ARTIFICIAL INTELLIGENCE

Documentation

Question 1

Changes to facilitate alpha-beta pruning in the original code:

1. **Function Parameters:** Added alpha and beta as parameters to the minimax function to keep track of the lower and upper bounds of possible scores for the maximizing player.

```
def minimax(board, depth, isMax, alpha, beta) :
```

2. **Alpha Update (Maximizer):** Inside the maximizer's branch of the minimax function, updated the alpha value by taking the maximum of the current alpha and the best value.

```
if (isMax) :
    best = -1000
    for i in range(3) :
        for j in range(3) :
            if (board[i][j]=='_') :
                board[i][j] = player
                best = max( best, minimax(board, depth + 1, not isMax, alpha, beta)
                alpha = max(alpha, best)  # Update alpha
                board[i][j] = '_'
                if alpha >= beta:  # Pruning condition
                    break
    return best
```

3. **Beta Update (Minimizer):** Inside the minimizer's branch of the minimax function, updated the beta value by taking the minimum of the current beta and the best value.

```
else :
    best = 1000
    for i in range(3) :
        for j in range(3) :
            if (board[i][j] == '_') :
                board[i][j] = opponent
                best = min(best, minimax(board, depth + 1, not isMax, alpha, beta))
                beta = min(beta, best)  # Update beta
                board[i][j] = '_'
                if beta <= alpha:  # Pruning condition
                    break
    return best
```

4. **Pruning Condition (Maximizer):** Added a condition to break the loop early if beta <= alpha (pruning condition) inside the maximizer's branch of the minimax function.
5. **Pruning Condition (Minimizer):** Added a condition to break the loop early if beta <= alpha (pruning condition) inside the minimizer's branch of the minimax function.

Question 2

Perform the following evaluations, using the *code in the previous question:*

1) Calculate the average number of nodes that would be evaluated using the minimax algorithm without pruning and compare it to the number of nodes evaluated with alpha-beta pruning.

	Without Alpha-Beta Pruning				With Alpha-Beta Pruning			
	Run1	Run2	Run3	Average	Run1	Run2	Run3	Average
No.of nodes evaluated	67,495	72,125	73,073	70,897	21,305	22,741	19,323	21,123

2) Implement a parallel version (2 threads) of alpha-beta pruning and compare its performance with sequential version.

	Parallelized Alpha-Beta Pruning				Serial Alpha-Beta Pruning			
No.of	Run1	Run2	Run3	Average	Run1	Run2	Run3	Average

nodes evaluated	8560	7167	5937	7221	24495	19323	17670	20496
Time Taken	0.058 45403 67126 4648	0.042 49501 22833 2519	0.0525 510311 12670	0.051166 6933695 47	0.1128 20625 30517 578	0.0932 879447 937011 7	0.1022 45569 22912 598	0.102784713 10933

3) Develop a heuristic for non-terminal states that would assist in reducing the search space (enhanced alpha-beta pruning) and compare it to the standard alpha-beta pruning algorithm.

Hint: *During the search process, when evaluating non-terminal states, use the heuristic to estimate the desirability of each state. Instead of exploring all possible moves to determine the exact value of a state, use the heuristic to prioritize which branches of the search tree are more promising. This involves comparing heuristic values and using them to guide the selection of moves and pruning of branches.*

	Alpha-Beta Pruning				Enhanced Alpha-Beta Pruning			
No.of nodes evaluated	Run1	Run2	Run3	Average	Run1	Run2	Run3	Average
	22766	21305	16321	20131	7331	8996	6493	7606

4) Develop a tic-tac-toe game, where the AI is a "weak" player. Modify the game in the following manner:

We'll use a human interaction approach for this one with AI being 'o' & our player as 'x'. Since we are relying on randomness, we don't need to make a game tree. We'll just randomly pick an empty node, given it is not forbidden node.