Mark Eatough

CSIS 2430 9:00 Class

Programming Project 10

Knights Tour/8Queens Program

Assignment objective:

My student number is odd so I was assigned the 8 queens problem. The goal of the 8 queens problem is to place 8 queens on an 8X8 chess board such that no two queen attack each other. Therefore no two queens can share the same row, column, or diagonal.

What Worked?:

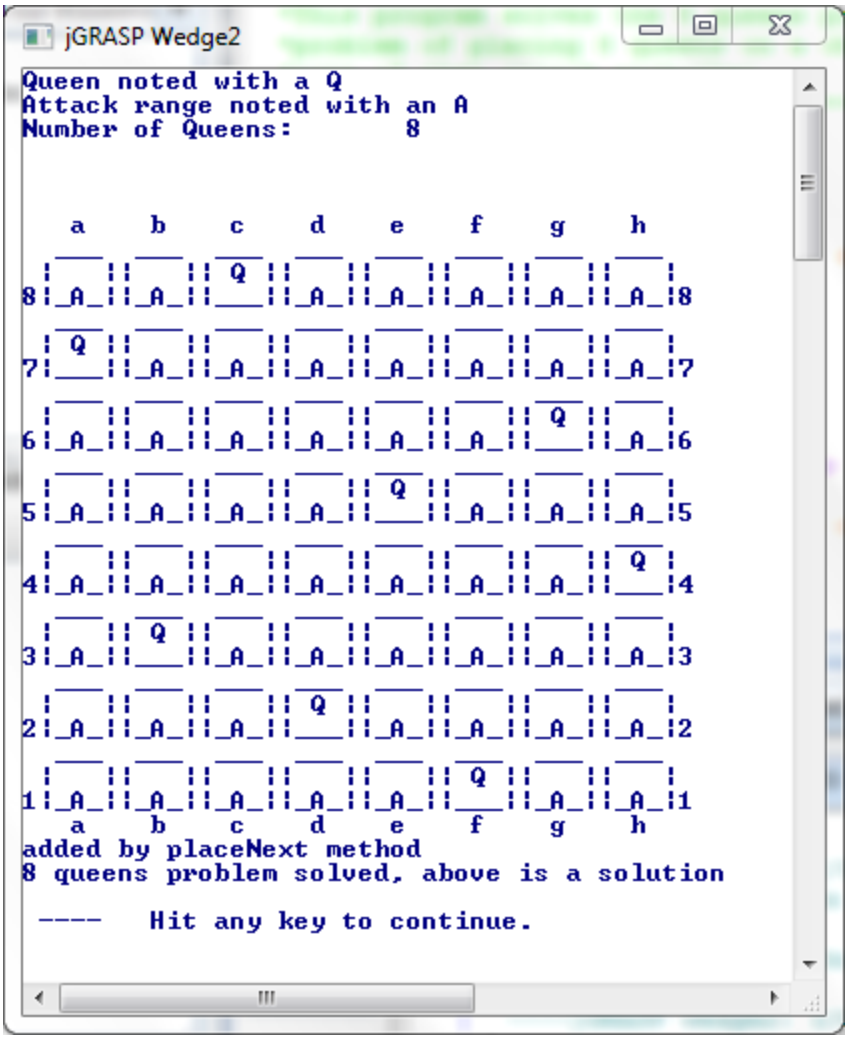
I created a two dimensional list to represent the chess board, created a chess coordinates object to represent the coordinates of the queens on the chessboard, and used a list as a stack to store the coordinates of the queens on the chessboard. I used "Q" to denote a queen on the board and "A" to denote an area of the board that was in the attack space of any queen or queens. Every time a queen was placed on the board I would fill out the board with all of the attack spaces, if any row was completely attacked, that is it had no open spaces for a queen then a queen was removed, and moved to a space later in the row that the queen was removed from. If the end of the row was reached with no space for a queen the next queen on the stack was popped. This process continued until the 8 queens problem was solved. I used a random number generator to get the row and column of the first queen, and then my algorithms placed the queens in their necessary positions after that.

What did not work?:

I tried to use a Unicode character for the queen icon, but python 2.X did not support this functionality in a way that was practical for this assignment. I was also hoping to be able to color coat the chess board like a normal board, or at least color the attack zones of the queens a different color, but I could not get this to work either. Finally I originally tried to use the same method to place a queen and the replace a queen. This did not work however, and I needed two separate methods.

Comments:

I really enjoyed this assignment, and feel I learned a lot. One of the things I read online said the 8 queens problem is often used to teach recursion. Earlier in this class we proved that iteration is always faster than recursion, so for this implementation I tried to use iteration rather than recursion as much as possible. I used recursion somewhat indirectly in my remove and replace queen methods. The remove queen method called the replace queen method. The replace queen method moved the queen to a column further down the same row if possible, if the end of the row was reached the replace queen method called the remove queen method and the process started all over again.


