# 使用 Minimax 实现 TicTacToe

言晓梅

**Abstract**

本次项目使用 Minimax 算法，实现了人机对弈 TicTacToe 井字游戏

## 1 问题分析

在我们之前学过的经典搜索算法中，都是从初始地点到目的地点的一个路径搜索过程，而搜索领域还有对抗性搜索这样超越经典的搜索算法。对抗性搜索算法面对的是想要实现相反目标的对手，例如在井字游戏中，就会遇到这种类型的搜索。


图 1: 井字游戏

井字游戏中，X 想要赢得比赛，O 也想赢得比赛，二者的目的是相反的，并且在进行决策时，倾向于使自己赢得比赛 (也可以理解为阻止对方赢得比赛)。作为人类，我们在玩游戏时，决定我们如何下这一步棋的因素是我们要“赢”，但是计算机无法理解“输赢”的概念，我们需要将游戏的输赢，翻译成计算机可以理解的数学语言。计算机不知道什么是“输赢”，但它可以解决谁大谁小的问题。因此我们可以将井字游戏的 3 种结果分别定义为-1(O 获胜)、0(平局)、1(X 获胜)，计算机要做的就是如何决策，使得结果最大或者最小，这样，我们就可以和 AI 对弈。

## 2 算法分析

通过上面的分析，我们可以使用一种 Minimax 算法来实现 AI 下棋，对抗双方 (X 和 O) 在下棋时，X 试图获得最高分数 1，O 试图获得最低分数-1，在这过程中产生的决策，需要综合考虑对方的“想法”。

使用规范的数学语言来描述：

S0：初始状态，在井字游戏中，表示一个 3 行 3 列的空棋盘

Players(s): 输入当前状态 s，返回轮到哪个玩家下棋 (X 或者 O)

Actions(s): 输入当前状态 s，返回在该状态下所有可选的下棋方案

Result(s，a): 输入状态 s 与动作 a，返回一个新的状态，即进行动作 a 之后产生的新的状态

Terminal(s): 输入当前状态 s，检查是否是游戏的最后一步，即有人获胜或者是打成平手则游戏结束，返回 True，否则返回 False。

Utility(s): 输入终端状态 s 的函数，返回状态的有效值：-1、0 或者 1。

当算法执行到最后一步 (也就是即将分出胜负) 时，结果是显而易见的，那么 Minimax 真正发挥作用的地方，就在游戏还没结束的过程中，决定一个玩家 (X 或者 O) 如何下棋。

如图 2，以这一步由 X 下棋为例。当 X 拿着棋子，看着眼前的棋盘的时候，有 3 个地方可以下棋，尽管作为人类，我们一眼就看出应该选择第 3 种方法就能获得胜利，但计算机需要通过计算才能做出决策。X 试图让分数接近

1 而 O 试图让分数接近-1，那么 X 就会考虑我下了这一步棋后，O 会如何应对，同时 O 的决策也会考虑 X 下一步的“想法”，这是一个函数递归的过程。

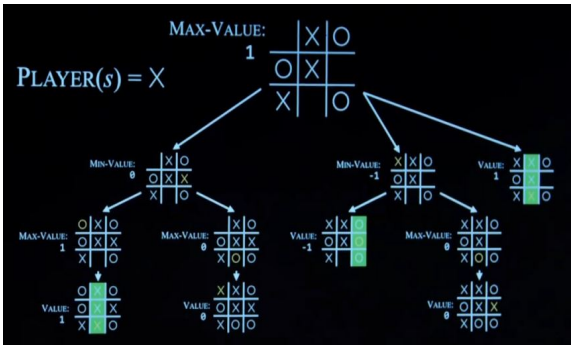


图 2: 算法简图

利用伪代码表示算法过程：

给定一个状态 s

玩家 X 在 Actions(s) 中会选择让 Min-Value(Result(s，a)) 最大的动作 a。玩家 O 在 Actions（s）中会选择让 Max-Value(Result(s，a)) 最小的动作 a。

其中

函数 Max-Value(s):

If Terminal(s):

Return Utility(s)

V=-∞

for action in Actions(s):

v=max(v, Min-Value(Result(s，a)))

return v

函数 Min-Value(s):

If Terminal(s):

Return Utility(s)

