

**课 程 实 验 报 告**

**课程名称： 人工智能导论**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师： 金 燕**

**报告日期： 2021.12.31**

**计算机科学与技术学院**

# 内容介绍

井字棋，英文名叫Tic-Tac-Toe，是一种在3\*3格子上进行的连珠游戏，和五子棋类似，由于棋盘一般不画边框，格线排成井字故得名。游戏需要的工具仅为纸和笔，然后由分别代表O和X的两个游戏者轮流在格子里留下标记，任意三个标记形成一条直线，则为获胜。此程序时简易的具有人机交互功能的井字棋游戏。由博弈树α-β剪枝以及估价函数实现。这种游戏的变化简单，常成为博弈论和赛局树搜寻的教学例子。这个游戏只有765个可能局面，26830个棋局。如果将对称的棋局视作不同，则有255168个棋局。

博弈树是指由于动态博弈参与者的行动有先后次序，因此可以依次将参与者的行动展开成一个树状图形。

对于任何一种博弈竞赛，我们可以构成一个博弈树。它类似于状态图和问题求解搜索中使用的搜索树。博弈树的结点对应于某一个棋局，其分支表示走一步棋；根部对应于开始位置，其叶表示对弈到此结束。在叶节点对应的棋局中，竞赛的结果可以是赢、输或者和局。

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，通过某搜索算法从中选出最优的走步。

其基本思想是：

(1) 设博弈的双方中一方为MAX，另一方为MIN。然后为其中的一方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。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。

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比如五子棋，第一步就有225种下法。而对手对应有225\*224=50400种决策方案；再往下一层，就有225\*224\*223=11,239,200种决策方案。这才第三层，就已经快爆炸了。一般五子棋的高手都能想到后面五六步，甚至十几步。想要与之对抗，我们必须得想办法减少我们的搜索数量，增加我们的搜索深度，这样我们的AI才能看得更远的未来，想得更多，这样棋力才会变强。因此试图利用完整的博弈树来进行极小极大分析是困难的。所以才有了α-β剪枝。

α-β剪枝为了提高搜索的效率，引入了通过对评估值的上下限进行估计从而减少需进行评估节点的范围。

α-β剪枝的主要概念：

MAX节点的评估下限值α：

作为MAX节点，假定它的MIN节点有N个，那么当它的第一个MIN节点的评估值为α时，则对于其它节点，如果有高于α的节点，就取那最高的节点值作为MAX节点的值；否则，该点的评估值为α。

MIN节点的评估上限值β：

作为MIN节点，同样假定它的MAX节点有N个，那么当它的第一个MAX节点的评估值为β时，则对于其他节点，如果有低于β的节点，就取最低的节点值作为MIN节点的值；否则，该点的评估值为β。

