Tic-Tac-Toe -CS50's Introduction to Artficial Intelligence with Python

（基于极大极小算法以及$\alpha-\beta$剪枝算法的井字棋游戏的Python 实现）

方翔

19211165@bjtu.edu.cn

摘要

本项目是在斯坦福 C50 课程中已给出的井字棋游戏交互项 python 代码（runner.py）的基础上，借助零和博弈中典型的极大极小算法以及在此基础上为了降低复杂度所提出的$\alpha-\beta$剪枝算法，完成包括盘面初始化，玩家判断，游戏结束判断，以及核心的极大极小算法和剪枝算法等的编写，之后通过运行，与机器对抗，最终完成项目。

1 项目背景

数学中的博弈论，是经济学的一个分支，把多 Agent 环境看成是博弈，其中每个 Agent 都会受到其他 Agent 的显著影响。而人工智能中的“博弈”通常专指有完整信息的、确定性的、轮流行动的、两个游戏者的零和游戏（如国际象棋）。在游戏中，两个 Agent 在确定的、完全可观察的环境中必须轮流行动，在游戏结束时效用值总是相等并且符号相反。

而本项目研究的是我们常玩的 Tic-Tac-Toe，即井字棋游戏。盘面为一个 3×3 的九宫格，游戏开始时，对决双方选择棋子种类：“X”或“O”，并默认选择“X”的一方先下。之后“O”方再下，一次类推。初始盘面为全空的九宫格，轮到哪一方下棋，他可以将他对应的棋子填入到还未被其他棋子填充的格子中，双方依次按照如上的规则下棋。直到九宫格的其中某一行或某一列或某一对角线被同一种棋子填充，游戏结束，满足上述条件的棋子对应的玩家获胜。

当然，游戏中也会出现当所有格子被填满后，仍不存在某一行或某一列或某一对角线被同一种棋子填充的情况，此时视为平均。

本项目就是要根据上述规则，利用 Python 实现可玩的井字棋游戏，用户作为玩家，与电脑进行对抗。当然，电脑所进行的下棋一定不是愚蠢的，是经过“思考”后决定的。所以我们将用到常见的极大极小算法作为电脑的决策策略。根据极大极小策略的原理，自然不能够出现用户玩家获胜的情况。

2. 项目对函数功能的要求

由于 runner 函数是提前给出

的，无需我们编写，因此我们这里不对 runner 函数进行研究。下面给出 tictactoe.py 中要求的一些函数。

- initial function：初始化盘面，所有格子置 EMPTY。
- player function：玩家函数，要以当前盘面状态作为输入，并返回当前是哪个玩家的回合（即当时是轮到“X”下还是“O”下）。
- actions function：行动函数，同样是以当前盘面状态作为输入，需返回以当前盘面为基础，下棋者还能采取的所有的行动组成的集合（即所有格子仍为 EMPTY 的位置），集合中的元素用（i，j）表示，其中 i 表示该可能行动对应的行（0、1 或 2），j 表示该可能行动对应列（0、1 或 2）。
- result function：结果函数，以当前盘面以及一个行动（action in actions）作为输入，返回由于该行动造成的盘面的更新的最新状态，但不要修改原始的版面
- winner function：胜者函数，同样是用当前盘面作为输入，判断当前盘面是否满足某方胜利的条件，若满足，则返回胜利方；否则范围 None 指示。
- terminal function：终止函数，同样也是用当前盘面作为输入，通过返回布尔值来指示游戏是否结束。如果游戏结束，要么是因为有人赢了游戏，要么是因为所有单元都被填满，没有人获胜，该函数应该返回 True。否则，如果游戏仍在进行中，该函数应该返回 False。
- utility function：用于判断结束时是哪一方获胜，或者是平局，用于 runner 函数调用并显示在游戏界面。
- minimax 函数：本项目最为核心的函数模块，以当前的版面作为输入，应返回一个以极大极小为策略的最佳行动。

