**实验目的：熟悉剪枝算法**

**实验内容：用剪枝算法实现井字棋（井字棋获胜步数较少，用起来方便）**

**算法理解：阿尔法贝塔剪枝算法其实是一种博弈的算法，两个人下棋，A如果下一步，B就会得到优势，那么A就一定不会下在那个位置，因此这条支路就可以被减掉，不需要进行搜索。**

**算法实现思路：创建一个列表，可以用来表示井字棋棋盘。**

**创建一个函数，可以用来打印棋盘。**

**创建一个函数用来判定是否胜利。凡是符合同一横排，同一竖排，同意斜排的胜利**

**创建一个函数判断是否有空位。从头到尾搜索一遍就行。**

**创建一个函数实现阿尔法贝塔剪枝。**

**最后还需要一个函数用于机器思考。**

**Python代码：**

**import** random  
  
*# 用如下的9个数字来表示棋盘的位置:  
# 0 1 2  
# 3 4 5  
# 6 7 8  
  
# 设定获胜的组合方式(横、竖、斜)*WINNING\_TRIADS = ((0, 1, 2), (3, 4, 5), (6, 7, 8),  
 (0, 3, 6), (1, 4, 7),(2, 5, 8),  
 (0, 4, 8), (2, 4, 6))  
*# 设定棋盘按一行三个打印*PRINTING\_TRIADS = ((0, 1, 2), (3, 4, 5), (6, 7, 8))  
*# 用一维列表表示棋盘:*SLOTS = (0, 1, 2, 3, 4, 5, 6, 7, 8)  
*# -1表示X玩家 0表示空位 1表示O玩家.*X\_token = -1  
Open\_token = 0  
O\_token = 1  
  
MARKERS = [**'\_'**, **'O'**, **'X'**]  
END\_PHRASE = (**'平局'**, **'胜利'**, **'失败'**)  
  
  
**def** alpha\_beta\_valuation(board, player, next\_player, alpha, beta):  
 *"""运用AlphaBeta剪枝来计算当前局面的分值  
 因为搜索层数少，总能搜索到最终局面，估值结果为[-1,0,1]  
 """* wnnr = winner(board)  
 **if** wnnr != Open\_token:  
 *# 有玩家获胜* **return** wnnr  
 **elif not** legal\_move\_left(board):  
 *# 没有空位,平局* **return** 0  
 *# 检查当前玩家"player"的所有可落子点* **for** move **in** SLOTS:  
 **if** board[move] == Open\_token:  
 board[move] = player  
 *# 落子之后交换玩家，继续检验* val = alpha\_beta\_valuation(board, next\_player, player, alpha, beta)  
 board[move] = Open\_token  
 **if** player == O\_token: *# 当前玩家是O,是Max玩家(记号是1)* **if** val > alpha:  
 alpha = val  
 **if** alpha >= beta:  
 **return** beta *# 直接返回当前的最大可能取值beta, 进行剪枝* **else**: *# 当前玩家是X,是Min玩家(记号是-1)* **if** val < beta:  
 beta = val  
 **if** beta <= alpha:  
 **return** alpha *# 直接返回当前的最小可能取值alpha, 进行剪枝* **if** player == O\_token:  
 retval = alpha  
 **else**:  
 retval = beta  
 **return** retval  
  
  
**def** print\_board(board):  
 *"""打印当前棋盘"""* **for** row **in** PRINTING\_TRIADS:  
 r = **' '  
 for** hole **in** row:  
 r += MARKERS[board[hole]] + **' '** print(r)  
  
  
**def** legal\_move\_left(board):  
 *""" 判断棋盘上是否还有空位 """* **for** slot **in** SLOTS:  
 **if** board[slot] == Open\_token:  
 **return True  
 return False  
  
  
def** winner(board):  
 *""" 判断局面的胜者,返回值-1表示X获胜,1表示O获胜,0表示平局或者未结束"""* **for** triad **in** WINNING\_TRIADS:  
 triad\_sum = board[triad[0]] + board[triad[1]] + board[triad[2]]  
 **if** triad\_sum == 3 **or** triad\_sum == -3:  
 **return** board[triad[0]] *# 表示棋子的数值恰好也是-1:X,1:O* **return** 0  
  
  
**def** determine\_move(board):  
 *"""决定电脑(玩家O)的下一步棋,若估值相同则随机选取步数"""* best\_val = -2 *# 本程序估值结果只在[-1,0,1]中* my\_moves = []  
 print(**"开始思考"**)  
 **for** move **in** SLOTS:  
 **if** board[move] == Open\_token:  
 board[move] = O\_token  
 val = alpha\_beta\_valuation(board, X\_token, O\_token, -2, 2)  
 board[move] = Open\_token  
 print(**"Computer如果下在"**, move, **",将导致"**, END\_PHRASE[val])  
 **if** val > best\_val:  
 best\_val = val  
 my\_moves = [move]  
 **if** val == best\_val:  
 my\_moves.append(move)  
 **return** random.choice(my\_moves)  
  
  
HUMAN = 1  
COMPUTER = 0  
  
  
**def** main():  
 *"""主函数,先决定谁是X(先手方),再开始下棋"""* next\_move = HUMAN  
 opt = input(**"请选择先手方，输入X表示玩家先手，输入O表示电脑先手："**)  
 **if** opt == **"X"**:  
 next\_move = HUMAN  
 **elif** opt == **"O"**:  
 next\_move = COMPUTER  
 **else**:  
 print(**"输入有误，默认玩家先手"**)  
  
 *# 初始化空棋盘* board = [Open\_token **for** i **in** range(9)]  
  
 *# 开始下棋* **while** legal\_move\_left(board) **and** winner(board) == Open\_token:  
 print()  
 print\_board(board)  
 **if** next\_move == HUMAN **and** legal\_move\_left(board):  
 **try**:  
 humanmv = int(input(**"你要落子的位置(0-8)："**))  
 **if** board[humanmv] != Open\_token:  
 **continue** board[humanmv] = X\_token  
 next\_move = COMPUTER  
 **except**:  
 print(**"输入有误"**)  
 **continue  
 if** next\_move == COMPUTER **and** legal\_move\_left(board):  
 mymv = determine\_move(board)  
 print(**"电脑决定下在"**, mymv)  
 board[mymv] = O\_token  
 next\_move = HUMAN  
print\_board(board)  
 print([**"平局"**, **"Computer赢了"**, **"你赢了"**][winner(board)])  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()