α-β剪枝的主要思想：

可以分为两个步骤，分别为α剪枝和β剪枝。

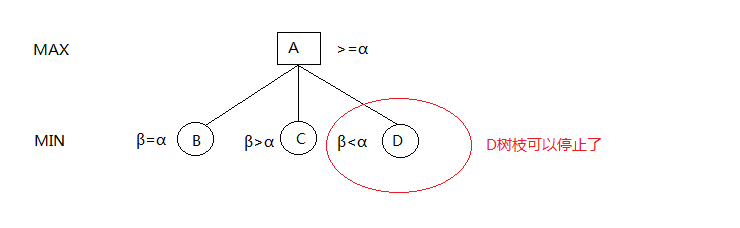
如图：

图1-1α剪枝示意图

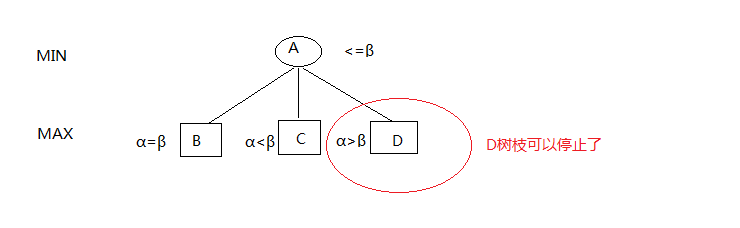


图1-2β剪枝示意图

# 算法分析

## 2.1程序总体分析

本程序为简单的具有人机交互功能的井字棋，程序共有8个函数，分别为printboard、playerselect、chesscountvalue、evaluate、searchtree、computerselect、winner、game，其中printboard函数用来显示当前的棋盘状态，palyerselect函数用来实现玩家的落子情况的改变，chesscountvalue函数用来实现返回当前棋子状态对应的估价值，evaluate函数作为估价函数判断当前棋盘的估价值，searchtree函数为深度优先搜索函数计算在落子一个位置后到棋盘满的过程产生的估价值，computerselect函数通过计算估价值确定电脑的落子位置，winner函数用来确定胜利者，game函数为主循环函数。

## 2.2函数分析

### 2.2.1 printboard函数

printboard函数为输出函数，输出当前的棋盘状态，此函数参数据为board用来存储棋盘的各个位置的信息，若值为0，表示未填写；若值为1，表示填写为“O”；其余表示填写为“X”。函数用dropshow存储board的各个值，用boardshow来存储棋盘输出格式，并用print函数输出。

### 2.2.2 playerselect函数

playerselect函数用于确定玩家落子位置并改变相应的棋盘布局。使用变量position存储玩家选择的落子位置，判断输入的position是否合法，不合法输出提示语句重新选择，否则返回position-1。函数流程图如图1-3所示：



图1-3playerselect函数

### 2.2.3 chesscountvalue函数

chesscountvalue函数用于返回对应棋子状态的估价，由于chesscountvalue是evaluate的子函数，只需要返回对应行或列或正负对角线的三个棋子的估价，其参数为chesscount，因此只需要判断chesscount各值的数量并返回相应值即可。这里我采用的是若所判断状态下均为电脑棋子返回10，均为玩家棋子返回-10；若有两个电脑棋子一个空白处返回5，两个玩家棋子一个空白处返回-5；若有一个电脑棋子两个空白处返回1，一个玩家棋子两个空白处返回-1；因为其余情况下三个位置里既有电脑棋子又有玩家棋子，不可能出现获胜的情况所以其余情况均为0。函数流程图如图1-4所示：



图1-4chesscountvalue函数流程图

### 2.2.4 evaluate函数

evaluate函数用于返回总体棋盘的估价值，由于棋盘共有9个位置，依据获胜条件，应该进行每行、每列以及正负对角线共8组估价，也就是调用chesscountvalue函数8次。依据chesscountvalue函数的返回值tempvalue进行下一步操作，若tempvalue绝对值为10，代表此时已经有一方获胜直接返回tempvalue的值即可；其余情况代表着此时并未有人获胜，将tempvalue的值加到value的值中以此估价目前整个棋局的值，8组数据都进行估价后返回value的值。函数流简化程图如图1-5所示：



图1-5evaluate函数简化流程图

### 2.2.5 searchtree函数

searchtree函数用于深度优先搜索，返回在落子一个位置后直到棋盘满这个过程所产生的估价值。函数通过递归进行深度优先搜索遍历，并且每一次都进行α-β剪枝，并依据whosturn的值返回相应的值。剪枝过程为根据whosturn进行相关操作：若当前为电脑下棋（whosturn=1），则判断tempvalue与alpha的关系，若tempvalue大于alpha，将alpha赋值为tempvalue，若alpha大于等于beta，返回beta；若当前为玩家下棋，则判断tempvalue与beta的关系，若tempvalue小于beta，将beta赋值为tempvalue，若beta小于等于alpha，返回alpha。并且若电脑下棋，最终返回alpha，玩家下棋返回beta。函数简化流程图如图1-6所示：



图1-6searchtree函数简化流程图

### 2.2.6 computerselect函数

computerselect函数用于遍历棋盘，在估价函数值最高的地方为电脑落子。使用mapgrid存储相应的值，对于每一个未填写的位置，调用searchtree函数计算估价值，将估价值存储在mapgrid中，对mapgrid进行递减排序，以此返回最大值。函数简化流程图如图1-7所示：



