MinMax with Alpha Beta Pruning

Assignment: With regular minmax, add a static int counter to the class that counts every move that is **searched**. Increment it in **makeMove** and never reset.. After main plays 100 games, print the total number of moves. Run a few times to make sure it's consistent. Then record the number of moves for search depth = 9,8,7,6,5 games in a spreadsheet.

Next implement alpha-beta pruning (in the same code is fine). Perform the same counts for 9,8,7,6,5. Make a graph of **moves-in-minimax (X) vs moves-in-alpha-beta (Y)** and determine -- what is the mathematical relationship between the two? E.g. does alpha beta cut the number in half? In third? Smaller? Is alpha-beta a non-linear function of mini-max?

Write a brief analysis with a graph and a trendline justifying your conclusion. Submit both the analysis pdf and AlphaBeta code to javadrop please.

Here is a standard minmax Algorithm

```
MINIMAX(board, depth, player):
  if game is over OR depth is 0:
    return score of board  (+1 = X win, -1 = O win, 0 = draw)

  if player is X (maximizer):
    best = -infinity
    for each empty space:
      make the move
      best = max(best, MINIMAX(new board, depth-1, O))
    return best

  if player is O (minimizer):
    best = +infinity
    for each empty space:
      make the move
      best = min(best, MINIMAX(new board, depth-1, X))
    return best


GET_MOVE(board, player):
  for each empty space:
    make the move
    score = MINIMAX(new board, depth, other player)
    keep track of the move with the best score
  return best move
```

Here is minimax with alpha-beta pruning

```
MINIMAX(board, depth, player, alpha, beta):
  if game is over OR depth is 0:
    return score of board  (+1 = X win, -1 = O win, 0 = draw)

  if player is X (maximizer):
    best = -infinity
    for each empty space:
      make the move
      best = max(best, MINIMAX(new board, depth-1, O, alpha, beta))
      alpha = max( alpha, best )
      if (alpha >= beta)
        break
    return best

  if player is O (minimizer):
    best = +infinity
    for each empty space:
      make the move
      best = min(best, MINIMAX(new board, depth-1, X, alpha, beta))
      beta = min( beta, best )
      if (alpha >= beta)
         break
    return best


GET_MOVE(board, player):
  for each empty space:
    make the move
    score = MINIMAX(new board, depth, other player, -infty, +infty)
    keep track of the move with the best score
  return best move
```

The idea is that alpha = the lowest score that MAX is guaranteed and beta = the highest score that MIN is guaranteed. Whenever they cross, you can stop searching.