上海交通大学安泰经济与管理学院

BUSS3620 人工智能导论

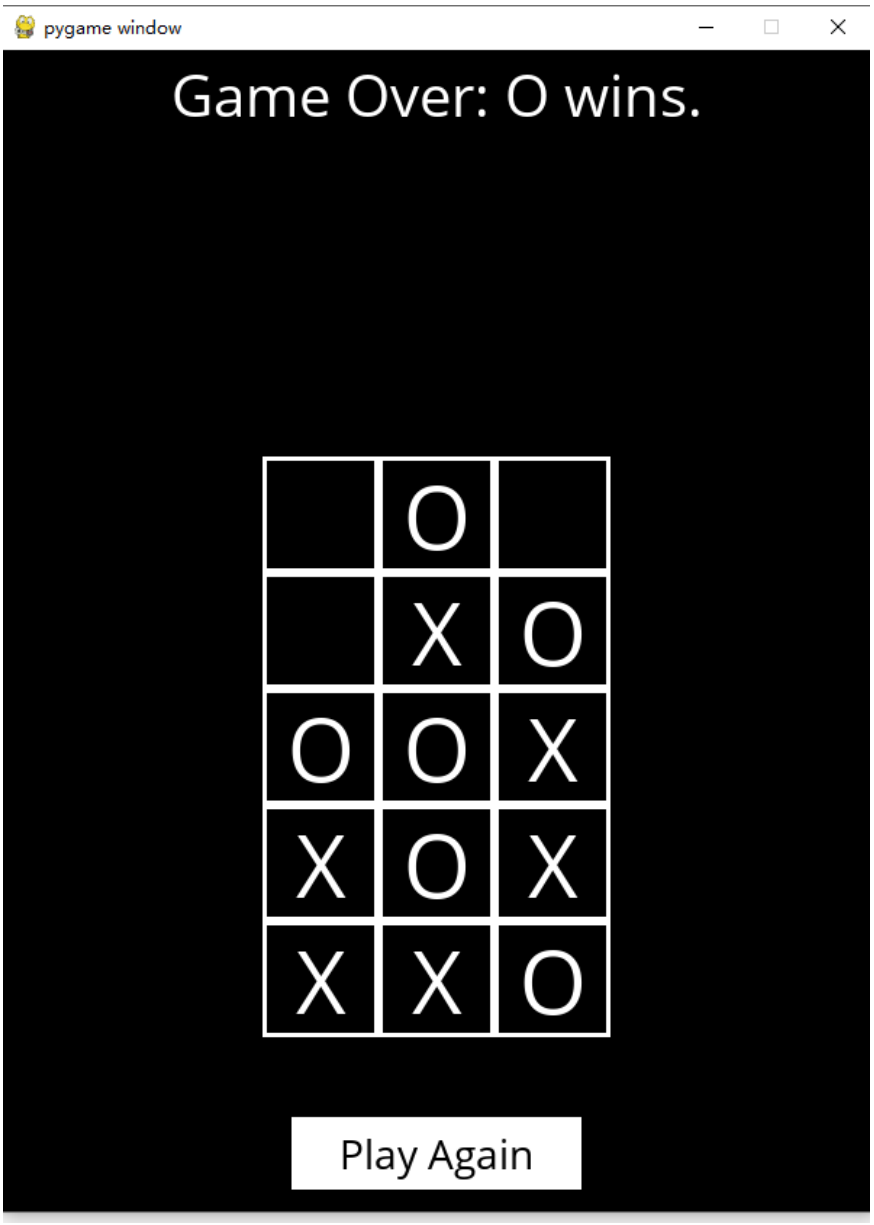
Project #2. 三子棋

刘佳璐 助理教授

三子棋

本项目会将使用人工智能技术来构建一个能够玩三子棋的 AI。您将通过 Minimax 以及 Alpha-beta 剪枝算法，计算出 AI 行动的最优解。

```
$ python runner.py
```

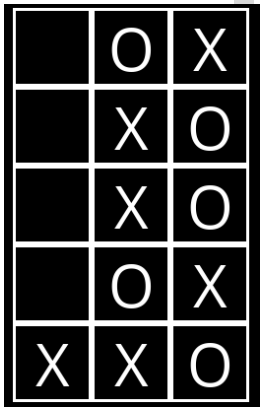


背景介绍

三子棋与经典的井字棋规则类似，玩家轮流将 `X` 或 `O` 放置在一个网格中，率先完成横排、竖排或对角线三连者获胜。但是有以下两个区别：

- 棋盘为 `5x3` 而非 `3x3`。
- 落子规则为，每一列必须从最底下的一格开始。依此可向上一格落子。

例如，在下图棋局中，根据规则，轮到 `O` 落子，此时唯一合法的落点是第一列第四行。`O` 落子后，轮到 `X` 落子，此时 `X` 可以落在第一列第三行，这样就连成了 3 连，所以当前局面中 `X` 必胜。



开始

- 从课程中心平台 Canvas 上下 `Week4 搜索 II` 单元中的 `week4_project.zip` 并且解压缩
- 当处于本项目文件所在的工作目录中时，在终端上运行 `pip3 install -r requirements.txt` 用来安装这次项目需要的 Python 包。