3. 基础函数 Python 实现

3.1 player 函数

player 函数的目的是根据已知的 board 盘面，判断目前是哪一位玩家的回合。由于我们默认“X”玩家第一个行动，因此我们可以知道：

1. 只要 board 上的“X”个数大于“O”的个数，自然当前是“O”的回合。
2. 只要 board 上的“X”个数等于“O”的个数，自然当前是“X”的回合。
3. board 上全为 EMPTY，也是“X”的回合。

据此，可以实现 player 函数的编写，代码如下：

```
    def player(board):
    """
    Returns player who has the next
turn on a board.
    """
    x_count = 0
    o_count = 0
    for i in board:
        for j in i:
            if j == "X":
                x_count += 1
            elif j == "O":
                o_count += 1
    if x_count == 0:
        return X
    elif x_count > o_count:
        return O
    elif x_count == o_count:
        return X
```

3.2 actions 函数

actions 函数较为简单，只需要读取版面中仍是 EMPTY 的位置并存入 list 即可。代码如下：

```
    def actions(board):

    action_list = []
    for row in range(3):
        for col in range(3):
            if board[row][col] ==
EMPTY:

action_list.append((row,col))
    return action_list
```

3.3 result 函数

该函数需要将行动值加入到盘面中，需要注意的是，加入该行动是以“X”加入还是“O”加入盘面，需要调用一次 player 函数来确定。代码如下：

```
    def result(board, action):

    board_dul =
copy.deepcopy(board)
    additive_player =
player(board_dul)

board_dul[action[0]][action[1]] =
additive_player
    return board_dul
```

3.4 winner 函数

该函数需扫描整个盘面，判断是否有某一行或某一列或某一对角线相同的情况，配合 terminal 函数以及 utility 函数判断游戏是否结束和判断胜败方。

```
    def winner(board):
    '''行相同'''
    for i in range(3):
        if board[i][0] == board[i][1] ==
board[i][2]:
            return board[i][0]
    '''列相同'''
    for j in range(3):
        if board[0][j] == board[1][j] ==
board[2][j]:
            return board[0][j]
    '''对角线相同'''
    if board[0][0] == board[1][1] ==
board[2][2]:
        return board[1][1]
    if board[0][2] == board[1][1] ==
board[2][0]:
        return board[1][1]
    return None
```

3.5 terminal 函数和 utility 函数

terminal 函数判断当前状态，是有一方获胜，还是格子全被填满（前面两种均返回 True），还是仍未结束（返回 Flase）。

在 terminal 函数返回 True 的基础上，通过判断结束类型，返回对应的 Flag 值用于显示。代码如下：

```
    def terminal(board):
    if  winner(board)  is  not
None:
        return True
    else:
        for i in board:
            for j in i:
                if j == EMPTY:
                    return False
        return True

def utility(board):
    if terminal(board) is True:
        flag = winner(board)
        if flag == X:
            return 1
        elif flag == O:
            return -1
        else:
            return 0
```

4. 极大极小算法以及 minimax 函数实现

4.1 极大极小算法介绍

常规的零和博弈游戏，都会有效用函数，即给输赢平一个效用值。对决的双方，其中一方是使得该效用值达到最大（MAX），而另一方是使得该效用值达到最小（MIN）。

我们规定，本项目的井字棋问题，下“X”为 MAX 方，下“O”为“MIN”方。X 方赢的效用值为 1，输的效用值为-1，平均的效用值为 0。

初始状态、ACTIONS 函数和 RESULT 函数定义了游戏的博弈树

下图给出了井字棋的部分博弈树。在初始状态 **MAX** 有九种可能的棋招。游戏 轮流进行，**MAX** 下 X，**MIN** 下 0,直到到达了树的终止状态即一位棋手的标志占领一行、 一列、一对角线或所有方格都被填满。叶结点上的数字是该终止状态对于 **MAX** 来说的效用值；值越高对 **MAX** 越有利，而对 **MIN** 则越不利。

极大极小算法是基于对决双方都是 smart 的。MAX 方的想法是：

MAX 下完后，MIN 方一定会从他所有的下法中选择一个使得效用值最小的一步行动。那 MAX 方就要使得 MIN 方的这个效用极小值最大，以满足他的 MAX 策略；

