COMPSCI 383 – Spring 2020

Homework 2

Due Monday, March 2nd at 11:59pm

You are encouraged to discuss the assignment in general with your classmates, and may optionally collaborate with one other student. If you choose to do so, you must indicate with whom you worked. Multiple teams (or non-partnered students) submitting the same code will be considered plagiarism.

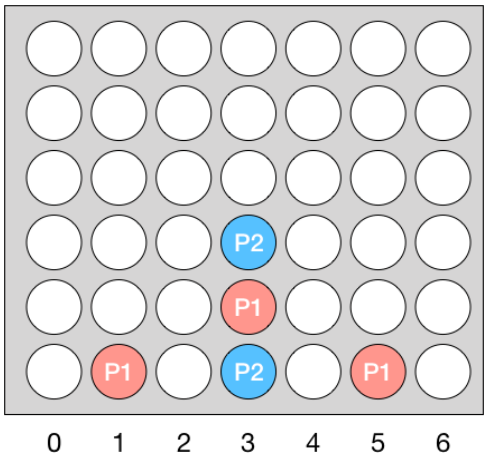
Code must be written in a reasonably current version of Python (>3.0), and be executable from a Unix command line. You are free to use Python's standard modules for data structures and utilities, as well as the pandas, scipy, and numpy modules. Do not use any modules or libraries that implement minimax.

Connect4

For this assignment, you will implement the Minimax algorithm for "Connect4", a vertical tic-tac-toe-like game (see https://www.mathsisfun.com/games/connect4.html for a playable demo). The board itself is made up of six rows and seven columns, which are filled with tokens. Players take turns dropping tokens into vertical columns, and the first player to get four in a row (horizontally, vertically, or diagonally) is the winner. If the board fills up and neither player has four in a row, the game is a tie.

The board shown on the right illustrates a game in progress after the following five moves have been made:

Player 1 to column 1
Player 2 to column 3
Player 1 to column 3
Player 2 to column 3
Player 1 to column 5



The utility of a win for Player 1 is 1, a draw 0, and a win for Player 2 is -1. Your solution will use the Minimax algorithm to calculate the utility of any given game state and dictate the next optimal move. In addition to "standard" Minimax, you will develop a depth-limited version that utilizes a heuristic for determining state utility, along with a more efficient version that utilizes alpha-beta pruning to limit the search.

Supplied Code

We've created a Python module that will handle representing the game state and manage play called `connect4.py`. You do not need to change any of the code in this file, but will need to understand how it works to program your game playing agent.

To play a game, you run `connect4.py` from a command line and supply it with two arguments that determine which type of agents are playing (one of 'r', 'h', 'c', or 'p'; see code for more details). For example, to play against a random opponent, you would type:

```
> python connect4.py r h
```

The Minimax code you will write should go into the file `agents.py`. Specifically, you must fill out the implementations for the following methods:

- `ComputerAgent.minimax()` – Here, you must implement the logic for Minimax. Your code should traverse the game tree and return the utility value for the given state.

- `ComputerAgent.evaluation()` – In cases where a depth limit was supplied, your Minimax method must stop its traversal when it reaches the given depth limit, and instead utilize this function to estimate the utility based on different features of the game state. How you do this is entirely up to you, subject to the restriction that your method must run in O(1) time.

- `ComputerPruneAgent.minimax_prune()` – this method will be similar to the `minimax()` method above, but will incorporate alpha-beta pruning (AIMA Sec. 5.3) to make the calculation more efficient.

See the comments in the code for more details. Make sure that you *do not change the API* (number of arguments or return values) for the above methods, as we'll be calling them directly to evaluate your code.

Testing

You should test your agent on a variety of boards to ensure that it's functioning correctly. To help with this, we've included a file called `tests.py` to hold test cases. You should modify the test boards that are there (you are also free to create more).

We will be checking your individual method implementations, as well as verifying that your agent can successfully beat some simple agents. Note that your solutions will only be tested on well-formed boards – we are not going to feed your program incorrectly formatted game states.

In addition, we will be pitting all of your submissions against each other, and will award bonus points to those whose code can consistently beat other members of the class. Note that since Minimax is optimal, your agents can only be differentiated when using a depth limit – your "secret sauce" will be in your implementation of the `evaluation()` method.

What to Submit

You should submit your versions of the three Python files (`connect4.py`, `agents.py`, and `tests.py`) required to run a game, along with a `readme.txt`, which should contain:

- Your name(s)
- The *worst* meal you've ever eaten
- A short description of your thinking and the design behind your `evaluation()` function, along with some thoughts on its effectiveness
- Short descriptions of the cases that the boards in `tests.py` are meant to test
- Notes or warnings about what you got working, what is partially working, and what is broken

Grading

We will run your program on a variety of test cases. Your grade will be proportional to the number of test cases you pass. The test cases for grading will not be available to you before grading, but we will make tests available that will check the format of your output.

Code that does not run will not receive credit.