V=∞

for action in Actions(s):

v=min(v, Max-Value(Result(s，a)))

return v

我们可以发现，玩家 X 在决策时，会调用玩家 O 的 Min 函数，同样的，玩家 O 也会调用玩家 X 的 Max 函数，递归由此进行。

## 3 算法精解

从哈佛 CS50AI 课程下的 project 目录下载的文件有两个主要文件，runner.py 和 tictactoe.py. tictactoe.py 包含玩游戏和做出最佳动作的所有逻辑。runner.py 已经实现，并包含运行游戏图形界面的所有代码。我们只需要完成 tictactoe.py 所有需要的功能，就可以在 runner.py 上与 AI 对弈。

在 tictactoe.py 中，首先调用 math、copy 库，定义变量 X、O、EMPTY，便于我们对棋盘上的状态进行描述。函数 initia_lstate 返回 board 的初始状态，即游戏开始我们的棋盘是 3 行 3 列的空板，将 board 定义为 3 行 3 列的二维列表（list）变量。

```python
"""
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]]

```

图 3: 初始化

根据前面的分析，在 tictactoe.py 中，我们要实现以下几个函数

### 3.1 player(board)

该函数 (图 4) 以状态 board 为输入，返回轮到哪个玩家下棋 (X 或者 O)。在初始状态下，X 先手，随后 X 与 O 交替进行移动遍历 board 的每个元素，以计数的方式统计棋盘上的 X 和 O 的数量，将两者进行比较，数量更少的一方则轮到其下棋。

### 3.2 actions(board)

action 函数 (图 5) 表示在输入状态为 board 的情况下，返回可以在棋盘上执行的所有操作。其中每个动作表示为一个元组 (i，j)，i 对应移动的行 (0、1、2)，j 对应该行中哪个单元格 (也

就是哪一列) 进行移动 (0、1、2)，使用 set 函数创建一个列表变量储存这些动作元组，合法的移动是棋盘上并未包含 X 或者 O 的单元格。

```python
def player(board):
    """
    Returns player who has the next turn on a board.
    """

    # If board is initial state, X gets the first move
    if board == initial_state():
        return X

    # Count numbers of X's and O's on board
    numX = 0
    numO = 0
    for row in board:
        numX += row.count(X)
        numO += row.count(O)

    # Next player is the one with fewer moves now
    if numX > numO:
        return O
    else:
        return X
```

图 4: player(board)

```python
def actions(board):
    """
    Returns set of all possible actions (i, j) available on the board.
    """

    # (row, cell) is legal move if its position on the board is empty now
    legal_actions = set()#set a list
    for row in range(3):
        for cell in range(3):
            if board[row][cell] == EMPTY:
                legal_actions.add((row, cell))

    return legal_actions
```

图 5: actions(board)

### 3.3 result(board，action)

result 函数将棋盘状态 board 和棋子的移动 action 作为输入，在不修改原始棋盘的基础上，返回一个新的棋盘状态，这个新的状态是当前下棋的玩家落子后产生的新的棋盘。

首先我们必须确保这个移动 action 对于棋盘是合法有效的，因此使用 raise 函数自定义一个异常类型，一旦执行了 raise 语句，raise 后面的语句将不能执行，以确保当 action 不合法时，程序不再无效进行。即当 action 元组 (i，j) 的 i 和 j 超出数组下标范围或者 (i，j) 表示的单元格并非空着的时候，程序将产生异常报错并结束。

更重要是，原始的 board 棋盘状态应保持不变，因为 Minimax 算法需要在过程中考虑所有不同的棋盘状态，如果简单地更新单元格则无法正确实现 result 函数。因此使用 copy.deepcopy 函数对棋盘状态 board 进行深层复制。

copy.deepcopy 函数是一个深复制函数。所谓深复制，就是从输入变量完全复刻一个相同的变量，无论怎么改变新变量，原有变量的值都不会受到影响。与等号赋值不同，等号复制类似于贴标签，两者实质上是同一段内存。像列表这样的变量，可以用深复制，从而建立一个完全的新变量，如果用等号给列表赋值，则新变量的改变将会引起原变量的随之改变。

```python
def result(board, action):
    """
    Returns the board that results from making move (i, j) on the board.
    """

    # Raise exception if index out of range or the cell is not empty
    if action[0] not in range(0, 3) or action[1] not in range(0, 3) or board[action[0]][action[1]] is not EMPTY:
        raise Exception("Invalid move")

    # Let player to make their move on the board
    new_board = copy.deepcopy(board)
    new_board[action[0]][action[1]] = player(board)
    return new_board
```