而 MIN 方的想法是：

MIN 自己下完后，MAX 方一定会从他所有的下法中选择一个使得效用值最大的一步行动。那 MIN 方就要使得他的那一步操作使得 MAX 方的这个效用极大值最小。

这就是极大极小算法的思想，上述的例子只是二层的极大极小。我们考虑 MAX 和 MIN 方都足够聪明，可以想到这一步后面的不止 2 步，可以考虑到多步，甚至走完整个游戏，因此可以分别编写 mini 函数和 maxi 函数，让他们互相嵌套，通过设置一定步长，到达步长后停止思考，或直到把整个游戏想完，terminal 函数返回 True 位置。由此分析为基础，极大极小算法的伪代码如下：

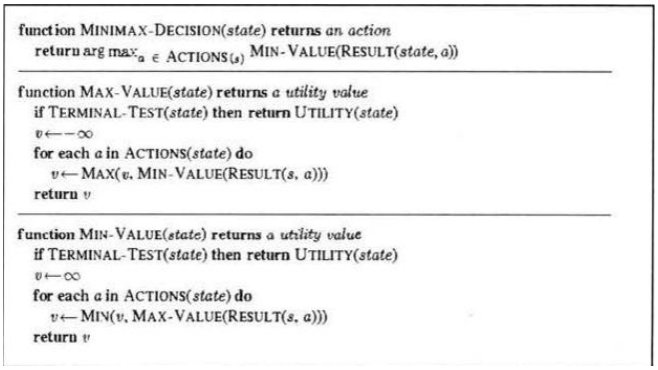

图 1 极大极小算法伪代码

4.2 井字棋 minimax 函数实现

根据人工智能课本中的上述伪代

码，进行了 Python 的实现。

python 代码见附录，其中 minimax 为主函数，mini 和 maxi 函数互相嵌套，直到判断到能使得游戏结束的一步，之后回溯，找到当前电脑应该执行的操作。代码的分析见注释，这里不做过多介绍，同时我们也会在项目演示视频中做简单的介绍。

5. $\alpha-\beta$剪枝算法及 Python 实现

传统的极大极小算法需要对每一种情况进行分析，特别是在这样的一种极大和极小层层嵌套的情况下，复杂度，电脑反应时间会非常的长（当然，随着游戏的进行，所有可能的情况逐渐减小，电脑的反应时间也会加快），但无论怎样，最开始的几步，都需要很长的计算时间。于是就有人提出的$\alpha-\beta$剪枝算法。其主要的思路通过剪去一些一定不会是最佳选择的枝叶，以减小一些不必要的时间和空间的浪费。

比如如下图，在 MAX 决策时，是选择所有的极小里的最大值，在第一个选择下，得到的极小值时 3.当继续判断第二个选择时，在搜索过程中发现了一个比之前发现的极小值 3 还要小的数 2，那剩下的节点就不需要再计算就可以知道，一定不能选择该行动（因为，起码这个行动的极小值是 2 甚至更低，肯定没有第一种选择好），于是就实现了α剪枝。

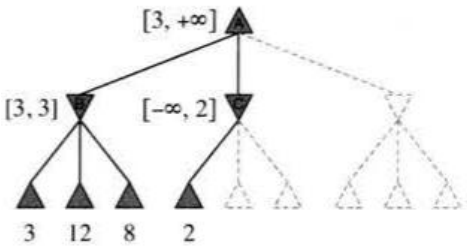

图 2 剪枝举例

同理 MIN 也有同样原理的β剪枝。将两者结合，于是就有了$\alpha-\beta$剪枝算法。

其伪代码如下图：

```
function ALPHA-BETA-SEARCH(state) returns an action
  v←MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state,α,β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v←−∞
  for each a in ACTIONS(state) do
    v←MAX(v, MIN-VALUE(RESULT(s,a),α,β))
    if v ≥ β then return v
    α←MAX(α, v)
  return v

function MIN-VALUE(state,α,β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v←+∞
  for each a in ACTIONS(state) do
    v←MIN(v, MAX-VALUE(RESULT(s,a),α,β))
    if v ≤ α then return v
    β←MIN(β, v)
  return v
```

