Group 26 - BME 122 - 2023
Assignment 2
Wallace Lee, Aric Quan, John Matti, Aaron Jumarang

bool isSafe: Checks to see if the position of a possible queen conflicts with another queen. It goes through the stack of queens to check if the possible queen is safe from other queens.

Input: it takes in the x and y position of a possible queen, the size of the board N, the current stack of queens

1. Make a copy of the stack of queens from previous rows so that it does not affect the main stack
2. Copies the queen in the top of the stack.
3. Checks to see if the possible queen is in the same column as the top stack queen, i.e. the same x value (preemptively ends and returns false)
4. If queen does not meet Step 3, then it checks to see if it's in the same diagonal towards the top left of the board of the top queen (if so, function returns false). Stops when iterators go out of bounds, meaning no queen was found.
5. If the no queen is detected in Step 4, it checks to see if it's in the same diagonal towards the top right of the board of the top queen (if so, function returns false). Stops when iterators go out of bounds, meaning no queen was found.
6. If no queen was found, pop the queen at the top of the stack.
7. Repeat steps 2 through 6 until the stack is empty.
8. Once Empty, this position of the possible queens works, and x and y position of a possible queen does not conflict with other queens previously placed

Output: true if it goes through the entirety of the function and the possible queen is safe, otherwise false in the conditions listed in the steps (see above)

bool solve: Places a queen in a row and recursively calls the function to place another queen in the next row until a queen placement is invalid or has filled every row.

Input: the current row number, the size of the board N, and the current stack of Queens.

1. The base case checks if the current row is the number N, which means that all queens have been placed successfully on each row and returns true (essentially ending the function as a valid path has been found).
2. If the current row is less than N, then it places a queen at the first column of the current row.
3. Checks if the queen at the current position (current row and column) is safe from attack from previous queens (i.e call the **isSafe** function)
4. If the queen is not safe, it will place the queen to the next column and repeat Step 3.
5. If the queen is safe, then it will push the queen to the main queen stack.
6. Recursively call the function for the next row, repeating Step 1.
7. If a valid path is true (i.e the recursive calls found a valid pathway where queens can be placed in each row and reached the base case in Step 1), it will return true.
8. If no valid path is found (i.e no valid queens can be placed in the next rows), it pops the queen from the stack and places the queen to the next column and repeats Step 3.
9. If each column has been checked, and no queen can be placed in the current row, it will return false and backtrack to the previous row, telling the previous function call that a queen cannot be placed in this current row (goes to Step 8 for the previous recursive call).

Output: true if all queens can be safely placed in a NxN board, false if a queen cannot be safely placed in the current row.

void printBoard: prints out board based and places queens that are in the stack

Input: size of the board N, and the stacks holding the queen positions

1. Create a 2D array of size NxN with N columns and N rows
2. Goes through each element in the 2D array and set them as 0
2. Go through each queen in the stack and sets the element at the the top queen's x and y position to 1 in the 2D array, signifying that a queen is present in that position
3. Prints lines of board
4. Goes through each element in the 2D array, print open space for each place on chess board that does not have queen (i.e. board has value 0), else, prints letter Q for queen
5. Goes through steps 3 through 4 until for loop reaches N
6. Print bottom line of chess board