图1-7computerselect函数简化流程图

### 2.2.7 winner函数

winner函数用于判断刚刚落子的一方是否获胜。对当前棋盘进行估价，若估价值为100，表示电脑获胜；若估价值为-100，表示玩家获胜；否则需要判断是平局还是未分出胜负继续游戏，依次遍历每个位置判断是否填写，若有一个位置未填写则继续游戏，否则均已填写则为平局。对于4中结果，返回值依次为1，2，0，3。函数简化流程图如图1-8所示：



图1-8winner函数简化流程图

### 2.2.8 game函数

game函数用于主循环实现游戏的进行。函数通过调用之前的函数实现相应功能，首先确定谁先手，然后显示棋盘，依次由玩家与电脑确定所选位置，进行判断是否胜利，最后结束游戏。函数简化流程图如图1-9所示：



图1-9game函数简化流程图

# 实验结果分析

### 3.1测试说明

测试在vscode中实现，并且测试用例较简单，但可以说明程序的正确性。测试用例分为两部分，一部分为玩家先手测试，一部分为电脑先手测试，测试结果以及说明见3.2、3.3。

### 3.2测试过程及结果

### 3.2.1玩家先手测试

当启动程序时会有提示语句提示选择谁先手，输入2选择玩家先手进行测试，运行截图如图1-10、1-11所示。

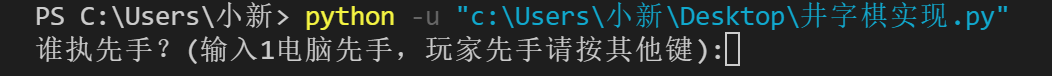


图1-10程序开始的提示语句

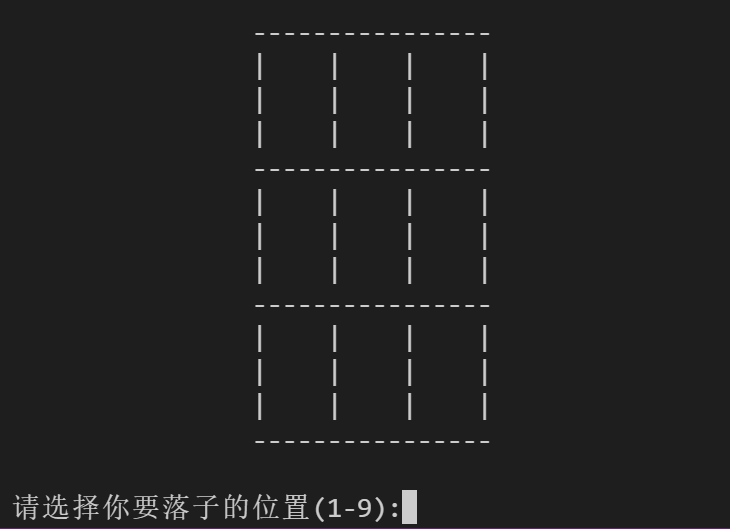


图1-11显示棋盘并提示选择位置

随后开始进行游戏，我选择了2作为第一个落子位置，并不断的下棋，最后电脑获胜，运行截图如图1-12、1-13所示。

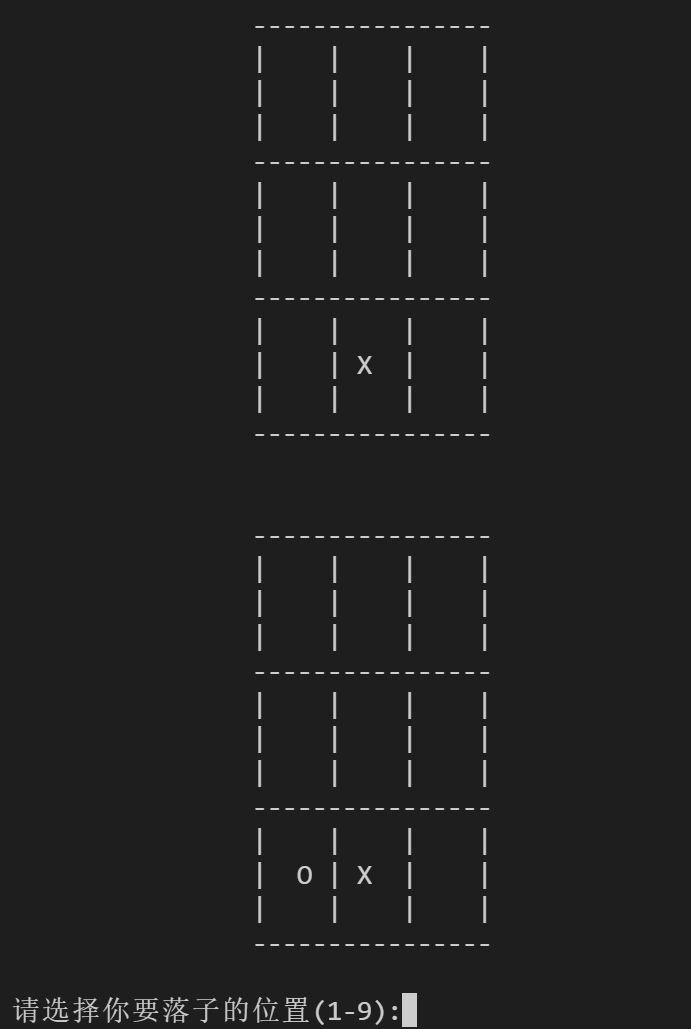


图1-12选择落子位置后电脑落子截图

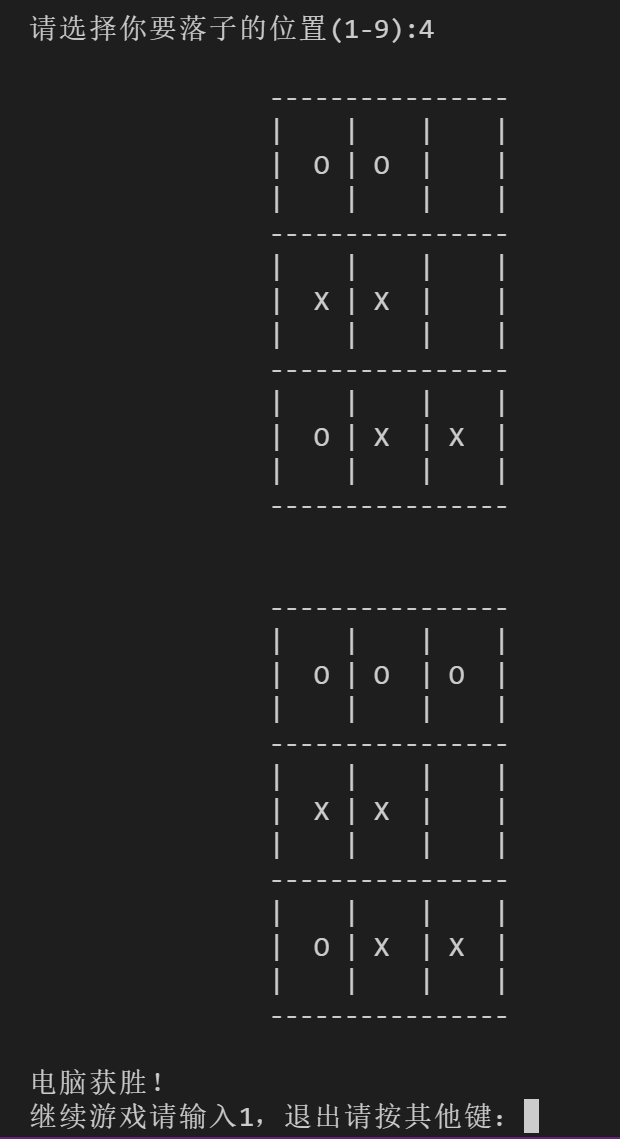


图1-13最终结果截图

玩家平局的运行截图如图1-14所示：

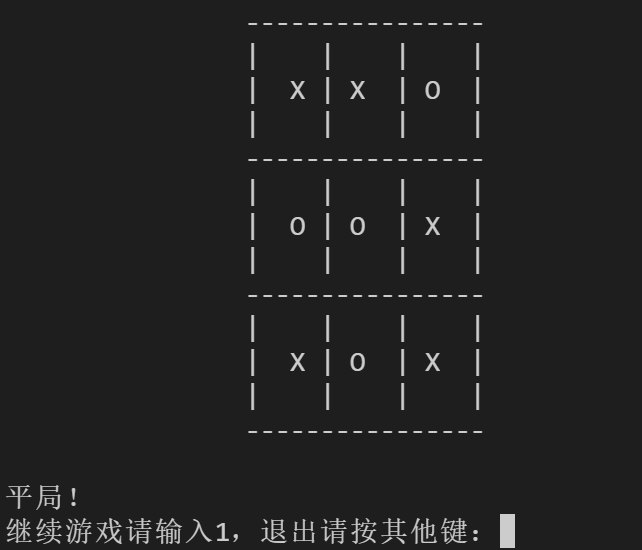


图1-14平局结果运行截图

由于自己棋艺有限，没有战胜电脑。最好也是平局的结果，故没有玩家获胜的截图，但是玩家获胜后与前两种结果基本一样，输出玩家获胜并输出最后一行的提示语句“继续游戏请输入1，退出请按其他键：”。

