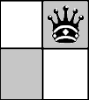
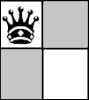
NQueens Backtracking and Output Analysis

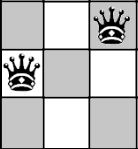
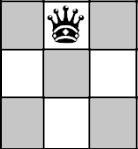
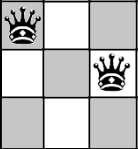
Example N-Values Without Solutions

For n = 2, nQueens reaches two unsolvable dead ends immediately after the first step:



Placing the queen at (1, 1) or (1, 2) doesn’t allow for queens to be placed in row 2. The available spaces are all in the same column or diagonal to a queen in either case.

When n = 3, there are three dead ends:



The first and last queen configurations blockade row 3 for similar reasons. The queen in row 1 attacks a tile in row 3 by column, and the queen in row 2 attacks the other two tiles in row 3 by another column and diagonally. The queen in the second configuration at (1, 2) covers all three spaces in row 2, one in its column and two diagonal spaces. Because all three tiles in row 1 lead to dead ends, NQueens has no paths left to search for solutions.

Recursion in NQueens

The NQueens problem is suitable for recursion because it involves solving smaller instances of a larger problem. Finding a solution to NQueens for the entire chessboard is less complex when it is split up into individual solutions for each row. In the program, the playQueens method uses recursion to handle each row individually and eventually build up the set of valid tiles for each NQueens solution.

Efficiency of Expand Compared to Checknode

Expand only places queens that are promising and skips over any others. A queen is placed at the next possible column index (i + 1) before being checked with the promising method. Checknode is less efficient because it places a queen first (at index i), then checks whether it is promising (moving on to index i + 1). There is less activity on the stack in expand. Pushing and popping occurs more often in checknode.