图 3 剪枝算法伪代码

通过分析容易发现，其与极大极小算法的区别仅在于 Max 和 Min 函数中增加了用于剪枝的两个参数“alph”和“beta”，并增加了函数返回途径，即 Min 操作后的值已经大于 beta 或者 Max 操作后的值已经小于 alph，这与我们之前的分析一致。

由此为基础，编写了 Python 代码。见附录。

6. 项目运行结果

6.1 minimax 算法运行结果

运行 project，此时弹出游戏界面如下：

图 4 游戏初始界面

选择“X”或“O”作为棋子，“X”为先手，“O”为后手。

比如选择“Play as O”开始游戏：

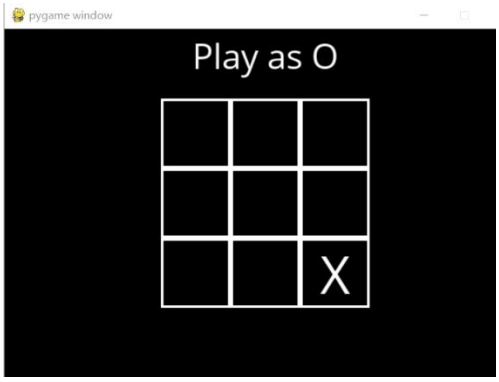

图 5 选择 O，电脑下第一步

以下是一些游戏结束情况。结果如下图：

图 6 用户（O）选择较好的策略

图 7 用户（X）选择不好的策略

通过多次尝试可以发现，用于电脑采用的始终的较好的策略，因此不存在用户胜利的情况，最多打成平局。一旦用户策略选择不当，用户就是输掉游戏。

另外，我们发现，若用户选择“O”开始游戏，由于初始盘面为空，电脑采取 minimax 算法，需要遍历所有的情况，用户需要等待较长的时间（作者试验计时第一步电脑需要将近 10s 才能完成第一步下棋），用户下完下一步后，电脑所用时显著减小到 1s 作用，之后时间越来越短。

不管怎么，10s 对于用户来说以及是很长了。

于是我们进行了剪枝算法编写和试验。

6.2 $\alpha-\beta$剪枝算法运行对比

在原 runner.py 中加入一行代码，并将对于的 minimax 行注释掉。

```
115        #    move = ttt.minimax(board)
116            move = ttt.alph_beta_prune(board)
117            board = ttt.result(board, move)
```

图 8 runner 中加入剪枝算法之后运行

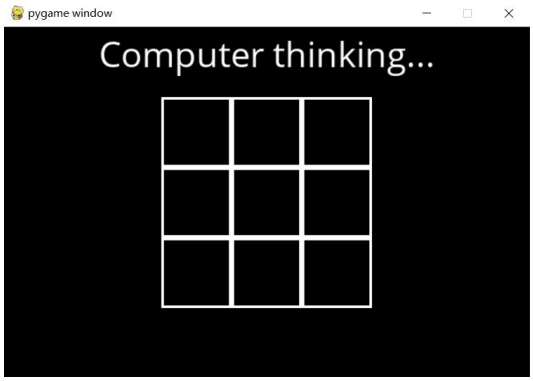

图 9 剪枝算法运行过程

作者计时得到应用剪枝算法，第一步的运行时间从原来的 10s 缩短到 3s 左右，时间显著减小。

7. 遇到的问题及解决方法

1）由于自己之前对 python 接触的较少，只是粗略的看过 python 的一些函数，没有真正的练习过 python 代码的编写。于是花了较长的时间学习

python 的基础知识。

2）由于要运行游戏，需要在 project 中导入 pygame，而安装的 python 一般都不自带 pygame，于是就广泛搜寻安装办法。按照一个较对的方法成功安装了 pygame 后，但发现 project 中的 pin 还是无法加入 pygame。

解决办法：因为之前下载 pycharm 的时候，还安装的 anaconda，于是想是否需要在 anaconda 里的 pin 安装 pygame，于是进行尝试，发现居然真的解决了这个问题。

具体流程为：运行“Anaconda Prompt”，利用“conda install pip”命令升级 pip，之后利用“pip install pygame”命令下载安装即可。

