Write up:

With the implementation of Expectimax in my connect 4, I found that it didn't change too much how often my AI would win. Even with the randomness that came from an equal probability of choosing any column, my expectimax ai was still able to win every time that I played it against the random opponent. There were some times when I felt like the random player had a better score on the board, like had 3 in a row or other similar layouts, however it never actually ended up winning against the ai. If I were to play my expectimax ai vs my minimax agent, I think that I would win a lot more consistently against my expectimax agent (given the fact that it would think I would be picking randomly, it is not well suited for strategic moves like the alpha beta search is). This lab was really interesting, and showed me just how much chance can alter the effect of any game, as well as the ai that is trying to beat it.

Code:

```
def get_expectimax_move(self, board):
    """
    Given the current state of the board, return the next move based on
    the expectimax algorithm.

    This will play against the random player, who chooses any valid move
    with equal probability

    INPUTS:
    board - a numpy array containing the state of the board using the
            following encoding:
            - the board maintains its same two dimensions
                - row 0 is the top of the board and so is
                  the last row filled
            - spaces that are unoccupied are marked as 0
            - spaces that are occupied by player 1 have a 1 in them
            - spaces that are occupied by player 2 have a 2 in them

    RETURNS:
    The 0 based index of the column that represents the next move
    """
    valid_moves = get_valid_moves(board)

    player = self.player_number
    opponent = 2

    if player == 1:
        opponent = 2
    elif player == 2:
        opponent = 1

    best_move = 0
    max_value = -math.inf
    depth = 4

    for action in valid_moves:
        new_board = np.copy(board)
        make_move(new_board, action, player)
        print(new_board)
        if is_winning_state(new_board, player):
            return action
        result = self.expectimax_value(new_board, depth, player, opponent)
        print(result)
        if result > max_value:
            max_value = result
            best_move = action

    return best_move
    # raise NotImplementedError('Whoops I don\'t know what to do')
```

```python
def expectimax_max(self, board, depth, player, opponent):
    valid_moves = get_valid_moves(board)

    if is_winning_state(board, player):
        return 1000000000
    elif depth == 0:
        return self.evaluation_function(board)

    v = -math.inf
    for action in valid_moves:
        new_board = np.copy(board)
        make_move(new_board, action, player)
        v = max(v, self.expectimax_value(new_board, depth - 1, opponent, player))
    return v

def expectimax_value(self, board, depth, player, opponent):
    valid_moves = get_valid_moves(board)
    probability = len(valid_moves)

    if is_winning_state(board, opponent):
        return -1000000000
    elif depth == 0:
        return self.evaluation_function(board)

    v = 0
    for action in valid_moves:
        new_board = np.copy(board)
        make_move(new_board, action, opponent)
        result = self.expectimax_max(new_board, depth - 1, opponent, player)
        v += result/probability
    return v
```