CS 470 Programming Assignments: AI for Connect-4

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

This handout describes three assignments that each involve you implementing an AI strategy for the game of Connect-4 in python. You should first download the `PA-Connect4.zip` file from Canvas and unzip in on your computer. You will use this same software for three programming assignments in the class (modules: Adversarial Search I, Adversarial Search II, and Monte Carlo Tree Search).

1 Software Overview

The folder you are provided contains:

- `ConnectFour.py` - The rules and GUI for Connect-4. This code parses command line arguments and runs the experiments.
- `Player.py` - This is the main file that you will be editing. It contains starter code and some utilities that you are welcome to use.

The software allows you to play one game interactively (where the game pauses and displays the current state between each move) or multiple games without interaction. You can also play as a human.

To run the code in its simplest form you should type:

```
python ConnectFour.py player1 player2
```

from the command line, where `playerN` is one of the following options for each player: `human, random, ab, expmax,& mcts`.

- `human` - this player will ask the human for moves
- `random` - this player will select actions uniformly at random
- `ab` - this player will use alpha-beta pruning (first Connect-4 assignment)
- `expmax` - this player will use expectimax search, assuming it is playing against a random agent (second Connect-4 assignment)
- `mcts` - this player will use MCTS (third Connect-4 assignment)

1.1 Command Line Options

Running the code without any options will have the two specified players compete in a single game, with the first specified player going first. This game will be run in interactive mode, where you will be able to see the game state before and after each turn. In interactive mode the game pauses between turns waiting for your input.

Other command line parameters can be specified as follows:

- `-p1` - the single string after this specifier will be passed as a parameter to the first player listed on the command line, if that player is an AI agent(`ab`, `expmax`, or `mcts`). (The parameter is ignored by the random and human players). You can parse and use this as you wish. This will allow you to modify some aspects of your agent from the command line, for example, you can play two copies of your alpha-beta agents against each other, where each uses a different search depth. The code to use the parameter in this way is given for you in the `AIPlayer.__init__` function. The default value for this is `None`.
- `-p2` - the single parameter string for the second player on the command line. This works in exactly the same way as the parameter for the first player.
- `-n` - the number of games that each player will go first in the match. If this number is 1, only 1 game will be played, with the order given on the command line. If the number is greater than 1, then twice this many games will be played, with each player going first in the game this many times. The default value for this parameter is 1.
- `-t` - the number of seconds each player has to make their move. The default value is 60.

1.2 Command Line Examples

Some examples of using the command line:

- `python ConnectFour.py ab expmax -p1 5` - this will run a single game, in interactive mode, between the alpha-beta player as player 1 and the expectimax player as player 2. The alpha-beta player will have a parameter of 5, which it could use to set its depth-limit.
- `python ConnectFour.py human ab -p2 2` - this will run a single game, in interactive mode, between a human (go you!) as player 1 and the alpha-beta player as player 2, with parameter of 2.
- `python ConnectFour.py mcts ab -p1 100 -p2 3 -n 5` - this will run 10 games (5 with each player going first) between the Monte Carlo Tree Search player with parameter 100 and the alpha-beta player with parameter 3.

2 PA: Adversarial Search I

In this assignment you will implement alpha-beta pruning to play Connect-4. The game is too large to search exhaustively, so you will use a depth-limited search, with alpha-beta pruning. The following tasks must be completed for this first Connect-4 assignment:

- **Task 1** - Implement an evaluation function for your alpha-beta search to evaluate non-terminal states. A function, called `evaluation_function` is provided for you to fill out. Look for the comment that says **#YOUR EVALUATION FUNCTION GOES HERE**, and add your code there.
- **Task 2** - Implement the alpha-beta search code. A function, called `get_alpha_beta_move` is provided for you to start from. You are welcome to add whatever helper functions you wish. Look for the comment that says **#YOUR ALPHA-BETA CODE GOES HERE** and add your code there.

