CSE 305 Spring 2011

Assignment #3: Game Trees and Minimax

Assigned: March 21

Due April 3 (11:59 pm)

The context for this assignment is a two-player game, such as chess, where one of the two players is the computer and the other is the user. When it is the computer's turn to make a move, it would construct a data structure known as a *gametree* in order to decide the best move to make. Each node of a gametree records some state of the game. The levels of a gametree alternate between states in which it is the computer's turn to make a move and those in which it is the user's turn. Each leaf node records the state of the game after a finite sequence of moves have been made. The branching factor of a gametree is not fixed, and could vary from one node to another – because the number of moves that one could make could vary from one state to another.

Assume that each leaf node can be mapped to a number indicating how "strong" the computer would be in that state. Then, the gametree is analyzed recursively as follows: At the root node, the computer chooses the next state (child node) that *maximizes* its strength; and, for each such next state, it assumes that the user would choose the next state that *minimizes* its strength. It carries out this analysis, recursively, for all levels of the gametree up to the leaf nodes. This is known as a 'minimax' algorithm.

Overall Objective: Write an ML function, **minimax: gametree → string list**, that returns the list of nodes in a game tree corresponding to a best sequence of moves that the computer could make in the state corresponding to the root of the tree, following the minimax algorithm sketched above.

Approach: Develop your ML program in three parts, but submit (online) one file "gametree.sml" containing all three parts.

1. Given datatype **piece** = king | queen of int | bishop of int | knight of int | rook of int | pawn of int;

For example, rook(2) represents two rooks; pawn(5) represents five pawns; etc.

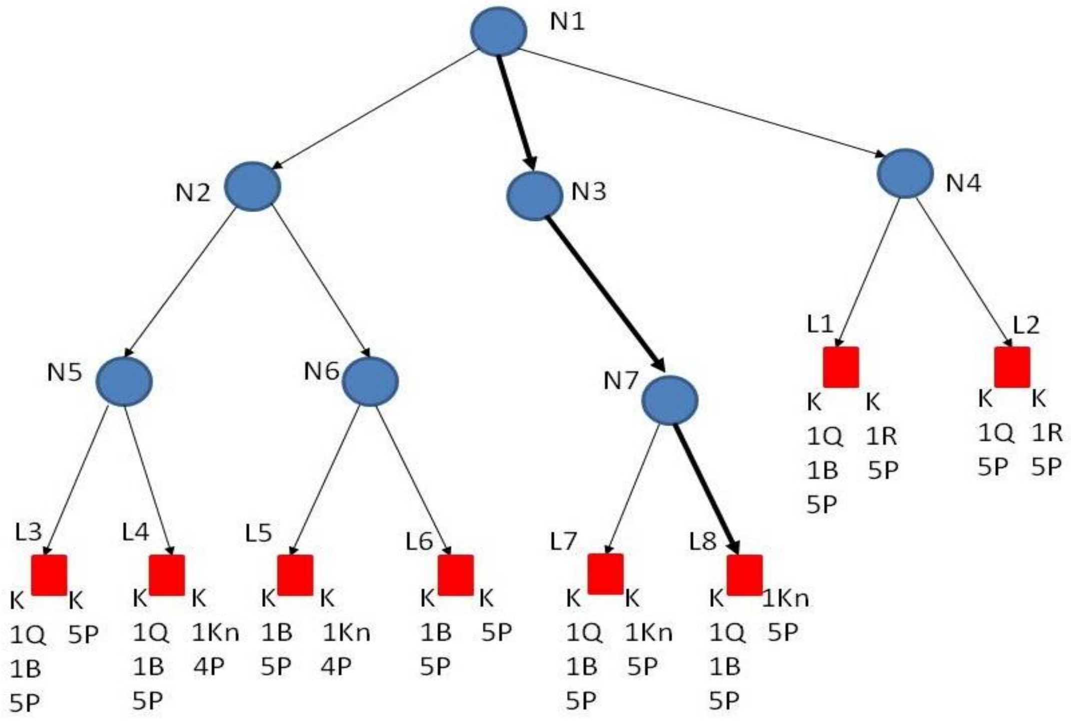
a. Write an ML function **netvalue**: piece list → int that takes a list of pieces and determines their net value, assuming that each piece has the following 'value': king = 100; queen = 50; bishop = 25; knight = 15; rook = 25; pawn = 3.

b. Write an ML function **strength**: piece list * piece list → int that computes the difference in the net values of two lists of pieces, the first being the computer's pieces and the second being the user's. (This is a very simplistic notion of state and strength; in practice, the state must take into account the board configuration, and the strength function would be more complex.)

2. Given datatype **gametree** = leaf of string * piece list * piece list
| node of string * gametree list;

Each node of the gametree has a label which is a string. Each leaf node also maintains the list of pieces for computer and the user, and each nonleaf node maintains a list of gametrees corresponding to the different possible next states. The datatype gametree is recursive since the subtrees of a gametree are also gametrees. Note: In this gametree, we are *not* recording the list of pieces for *nonleaf* nodes.

In the gametree shown below, listed under each *leaf node* are the pieces belonging to the computer (shown to the lower left) as well as the user (shown to the lower right). An abbreviated notation is used: K = king, Kn = knight, Q = queen, B = bishop, R = rook, and P = pawn.



Represent the above gametree in ML as follows: Create a value binding for each of the eight leaf nodes, and then use them to create value bindings for each of the seven non-leaf nodes. For example, the value bindings for the leaf node "L8" and the nonleaf node "N2" would be as follows.

val **leaf8** = leaf("L8", [king,queen(1),bishop(1),pawn(5)], [knight(1),pawn(5)]);
val **node2** = node("N2", [node5, node6]);

That is, introduce variables node1 … node7 for the nonleaf nodes and variables leaf1 … leaf8 for the leaf nodes, and perform 7+8 = 15 value bindings.

3. Write the function **minimax:** gametree → string list, incorporating the minimax algorithm. Suppose the above gametree is bound to a variable **node1**. Then, evaluating

minimax(node1);

would return the string list ["N1", "N3", "N7", "L8"], which corresponds to the sequence of moves highlighted by the thick arrows. At node L8, the computer's strength = 160, by our strength function.

Since minimax needs to alternate between phases of maximizing and minimizing, you may find it convenient to define minimax in terms of an auxiliary (nested) function

minimax2: bool * gametree → string list * int

which takes a boolean value indicating whether this is a maximizing phase or a minimizing phase. It would be convenient to return a list of node names as well as the strength (an integer) at each stage.

Further guidance on the assignment will be given in the recitations.

End of Assignment # 3