理解项目的相关文件

这个项目最主要的两个文件为：`runner.py` 和 `connect_three.py`。

`runner.py` 已经写好，包含运行游戏图形界面的所有代码。`connect_three.py` 包含玩游戏的 AI，需要同学你来完成。完成后，你可以通过运行 `python runner.py` 与 AI 进行对战！

首先打开 `connect_three.py`。我们定义三个变量：`X`、`O` 和 `EMPTY`，以表示棋盘每个位置可能的情况。`initial_state` 函数返回棋盘的起始状态。我们用五个列表（代表棋盘的五行）来表示棋盘状态 `board`，其中子列表包含三个可能的值，即 `X`、`O` 或 `EMPTY`。其他函数将留给同学们你来完成！

`Week4_project.zip` 中还包含几个文件。`OpenSans-Regular.ttf` 是字体文件，用来绘制游戏界面。`autograde` 文件夹包含测试代码的相关文件。

要求

`player` 函数

- 输入：当前棋盘状态 `board`
- 功能：确定当前是 `X` 还是 `O` 的回合
    - 在初始游戏状态下，`X` 玩家先走。随后，玩家交替进行每一步移动。
    - 如果输入的是一个终止状态的棋盘（即游戏已经结束），任何返回值都是可以接受的。
- 输出：当前该轮次的玩家（`X` 或 `O`）

`actions` 函数

- 输入：当前棋盘状态 `board`
- 功能：返回当前棋盘上所有合法的落子位置
  - 每一列必须从最底下的一格开始落子。依此可向上一格落子。
- 输出：
  - 一个集合，包含所有可供选择的行动，每个行动形式为`(i, j)`，其中 `i` 为行，`j` 为列。

`result` 函数

- 输入：当前棋盘状态 `board` 及某一行动 `action`
- 功能：返回在当前棋盘上进行某个行动后的新的棋盘状态
  - 原始棋盘应该保持不变。 `Minimax` 算法计算过程中会考虑许多不同的棋盘状态，计算过程不应该改变原始棋盘。这意味着简单地更新棋盘状态 `board` 并不是该函数的正确实现。在进行任何更改之前，您可能希望先对棋盘状态进行深拷贝（deep copy）
- 输出：
  - 一个新的棋盘状态

`winner` 函数

- 输入：当前棋盘状态 `board`
- 功能：判断棋盘上是否有胜者，并返回胜者（`X` 或 `O`）
  - 玩家可以通过在水平、垂直或对角线上连续三个棋子来赢得游戏
  - 你可以假设最多只有一个赢家（也就是说，不会有棋盘同时有两个玩家在三连线上，那将是一个无效的棋盘状态）
- 输出：
  - 若某玩家获胜，返回胜者（`X` 或 `O`），否则返回 `None`

`terminal` 函数

- 输入：当前棋盘状态 `board`
- 功能：判断游戏是否结束（胜者已定或棋盘已满）
  - 如果游戏结束，可能是因为有人赢得了游戏，或者因为所有单元格都被填满而没有人获胜，此时函数应该返回 `True` ，否则返回 `False`
- 输出：
  - `True` 或 `False`

`utility` 函数

- 输入：终止状态的棋盘状态 `board`
- 功能：返回该状态的效用值（`X` 胜为 `1`，`O` 胜为`-1`，平局为 `0`）
  - 你可以假设 `utility` 函数 只会在 `terminal(board)` 为 `True` 的情况下被调用
- 输出：
  - 该状态的效用值

`minimax` 函数

- 输入：当前棋盘状态 `board`，`alpha`, `beta`
  - `alpha/beta` 在初次调用时使用默认值负无穷/正无穷，在 `Alpha-beta` 剪枝算法中递归调用 `minimax` 时，你需要选择合适的值作为输入。
- 功能：返回 `alpha-beta` 剪枝算法找到当前棋盘状态下的最佳行动 `action` 以及对应最终动作的 `utility`

- 返回的 `action` 应该是棋盘上合法的动作之一的最优动作 `(i, j)`。如果有多个同样最优的移动，任何一个都是可以接受的。如果棋盘是终止状态的棋盘，`minimax` 函数应该返回 `None, 0`。
- `utility` 的含义是双方均使用最优策略时的最终 `utility`，这个值在 `Alpha-beta` 剪枝算法里需要被使用

- 输出：
  - 最佳 `action` 以及对应最终动作的 `utility`，形式为 `(i, j), utility`

你可以借鉴参考我们课上的代码案例。

你不应该修改 `connect_three.py` 中已经写好的其他部分，或者使用 Python 中的一些标准的程序包。一旦所有功能都正确实现，你就应该能够运行 `python runner.py` 并与 AI 对抗。如果你构建的游戏 AI 是正确的，你应该永远无法击败 AI。

测试代码

- 你可以使用代码 `pytest autograde/autograde.py --tb=no` 自行测试自己的代码是否满足要求。您需要安装 `requirements.txt` 中的 `pytest` 包。
- 请先确保你的程序能够成功运行并输出结果。请确保你的工作目录中包含 `connect_three.py`。