Marco A Cortes **Andrew Envia**

Objective:

The goal of the this program is to create a perfect playing Tic-Tac-Toe AI implementing the Minimax Algorithm.

Problem Definition & Criteria:

The problem is that the player must challenge the AI to a Tic-Tac-Toe game and try to beat as best as they can. However, the main point of this program is for the AI to never lose to the player by utilizing the MiniMax algorithm. For the heuristic of the MiniMax algorithm, it traverses through every possible decision and then once it has gotten to a terminal node, it will return the heuristic value of that path.

The Criteria of whether this programs works is that the AI should never lose, however draws are acceptable for the AI. The AI should try to make moves that prevents it from losing as much as possible.

Implementation:

A majority of this program exists within the gameboard class, and the tttrules class. The former contains a multidimensional list which represents the tic tac toe board being playing on. Going further, it contains all functions used for any actions done on the board. The printboard() function takes no arguments and will print out the current tic tac toe board in the correct format. The clearboard() function empties out the board in case the user wants to play additional games.

The getchar() function takes in a coordinate argument and returns the whatever is currently occupying that slot in the tic tac toe board. Both setCharAI() and setChar() functions will take in two arguments, the first being the player that is attempting to make the move, the second being the coordinate for that move. What differentiates the two is that setChar() has restrictions where it can't place a move if it's being occupied, this one is used by the human player. The setChar() function has no such restrictions, as it's used by the algorithm to test different game possibilities.

The rrrtules class contains four counters for checking who's turn it is, and counting player wins and/or draws. The turn2() function is what invokes the minimax implementation. It will first call the returnbestmove() function with the current tic tac toe board as it's argument and will return the move to be played. What the returnbestmove() does is it calls the possiblemoves() function, which takes a tic tac toe board as an argument and returns a list with the coordinates of all slots that are empty. Then, for every coordinate in that list, it will call the setCharAI() function with both that coordinate and the char that indicates player two ('x' indicates a move by player 1, 'o' a move by player two.) After playing that move, the minimax() function will be called with the tic tac toe board that was previously modified and string 'x' (indicating the opposing player) as arguments. It will return an integer, which will be assigned to variable 'moveScore' after which the move that was previously added to the tic tac toe board will be cleared using the setCharAI() function. It then checks if the moveScore value it received is greater than the previous best score, if so it will set variable bestMove to the current coordinate being checked and variable bestScore to the score the minimax function returned when given the current coordinates. After iterating through every move and finding the one with the highest score, it will return the coordinates for the move with the highest score.

The minimax() function does everything the getbestmove() function does, with the addition of a couple of things. Along with taking in a gameboard object as an argument it also takes in currentPlayer arguments, which represents whether it is the maximizing or minimizing player's turn. This is represented by either 'x' or 'o.' Additionally the minimax() function will also check for a terminal state and return an integer based on whether player1 or player2 is winning or if there's a draw. Assuming it isn't a terminal state, it will then go to one of two parts of code which depends on whether it's currently player1 or player2's turn (the value of the currentPlayer variable brought in as an argument.) It will set a variable bestscore to -1000 if currentPlayer is 'o' (maximizing player) or set it to 1000 if 'x' (minimizing player.) It will then iterate through every possible move, playing that move, then calling minimax() recursively but with the opposing player as the argument this time. In the end it will return the best possible score if it's the maximizing player, or the worst possible score if it's the minimizer.

The reason for returnbestmove() being a seperate function than minimax() while being functionally very similar is that minimax() only has to return the scores for the assigned moves. The returnbestmove() function will have to return the coordinate for the move with the highest score.