CS 1102

A term. 2013.

Assignment #4

Due: Tuesday October 1, 2013 @ 9:00 a.m. via turnin

Assignment guidelines

1. You should use the intermediate student language. You may use any constructs in ISL, unless doing so would contradict the assignment details.
2. You must provide data definitions and templates. As data get more complex, being careful with getting started correctly becomes more important.
3. You should spend time thinking about good test cases for each part. Explain why you chose the test cases you did (a quick note in a comment in the line before the test case will usually suffice). Show us that you've thought through boundary conditions and ways your program could break.
4. For complicated test cases, provide an explanation for why you believe your output is correct.
5. You should not modify the state representation, the provided code (except where noted in the starter file), or any of the constants (you may adjust ROWS and COLUMNS to see how that impacts your code).

Connect Four the game

In this assignment, you will create a computer player for the game Connect Four. You should familiarize yourself with the rules for the game (they're fairly simple). The computer will play the black pieces and move second. Your job is to create a computer player that is able to play a passable game of Connect Four, using what you've learned about templates and generative recursion.

Provided functionality

For this assignment, you are given a Connect Four program that displays the board, accepts user input for moves and places the checker at the correct location, and has a sample (very crude) computer player that makes random, but legal, moves. Moves are represented as a list of length 2 with X and Y coordinates of the move. The world state is represented as a structure that contains the board position, whose turn it is, and additional configuration information (at present, unused).

The following functions are provided for your use:

1. legal-next-moves: world-state→list of moves. From the given world-state, it creates a list of legal moves the current player may make.
2. check-win? World-state → boolean. Determines whether the current world-state is a winning position.
3. make-move: world-state move → world-state. Takes as input the present state and a proposed move, and returns the world-state, which has been updated to include the current configuration of pieces on the board and changes who is the current player.

4. main. Initializes the world. The starter file is designed to create a game board.

Creating an evaluation function

Your first task is to create an evaluation function. The input for this function is a world-state, and it outputs a Number. The evaluation function determines how good a board is for each player. Positive values indicate a strong board position for the human, while negative values indicate a strong board position for the computer. This function must be called *evaluation-function*.

A key bit is that your evaluation function should recognize a won (or lost) game, and value it accordingly. It should value game positions between winning and losing according to how good they look for each player. You are not expected to spend the majority of your time on this part of the problem. A function that reasonably orders positions is fine.

Searching for good moves

For this assignment you will implement MiniMax search to find a good move for the computer. You should implement MiniMax search as was discussed in class so that both players are making good moves (from their opposing perspectives). The search should stop when it reaches the specified search depth and then call the evaluation function.

You should experiment and find a search depth that will have the computer making a move within 5 approximately seconds (on your computer).

Extra credit

Write a player whose *settings* field in world-state takes a Number which represents the amount of time, in seconds, the computer has to make a move (there is no restriction on how long the human may take). Stop your search and return a reasonable move before this threshold is exceeded.

Note: the search you are performing is a depth-first search. As a result, to make a reasonable move you cannot simply perform your search as normal and then stop when the time limit is reached. For example, if the computer had an immediate winning move, but that move was not yet reached in the search, it would bypass the opportunity for an immediate win. You will need to read up on a technique called iterative deepening and implement it. The technique is relatively straightforward, and is used for programs that are working under a time limit but must return a reasonable result.

Hints

1. Conceptual complexity is lower than assignment #3, but there are some tricky details .
2. Use the generative recursion template and think about your "*trivial?*" cases that should terminate the recursion. Hint: there are two distinct cases under which the search should stop (and possibly a third if you are implementing the extra credit).
3. Add functionality incrementally and remember test cases. If you are sure your evaluation function is working correctly, it will be easier to add search functionality. Similarly, creating some test boards with clearly correct answers makes it easier to check your search functionality.