CS 470 Programming Assignment:
Monte Carlo Tree Search for Connect-4

Honor code note: You are welcome to discuss the assignment with other students, but you must individually and independently write the code and other deliverables that you submit for the assignment. Any copying of code from outside sources or other students will be considered an honor code violation.

This assignment involves implementing MCTS in the (now familiar) game ofConnect-4. You should first download the `PA-MCTS.zip` file from Canvas and unzip in on your computer. This code has minor changes from the previous Connect-4 assignments to support MCTS. You will need to copy over your code from the previous assignments where necessary to
The folder you are provided contains:

- `ConnectFour.py` - The rules and GUI for Connect-4.
- `Player.py` - This is the main file that you will be editing.
- `Player_blank.py` - If you would prefer to write all of the MCTS code yourself, this is a blank `Player.py` file that you can start from.

To run the code you should type:

```
python ConnectFour.py player player
```

from the command line, where `player` is one of the following options for player: `human, ai, random, mcts`.

- `human` - this will ask the human for moves
- `ai` - this will use alpha-beta pruning, unless the opponent is random, when it will use expectimax
- `mcts` - this will use mcts
- `random` - this will randomly select actions

Tasks

For this assignment you need to complete the following tasks

- **Task 1** - Copy over your alpha-beta and expectimax code from the previous assignments into the appropriate places in the new code. Then read through the new code related to MCTS. Specifically look at:

  - `MCTSNode` class - the node for our MCTS search tree. You should be familiar with its structure and member functions, especially the `select()` function, which is implemented for you as an example of how to work with the MCTS nodes
  - `get_mcts_move()` inside the `AIPlayer` class. This is implemented for you, and will call use the MCTS code to determine the action to take from the given state. You should be familiar with this, as you might need to adjust the number of iterations, which is how we will control how long the MCTS agent takes.

- **Task 2** Implement the `upper_bound()` function in the `MCTSNode` class. This function will return the UCB for the node, and takes as an input parameter the number of visits of the parent node. Look for the place in the code marked "TASK 2".
- **Task 3** Implement the `simulate()` function in the `MCTSNode` class. This function will run a random game to completion from that node's state and then back-propagate the result through the search tree's parent pointers. Comprehensive psuedocode is provided to help you understand what the steps will look like within the provided framework. Look for the place in the code marked "TASK 3".

You now should have working MCTS code. Once it seems to be working, you can play against it, and then you should play it against your alpha-beta code as well. Then please complete the following experiments and answer the associated questions.

Questions

1. Experiment with different c values in your MCTS code. (Unfortunately, the code isn't set up right now to play different c valued MCTS agents against each other). For each, you can look at things like how many times the different actions were sampled at the root node, and of course, how well it seems to perform against either you or your alpha-beta pruning code. What c value seems to work best?
2. Play your MCTS agent against your alpha-beta pruning agent for 5-10 games. Which is victorious more?
3. Write a few paragraphs comparing and contrasting the way that the alpha-beta and MCTS agents work.

Deliverables

You are required to hand in the following for this assignment:

1. Your code with the required parts implemented or changed.
2. Your answers to the questions from the previous section.