3）按照人工智能方法课本中给出的剪枝算法的伪代码编写后，运行发现，电脑选择的策略竟然不是最好的，会使得用户胜利。通过多次比对我编写的代码与书中给出的伪代码，始终没有头绪。

解决方法：思考修改了几天代码，决定从头分析原理，发现，剪枝处的算法不应该用“<=”和“>=”，搜索时，若碰到两个值相同，不应该被剪枝，因为可能当前枝的叶子结点甚至之后的节点很有可能比上一个相同值的叶子结点好。那为什么课本中给出的是小于等于，我的想法是当只是看两层的极大极小时，可以用小于等于或大于等于，因为他们不需要再往下看其叶子结点，因此二选一即可，于是可以被剪枝减掉。

8. 总结

本项目以斯坦福 C50 课程给出的部分代码为基础，按照文档中提出的各项要求，编写了对应的基础函数，以及最为重要的 minimax 函数，同时，也在此基础上，完成了$\alpha-\beta$剪枝算法的编写，通过对比，证实了$\alpha-\beta$剪枝算法在复杂度上的优越性。并在项目过程中国，针对遇到的问题，不怕困难，积极解决，并对课本中提出的部分伪代码做出的自己的理解有修改后用于 python 代码的编写。

总的来说，这次项目对本人的 python 代码编写以及对 minimax 算法和$\alpha-\beta$剪枝算法等人工智能算法有个更深的理解。

9. 感谢

感谢陈一帅老师的悉心教学，使得作者本人在进行项目的过程中不至于过于困难。

感谢自己的不放弃，最终安装成功了 pygame，也将$\alpha-\beta$剪枝算法课本中的伪代码修改后实现。

感谢斯坦福的代码资源，给了我部分框架，使得我在完成项目时有了方向。

10.参考文献

[1]Russell, Stuart J . Artificial Intelligence: A Modern Approach[J]. 人民邮电出版社, 2002.

