This assignment doesn't requires coding in Javascript. But you can use the template provided

Ecoworld Testing.7z to get started easily.

1. Game (tic-tac-toe) playing.

a. Implement terminal check and utility functions for tic-tac-toe. See the is_terminal and utility functions in tictactoe.js for more detailed instructions. (2 pts)

b. Create a function to play tic-tac-toe, using the Minimax algorithm. Modify the template function tictactoe_minimax, provided in tictactoe.js, to accomplish this. (Note: You have already been provided a working recursive depth-first search implementation; you need only add Minimax to it.) (3 pts)

c. Use the provided tictactoe.htm file to play a few games against your code from (1b). Does it play the way you expect? Can you beat it? Also, notice that the evaluated/expanded state counts are reported while you are playing a game. Record these values as you play. How do they change over the course of the game? Is this what you expected? Discuss. (1 pt)

d. Modify your algorithm to use alpha-beta pruning. (Copy your code from (1b) into the provided tictactoe_minimax_alphabeta function. Modify this.) (3 pts)

e. Repeat (1c) using your new function. (1pt)

f. Configure the game so the human player goes second (and make sure the initial board is empty). Set for standard MiniMax and start the game. Report how many nodes are expanded to calculate the first move. Repeat for MiniMax w/ Alpha-Beta Pruning. How do the two algorithms compare? Are the results what you expected? Discuss.

Now, look again in tictactoe.js and notice the variable move_expand_order, which controls the order in which different possible moves are examined. An alternate ordering has been provided but commented out. Uncomment this and repeat the above experiment. How do the results change? Try to explain why you get this result. (1pt)

2. Use truth tables to show whether or not the following sentences are valid (make sure to state whether or not it is valid!). (2 pts each)

$[P \wedge Q] \Leftrightarrow [\neg Q \Rightarrow P]$

$[P \Rightarrow (Q \vee R)] \Leftrightarrow [\neg P \vee Q \vee R]$

Create a report including answers to the asked questions and a printout of your code.

Tips: If you need to print out debug statements, you may use the console.log() function to print out to the browser's debug console. To access this log, use CtrlShift-J in Chrome or F12 in Firefox. Alternately, you can use the helper_log_write() function (from tictactoe_helper.js) to output to the log region on the web page. A debug function (and a button to run it) has been provided in tictactoe.js to help you if you need to run any code for testing purposes.

Extra Credit: Try improving on the ordering from (1f). Can you explain why your result works better? 0.5pts: Any improvement over the results from (1f). 1pt: Expands fewer than 50,000 nodes and can explain why the changes work. 2pts: Expands fewer nodes than any other student submission (and meets the requirements for getting 1pt) 2pts: Alternately, find the worst possible ordering (must expand more than 115,000 nodes). (You may try for both 2pt options, but 2pts total is the max extra credit you may receive. Also, note that the node counts above assume Alpha > Beta.)