图 6: result(board，action)

### 3.4 winner(board)

winner 函数棋盘状态以 board 为输入，若 X 赢得游戏，则函数返回 X, 若 O 赢得游戏，则返回 O。获胜的条件是玩家的棋子在水平、垂直或者对角线上连续出现 3 个，若游戏没有赢家（可能是游戏正在进行或者游戏以平局结束），则返回 None。

```python
def winner(board):
    """
    Returns the winner of the game, if there is one.
    """

    # Evaluate board for each player
    for mark in [X, O]:
        # Check the horizontal chess pieces
        for row in range(0, 3):
            if all(board[row][col]==mark for col in range(0, 3)):
                return mark
        # Check the vertical chess pieces
        for col in range(0, 3):
            if all(board[row][col]==mark for row in range(0, 3)):
                return mark
        # Check the pieces in the diagonal direction
        diagonals = [[(0, 0), (1, 1), (2, 2)], [(0, 2), (1, 1), (2, 0)]]
        for diagonal in diagonals:
            if all(board[row][col]==mark for (row, col) in diagonal):
                return mark

    # Game is a tie or still in progress
    return None
```

图 7: winner(board)

### 3.5 terminal(board)

terminal 函数以棋盘状态 board 为输入，返回一个布尔值，表示游戏是否结束。若有人赢得比赛或者所有单元格被填满后打成平手，则函数返回 Ture，表示游戏结束，否则游戏还在进行，返回 False。

### 3.6 utility(board)

utility 函数以终态棋盘状态 board 为输入，返回一个实际的数值。若 X 赢得比赛，则返回

```python
def terminal(board):
    """
    Returns True if game is over, False otherwise.
    """

    # If there is a winner
    if winner(board) is not None:
        return True

    # If all cells have been filled
    all_moves = [cell for row in board for cell in row]
    if not any(move==EMPTY for move in all_moves):
        return True

    # Otherwise game in progress
    return False
```

图 8: terminal(board)

1，O 赢得比赛返回-1，平局则返回 0。

```python
def utility(board):
    """
    Returns 1 if X has won the game, -1 if O has won, 0 otherwise.
    """

    if winner(board) == X:
        return 1
    if winner(board) == O:
        return -1
    return 0
```

图 9: utility(board)

### 3.7 minimax(board)

minimax 以棋盘状态 board 为输入，返回当前玩家在棋盘上进行的最佳的移动。返回的最佳移动 (i，j) 应为棋盘上允许进行的动作之一，若 board 是终态棋盘，则 minimax 返回 None。

以 X 下棋为例，X 会考虑所有可能的移动，在其中考虑对手 O 的选择，在 O 的选择中，选择使得数值最大的那一个作为自己的落子选择。O 的选择同理 X，但目的与 X 恰恰相反，它试图使比分最小化。

```python
def minimax(board):
    """
    Returns the optimal action for the current player on the board.
    """

    if terminal(board):
        return None
    #X tried to maximize the score
    if player(board) == X:
        best_v = -math.inf#initialize the v
        for move in actions(board):
            max_v = min_value(result(board, move))
            if max_v > best_v:
                best_v = max_v
                best_move = move
    #O tried to minimize the score
    elif player(board) == O:
        best_v = math.inf
        for move in actions(board):
            min_v = max_value(result(board, move))
            if min_v < best_v:
                best_v = min_v
                best_move = move
    return best_move
```

图 10: minimax(board)

### 3.8 min_value/max_value

在 minimax 函数中调用的 min_value 函数以棋盘状态 board 为输入返回在所有可选的移动中 (考虑了 X 的选择)，选择使比分最小的那一个。max_value 与 min_value 互相调用，实现递归运算。

```python
def min_value(board):
    """
    Returns the minimum utility of the current board.
    """
    if terminal(board):
        return utility(board)

    v = math.inf
    for move in actions(board):
        v = min(v, max_value(result(board, move)))
    return v

def max_value(board):
    """
    Returns the maximum utility of the current board.
    """
    if terminal(board):
        return utility(board)

    v = -math.inf
    for move in actions(board):
        v = max(v, min_value(result(board, move)))
    return v
```

