Names: Anuraj Dubey, Taksh Patel, Aneesh Reddi, James Preteroti

Basic Overview of Submitted Files

Firstly, our submitted folder includes a Tree.java file to create a tree containing all possible game states, a Node.java file that defines a node, or each state, and its respective values, a GUi.java file that holds the method to creating our initial gameboard (currently set to a 3x3 board) in a 2d array, and lastly WumpusMinimax.java which holds our main method and our actual algorithm for using minimax. In all honesty, we unfortunately were not able to get our game to run successfully, however please look at our code and the comments to see our implementation of Minimax. Apologies for this.

Overall Implementation of Minimax

So, our minimax algorithm first creates a tree from which we can check all possible states. Within our algorithm, we implemented a recursive base case for which Minimax recursively reaches the bottom-most nodes in the tree. If it does, we of course returned these leaf values so the recursion can progress. If we are on the maximizing player's (the agent's turn) in the tree, we recursively call Minimax on each child of the binary tree and set alpha to the max of the resultant value. We then check if we can prune this current part of the tree by checking if alpha is greater than or equal to the current beta value. If it is, we prune by calling Minimax and backtracking back to the original node and ignoring the pruned part of the tree by leaving the Minimax boolean parameter as true. Of course, each time we recursively call Minimax here (with the exception of pruning) we set the Minimax boolean value to false, in order to check the turn of the minimizing player's turn (the human player). The same exact logic applies to the human player's Minimax iteration, except we are simply concerned with the resulting min value rather than the max, and thus return that.

Proposed Metric of Evaluation (#2 in Deliverables)

Our proposed metric for evaluating the score of each player’s state is quite simple. We determine the score of each state depending on the number of remaining pieces they possess. Of course this is dependent on d. Thus, this value initially is (d/3) * 3 , as this is the number of pieces each player initially starts off with. So the bottom leaf nodes, must be +(d/3) * 3 and -(d/3) * 3 as these two states represent the extreme states in which the game would be over and a certain player wins.