实验报告

（ 2021/2022 学年 第 二 学期）

题 目：**基于Minimax和α-β剪枝的井子棋游戏**

|  |  |
| --- | --- |
| **专 业** |  |
| **学号姓名** |  |
| **指 导 教 师** |  |
| **指 导 单 位** |  |
| **日 期** |  |

**基于Minimax和α-β剪枝的井子棋游戏**

**一、课题内容和要求**

井子棋是黑白棋的一种。要求玩家轮流在有九个小方格的正方形内划叉号或者圆圈，谁先把三个同一种标记横着、竖着、斜着排成一条线，谁就是胜者。本程序可以让两个玩家或者一个玩家一个基于Minimax算法的AI来进行井子棋对弈。

**二、问题分析**

井字棋的结构框架如图1所示。

井字棋

棋盘

胜负判定

选手定义

落子

图１结构框架图

落子：应该用什么来落子，如何落到棋盘里，什么样的落子是合法落子。

选手定义：人类玩家和AI玩家如何定义，如何进行动作，进行动作的方式是什么，AI玩家如何来“思考”棋局。

胜负判定：当棋盘满足什么条件时判断胜负。

棋盘：棋盘的储存方式，棋盘的打印方式等

**三、概要设计**

主要用到的类以及其继承关系：

基本

玩家

棋盘

AI

人类

游戏本体

棋盘类：

在本程序中棋盘是以二维矩阵存储的，空状态以“-”来表示，被玩家占用后使用玩家的标志来储存。棋盘通过遍历的方法来判断胜利，同样通过遍历的方法来判断是否平局。棋盘还有判断并得出合法落子的函数，结果以元组数组储存。

基本玩家：

其属性包括了玩家的落子标志，定义了两类玩家共同的行为，比如落子。

AI玩家：

其主要包含Minimax算法和α-β剪枝来进行对弈的“思考”行为。

人类玩家：

其主要包含接收外部玩家发送的消息并作出相应的动作。

游戏本体：

其主要包括了游戏初始化，玩家初始化，如选择玩家类型，交换落子玩家，以及通过上述的类和方法进行整个游戏的进行。

游戏的主要流程图：

选择两个玩家的属性

开始游戏

交换玩家

玩家思考并得出落子位置

玩家落子

判断结束

否

打印结果

游戏结束

是

主要函数解析：

Minimax算法计算最佳落子位置：

**def** minimax(self, board, depth, player, alpha, beta): *# , alpha, beta):* **if** self.take == **'O'**:  
 bestval = -10  
 **else**:  
 bestval = 10  
 score = board.check\_winner()  
 **if** score == 10:  
 **return** -10 + depth, **None  
 elif** score == -10:  
 **return** 10 - depth, **None  
 elif** board.is\_left():  
 **return** 0, **None** *# 返回0表示平局* **for** action **in** board.get\_legal\_action():  
 board.move(action, self.take)  
 val, \_ = player.minimax(board, depth + 1, self, alpha, beta)  
 board.move(action, **'-'**)  
  
 **if** self.take == **'O'**:  
 **if** val > bestval:  
 alpha = val  
 bestval, bestaction = val, action  
 **if** alpha > beta:  
 **break  
 else**:  
 **if** val < bestval:  
 beta = val  
 bestval, bestaction = val, action  
 **if** alpha > beta:  
 **break  
 return** bestval, bestaction

Minimax算法把一个局面映射成一个值，这个值就代表着当下赢面的大小，我们要在遍历树的过程中找到值最大的局面。然后通过for语句在合法的落子范围内进行遍历下棋，并且模拟出一个“假想敌”且他足够聪明来使我的值最小。规定人类方胜利的时候值为-10加上深度，我方胜利的时候值为10-深度，值与深度也有关系，在值越大的时候深度必然越小，这就意味着AI必须尽可能快的结束这场游戏。最后加上α-β剪枝来减少树展开的时间复杂度。

AI的思考过程如下：

**def** think(self, board):  
 take = [**'X'**, **'O'**][self.take == **'X'**]  
 player = AIPlayer(take) *# 创建假想敌*  
 \_, action = self.minimax(board, 0, player, -10000, 10000)

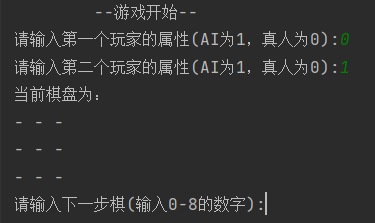
游戏运行函数：

**def** run(self):  
 print(**'--游戏开始--'**)  
 p1 = int(input(**"请输入第一个玩家的属性(AI为1，真人为0):"**))  
 p2 = int(input(**"请输入第二个玩家的属性(AI为1，真人为0):"**))  
 player1 = self.initialize\_player(p1, **'X'**)  
 player2 = self.initialize\_player(p2, **'O'**)  
 **while True**:  
 self.current\_player = self.switch\_player(player1, player2)  
 action = self.current\_player.think(self.board)  
 self.board.move(action, self.current\_player.take)  
 self.board.display()  
 **if** self.board.check\_winner() == 10:  
 print(**"Player1胜利啦！"**)  
 **break  
 elif** self.board.check\_winner() == -10:  
 print(**"Player2胜利啦！"**)  
 **break  
 elif** self.board.is\_left():  
 print(**"你们不分上下噢"**)  
 **break**