### 3.2.2电脑先手测试

输入1进行电脑先手测试，运行结果如图1-15所示。

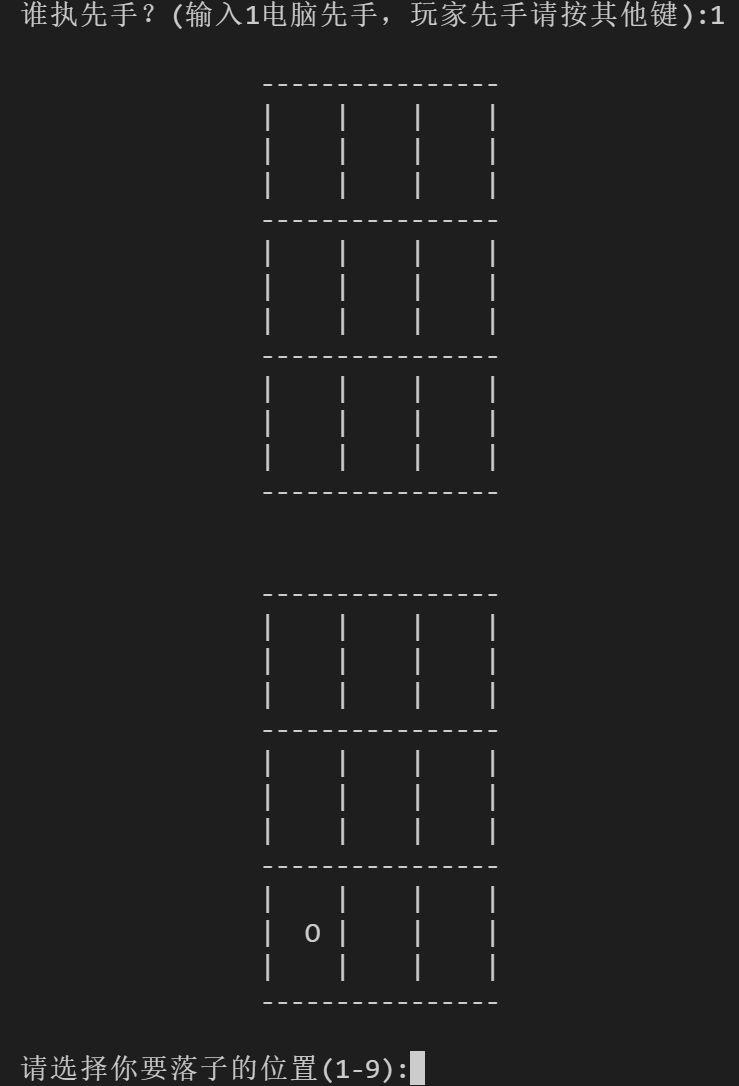


图1-15电脑先手测试截图

最终结果截图如图1-16所示：

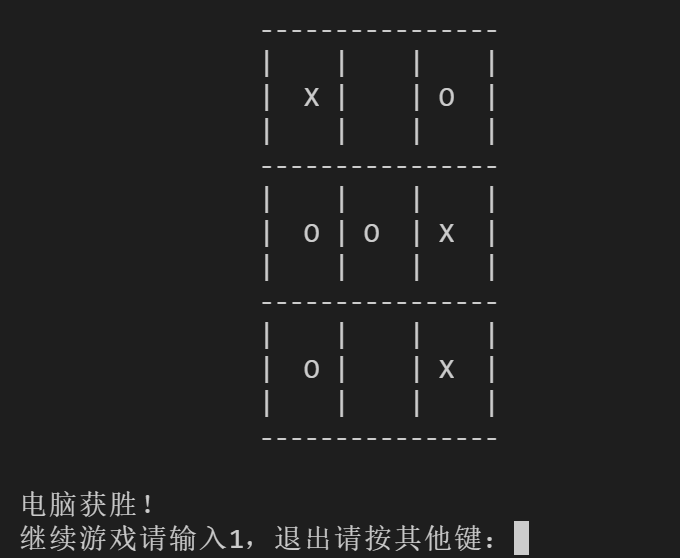


图1-16最终结果运行截图

## 3.3测试分析

经过测试发现，当电脑先手时它会选择左下角的位置，这是因为在计算估价值时，左下角是最优的选择，这也有悖与我们平时的经验，当我们先手不选择左下角时，电脑也会优先占据左下角的位置。但是当我们选择了左下角的位置后，电脑会占据中间的位置，这是因为除了左下角，中间是最有利的位置，此外发现电脑下棋并不是随意乱下的，显然电脑经过计算后进行的选择比我们普通的选择更加可能获胜。程序的运行速度较快，一方面是由于采用了博弈树进行搜索，合理运用了α-β剪枝提高了计算力，另一方面是由于井字棋的规模并不太大所以计算速度较快。另外我首先使用极大极小值搜索实现了井字棋，发现α-β剪枝的速度明显快于极大极小值搜索，见识了α-β剪枝的速度及性能。α-β剪枝是对极小化极大算法的一种改进，但是在实际应用过程中，α-β剪枝首先要区分出博弈双方谁是取极大值者，谁是取极小值者，达到剪枝条件时才会进行剪枝。