CS 370 – Project 2: Connect Four

Due Date: Sunday, October 29, 2017

Introduction

In this project, you will implement Minimax and Alpha-Beta Pruning to build a competitive Connect Four-playing agent. I supply code (in Java and Python) for the game itself and an agent that makes random moves. The only thing you need to do is create another agent class. If you have never heard of Connect Four, read up on the game on Wikipedia.

Included Classes

Minor differences exist between the provided Java and Python code. The notes below are based on Java, but Python equivalents should be clear.

Classes fall into three categories. Classes in the `c4.mvc` package define the game rules and the interface for playing the game. Three interfaces are available: A GUI, a console interface, and a "silent" interface. The `ConnectFourModel` class contains the game logic and has the functions that players can use to get information about the game (e.g., the current state of the grid, what moves are available). Classes in the `c4.players` package define players in the game, including the agent you will develop for this project. You have been provided with a `HumanPlayer` class that allows you to play, and a `RandomPlayer` class that makes random moves. The `c4.ConnectFour` class is your main class to start the game.

Note: The GUI view is only available in Java and was originally designed for Human vs. Human play. I'm not 100% confident in its use for Human vs. AI games or AI vs. AI games.

Note 2: I have also provided code with a similar class structure for Tic-Tac-Toe. This code is entirely to help you diagnose errors in your Minimax and/or Alpha-Beta Pruning implementations. I will not grade, or even look at, anything related to Tic-Tac-Toe.

File to Edit: For either language, you will create a subclass of `ConnectFourPlayer`, which can be found in `c4players.py` or `ConnectFourPlayer.java`. You will probably also want to briefly modify the `main` function in `connect4.py` or `ConnectFour.java` to use your player. Do not modify any other code, except Tic-Tac-Toe code for debugging.

Evaluation: Your code will be evaluated based on sample test-cases (in the case of the functions you write) and in-game performance (for the overall agent). Please *do not*

change the names of any of the provided functions or classes. Modifying code that you were instructed not to modify is grounds for losing credit on this assignment.

Getting Help: You are not alone! If you find yourself stuck on something, contact me for help sooner rather than later. I want these projects to be rewarding and instructional, not frustrating or demoralizing. But I do not know how or when to help unless you ask.

Also, remember, discussion of the assignment is perfectly acceptable. However, this discussion should not extend to sharing code. If in doubt of what would constitute academic dishonesty, contact me.

Assignment Questions

For all questions below, function names in camelCase refer to Java, while names in underscore_case refer to Python. Name your functions according to the style expected in your chosen language.

Question 1 (2 points) – Dumb Player

Implement a subclass of the `ConnectFourPlayer` class called `ConnectFourAIPlayer`. For now, follow the guide of `ConnectFourRandomPlayer` by having the constructor get a copy of the game model. (You may wish to review the public functions of the game model.)

Override the `getMove/get_move` function so that it always makes a legal move, without having the choice of move be completely random. (That is, you should do this differently from the `getMove` function in `RandomPlayer`.)

Note that for this question and all other questions in this assignment, it is the responsibility of the agent to make sure its move is legal, where a legal move is a number between 0 and 6 (inclusive) and the column specified is not already full. The game model will consider an illegal move to be a forfeit.

When you complete this, congratulations! We have a (barely) intelligent agent. It probably performs slightly better than the random agent, but you should still have no problem beating it.

Question 2 (2 points) – Terminal Test

The minimax and alpha-beta algorithms require a number of helper functions in order to work properly. Implement a function `terminalTest/terminal_test` that takes in a board state as an argument and returns `True` if the game would terminate in this state – that is, if a player has won or if the game has reached a full board (a draw).

In Python, the game board is represented as a 2-d list of integers. In Java, the game board is represented as a 2-d array of ints.

The `TicTacToeAIPlayer` class provides a sample of this function for Tic-Tac-Toe.

Question 3 (3 points) – Actions

For any potential board state, the algorithm must be able to identify what actions are valid. As noted above, an action is a number between 0 and 6 (inclusive) where the number represents a non-full column. Implement a function `actions` that takes in a board state as an argument and returns a list (Python) or `int[]` (Java) representing the valid moves from that state.

The `TicTacToeAIPlayer` class provides a sample of this function for Tic-Tac-Toe.

Question 4 (3 points) – Results

Naturally, if we know the actions, then we need to be able to follow through with creating the game state that follows from those actions. Write a function `result` that takes two arguments: the board state and an action (an integer between 0 and 6). Return the board state that exists after that action is carried out.

The `TicTacToeAIPlayer` class provides a sample of this function for Tic-Tac-Toe.

Hint 1: Typically a game state includes whose turn it is, but our game state only includes the game board. How can you deduce whose turn it is? (You'll need to do this since the grid uses 1s and 2s in the grid to represent whose tokens are whose.)

Hint 2: Make sure that you're making a sufficiently deep copy of the grid when adding the new move. If your copies are too shallow, then your minimax algorithm may not be working from the correct game state as it backtracks.

