Homework 2: Connect Four Bot

Create an AI to play "Connect Four", using the minimax algorithm.

Download the files `connect4.py` and `connect4player.py`. `connect4.py` contains the Connect Four game, which you will not modify at all. Your job is to modify the `ComputerPlayer` class in `connect4player.py`. It has two public methods:

- `__init__(self, id, difficulty_level)`: the constructor. `id` will hold either `1` or `2`, depending on whether this player is to be Player 1 or Player 2. `difficulty_level` is a positive integer, that represents the number of plies to look ahead.
- `pick_move(self, rack)`, which determines the move to make. `rack` holds a 2D tuple indicating the current board state. It is column-major, with element `[c,r]` indicating the position `c` from the left, `r` up from the bottom. A `0` indicates that no disc is there, `1` indicates that Player 1 has a disc there, and a `2` indicates that Player 2 does. It must return an 0-indexed integer, indicating the column in which to play.

At present, `ComputerPlayer` just plays in a random location (after pausing dramatically).

The evaluation function you should use is as follows. You must inspect every "quartet", in which there are four spaces in a row in which a player could potentially win. A standard rack contains 24 horizontal quartets, 21 vertical quartets, and 24 diagonal quartets (12 going up-right and 12 going down-right). For each of these 69 quartets:

- Point value is positive if it favors the AI, and negative if it favors its opponent.
- If it contains at least one disc of each color, it cannot be used to win. It is worth 0.
- If it contains 4 discs of the same color, it is worth $\pm\infty$ (since one player has won).
- If it contains 3 discs of the same color (and 1 empty) it is worth ± 100.
- If it contains 2 discs of the same color (and 2 empties) it is worth ± 10.
- If it contains 1 disc (and 3 empties) it is worth ± 1.

Some hints:

- If you need a refresher on how the game works, go to `https://en.wikipedia.org/wiki/Connect_Four`.
- Do not assume that the board is a standard-sized one. It may have unusual dimensions.
- Be sure to bug test extensively, using different levels. This code can get twisted, fast.
- You may find it easier to implement negamax rather than classic minimax. (Mathematically, the two algorithms are identical.)
- Run the program with the `-h` command-line option to learn some useful commands.

Extra credit: Implement alpha-beta pruning. If you choose to do this, make sure that you can turn it on and off easily, so that you can check your results. Remember, proper alpha-beta pruning will never change a move, but will calculate them more quickly.