这一优化方法虽然简洁明了，但在一定程度上让算法的效率打了折扣。因此在具体的博弈中，结合博弈的特定规则进行优化，比如说，将一些先验知识纳入剪枝条件中，这种基于具体应用的优化将是α-β剪枝的重要发展方向。

# 心得体会

此次作业是五选一，由于自己的能力有限，选择了较为简单的井字棋实现作为自己的作业。自己在大一时的一门课上就实现过井字棋，但是当时并不是很懂很多都是搜集的资料或者看书写的，这次重新选择了井字棋并自己动手又实现了一次，比之前对于实现原理的理解更加深刻了。自己也见识到了α-β剪枝的巧妙之处，更是亲身感受到了人工智能的“厉害”，在我与自己所写的程序进行对战时，我的胜率是0%，这也加深了我对于人工智能的理解。自己本身就对于人工智能很感兴趣，但是能力有限一直没有进行相关方面的学习，这学期选择了人工智能导论这门课就学到了很多知识也了解了更多有关人工智能内容。自己下学期想选择机器学习这门课，我认为人工智能导论为我打开了人工智能的大门，但是进一步的学习还是得靠自己，自己课下也一定会更加认真地学习人工智能相关方面地知识，争取自己可以将老师所布置的5个内容都自己动手实现一次。最后感谢老师为这一门课的付出，让我收获了许多，视野也开拓了许多！