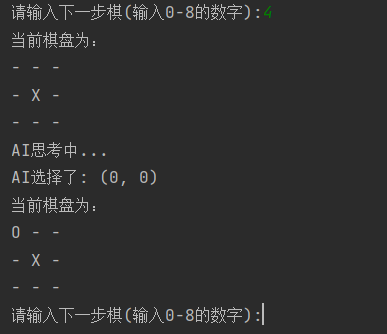
通过一个永真的循环来不停地让两个玩家交替下棋，知道分出胜负通过break来结束循环。

**四、运行结果**

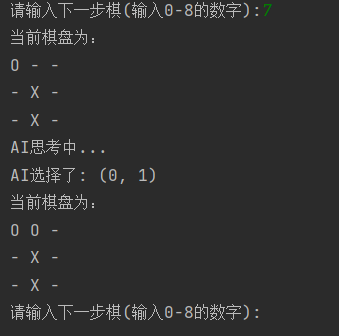
选择真人对战AI



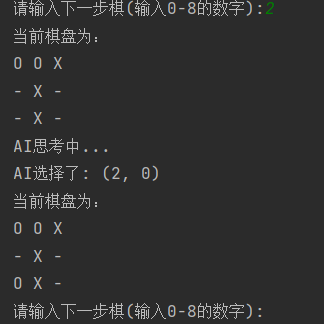
**下4号位**



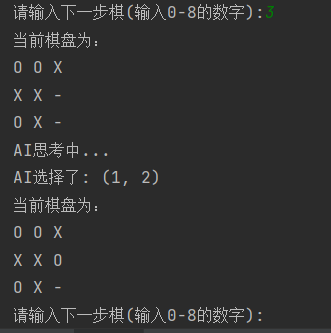
**下7号位**



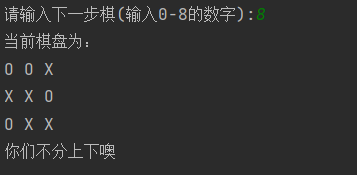
**下2号位**



下3号位



下8号位



源代码：

Player = **'X'** *# 玩家的标志*Opponent = **'O'** *# 对手的标志*transform = {  
 0: (0, 0),  
 1: (0, 1),  
 2: (0, 2),  
 3: (1, 0),  
 4: (1, 1),  
 5: (1, 2),  
 6: (2, 0),  
 7: (2, 1),  
 8: (2, 2)  
} *# 建立玩家输入位置与矩阵坐标的对应关系  
  
  
# 棋盘类***class** Board:  
 **def** \_\_init\_\_(self):  
 self.board = [[**'-'**] \* 3 **for** \_ **in** range(3)] *# 初始化棋盘，以二维数组来储存* **def** check\_winner(self): *# 遍历来判定胜负* board = self.board  
  
 *# 判断横行是否胜利* **for** i **in** range(3):  
 **if** board[i][0] == board[i][1] **and** board[i][1] == board[i][2]:  
 **if** board[i][0] == Player:  
 **return** 10 *# 如果X胜返回10* **elif** board[i][0] == Opponent:  
 **return** -10 *# O胜返回-10  
 # 判断竖列是否胜利* **for** i **in** range(3):  
 **if** board[0][i] == board[1][i] **and** board[1][i] == board[2][i]:  
 **if** board[0][i] == Player:  
 **return** 10  
 **elif** board[0][i] == Opponent:  
 **return** -10  
 *# 判断对角线是否胜利* **if** (board[0][0] == board[1][1] **and** board[1][1] == board[2][2]) \  
 **or** (board[0][2] == board[1][1] **and** board[1][1] == board[2][0]):  
 **if** board[1][1] == Player:  
 **return** 10  
 **elif** board[1][1] == Opponent:  
 **return** -10  
 **return** 0  
  
 **def** is\_left(self):  
 **for** i **in** range(3):  
 **for** j **in** range(3):  
 **if** self.board[i][j] == **'-'**:  
 **return False** *# 当出现'\_'时没有结束棋局* **return True** *# 遍历之后仍没有出现'\_'，表明已经结束* **def** display(self): *# 打印棋盘* print(**"当前棋盘为："**)  
 **for** i **in** range(3):  
 print(self.board[i][0], self.board[i][1], self.board[i][2])  
  
 **def** is\_legal\_position(self, i, j): *# 判断棋盘的[i][j]位是否被占用* **if** self.board[i][j] == **'-'**:  
 **return True  
 else**:  
 **return False  
  
 def** move(self, action, player): *# 落子，第一个参数为落子位置，第二个参数为落子的内容* self.board[action[0]][action[1]] = player  
  
 **def** get\_legal\_action(self): *# 遍历棋盘得到所有合法的落子位置并返回其数组* actions = []  
 **for** i **in** range(3):  
 **for** j **in** range(3):  
 **if** self.is\_legal\_position(i, j):  
 actions.append((i, j))  
 **return** actions  
  