附录 runner.py 源代码

```
import pygame
import sys
import time

import tictactoe as ttt

pygame.init()
size = width, height = 600, 400

# Colors
black = (0, 0, 0)
white = (255, 255, 255)

screen =
pygame.display.set_mode(size)

mediumFont =
pygame.font.Font("OpenSans-Regul
ar.ttf", 28)
largeFont =
pygame.font.Font("OpenSans-Regul
ar.ttf", 40)
moveFont =
pygame.font.Font("OpenSans-Regul
ar.ttf", 60)

user = None
board = ttt.initial_state()
ai_turn = False

while True:

    for event in pygame.event.get():
        if event.type ==
pygame.QUIT:
            sys.exit()

    screen.fill(black)

    # Let user choose a player.
    if user is None:

        # Draw title
        title =
largeFont.render("Play
Tic-Tac-Toe", True, white)
        titleRect = title.get_rect()
        titleRect.center = ((width
/ 2), 50)
        screen.blit(title,
titleRect)

        # Draw buttons
        playXButton =
pygame.Rect((width / 8), (height /
2), width / 4, 50)
        playX =
mediumFont.render("Play as X",
True, black)
        playXRect = playX.get_rect()
        playXRect.center =
playXButton.center
        pygame.draw.rect(screen,
white, playXButton)
        screen.blit(playX,
playXRect)

        playOButton = pygame.Rect(5
* (width / 8), (height / 2), width
/ 4, 50)
        playO =
mediumFont.render("Play as O",
True, black)
        playORect = playO.get_rect()
        playORect.center =
playOButton.center
        pygame.draw.rect(screen,
white, playOButton)
        screen.blit(playO,
playORect)

        # Check if button is clicked
        click, _, _ =
pygame.mouse.get_pressed()
        if click == 1:
            mouse =
```

```
pygame.mouse.get_pos()
            if
playXButton.collidepoint(mouse):
                time.sleep(0.2)
                user = ttt.X
            elif
playOButton.collidepoint(mouse):
                time.sleep(0.2)
                user = ttt.O

    else:

        # Draw game board
        tile_size = 80
        tile_origin = (width / 2 -
(1.5 * tile_size),
                       height / 2 -
(1.5 * tile_size))
        tiles = []
        for i in range(3):
            row = []
            for j in range(3):
                rect = pygame.Rect(
                    tile_origin[0] +
j * tile_size,
                    tile_origin[1] +
i * tile_size,
                    tile_size,
tile_size
                )

pygame.draw.rect(screen, white,
rect, 3)

                if board[i][j] !=
ttt.EMPTY:
                    move =
moveFont.render(board[i][j], True,
white)
                    moveRect =
move.get_rect()
                    moveRect.center
= rect.center
                    screen.blit(move,
moveRect)
                row.append(rect)
            tiles.append(row)

        game_over =
ttt.terminal(board)
        player = ttt.player(board)

        # Show title
        if game_over:
            winner =
ttt.winner(board)
            if winner is None:
                title = f"Game Over:
Tie."
            else:
                title = f"Game Over:
{winner} wins."
        elif user == player:
            title = f"Play as {user}"
        else:
            title = f"Computer
thinking..."
        title =
largeFont.render(title, True,
white)
        titleRect = title.get_rect()
        titleRect.center = ((width
/ 2), 30)
        screen.blit(title,
titleRect)

        # Check for AI move
        if user != player and not
game_over:
            if ai_turn:
                time.sleep(0.5)
             #   move =
ttt.minimax(board)
                move =
ttt.alph_beta_prune(board)
                board =
ttt.result(board, move)
                ai_turn = False
```

```
            else:
                ai_turn = True

        # Check for a user move
        click, _, _ =
pygame.mouse.get_pressed()
        if click == 1 and user ==
player and not game_over:
            mouse =
pygame.mouse.get_pos()
            for i in range(3):
                for j in range(3):
                    if (board[i][j]
== ttc.EMPTY and
tiles[i][j].collidepoint(mouse)):
                        board =
ttc.result(board, (i, j))

        if game_over:
            againButton =
pygame.Rect(width / 3, height - 65,
width / 3, 50)
            again =
mediumFont.render("Play Again",
True, black)
            againRect =
again.get_rect()
            againRect.center =
againButton.center

pygame.draw.rect(screen, white,
againButton)
            screen.blit(again,
againRect)
            click, _, _ =
pygame.mouse.get_pressed()
            if click == 1:
                mouse =
pygame.mouse.get_pos()
                if
againButton.collidepoint(mouse):
                    time.sleep(0.2)
                    user = None
                    board =
ttc.initial_state()
                    ai_turn = False

        pygame.display.flip()
```