Additionally note that there are a few helper/utility functions provided for you in the `Player.py` file, at the bottom of the file. You are welcome to use these, or add to them as you see fit.

2.1 Questions

Once you have completed and debugged your code, you should answer the following questions:

1. What is a high level overview of how your evaluation function works?
2. For depth limits of 1-5, how long does it take your alpha-beta code to find a move at the beginning of the game? (If any level takes way too long you can kill the code and report that fact)
3. Play against your agent at several depth limits. At what depth limit does your program beat you (if any)?
4. Play your agent with different depth limits against a random agent, for 10 games. Does depth impact its ability to win?
5. Play two versions of your code with different depth limits against each other. Do this for several combinations of depth-limits. How does depth of search seem to impact the ability of the agent?
6. Do you think there is an advantage to going first or second in Connect4? Devise at least one experiment you can do to shed light on this question and report the results, and how they support your answer.
7. How long did this assignment take you? Were any portions especially frustrating? Anything you would change next time around?

2.2 Deliverables

For this assignment you need to turn a `.pdf` file with the following:

- Your `Player.py` file with your evaluation function and alpha-beta code implemented.
- Your answer to the questions.

3 PA: Adversarial Search II

In this assignment you will implement expectimax search to play Connect-4. This approach assumes that the agent is playing against a random opponent. The following task must be completed for this first Connect-4 assignment:

- **Task 1** - Implement the expectimax search code. A function, called `get_expectimax_move` is provided for you to start from. You are welcome to add whatever helper functions you wish. Look for the comment that says **#YOUR EXPECTIMAX CODE GOES HERE** and add your code there.

As before, you are welcome to add helper functions to your hearts content to help your code stay well-organized.

3.1 Questions

Once you have completed and debugged your code, you should answer the following questions:

1. For depth limits of 1-5, how long does it take your expectimax code to find a move at the beginning of the game? (If any level takes way too long you can kill the code and report that fact)
2. Play against your agent at several depth limits. At what depth limit does your program beat you (if any)?
3. How does the play style of this agent differ from your alpha-beta agent?
4. Play your agent with different depth limits against a random agent, for 10 games. Does depth impact its ability to win?
5. Play your alpha-beta agent against your expectimax agent for 10 games. Do this for several combinations of depth-limits. How does depth of searches impact the effectiveness of the agent?
6. How long did this assignment take you? Were any portions especially frustrating? Anything you would change next time around?

3.2 Deliverables

For this assignment you need to turn a `.pdf` file with the following:

- Your `Player.py` file with your evaluation function and alpha-beta code implemented.
- Your answer to the questions.

4 PA: Monte Carlo Tree Search

For this assignment you will implement an MCTS agent to play Connect-4. To begin, read through the provided starter MCTS code. If you prefer, you can start from scratch; you are not required to use the MCTS starter code, but your agent needs to use MCTS to make its decision. You should 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. This can be hard-coded in the program or passed in on the command line when running the code.

The following tasks need to be completed.

- **Task 1** 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 "YOUR MCTS TASK 1 CODE GOES HERE".
- **Task 2** 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 "YOUR MCTS TASK 2 CODE GOES HERE".

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.

4.1 Questions

1. Experiment with at least three different c values in your MCTS code. You should be able to modify the code to set that value from with the command line parameter, or just hard-code the change. For each value of c, you should observe how many times the different actions were sampled at the root node, and also how well it performs against your alpha-beta pruning agent in 10 games. What c value seems to work best?
2. Play your MCTS agent against your alpha-beta pruning agent for 5-10 games. Use a combination of different max-iterations for MCTS and depth-limits for alpha-beta. How do the agents compare?
3. Play a few games against your MCTS agent. How does it do, compared to alpha-beta and expectimax? Which of your agents is the strongest?

4. How long did this assignment take you? Were any portions especially frustrating? Anything you would change next time around?

4.2 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.