图 11: min_value/max_value

## 4 运行结果

在 runner.py 调用 tictactoe.py 作为一个 python 库使用，实现和 AI 的对弈！

### 4.1 选择成为 X 或者 O 玩家

开始游戏，选择自己的身份

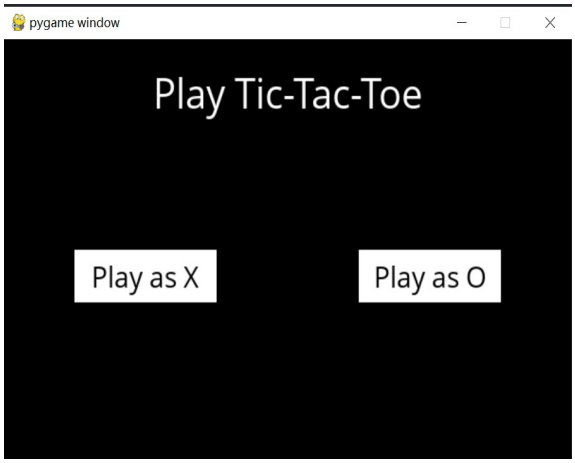


图 12: 开局

### 4.2 X 玩家先落子，AI 在思考后做出判断

作为 X 玩家，我们落子后，AI 通过计算所有可能的情况后选择最佳的移动。同时注意

到，当我们选择成为 O 玩家时，AI 成为 X 玩家，在 AI 进行第一步时，由于棋盘上没有参考的棋子，因此计算时间相对较长。

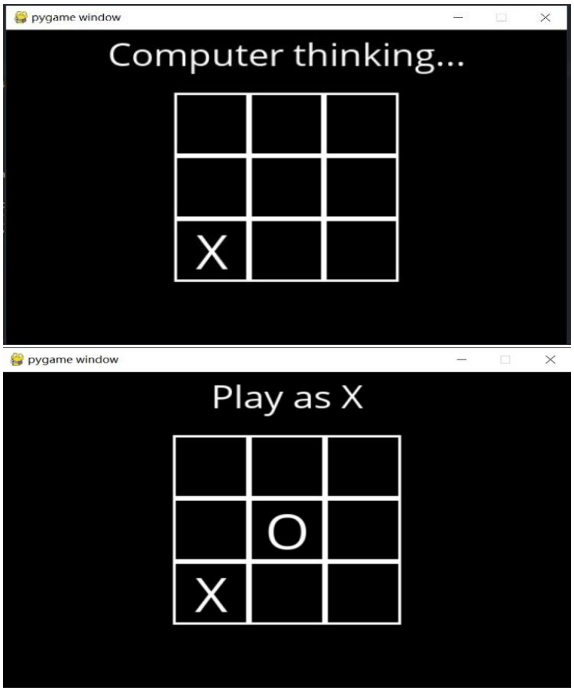


图 13: 对弈

### 4.3 对局结果

由于井字游戏是双方都给予最佳发挥的对局，因此我们永远无法击败 AI, 只有可能和它打成平手。(图 14、15)

## 5 总结

本次项目使用 minimax 算法，实现了人机对弈，从算法的层面更加深入地了解了人工智能在进行搜索时的一些处理。本项目在人机交互方面并未下太多功夫，主要是对 minimax 算法进行复现。将所要实现的功能进行模块划分，创建多个子函数来实现代码，这是编程中的模块化思想，能使得我们的代码更有条理。Minimax 是一个优秀的算法，本次实现的是最普通的 minimax，关于 minimax 的优化，例如 Alpha-beta 修剪算法等可以使得我们的 AI 在处理问题时更加高效，但由于个人能力有限，在本项目中并未体现。