  
*# 基本玩家类***class** BasePlayer(object):  
 **def** \_\_init\_\_(self, take=**'X'**):  
 self.take = take *# 玩家执的棋子类型，默认为X* **def** move(self, board, action): *# 玩家落子* board.move(action, self.take)  
  
 **def** think(self, board): *# 玩家思考，在派生类中具体实现* **pass***# 人类玩家，继承于基本玩家***class** Human(BasePlayer):  
 **def** \_\_init\_\_(self, take):  
 super().\_\_init\_\_(take)  
  
 **def** think(self, board): *# 玩家思考* **while True**: *# 永真循环，直到输入正确的数字* position = int(input(**"请输入下一步棋(输入0-8的数字):"**)) *# 获得玩家落子位置* **try**:  
 action = transform[position]  
 **while not** board.is\_legal\_position(action[0], action[1]): *# 直到输入合法落子位置* print(**"输入错误"**)  
 position = int(input(**"请输入下一步棋(输入0-8的数字):"**))  
 action = transform[position] *# 把数字转换为坐标* **return** action  
 **except**:  
 print(**"输入错误，请重新输入"**)  
  
  
*# AI玩家***class** AIPlayer(BasePlayer):  
 **def** \_\_init\_\_(self, take):  
 super().\_\_init\_\_(take)  
  
 **def** think(self, board): *# 玩家思考* print(**"AI思考中..."**)  
 take = [**'X'**, **'O'**][self.take == **'X'**] *# 得到与AI相反的标志* player = AIPlayer(take) *#使用上述标志创建假想敌* \_, action = self.minimax(board, 0, player, -10000, 10000) *# Minimax算法来计算最佳落子* print(**"AI选择了:"**, action)  
 **return** action  
  
 *# minimax算法来计算最佳落子位置  
 # 第一个参数为棋盘，第二个参数为深度，第三个参数为假想敌，第四个参数为alpha-beta剪枝中的下界，第五个参数为alpha-beta剪枝中的上届* **def** minimax(self, board, depth, player, alpha, beta):  
 **if** self.take == **'O'**:  
 bestval = -10 *# 初始化让局面值最小* **else**:  
 bestval = 10 *# 初始化让局面值最大* score = board.check\_winner() *# 判断是否胜利* **if** score == 10:  
 **return** -10 + depth, **None** *# 人类赢了使值尽可能小* **elif** score == -10:  
 **return** 10 - depth, **None** *# AI赢了使值尽可能大* **elif** board.is\_left():  
 **return** 0, **None** *# 返回0表示平局* **for** action **in** board.get\_legal\_action(): *# 遍历每一个合法走法* board.move(action, self.take) *# 假设走一步* val, \_ = player.minimax(board, depth + 1, self, alpha, beta) *# 假想敌来想对策* board.move(action, **'-'**) *# 假设的一部消失* **if** self.take == **'O'**: *# 得到最大值* **if** val > bestval:  
 alpha = val  
 bestval, bestaction = val, action  
 **if** alpha > beta:  
 **break  
 else**: *# 得到最小值* **if** val < bestval:  
 beta = val  
 bestval, bestaction = val, action  
 **if** alpha > beta:  
 **break  
 return** bestval, bestaction *# 返回最大值以及其行动***class** Game:  
 **def** \_\_init\_\_(self):  
 self.board = Board() *# 游戏棋盘初始化* self.current\_player = **None** *# 现在执子一方* **def** initialize\_player(self, p, take=**'X'**): *# 初始化玩家* **if** p == 0:  
 **return** Human(take) *# 0代表人类* **else**:  
 **return** AIPlayer(take) *# 1代表AI* **def** switch\_player(self, p1, p2):  
 **if** self.current\_player == **None**: *# 第一次进入循环时让p1执子* **return** p1  
 **else**:  
 **return** [p1, p2][self.current\_player == p1] *# 交换双方* **def** run(self):  
 print(**'\t\t--游戏开始--'**)  
 p1 = int(input(**"请输入第一个玩家的属性(AI为1，真人为0):"**))  
 p2 = int(input(**"请输入第二个玩家的属性(AI为1，真人为0):"**))  
 player1 = self.initialize\_player(p1, **'X'**) *# 初始化玩家1* player2 = self.initialize\_player(p2, **'O'**) *# 初始化玩家2* self.board.display() *# 打印空棋盘* **while True**:  
 self.current\_player = self.switch\_player(player1, player2) *# 交换玩家* action = self.current\_player.think(self.board) *# 执子玩家思考* self.board.move(action, self.current\_player.take) *# 执子玩家落子* self.board.display() *# 打印棋盘* **if** self.board.check\_winner() == 10:  
 print(**"Player1胜利啦！"**)  
 **break  
 elif** self.board.check\_winner() == -10:  
 print(**"Player2胜利啦！"**)  
 **break  
 elif** self.board.is\_left():  
 print(**"你们不分上下噢"**)  
 **break  
  
  
if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 Game().run()