TIC TAC TOE Q&A

1. What are the “node” in our tic-tac-toe game? What are the children of “node” then?

   The nodes are the present of board condition with possible moves ahead. The number of generation of children represents number of move of look ahead. The alternative of generation represents the player and computer’s move.

2. What is the “terminal node” in our tic-tac-toe game?

   Terminal node is theoretically the end of a game, resulting in a win, lose, or draw, but for the sack of running time, terminal node is the depth of lookahead.

1. How to make sure your AI always prevent the opponent from winning in the next move if your AI still have a chance to do so by correct values for nodes?

   To prevent opponent from winning we need to implement the correct evaluation function where it will set the move that lead to a win positive score and an opponent’s win a negative score. Also, we give a higher positive score for more AI’s symbol in line and more negative score for more opponent’s mark in line. By running minimax algorithm we will only use the maximum score of us and minimum score of opponents that we can find in the tree. Thus, this algorithm make sure AI always choose correct value for nodes.

2. If there is a move for your AI to win in the next move, is your AI able to find out the move also by correct values for nodes?

   Yes, because evaluate function will assign largest score 1000 for definite winning move. Also, this node will be a terminal node, so it definitely be chosen to be the next move unless there is an alternative move that also result a definite win.

Method and Implementation (including description of minimax and heuristic rule):

1. We consider number of depth, which is number look ahead, and connect of each look ahead as levels of tree. The possible moves will take current condition of board into consideration and will not place marking on already occupied cell.
2. For each lookahead possibility we give it an evaluation score.

3. <u>(heuristic rule)</u> The evaluation score will be calculated where the moves that lead to the winning of AI will have a higher positive score and the winning of opponent will lead to a lower negative score.
    a. The evaluation will use a precomputed winning circumstances that stored in several loops in total of 76 possibilities.
    b. The line that has different markings, of both AI and opponent, will have an automatic score of 0. The evaluation will count score with following rules:
        i. +1000 for each 4 in a line for AI
        ii. +100 for each 3 in a line for AI with 1 empty block
        iii. +10 for each 2 in a line for AI with 2 empty block
        iv. +1 for each 1 in a line for AI with 3 empty block (all scores will not be repeatedly counted)
        v. Negative score for all x in line for opponent
    c. There are 2 functions for evaluate the board: evaluate and evaluatList
        i. Evaluate lists all the possibilities of winning
        ii. evaluateList calculate score based on whether you or opponent will have a win
4. <u>(minimax)</u> Then, we used the tree with evaluated scored stored in and run it with Minimax, alpha-beta pruning where for each tree we created, we use the algorithm to find the best move for the player. (The algorithm used is from Wikipedia, derived from pseudo code)
    a. Alpha-beta pruning: In the minimax algorithm, we created a variable alpha to hold the best max value and beta that holds the best min value, which is initialized to be positive and negative infinite respectively. we find the best next move by number of lookahead, and terminate the branch that reaches condition alpha >= beta where it is impossible to win.
    b. We created 2 function, getMin and getMax. getMin finds minimum of all its child node values and getMax chooses the maximum node values of its child nodes in addition to alpha beta pruning algorithm.
    c. We created 3 global variables called bestMove1 2 3 was calculated by minmax function using getmin and getmax

Experiment and results:

	Win (out of 100 test)	Lose	Draw	Efficiency (time used)
Random vs AI	100	0	0	~0 millisec for depth 7

AI vs AI with similar setting (with depth 2)	42	58	0	~0 millisec for depth 7
AI vs AI with different depth (depth 2 vs depth 7)	61 win for depth 7	39 lose for depth 7	0	<0 sec for depth 2 ~0 sec for depth 7 Avg 0 sec in total

The efficiency of the system: For random vs AI, and 2 AI vs AI game, it uses almost no time to run.

Description of experiment design: we created 3 data sets with different setting. We used data set random vs AI with depth 7 for AI, AI vs AI in similar setting with depth 2 on both AI, and AI vs AI with different depth 2 and 7. We experiment with data size with 100 games per set with number of win, lose and draw.

Discussion:

Sometimes when we use our AI against itself with exactly same setting, it doesn't tend to move with strategy that benefits itself the most and minimize its opponent's winning route but rather stays in its own route. In another word, it cannot predict what is the opponent move and make a move that correspond to predicted move. Depth used in the algorithm doesn't need to be set too high, a overly high depth value will cause the AI to overthink situation and made bad decision, and thus make bad choices compare to medium value depth.

Conclusion:

For the algorithm used in this programming assignment, we used heuristic evaluation system to count the score in the game and we used alpha beta pruning to calculate the best move for AI using different depth. We log the number of markings (all the winning conditions) per row/column/diagonal in an aggregated precomputed arrays and later iterate over them. In experiment we found that as depth increase, the chance of AI with a higher depth have a significantly larger winning chance, but the cost is increasing large running time to compensate for the larger depth. However, there are still some confusion for AI when it is running against itself in similar setting by not accurately predicting opponent's move and make the best move for itself.