''该层为min'''
                        elif depth%2==0 and cs.team!=self.team:
                                next=chessboard.get_put_down_position(cs)
'''获取当前棋子可以走的列表'''
                                for new_x,new_y in next:
                                        last_x=cs.row
                                        last_y=cs.col
                                        '''保存下一步位置的棋'''

origin_chess=chessboard.chessboard_map[new_x][new_y]
                                        '''走到下一步'''
                                        chessboard.chessboard_map[new_x]
[new_y]=chessboard.chessboard_map[last_x][last_y]
                                        '''更新图片'''
                                        chessboard.chessboard_map[new_x]
[new_y].update_position(new_x,new_y)
                                        chessboard.chessboard_map[last_x]
[last_y]=None
                                        '''深度优先搜索'''

temp=self.alpha_beta(depth+1,a,b,chessboard)
                                        '''复原当前棋局'''
                                        chessboard.chessboard_map[last_x]
[last_y]=chessboard.chessboard_map[new_x][new_y]
                                        chessboard.chessboard_map[last_x]
[last_y].update_position(last_x,last_y)
                                        chessboard.chessboard_map[new_x]
[new_y]=origin_chess
                                        b=min(b,temp)
                                        if b<=a:#剪枝
                                                return b
                if depth%2==1:
                        return a
                else:
                        return b
#raise NotImplementedError("Method not implemented!!!")
```

4、创新点&优化

1.界面优化

为了使下棋的界面更加完善和美观，看起来像下象棋的小游戏一样，我给程序增加了背景音乐、获胜的特效、更换了背景图还有增加了必要的组件，具体代码如下：

```python
def main():
    # 初始化pygame
    pygame.init()
    #设置字体
    font1=pygame.font.SysFont('arial', 30, bold=True)
    font2=pygame.font.SysFont('arial', 20, bold=True)
    # 创建用来显示画面的对象（理解为相框）

    pygame.mixer.init()
    pygame.mixer.music.load('F:\chess\\alpha-beta-AIchess\images\\bgm.mp3')
    pygame.mixer.music.play(-1,0)


    screen = pygame.display.set_mode((1000, 730))

    #broadcast=pygame.display.set_mode((100,200))
    # 游戏背景图片
    background_img = pygame.image.load("F:\chess\\alpha-beta-AIchess\images\\bg2.jpg")
```

实现效果如下：

2.增加新的功能

在原有的程序下增加了以下功能：

1、认输按钮，玩家可以通过认输按钮重置棋局。具体实现为创建一个投降类，通过`class game`统一管理，再为其编写按钮元件，当事件触发（用户按下认输按钮）时进行投降，系统判定AI胜利并重置棋局，核心代码如下：

```python
class Reset(pygame.sprite.Sprite):
    def __init__(self,screen):
        self.screen=screen
        self.image=pygame.image.load("F:\chess\\alpha-beta-AIchess\images\\btn_lose.png")
        self.rect = self.image.get_rect()
        self.rect.topleft = (615, 250)

    def show(self):
        self.screen.blit(self.image, self.rect)

    def clicked_back(self, chessboard: ChessBoard, event):
        if event.type == pygame.MOUSEBUTTONUP and self.rect.collidepoint(event.pos):
            print("红方认输，重新开始！")
            return True
```

2、退出游戏按钮，玩家可以通过此按钮进行退出，同样创建一个退出类，通过`class game`统一管理，再为其编写按钮元件，关键代码如下：

```
class Exit(pygame.sprite.Sprite):
    def __init__(self,screen):
        self.screen=screen
        self.image=pygame.image.load("F:\chess\\alpha-beta-AIchess\images\\btn_exit.png")
        self.rect = self.image.get_rect()
        self.rect.topleft = (200, 660)
    def show(self):
        self.screen.blit(self.image, self.rect)

    def clicked_back(self, chessboard: ChessBoard, event):
        if event.type == pygame.MOUSEBUTTONUP and self.rect.collidepoint(event.pos):
            print("退出游戏！！")
            return True
```

3、即使战报，在界面的右侧，可以即使播报战局的情况。主要通过设置字体、创建消息数组，当用户或AI进行操作时记录具体操作然后每次刷新时进行播报，关键代码如下：

```
......
#设置字体
font1=pygame.font.SysFont('arial', 30, bold=True)
font2=pygame.font.SysFont('arial', 20, bold=True)
screen.blit(font1.render('War Report',True,[0,0,0]),[750,100])
first=150
if len(message)>9:
        templ=len(message)
message=message[templ-9:templ]
for i in message:
        screen.blit(font2.render(i,True,[0,0,0]),[750,first])
        first+=50
......
```

3.参考已有文献实现其他评估函数

参考西安理工大学谢艳茹硕士的论文《中国象棋计算机博弈数据结构与评估函数的研究和实现》，得到一个可靠的评估函数为 **棋力值 + 位置值或残局时的值 +灵活性**，给出的样例程序中每个棋子的位置值是固定的，但我们知道，同一个棋子在开中局和残局时对局面的影响也是不同的，比如残局的兵如果运用得当，甚至可以将死对方，中局过河兵和残局兵的数量非常重要。因此我参考资料，增加了棋子在残局时的位置值，根据步数来判断是否进入残局，切换位置值数组，并据此修改了`get_chess_pos_point()`函数和增加了灵活度的函数

`get_chess_linhuo_point`关键代码如下：

```
def evaluate(self, chessboard: ChessBoard):
    point = 0
    for chess in chessboard.get_chess():
        point += self.get_single_chess_point(chess)
        point += self.get_chess_pos_point(chess)
        point += self.get_chess_linhuo_point(chess)
        #根据参考资料的评估函数，返回子力值 + 位置值或残局值 +灵活性
    self.judge_count+=1
    if self.judge_count>50 :
        self.judge_count=0
    return point
```

三、实验结果及分析

运行程序，并进行和程序进行博弈，可以看到对于我们的操作，程序能够在较快时间内做出反应，且具有一定的实力，能够对付象棋初学者，具体演示结果在附件`/result`中有由视频演示。
但对于较高水平的玩家则程序十分吃力，且会出现很多“无用步”，因此程序还有很大的提升空间。

四、参考资料

- 《中国象棋计算机博弈数据结构与评估函数的研究和实现》