jGRASP Wedge2
Queen noted with a Q
Attack range noted with an A
Number of Queens: 8
a b c d e f g h
8 Q 8
7 Q 7
6 Q 6
5 Q 5
4 Q 4
3 Q 3
2 Q 2
1 Q 1
a b c d e f g h
added by placeNext method
8 queens problem solved, above is a solution
---- Hit any key to continue.

```
'''
************************************************************
 * Discrete Structures
 * 8 Queens Program
 * Programmer: Mark Eatough
 * Course: CSIS 2430
 * Created Novermber 4, 2013

 *This program solves the 8 queens problem, which is the
 *problem of placing 8 queens on a chess board so that none
 *none of them attack eachother.
 ************************************************************
'''
from random import randint
import os
import time
#my s number ends in an odd number, so I will implement the 8 queens
class ChessCoord:
   def __init__(self, row, col):
      self.row = row
      self.col = col

queenStack = []
matrix = [[0 for x in xrange(8)] for x in xrange(8)]

#method to track horizontal attacks of queens
def verticleAttack(m,n):
   for i in range(m+1,len(matrix)):
      if(matrix[i][n] != 1):
         matrix[i][n] += 1
   for j in range(0,m):
      if(matrix[j][n] != 1):
         matrix[j][n] += 1
#method to track horizontal attacks of queens
def horizontalAttack(m,n):
   for i in range(n+1,len(matrix)):
      if(matrix[m][i] != 1):
         matrix[m][i] += 1
   for j in range(0,n):
      if(matrix[m][j] != 1):
         matrix[m][j] += 1
#method to track left diagonal attacks of queens
def leftDiagonal(m,n):
   #check diagonal starting from origin to bottom right corner
   i = m+1
```

```
    j = n+1
    while(i<len(matrix) and j<len(matrix)):
        if(matrix[i][j]!=1):
            matrix[i][j]+=1
        i+=1
        j+=1
    #check diagonal starting from origin to top left corner
    k = m-1
    l = n-1
    while(k>=0 and l>=0):
        if(matrix[k][l]!=1):
            matrix[k][l]+=1
        k-=1
        l-=1
#method to track right diagonal attacks of queens
def rightDiagonal(m,n):
    #check diagonal starting from origin to bottom left
corner
    i = m-1
    j = n+1
    while(i>=0 and j<len(matrix)):
        if(matrix[i][j]!=1):
            matrix[i][j]+=1
        i-=1
        j+=1
    #check diagonal starting from origin to top right corner
    k = m+1
    l = n-1
    while(k<len(matrix) and l>=0):
        if(matrix[k][l]!=1):
            matrix[k][l]+=1
        k+=1
        l-=1
#calls all directional attacks
def attackRange(m,n):
    horizontalAttack(m,n)
    verticleAttack(m,n)
    rightDiagonal(m,n)
    leftDiagonal(m,n)
#method to make the mid portion of individual chess square
def makeMidChessSquare(n):
    if(n < 2):
        middle = "|   |"
    else:
        middle = "| Q |"
    return middle
#method to make the bottom portion of individual chess
```

```
square
 92 def makeBotChessSquare(n):
 93    if(n == 0 or n == 2):
 94       bottom = "|___|"
 95    else:
 96       bottom = "|_A_|"
 97    return bottom
 98 #method to display entire chess board
 99 def displayChessBoard(twoD):
100    os.system('cls')
101    print "Queen noted with a Q"
102    print "Attack range noted with an A"
103    print "Number of Queens:\t", len(queenStack), "\n\n\n"
104    tab = " "
105    num = 8
106    a=ord('a')
107    h=ord('h')
108    columns = tab
109    for letter in range(a, h+1):
110       columns+="  "+chr(letter)+"  "
111    top = " ___ "
112    middle = "|   |"
113    bottom = "|___|"
114    print columns
115    for i in range(8):
116       num = 8-i
117       middle = tab + makeMidChessSquare(twoD[i][0])
118       gTop = tab + top
119       gBottom = str(num) + makeBotChessSquare(twoD[i][0])
120       for j in range(1,8):
121          gTop+=top
122          middle+= makeMidChessSquare(twoD[i][j])
123          gBottom+=makeBotChessSquare(twoD[i][j])
124       print gTop
125       print middle
126       print gBottom+str(num)
127    print columns
128 #method to add a queen
129 def addQueen(m,n):
130    matrix[m][n]+=2
131    attackRange(m,n)
132    queenStack.append(ChessCoord(m,n))
133 #method to remove a queen
134 def removeQueen():
135    rq = queenStack.pop()
136    for i in range(len(matrix)):
137       for j in range(len(matrix)):
```

```
138             matrix[i][j] = 0
139       for k in range(len(queenStack)):
140          tempQueen = queenStack[k]
141          matrix[tempQueen.row][tempQueen.col] += 2
142          attackRange(tempQueen.row,tempQueen.col)
143       displayChessBoard(matrix)
144       print "Queen removed"
145       time.sleep(.5)
146       replaceQueen(rq)
147
148 #method to replace removed queen
149 def replaceQueen(rq):
150       myCol = rq.col+1
151       myRow = rq.row
152       while(myCol < 8):
153          if(matrix[myRow][myCol]==0):
154             addQueen(myRow, myCol)
155             displayChessBoard(matrix)
156             print "Queen replaced in different column in same
row"
157             time.sleep(.5)
158             return
159          myCol+=1
160       print "No more valid spaced for queens, call remove queen
method again"
161       time.sleep(.5)
162       removeQueen()
163 #method to check if a row has a problem
164 def checkProblemRows():
165       for i in range(len(matrix)):
166          for j in range(len(matrix)):
167             k = matrix[i][j]
168             if(k==0 or k==2):
169                break
170          if(k!=0 and k!=2):
171             return i
172       return -1
173 #randomly select a position for first queen
174 def firstQueen():
175       m = randint(0,7)
176       n = randint(0,7)
177       addQueen(m,n)
178       displayChessBoard(matrix)
179 #method to place the next queen
180 def placeNext():
181       for i in range(len(matrix)):
182          for j in range(len(matrix)):
```

```
183          if(matrix[i][j] == 0):
184             addQueen(i,j)
185             displayChessBoard(matrix)
186             print "added by placeNext method"
187             time.sleep(.5)
188             return
189 #use algorithm to select position for queens without any
attacking eachother
190 def placeQueens():
191    while(len(queenStack) < 8):
192       placeNext()
193       while(checkProblemRows()>-1):
194          removeQueen()
195    print "8 queens problem solved, above is a solution"
196 displayChessBoard(matrix)
197 firstQueen()
198 placeQueens()
```