# 附录代码

#使用博弈树、估价函数，确定电脑最佳落子处

def printboard(board):

    #显示棋盘当前状态

    boardshow='''

                ----------------

                |    |    |    |

                |  %s | %s  | %s  |

                |    |    |    |

                ----------------

                |    |    |    |

                |  %s | %s  | %s  |

                |    |    |    |

                ----------------

                |    |    |    |

                |  %s | %s  | %s  |

                |    |    |    |

                ----------------

            '''

    dropshow=[]

    for c in board:

        if c==0:

            dropshow.append(' ')

        elif c==1:

            dropshow.append('O')

        else:

            dropshow.append('X')

    print(boardshow%tuple(dropshow[6:9]+dropshow[3:6]+dropshow[0:3]))

def playerselect(board):

    #玩家落子，改变棋盘状态即可

    position=int(input('请选择你要落子的位置(1-9):'))

    while position not in range(1,10) or board[position-1]!=0:

        position=int(input('非法落子位置，请重新选择(1-9):'))

    return position-1

def chesscountvalue(chesscount):

    #返回对应棋子状态的估价

    num1=chesscount[1]

    num2=chesscount[2]

    num0=chesscount[0]

    if num1==3:

        return 10

    elif num2==3:

        return -10

    elif num1==2 and num0==1:

        return 5

    elif num1==1 and num0==2:

        return 1

    elif num2==2 and num0==1:

        return -5

    elif num2==1 and num0==2:

        return -1

    return 0

def evaluate(board):

    #估价函数，判断当前棋盘的估价值

    # 判断每一行、每一列、每条对角线的棋子状态，不同的状态有不同的估价值

    board=[board[6:9],board[3:6],board[0:3]]

    chess=[0,0,0] # 记录棋子状态

    value=0

    for i in range(0,3):

        chess=[0,0,0]

        for j in range(0,3):

            chess[board[i][j]]+=1 # 判断行

        tempvalue=chesscountvalue(chess)

        if abs(tempvalue)==10:

            return tempvalue

        else:

            value+=tempvalue

        chess=[0,0,0]

        for j in range(0,3):

            chess[board[j][i]]+=1 # 判断列

        tempvalue=chesscountvalue(chess)

        if abs(tempvalue)==10:

            return tempvalue

        else:

            value+=tempvalue

    chess=[0,0,0]

    for p in range(0,3):

        chess[board[p][p]]+=1 # 判断正对角线

    tempvalue=chesscountvalue(chess)

    if abs(tempvalue)==10:

        return tempvalue

    else:

        value+=tempvalue

    chess=[0,0,0]

    for p in range(0,3):

        chess[board[p][2-p]]+=1 # 判断负对角线

    tempvalue=chesscountvalue(chess)

    if abs(tempvalue)==10:

        return tempvalue

    else:

        value+=tempvalue

    return value

def searchTree(board,whosturn,alpha,beta):

    #α-β剪枝实现

    value=evaluate(board) # 计算当前棋盘状态的价值

    isfull=True # 棋盘是否已经下满

    if abs(value)>=10: # 若当前棋盘状态已经达到最大值或最小值，返回即可

        return value

    for i in range(0,9):

        if board[i]!=0: # 不等于0表示已经有棋子落在了该处，则继续判断下一个位置

            continue

        isfull=False # 执行到这一步说明棋盘没有满，将isFull赋值为False

        board[i]=whosturn # 将当前落子方的棋子下在当前遍历的位置

        tempvalue=searchTree(board,((whosturn-1)^1)+1,alpha,beta)

        board[i]=0

        if whosturn==1:

            if tempvalue>alpha:

                alpha=tempvalue

            if alpha>=beta:

                return beta

        if whosturn==2:

            if tempvalue<beta:

                beta=tempvalue

            if beta<=alpha:

                return alpha

    if isfull: # 若isFull为True，表示棋盘已满，返回进入该函数时棋盘的价值

        return value

    if whosturn==1:

        return alpha

    if whosturn==2:

        return beta

#def searchtree(board,whosturn):

