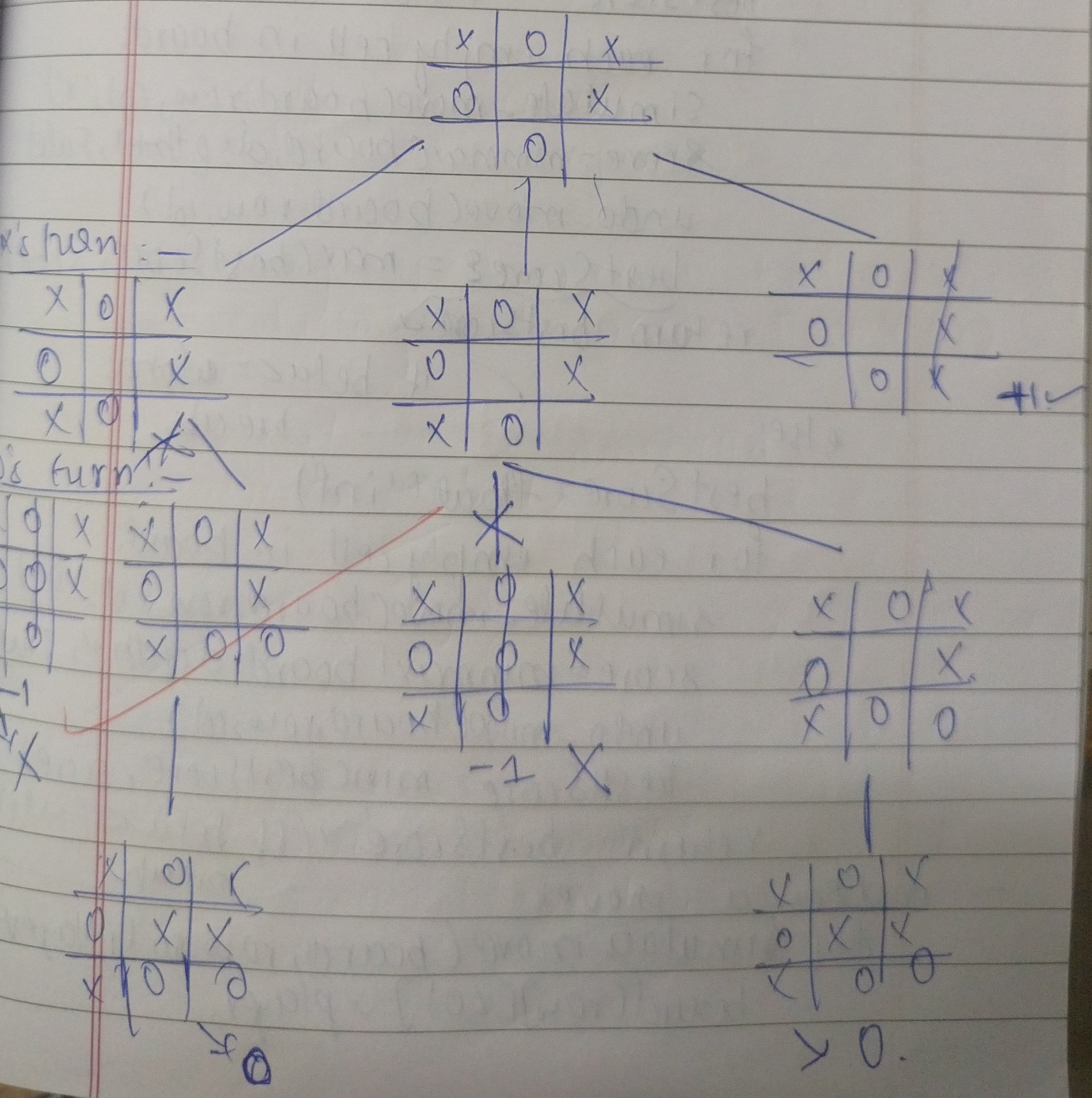
LAB 10:-

=> Solving ~~minimax~~ Tic-Tac-Toe game using minimax ~~and~~ with alpha beta pruning.

In this pruning happens when we can confidently say that a given path will not affect the final result (because the opponent will avoid it)

Example:-



Algorithm

Algo for minmax:-

```
def minmax(board, depth, isMaximizingPlayer):
    result = check_game_over(board)
    if result != 0:
        return result

    if isMaximizingPlayer:
        bestScore = -float('inf')
        for each empty cell in board:
            simulate_move(board, row, col, 'X')
            score = minmax(board, depth+1, False)
            undo_move(board, row, col)
            bestScore = max(bestScore, score)
        return bestScore
            if beta <= alpha:
                break
    else:
        bestScore = float('inf')
        for each empty cell in board:
            simulate_move(board, row, col, 'O')
            score = minmax(board, depth+1, True)
            undo_move(board, row, col)
            bestScore = min(bestScore, score)
        return bestScore
            if beta <= alpha
                break
```

-> make a move:

```
def simulate_move(board, row, col, player):
    board[row][col] = player
```

```
def undo_move(board, row, col):
    board[row][col] = '_'
```