Artificial Intelligence	Assignment #3	Due: start of class, Fri. 3/20

This is an individual assignment. Do your own work. You may not discuss any aspect of the problem (design, approach, code) with anyone but the course instructor.

Problem: The game of Go Moku

Requirements: You will **create an F# game-playing program** that allows the user **to play** the game **Go Moku** against the computer. The attached sheet gives the rules of the game. Use the given TicTacToe implementation as a starting point, and modify this code to create your program.

It is impractical to do a complete minimax search all the way to a terminal state before making a move in this game. Therefore, you should modify the code to **search only to a predetermined depth (ply**). Read the user's choice of ply/difficulty from a control on the Windows form or a Console window when the Restart button is clicked (set it to 1 when the form is initially loaded into memory). Use the ply setting as a parameter that gets passed to all relevant functions and gets decremented whenever the player changes during a single search. When ply = 0, it's as if the terminal test were true and the current state must be evaluated without further search. This implies that your evaluation function must distinguish between terminal states and states where the game is not over. **Terminal states should return very large numbers (in absolute value) for a win**. For non-terminal states, make **a heuristic evaluation** of how good the current state is for max vs. min (a positive value means it's better for max, negative is better for min). This function should be as "smart" as possible. The large game board won't allow many levels of look-ahead, so the quality of the computer's **game play is highly dependent on the thoroughness of this evaluation function.**

If the computer uses **a 1-ply search**, you will probably be able to beat it, but it **should give a quick response and play a fairly smart game**. Specifically, the evaluation function should make the computer always choose a winning move, if available, and work towards a win if you aren't currently a threat. It should also play defensively if you are becoming a threat. In particular, it should definitely make a blocking move when you have 4 in a row or any 4 of 5 pieces that would give you a win, and should even try to block you when you have 3 of 5 pieces towards a win (not necessarily consecutive). None of these behaviors are coded explicitly; they should happen automatically using a minimax search and your heuristic evaluation function. The magnitude of the evaluation function's return value should reflect the relative advantage that player MAX has with respect to MIN. Use a linear combination (weighted sum) of various features of the game board.

Modify the Tic Tac Toe program as needed to deal with the larger game board and different graphics. Use a brown wood looking game board with a 19x19 grid of squares and a larger overall size than the tic tac toe game board. Instead of drawing characters (X & O), draw a solid circle (black or white) to represent a player's placement of a pebble on the game board.

Hints - There may be other approaches that will work, but here is one approach to implementing the heuristic evaluation function. It kind of assumes that a 1-ply search is being used, but my implementation still seems to work ok with a 2-ply search. First, for every possible sequence of 5 spaces on the board, determine counts of how many pieces each player has. If one player has 3 or more pieces while his opponent has none, he is becoming a threat in that area, and you should add an amount to that player's total. After checking the entire board, calculate a return value based on the totals calculated for each player. Bias the evaluation significantly so that player MAX has more weight. This is necessary for the 1-ply search since it will be used to evaluate states that result from a move by MIN. Getting to a state where both players have equivalent pieces after MIN makes its move (e.g., both have 4 in a row) does not mean both players have an equal chance of winning. MAX will have the next move when that state is reached and will easily win. A state like that should have a big positive evaluation (MAX has the advantage) so that MIN will try to avoid it.

Extra Credit (15 pts): A 2-ply search should play a better game and be much harder to beat, but may take too long to be practical unless you use alpha-beta pruning. Modify the functions used to carry out searches in your implementation of a depth-limited matrix game so that all functions used to calculate values and decisions use alpha-beta pruning in addition to a depth cutoff. These functions should stop generating child states whenever the alpha-beta conditions are met.