    #极大极小值搜索实现

    #深度优先搜索，在落子一个位置后，直到棋盘满，这个过程产生的估价值

#    maxvalue=-100 # 用来存储对当前落子方来说最大的价值

#    minvalue=100 # 用来存储对当前落子方来说最小的价值

#    value=evaluate(board) # 计算当前棋盘状态的价值

#    isfull=True # 棋盘是否已经下满

#    if abs(value)>=10: # 若当前棋盘状态已经达到最大值或最小值，返回即可

 #        return value

    # 遍历每个棋盘位置

#    for i in range(0,9):

#         if board[i]!=0: # 不等于0表示已经有棋子落在了该处，则继续判断下一个位置

#             continue

#         isfull=False # 执行到这一步说明棋盘没有满，将isFull赋值为False

 #        board[i]=whosturn # 将当前落子方的棋子下在当前遍历的位置

        # 递归调用，继续向下进行搜索

#         tempvalue=searchtree(board,((whosturn-1)^1)+1)

#         maxvalue=max(maxvalue,tempvalue) # 判断下在当前遍历的位置后产生的价值是否大于maxValue

#        minvalue=min(minvalue,tempvalue) # 判断下在当前遍历的位置后产生的价值是否小于minValue

 #        board[i]=0 # 将被落子的位置清空，继续循环判断下在其他位置后棋盘的价值

 #    if isfull: # 若isFull为True，表示棋盘已满，返回进入该函数时棋盘的价值

  #       return value

 #    if whosturn==1: # 若当前落子方为电脑，返回最大价值

 #       return maxvalue

  #   if whosturn==2: # 若当前落子方为玩家，返回最小价值

  #       return minvalue

def computerselect(board):

    #遍历棋盘，计算在可以落子的位置落子后的估价函数值，选择值最高的位置落子

    mapgrid=[]

    for i in range(0,9):

        if board[i]!=0:

            continue

        board[i]=1

        mapgrid.append([searchtree(board,2),i])

        board[i]=0

    mapgrid=sorted(mapgrid,key=lambda x: x[0],reverse=True)

    return mapgrid[0][1]

def winner(board):

    #判断刚刚落子的一方是否获胜（电脑胜出还是玩家胜出）

    value=evaluate(board)

    if value==10:

        return 1 # 电脑获胜

    elif value==-10:

        return 2 # 玩家获胜

    else:

        for i in range(0,9):

            if board[i]==0:

                return 0

        return 3

def game():

    #主循环，电脑玩家依次落子，电脑每次落子前选择最优位置落子，玩家根据输入落子

    continuegame=True

    while continuegame:

        whofirst=(input('谁先手？(输入1电脑先手，玩家先手请按其他键):'))

        computersturn=1

        playersturn=2

        whosturn=computersturn if whofirst=='1' else playersturn

        # 0表示可以落子，1表示电脑已落子，2表示玩家已落子

        board=[0,0,0,0,0,0,0,0,0]

        # 游戏开始

        gaming=True

        printboard(board)

        while gaming:

            if whosturn==playersturn:

                playerdrop=playerselect(board)

                board[playerdrop]=2

                whosturn=computersturn

            elif whosturn==computersturn:

                computerdrop=computerselect(board)

                board[computerdrop]=1

                whosturn=playersturn

            printboard(board)

            state=winner(board)

            if state==1:

                print('电脑获胜！')

            elif state==2:

                print('你获胜！')

            elif state==3:

                print('平局！')

            if state!=0:

                continueGame=(input('继续游戏请输入1，退出请按其他键：'))

                if continueGame=='1':

                    continuegame=True

                else:

                    continuegame=False

                break

game()