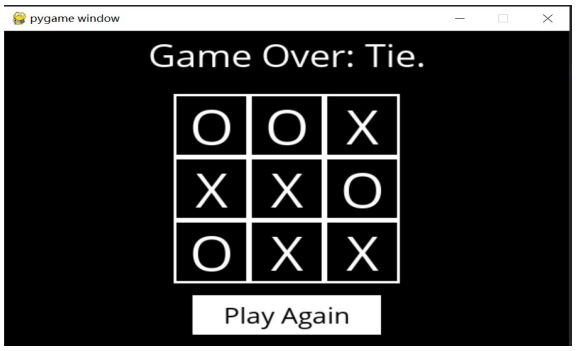


图 14: 平局

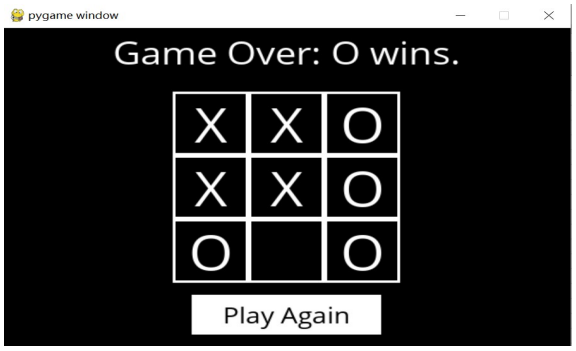


图 15: AI 获胜

## 6 感想

于我这样的 python 小白来说，本次大作业可以说是相当硬核。之前在中国大学 MOOC 跟随北京理工大学嵩天老师学习过一段时间《Python 语言程序设计》，有一点 Python 语法的基础，能看得懂一些简单的代码，但并未自己开发过项目。由于过了很长时间没有再接触 python 并且当时学习时只是在 python 自带的编译环境中学习，并没有使用 python 进行数据分析的能力。

这次大作业我首先还是对老师讲授过的内容进行复习，不懂的地方就上 B 站搜学习视频，磕磕绊绊地也大致理解了 minimax 算法是在讲什么。接下来就是最难的编程环节，总所周知，编程第一步就是要配置环境，这也是我比较头疼的一步。无论是 Anaconda 还是 VScode 亦或是 Pycharm，对于我来说都显得比较复杂，虽然跟着老师给的指导文档一步步安装，但在过程中难免会遇到很多奇奇怪怪的问题，我只能通过百度搜索，一个一个地解决。在环境配置

好之后，就要开始进行编程，我的编程能力是极差的，逻辑什么的不太拎得清，很多语法也忘了，一些本次项目中要用到的函数我也不会用，整个编程过程可以说是相当艰难。我认识到我自己是无论如何也写不出这个代码了，毕竟我是一点项目开发的经验也没有。之后就开始上网查找关于井字游戏的相关代码实现，这方面的代码还是很多的，对阅读的资料进行综合以及参考其中的代码后，我对网上的代码进行了自己的复现。但尽管是简单地将现成的代码在 VScode 中进行复现，也会出现各种各样的问题，文件读取失败或者是变量命名重复等等，都需要通过查阅资料来解决。对于参考的代码，我力求做到理解它们每一步的逻辑以及各种变量之间的关系，最终对于这个项目也有了比较明晰的认识。

总的来说，本次大作业我收获颇丰，既加深了对人工智能的认识，也锻炼了自己的编程能力，感觉自己离成为一名优秀的程序员又近了一步！

## 7 附录

GitHub 仓库链接

https://github.com/yanxiaomei-github/tictactoe

B 站展示视频链接

https://www.bilibili.com/video/BV1aY4y157VL/

## References


[1]B 站视频

https://www.bilibili.com/video/BV1tT4y137LB?p=2

[2]B 站视频

https://www.bilibili.com/video/BV1fA411W7kZ?spm_id_from=333.337.search-card.all.click

[3] 知乎文章

https://www.zhihu.com/question/46309360/answer/254638807

[4] 知乎文章

https://www.zhihu.com/question/41206352

[5]CSDN https://blog.csdn.net/qazwsxpcm/article/details/68946736

[6]CSDN https://ask.csdn.net/questions/7498273

[7]CSDN https://blog.csdn.net/qq_43511299/article/details/115260534

[8]CSDN

https://blog.csdn.net/gouqinan/article/details/122411736

[9]CSDN https://blog.csdn.net/kissmoon_/article/details/117437893

[10]MOOC 北理《Python 语言程序设计》

https://www.icourse163.org/course/BIT-268001

[11]CSDN https://blog.csdn.net/qq_38981614/article/details/115013188

[12]CSDN https://blog.csdn.net/weixin_39876450/article/details/112287980

[13]CSDN https://blog.csdn.net/qq_36931982/article/details/90551356

[14] 简书

https://www.jianshu.com/p/86117613b7a6

[15] 代码参考

https://download.csdn.net/download/weixin_42131316/16220671