**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?:**

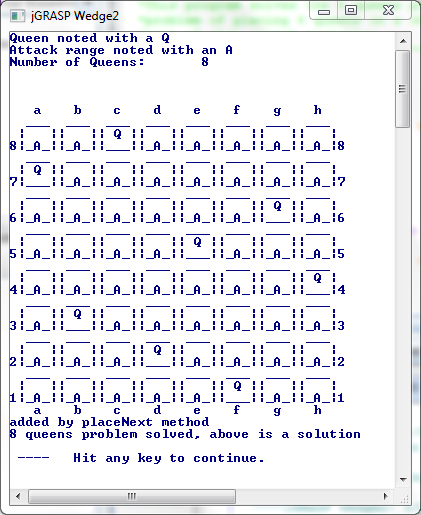
I created a two dimensional list to represent the chess board, created a chess coordinates object to represent the coordinates of the queens on the chessboard, and used a list as a stack to store the coordinates of the queens on the chessboard. I used “Q” to denote a queen on the board and “A” to denote 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.



1 '''  
 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   
 3 \* Discrete Structures  
 4 \* 8 Queens Program  
 5 \* Programmer: Mark Eatough  
 6 \* Course: CSIS 2430   
 7 \* Created Novermber 4, 2013  
 8   
 9 \*This program solves the 8 queens problem, which is the   
 10 \*problem of placing 8 queens on a chess board so that none  
 11 \*none of them attack eachother.  
 12 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 13 '''  
 14 from random import randint  
 15 import os  
 16 import time  
 17 #my s number ends in an odd number, so I will implement the 8 queens   
 18 class ChessCoord:  
 19 def \_\_init\_\_(self, row, col):  
 20 self.row = row  
 21 self.col = col  
 22   
 23 queenStack = []  
 24 matrix = [[0 for x in xrange(8)] for x in xrange(8)]   
 25   
 26 #method to track horizontal attacks of queens  
 27 def verticleAttack(m,n):  
 28 for i in range(m+1,len(matrix)):  
 29 if(matrix[i][n] != 1):  
 30 matrix[i][n] += 1  
 31 for j in range(0,m):  
 32 if(matrix[j][n] != 1):  
 33 matrix[j][n] += 1  
 34 #method to track horizontal attacks of queens  
 35 def horizontalAttack(m,n):  
 36 for i in range(n+1,len(matrix)):  
 37 if(matrix[m][i] != 1):  
 38 matrix[m][i] += 1  
 39 for j in range(0,n):  
 40 if(matrix[m][j] != 1):  
 41 matrix[m][j] += 1  
 42 #method to track left diagonal attacks of queens  
 43 def leftDiagonal(m,n):  
 44 #check diagonal starting from origin to bottom right corner  
 45 i = m+1  
 46 j = n+1  
 47 while(i<len(matrix) and j<len(matrix)):  
 48 if(matrix[i][j]!=1):  
 49 matrix[i][j]+=1  
 50 i+=1  
 51 j+=1  
 52 #check diagonal starting from origin to top left corner  
 53 k = m-1  
 54 l = n-1  
 55 while(k>=0 and l>=0):  
 56 if(matrix[k][l]!=1):  
 57 matrix[k][l]+=1  
 58 k-=1  
 59 l-=1  
 60 #method to track right diagonal attacks of queens  
 61 def rightDiagonal(m,n):  
 62 #check diagonal starting from origin to bottom left corner  
 63 i = m-1  
 64 j = n+1  
 65 while(i>=0 and j<len(matrix)):  
 66 if(matrix[i][j]!=1):  
 67 matrix[i][j]+=1  
 68 i-=1  
 69 j+=1  
 70 #check diagonal starting from origin to top right corner   
 71 k = m+1  
 72 l = n-1  
 73 while(k<len(matrix) and l>=0):  
 74 if(matrix[k][l]!=1):  
 75 matrix[k][l]+=1  
 76 k+=1  
 77 l-=1   
 78 #calls all directional attacks  
 79 def attackRange(m,n):  
 80 horizontalAttack(m,n)  
 81 verticleAttack(m,n)   
 82 rightDiagonal(m,n)  
 83 leftDiagonal(m,n)  
 84 #method to make the mid portion of individual chess square  
 85 def makeMidChessSquare(n):  
 86 if(n < 2):  
 87 middle = "| |"  
 88 else:  
 89 middle = "| Q |"  
 90 return middle  
 91 #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()