Question 5 (8 Points) – Alpha-Beta

Implement the alpha-beta pruning algorithm described in the textbook (pseudocode on p. 170), including the minimum-value and maximum-value functions as described. You'll also need a utility function. Since we're currently not cutting off the depth of our search, define your utility function so that winning states for your player return 1000, losing states for your player return -1000, and draw states return 0.

Refactor your old `getMove`/`get_move` function to be called `dumbGetMove`/`dumb_get_move`. Create a new `getMove`/`get_move` function that uses your alpha-beta search functions to decide which function to carry out.

How to Debug For This Question

The `TicTacToeAIPlayer` class provides skeleton code for the functions related to getMove and Alpha-Beta Pruning. You should be able to paste your implementation for Connect Four directly into Tic-Tac-Toe (or vice versa) and see results, since the algorithm is independent of the game being played, as long as you've implemented Q2-Q4 correctly. (In Java, you may change the return types on any of these functions for Tic-Tac-Toe, except `getMove()`.) You can run the game from `TicTacToe.java` or `tictactoe.py`.

In my tests, the `TicTacToeAIPlayer` comes up with a move pretty much instantaneously (except on the first move). If your code is correct, the AI player should NEVER lose. If you (the human player) play correctly, the AI player will always force a draw. If you screw up, the AI player may win. If all of these factors check out in Tic-Tac-Toe, then you should feel confident in your code for this question. If the code works for Tic-Tac-Toe, but not Connect Four, then there is probably an issue with your solution(s) to earlier questions.

Question 6 (8 Points) – Difficulty and Mid-Game Utility

If you've written your code correctly, you probably have an agent that is incredibly difficult to beat. That's no fun. It may also take a fair bit of time to decide what move to make because of the game's branching factor. Now, we are going to modify our code so that the agent cuts off its search after a certain number of plies. The more plies that our agent investigates, the more challenging it will be to play against.

Modify your agent by adding an instance variable that represents the maximum depth that you will search to. Add an optional integer parameter to the constructor (Python) or a second constructor that takes the model and an integer (Java) to represent this cut-off level. (For your default max-depth agent, you could use 42, since that is the maximum number of turns in the game.)

Modify your minimum-value and maximum-value functions so that, in addition to returning a state's utility if we've reached a terminal state, it also does so if we've reached the cutoff depth. You should also modify each of these functions so that they have a parameter representing the current depth of search. (Section 5.4.2 describes the necessary changes to the Alpha-Beta Search pseudocode.)

Of course, now our utility function is insufficient, since we wrote it to only measure terminal states. Here is where your creativity comes in. Modify your utility function so that it is capable of measuring a state's utility when given a non-terminal state. What this function looks like is up to you. Give some thought to what a high-quality or low-

quality state looks like. Remember that winning and losing states should still return the highest and lowest possible values, respectively.

Question 7 (4 points) – Explanation of Utility

Write a brief summary of the utility function that you developed for Question 6. Explain the logic that went into measuring a game state's utility. I should be able to follow the logic of how your agent determines the quality of a state without much effort.

Tournament! (3 points extra credit)

I will pit all agents submitted against each other in a round-robin battle to determine whose agent is the best. The winner(s) receive the extra credit points. The tournament will work as follows:

- Python and Java agents will participate in separate tournaments.
- The tournament will be held seven days after the project deadline. Agents not submitted by this time will be ineligible to participate in the tournament/receive extra credit.
- An agent will play each other agent twice, once as Player 1 and once as Player 2.
- The winner of each game receives 2 points. In the case of a draw, both players receive 1 point. Losses receive 0 points. Forfeits due to illegal moves (or any other uncaught Exception) are considered losses.
- The tournament will be conducted using a maximum-depth to be determined. It will not be a full-game search. It is likely that the tournament will be held with a depth parameter such that `5 <= depth <= 10`.
- The winner of the tournament will be the agent that racks up the highest point total. A winner will be selected to receive extra-credit in both the Java and the Python tournament.
- Tiebreaking: In the event that two or more agents have the same number of points, tiebreaking will occur as follows:
    - Rank all tied agents according to points earned in games against the other tied agents.
    - If one agent leads this ranking, that agent is the winner. If multiple agents are tied to lead this ranking, but the number of agents is fewer than the number of originally-tied agents, then repeat the process using the remaining agents.
    - If all agents have the same score in the tiebreaker ranking, then all remaining agents receive the extra credit.

Testing your code: Functions have been provided in `connect4.py` and `Connect4.java` to play a single game or to play many consecutive games. You may wish to pit your agent against the `RandomPlayer`, other people's agents, or yourself.

You should also evaluate the individual functions that you have been tasked with writing to make sure they do what they say they do. Make use of the `TicTacToeAIPlayer` where you can.

What to Submit: Zip all project files for your chosen language, including your analysis for Q7, and submit it on Kodiak by the due date. Any submission after the due date is subject to the late penalties described in the syllabus.