ITE 365 Lab05: The Eight Queens Problem

Adam Lewis

May 18, 2018

Contents

1 Objectives

1. Continue to work with arrays in C#

2. Study a classic problem from AI and algorithm theory

2 Background

A classic problem in chess is the "Eight Queens Problem": Is it possible to place eight queens on an empty chessboard so that no queen is "attacking" any other queen. This means that no two queens are in the same row, in the same column, or along the same diagonal.

2.1 Algorithm

There are a number of ways to attack this problem. In our case, we will start by assigning a number in each square of an empty chessboard that would indicate how many squares would be "eliminated" if we were to place a queen in that square. For example, each of the corner squares will be assigned the number 22 as that is the number of squares "eliminated" along that row, column, and diagonal.

Once we have done this, then we can attempt to place queens on the board by putting a queen onto the square that has the "smallest" elimination number.

3 Instructions

Start a new application in Visual Studio named "EightQueens". Rename the file and class as normal

3.1 Data structures

We will need to keep track of the location of each queen on the board + the current set of access values:

```
static Random randomNumbers = new Random();

static bool[ , ] board; // gameboard

// accessibility values for each board position
static int[ , ] access = { { 22, 22, 22, 22, 22, 22, 22, 22 },
                           { 22, 24, 24, 24, 24, 24, 24, 22 },
                           { 22, 24, 26, 26, 26, 26, 24, 22 },
                           { 22, 24, 26, 28, 28, 26, 24, 22 },
                           { 22, 24, 26, 28, 28, 26, 24, 22 },
                           { 22, 24, 26, 26, 26, 26, 24, 22 },
                           { 22, 24, 24, 24, 24, 24, 24, 22 },
                           { 22, 22, 22, 22, 22, 22, 22, 22 } };
static int maxAccess = 99; // dummy value to indicate that queen placed

static int queens; // number of queens placed on the board
```

3.2 Now let's try to place things onto the board

In your **main()** function, define two variables to keep track of the current row and column:

```
int currentRow; // the row position on the chessboard
int currentColumn; // the column position on the chessboard

board = new bool[ 8, 8 ]; // all elements default to false
```

We pick a random position for the first queen. We now need to update the access values for the selected position by calling a method **UpdateAccess()** (which we'll define shortly):

```
// randomize initial first queen position
currentRow = randomNumbers.Next( 8 );
currentColumn = randomNumbers.Next( 8 );

board[ currentRow, currentColumn ] = true;
++queens;

UpdateAccess( currentRow, currentColumn ); // update access
```

Now we can look for the square on the board that has the lowest access number:

```
bool done = false;

// continue until finished traversing
while ( !done )
{
   // the current lowest access number
   int accessNumber = maxAccess;

   // find square with the smallest elimination number
   for ( int row = 0; row < board.GetLength( 0 ); row++ )
   {
      for ( int col = 0; col < board.GetLength( 1 ); col++ )
      {
         // obtain access number
         if ( access[ row, col ] < accessNumber )
         {
            accessNumber = access[ row, col ];
            currentRow = row;
            currentColumn = col;
         } // end if
      } // end inner for
   } // end outer for

   // traversing done
   if ( accessNumber == maxAccess )
      done = true;
   // mark the current location
   else
   {
      board[ currentRow, currentColumn ] = true;
      UpdateAccess( currentRow, currentColumn );
      ++queens;
   } // end else
} // end while
```

Finally we call a method to display the board:

```
PrintBoard();
```

3.3 Update the access for a new piece on the board

Now we can deal with updating the access values for a particular entry on the board:

```
public static void UpdateAccess( int row, int column )
{
   for ( int i = 0; i < 8; i++ )
   {
      // set elimination numbers to 99
      // in the row occupied by the queen
      access[ row, i ] = maxAccess;

      // set elimination numbers to 99
      // in the column occupied by the queen
      access[ i, column ] = maxAccess;
   } // end for
   // set elimination numbers to 99 in diagonals occupied by the queen
   UpdateDiagonals( row, column );
} // end method UpdateAccess
```

3.3.1 Updating the diagonals

Now we can deal with each of the diagonals. Here, we have to take advantage of a helper function that checks to see if we have a valid move:

```
public static void UpdateDiagonals( int rowValue, int colValue )
 {
    int row = rowValue; // row position to be updated
    int column = colValue; // column position to be updated

    // upper left diagonal
    for ( int diagonal = 0; diagonal < 8 &&
       ValidMove( --row, --column ); diagonal++ )
       access[ row, column ] = maxAccess;

    row = rowValue;
    column = colValue;

    // upper right diagonal
    for ( int diagonal = 0; diagonal < 8 &&
       ValidMove( --row, ++column ); diagonal++ )
       access[ row, column ] = maxAccess;

    row = rowValue;
    column = colValue;

    // lower left diagonal
    for ( int diagonal = 0; diagonal < 8 &&
       ValidMove( ++row, --column ); diagonal++ )
       access[ row, column ] = maxAccess;

    row = rowValue;
    column = colValue;

    // lower right diagonal
    for ( int diagonal = 0; diagonal < 8 &&
       ValidMove( ++row, ++column ); diagonal++ )
       access[ row, column ] = maxAccess;
 } // end method UpdateDiagonals
```

3.4 Testing to see if we have a valid move

Testing for a valid move requires that we keep the row and column indexes with the valid range:

```
public static bool ValidMove( int row, int column )
  {
     return ( row >= 0 && row < 8 && column >= 0 && column < 8 );
  } // end method ValidMove
```

3.5 One last method: printing the board

Our last task is to print the board:

```
// display the board
public static void PrintBoard()
{
   Console.Write( "  " );

   // display numbers for column
   for ( int k = 0; k < 8; k++ )
      Console.Write( " {0}", k );

   Console.WriteLine( "\n" );

   for ( int row = 0; row < board.GetLength( 0 ); row++ )
   {
      Console.Write( "{0} ", row );

      for ( int column = 0; column < board.GetLength( 1 ); column++ )
      {
         if ( board[ row, column ] )
            Console.Write( " Q" );
         else
            Console.Write( " ." );
      } // end for
      Console.WriteLine();
   } // end for

   Console.WriteLine( "\n{0} queens placed on the board.", queens );
} // end method PrintBoard
```

4 Submission instructions

Combine a copy of your source code and a screen-shot of your program into a PDF file (you can use Microsoft Word to do this) and attach it to the assignment on Blackboard.