```
using System;
using System.Collections.Generic;
using System.Text;

namespace NQueens
{
    class BlindSearch
    {
        //Private vars
        Stack<ChessBoard> stack;
        int gridSize;
        int moveCounter = 0;
        int failedBoards = 0;
        ChessBoard solution;

        /// <summary>
        /// Creats variables and calls main function
        /// </summary>
        /// <param name="size">size of board to make</param>
        public BlindSearch(int size)
        {
            Printer printer = new Printer();
            gridSize = size;
            stack = new Stack<ChessBoard>();
            stack.Push(new ChessBoard(size));

            solution = DepthSearch();
            if (solution != null)
            {
                printer.Print(solution.board);
                Console.WriteLine(String.Format("Solution found! \nTotal moves: {0}\nDead Ends: {1}", moveCounter, failedBoards));
            }
            else
            {
                Console.WriteLine("No solution found");
            }
        }

        /// <summary>
        /// Main function.
        /// Attempts to find solution by pushing valid moves onto the stack at every stage
        /// Exausts stack until solution is found
        /// </summary>
        /// <returns>solution if found</returns>
        ChessBoard DepthSearch()
        {
            ChessBoard currentBoard;
            while (stack.TryPop(out currentBoard)){
                //Check if done
                if(currentBoard.GetNumQueens() == gridSize)
                {
                   return currentBoard;
                }
                else
                {
                    int stackCount = stack.Count;
                    PushValidMoves(currentBoard);

                    if (stack.Count == stackCount) //Current board did not add to stack. Deadend hit
                    {
                        failedBoards++;
                    }
                }

                moveCounter++;
            }

            return null;
        }

        /// <summary>
        /// Pushes the valid moves for the next cycle onto the stack
        /// </summary>
        /// <param name="currentBoard">board on which to run the push</param>
        void PushValidMoves(ChessBoard currentBoard)
        {
            int currentRow = currentBoard.GetNumQueens();
            Square[] row = currentBoard.board[currentRow];
            for (int i = gridSize - 1; i >= 0; i--)
            {
                if(row[i].GetNumHits() == 0)
                {
                    ChessBoard tmpBoard = currentBoard.Clone();
                    tmpBoard.AddQueen( new Coord(currentRow, i));
                    stack.Push(tmpBoard);
                }
            }
        }


        public ChessBoard GetSolution() { return solution; }
    }
}

```