CS007A: Queens Problem

A classic family of chess problems is that of finding the maximum number of pieces, of a given type, that can be placed on a board of given size, so no piece attacks another, and enumerating all possible solutions.

The most analyzed and generalized of all non-attacking tasks involves queens. On the standard 82 chessboard there are exactly 12 different ways (rotations and reflections are never considered different) of placing eight queens, obviously the maximum number, so no queen attacks another. These are shown below (From Chess Queens and Maximum Unattacked by MARTIN GARDNER, Math Horizons, Vol. 7, No. 2 (November 1999), pp. 12-16.)

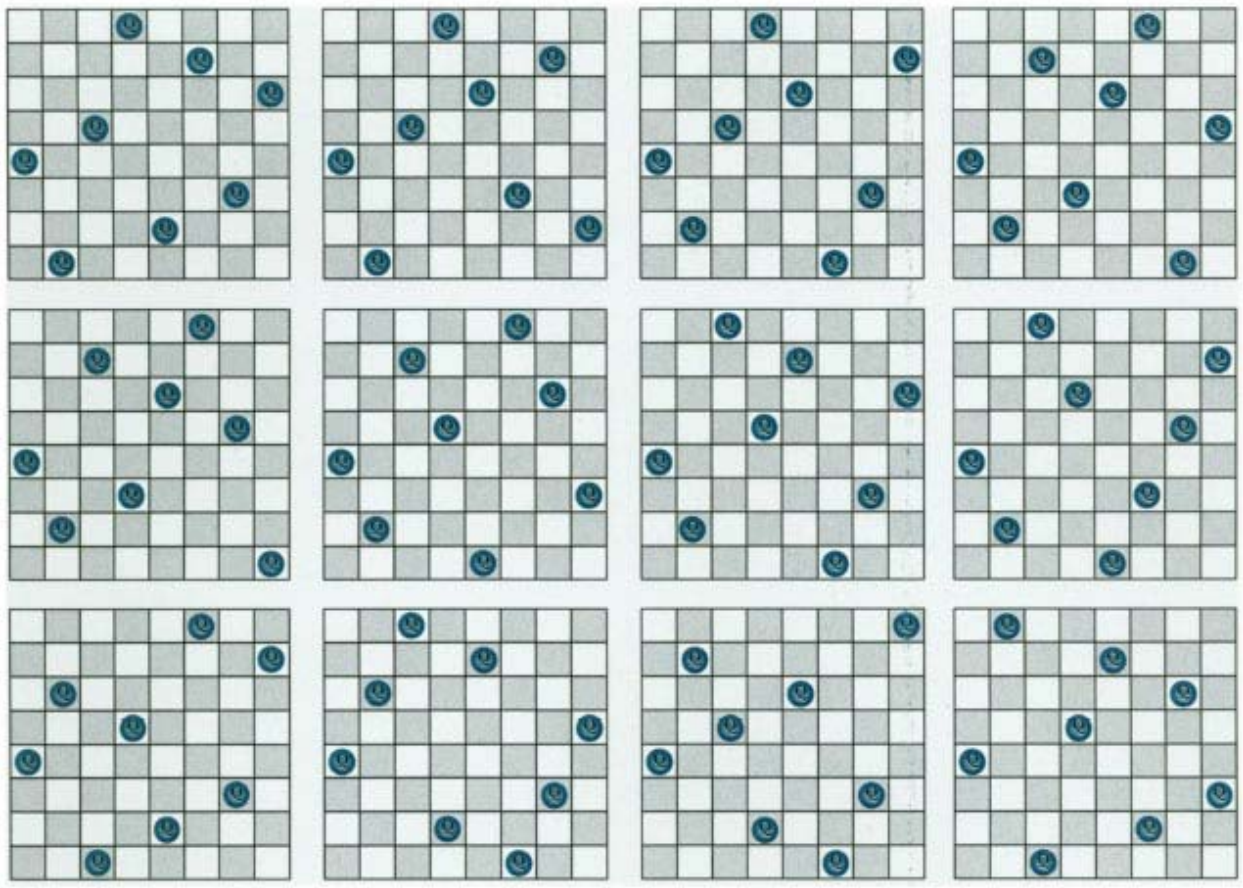

Figure 1. *The twelve solutions to the nonattacking queens.*

In lab on Tuesday 4/10/12 we developed fleshed out the pseudo code below to implement the positioning of queens on a chessboard and the facilitate the user's ability to position them and keep track of the queens' influence throughout the board.

```
initialize board and initial position
while (it's possible to move)
   display board
   get move
   update board
```

The variables needed include the state of the board (a two-dimensional array, only part of which is used – giving the user flexibility as to how big a board they want, up to 100 by 100), the user-supplied dimensions of the board (`dim`), the number of queens placed so far (`qCounter`), the position of the last queen placed (`qpos` – a number between 0 and `dim`^{2} – 1, computed as **`row*dim+col`** for a queen in `row` and `col`), and `stck`, a boolean returned by the function `stuck()`, which is true if there are no more free squares to put the queen.

