Nicholas Callahan

Connect-4 AI Honors Contract

This program replicates the common Connect-4 board game using Java in the terminal. It allows the user to do two things:

1. Simulate two AI algorithms that compete at Connect-4:
   MinMax (using alpha-beta pruning) and Monte Carlo Tree Search
2. Allow the user to play against either algorithm with any depth or # of iterations.

Since I already have learned and implemented Min-Max in our classwork, applying the logic to Connect-4 wasn't much more difficult. The most difficult work of this project came with everything else, like making Connect-4 work in the terminal, and learning, then implementing, Monte Carlo Tree Search.

My actual implementation of Connect-4 ended up being about as simple as our TicTacToe. It uses a single 2D array to store the pieces, 0 for empty, 1 for player1, and 2 for player2. By default, player 1 is MinMax and player 2 is MCTS, but when you choose to play against either AI, you choose which player to be once prompted. So once the internal logic of the board, in board.java, was properly planned out and working, that part became easier and easier. But at first, putting together the game from scratch instead of having code to start with, like in TicTacToe, was daunting.

The second main difficulty was learning and understanding Monte Carlo, so I turned to YouTube and watched a few videos to understand conceptually how it works, and then began planning and implementing it in Java. This is another circumstance where it seemed more daunting from the beginning, but began to work itself out as I chipped away at it. At the heart of this is understanding UCB1, which is the node selection strategy MCST uses. This is a big part of the MCTSPlayer.java implementation, including step 1, which also uses a sqrt(2) as the exploration constant. This is diving a little too deep into the theory behind MTCS for the length of this paper allows, but the main point is that learning MTCS was a large hurdle in the creation of this program.

All in all, the main takeaways I've learned from creating this project are something very cool - a full top-to-bottom creation. In some classes, I've gotten close or similar to this amount of involvement, but I'm not sure I've had to learn something entirely separate from my class work and apply it to my own game concept. When I decided to do this for my honors contract, I knew basically nothing about Monte Carlo search, and we were barely at the point in the semester to be able to create any game AI or just make a game run in the terminal at all. So to go from the beginning of the semester to this fully fleshed out project is very satisfying.

Nicholas Callahan

Program instructions: to play against an AI

Commands:

javac *.java

java Play.java

The program will ask you to choose:

- Opponent: MinMax or MCTS
- Turn order: go first (X) or second (O)
- Difficulty: search depth (MinMax) or iteration count (MCTS)

Enter columns 1-7 on your turn. Invalid or full columns are rejected, and you will be re-prompted.

Program instructions: to simulate the AIs playing against each other

Commands:

javac *.java

java Simulate.java

The program will ask you to enter:

- Number of games (default 100)
- MinMax search depth (default 8)
- MCTS iterations (default 5000)

Press Enter on any prompt to use the default. Enter 1 for number of games to watch a single verbose game with the board printed after every move.

During a multiple-game simulation:

Players alternate who goes first each game, so neither side has a permanent advantage.

A live result line prints after each game, followed by a full summary when all games finish.