tictactoe.py 源代码

```
    """
Tic Tac Toe Player
"""

import math
import copy

X = "X"
O = "O"
EMPTY = None


def initial_state():
    """
    Returns starting state of the board.
    """
    return [[EMPTY, EMPTY, EMPTY],
            [EMPTY, EMPTY, EMPTY],
            [EMPTY, EMPTY, EMPTY]]


def player(board):
    """
    Returns player who has the next turn on a board.
    """
    x_count = 0
    o_count = 0
    for i in board:
        for j in i:
            if j == "X":
                x_count += 1
            elif j == "O":
                o_count += 1    #对X和O计数
    if x_count == 0:          #判断哪位玩家的回合
        return X
    elif x_count > o_count:
        return O
    elif x_count == o_count:
        return X
    # raise NotImplementedError


def actions(board):
    """
    Returns set of all possible actions (i, j) available on the board.
    """
    action_list = []
    for row in range(3):
        for col in range(3):
            if board[row][col] == EMPTY:  #为EMPTY的位置入list

action_list.append((row,col))
    return action_list
    # raise NotImplementedError


def result(board, action):
    """
    Returns the board that results from making move (i, j) on the board.
    """
    board_dul = copy.deepcopy(board)
    additive_player = player(board_dul)  #调用player 确定应置入X还是O

board_dul[action[0]][action[1]] = additive_player
    return board_dul
  #  raise NotImplementedError


def winner(board):
    """
    Returns the winner of the game, if there is one.
    """
    '''行相同'''
    for i in range(3):
```

```
        if board[i][0] ==
board[i][1] == board[i][2]:
            return board[i][0]
    '''列相同'''
    for j in range(3):
        if board[0][j] ==
board[1][j] == board[2][j]:
            return board[0][j]
    '''对角线相同'''
    if board[0][0] == board[1][1]
== board[2][2] :
        return board[1][1]
    if board[0][2] == board[1][1]
== board[2][0] :
        return board[1][1]
    return None
 #   raise NotImplementedError


def terminal(board):
    """
    Returns True if game is over,
False otherwise.
    """
    if winner(board) is not None:
        return True
    else:
        for i in board:
            for j in i:
                if j == EMPTY:
                    return False
        return True

  #  raise NotImplementedError


def utility(board):
    """
    Returns 1 if X has won the game,
-1 if O has won, 0 otherwise.
    """
    if terminal(board) is True:
        flag = winner(board)
        if flag == X:
            return 1
        elif flag == O:
            return -1
        else:
            return 0
 #   raise NotImplementedError


def minimax(board):
    """
    Returns the optimal action for
the current player on the board.
    """
    m = actions(board)
    player1 = float('-inf')
    player2 = float('inf')  #MAX 和
MIN 策略置初值
    player_judgement =
player(board)
    if player_judgement is X: #若
为 X 的回合
        for i1 in m:
            player1_new =
mini(result(board,i1)) # MIN 的策略，
找每一步下的极小的效用值
            if player1 <=
player1_new:  #MAX 在所有的极小中找
最大的效用值，并确定对应的行动
                player1 =
player1_new
                final_action = i1
    else:               #若为 Y 的回
合
        for i2 in m:
            player2_new =
maxi(result(board,i2))# MAX 的策略，
找每一步下的极大的效用值
            if player2 >=
player2_new:  #MIN 在所有的极大中找
最小的效用值，并确定对应的行动
                player2 =
player2_new
                final_action = i2
```

```
    return final_action
#    raise NotImplementedError


def mini(board):
    if terminal(board):  #直到terminal判断游戏结束，返回对应的效用值
        return utility(board)
    p1 = float('inf')
    for i in actions(board):
        p1_new = maxi(result(board,i)) #mini中嵌套maxi 寻找极大
        if p1 >= p1_new: #MIN策略在所有极大中找最小的效用值
            p1 = p1_new
    return p1

def maxi(board):
    if terminal(board):
        return utility(board)
    p2 = float('-inf')
    for j in actions(board):
        p2_new = mini(result(board,j))#maxi中嵌套mini 寻找极小
        if p2 <= p2_new: #MAX策略在所有极小中找最大的效用值
            p2 = p2_new
    return p2

def alph_beta_prune(board):
    """
    Returns the optimal action for the current player on the board.
    """
    m = actions(board)
    player1 = float('-inf')
    player2 = float('inf')
    player_judgement = player(board)
    if player_judgement is X:
        for i1 in m:
            player1_new = mini_ab(result(board,i1),player1,player2) #传alph和beta参数的初始值
            if player1 <= player1_new:
                player1 = player1_new
                final_action = i1
        return final_action
    else:
        for i2 in m:
            player2_new = maxi_ab(result(board,i2),player1,player2)
            if player2 >= player2_new:
                player2 = player2_new
                final_action = i2
        return final_action

def mini_ab(board,alph,beta):
    if terminal(board):
        return utility(board)
    p1 = float('inf')
    for i in actions(board):
        p1 = min(p1,maxi_ab(result(board,i),alph,beta))
        if p1 < alph:    #判断MAX策略下当前值是否已经比之前搜索到的极小值小
            return p1   #是，则剪枝，提前返回
        beta = min(beta,p1) #更新beta
    return p1

def maxi_ab(board,alph,beta):
    if terminal(board):
        return utility(board)
    p2 = float('-inf')
    for j in actions(board):
        p2 = max(p2,mini_ab(result(board,j),a
```

```
lph,beta))
        if p2 > beta:   #判断 MIN 策略
下当前值是否已经比之前搜索到的极大值大
            return p2   #是，则剪枝，
提前返回
        alph = max(alph,p2) #更新
beta
    return p2
```