Here's some code we came up with to make this work during lab on 4/10/12:

```
//Geoff Hagopian  Queens

#include <iostream>
#include <iomanip>
using namespace std;

const int SIZE = 100;

int initBoard(void);
int placeQueen(char board [][SIZE], int);
void updateBoard(char board [][SIZE], int, int, int);
bool stuck(char board [][SIZE], int);
void displayBoard(char board [][SIZE], int);

int main() {
      int qCounter = 1;
      char board[SIZE][SIZE] = {0};
      int dim = initBoard();
      int qpos = placeQueen(board, dim);
      updateBoard(board, dim, qpos, qCounter);
      bool stck = false;
      while(!stck)      {
            ++qCounter;
            displayBoard(board,dim);
            qpos = placeQueen(board, dim);
            updateBoard(board, dim, qpos, qCounter);
            stck = stuck(board, dim);
      }
      return 0;
}

int initBoard(void) {
      int dim;
      cout << "\nWhat is the width of your square chess board? ";
      cin >> dim;
      return dim;
}

int placeQueen(char b[][SIZE], int dim)
{
      int row = dim, col = dim;
      cout << "\nIndicate row and column where the queen is to go: \n";
      cin >> row >> col;
      while(row >= dim || row < 0 || col >= dim || col < 0)
      {
            cout << "\nThat is not a legal position.\n";
            cin >> row >> col;
      }
      return row*dim+col;
```

```
}

void updateBoard(char b[][SIZE], int dim, int qpos, int qCounter)
{
	int row = qpos/dim;
	int col = qpos%dim;
	b[row][col] = 'Q';
	for(int i = 0; i < dim; ++i)
	{
		if(b[row][i]==0)
			b[row][i] = qCounter;
		if(b[i][col]==0)
			b[i][col] = qCounter;
		if(b[(row+i)%dim][(col+i)%dim]==0)
			b[(row+i)%dim][(col+i)%dim] = qCounter;
	}
}

bool stuck(char board [][SIZE], int dim)
{
	for(int i = 0; i < dim; ++i)
		for(int j = 0; j < dim; ++j)
			if(board [i][j] == 0) return false;
	return true;
}

void displayBoard(char board [][SIZE], int dim)
{
	cout << endl;
	for(int i = 0; i < dim; ++i)
	{
		for(int j = 0; j < dim; ++j)
			if(board[i][j]=='Q') cout << setw(1+dim/10) << 'Q';
			else cout << setw(1+dim/10) << (int)board[i][j];
		cout << endl;
	}
}
```

Observe that we have three functions corresponding roughly to our pseudocode. (1) `initBoard()` allows the user to choose a board size. **`placeQueen()`** prompts the user to place the queen and checks that the position is on the board, but not whether it is in harm's way. **`updateBoard()`** attempts to fill in the rows, columns and diagonals attacked by the new queen, but doesn't yet work quite right, and `stuck()` determines if there are still places to put a queen or not.

Here's some out put from the play of the game (below). Note that, on the first move, things go pretty well, but only one of the two diagonals is filled in. Then on the second move the second queen exhibits wrap-around, appearing to control the diagonal extending down the lower left of the board, which is a mistake.

Your task is to fix these mistakes so that the game works properly, and a user can attempt to place as many queens as possible on the board. Can you find a pattern for how many queens can be put on a board of size `dim`?

```
What is the width of your square chess board? 10

Indicate the row and column where the queen is to go:

1 1

 1 1 0 0 0 0 0 0 0 0
 1 Q 1 1 1 1 1 1 1 1
 0 1 1 0 0 0 0 0 0 0
 0 1 0 1 0 0 0 0 0 0
 0 1 0 0 1 0 0 0 0 0
 0 1 0 0 0 1 0 0 0 0
 0 1 0 0 0 0 1 0 0 0
 0 1 0 0 0 0 0 1 0 0
 0 1 0 0 0 0 0 0 1 0
 0 1 0 0 0 0 0 0 0 1

Indicate the row and column where the queen is to go:

2 9

 1 1 0 0 0 0 0 2 0 2
 1 Q 1 1 1 1 1 1 1 1
 2 1 1 2 2 2 2 2 2 Q
 2 1 0 1 0 0 0 0 0 2
 0 1 0 0 1 0 0 0 0 2
 0 1 2 0 0 1 0 0 0 2
 0 1 0 2 0 0 1 0 0 2
 0 1 0 0 2 0 0 1 0 2
 0 1 0 0 0 2 0 0 1 2
 0 1 0 0 0 0 2 0 0 1
```