Pattern matching is a kind of solitaire game

Scoring Algorithm Idea:
Arguments: Player (X or Y), board (with pieces placed, a game state.), level
Return value: The score for the board relative to X.
(1) Test of the game is over. If so return the score.
(2) Find (Kumar) OR loop through (sdc) all legal moves.
(3) Pick a "best" or somehow otherwise guessed legal move.
(4) If this is for the real game, make that move on the board(changes it!)
(5) Return the score for the board with the picked move made.
IRAT: Finish code to find the minimum score for Y.
TRAT: Pick the best move.
Clarification:
Group: Count the number of possible moves:
Individual: pick the k'th move.

Notes:
Declare local ints fRow, fCol for final Row, final Column
Make code that leaves fRow, fCol holding the best or
guessed move for the player, AT THE END OF THE
nested loops.

LAST THING:
if( level==0) {board.move(player, fRow, fCol)};
static int nMoves(Board b)
{
    int retVal = 0;
    int row = 0;
    while( ... )
    {
        int col = 0;
        while( ... )
        {
            if( .... ) { retVal++;}
            col++;
        }
        row++;
    }
    return retVal;
}

Individual problem: SUBMIT AS CODE IN YOUR PROJECT
static void doKthMove(char player, int k, Board b)
{ //Algorithm ideas:
    //(1) Nested loops as above to count open spaces.
    //(2) Count up to k.
    //(3) When the k-th open space is found,
    //    call move on b like b.move(player, row, col);
    //(4) return immediately after the move is made.
}

Finishing part 1 (put this code in computerMove()):
(1) Call int nM = nMoves(b);
(2) Calculate a random number K in the range
    1, 2, 3, ..., nM
(3) Call doKthMove(player, K, b)

How to generate a random number in range 1, .., nM:
    (int) (Math.random() * nM + 1.0)