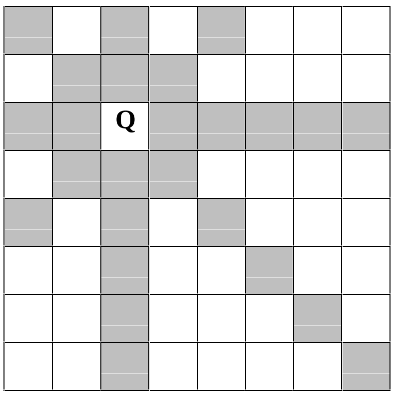
CS 100: Programming Assignment P7 (Cont'd)

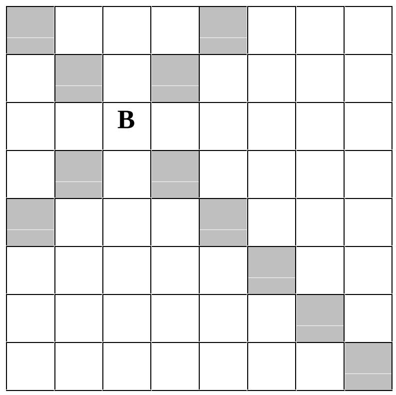
Due: Friday, May 7, 5pm, Carpenter Lab (or in lecture)

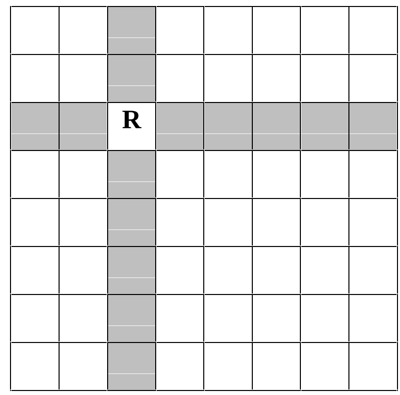
You may work in pairs. Do not submit your assignment for grading unless you have read and understand the CS100 webpage on Academic Integrity. Follow the course rules for the submission of assignments or lose points.

Part B. Threats on a Chessboard

In chess the major pieces are Queen (Q) , Bishop (B), Rook (R), King (K) and Knight (Kn). Each piece can only move in a certain way. A tile on the chessboard is *threatened* if it can be reached by a piece. A piece does not threaten the tile it occupies. Here is how the Queen, Bishop, and Rook threaten:



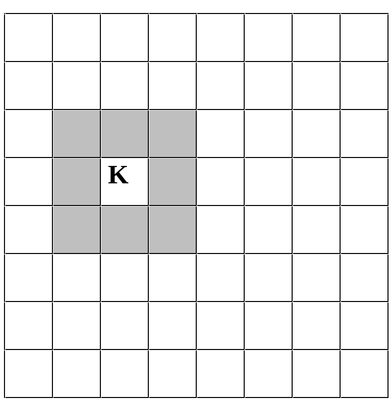


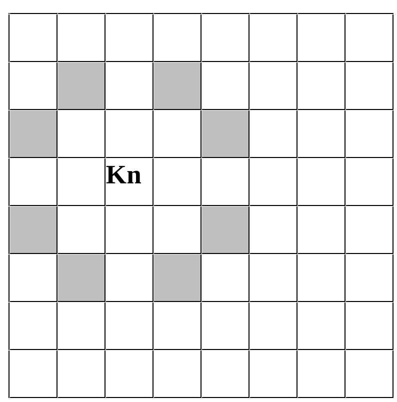


In talking about locations on the 8-by-8 chessboard we use a "(row index,column index)" notation. The upper left tile is the (0,0) tile. In the middle example the Bishop can reach tiles (0,0), (1,1), (3,3), (4,4), (5,5), (6,6), (7,7), (4,0), (3,1), (1,3) and (0,4). Since those tiles are threatened, they are shaded. More generally,

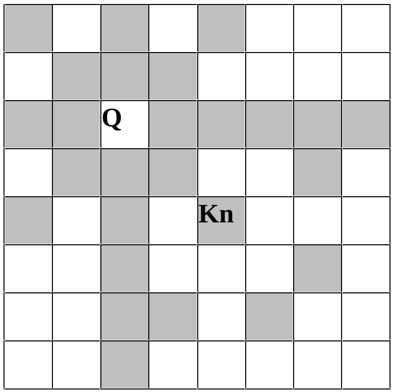
- The Queen can move left along a row, right along a row, up along a column, down along a column, diagonally "northeast", diagonally "southeast", diagonally "southwest", or diagonally "northwest".
- The Bishop can move diagonally "northeast", diagonally "southeast", diagonally "southwest", or diagonally "northwest".
- The Rook can move left along a row, right along a row, up along a column, or down along a column.

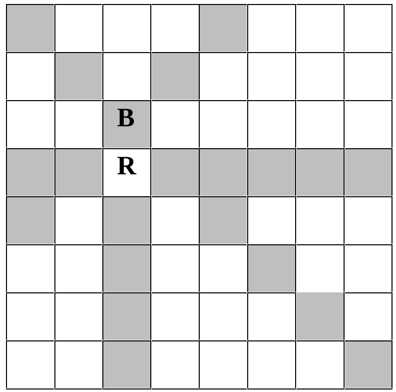
The King and the Knight have a more local impact:

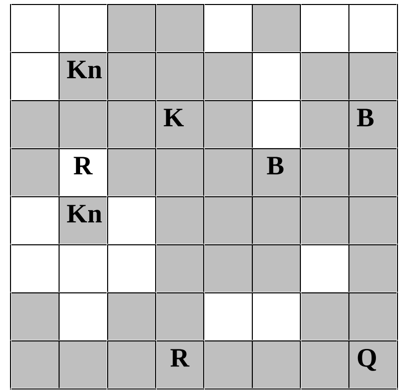




When several pieces are on the board, there can be “shadows”, i.e., a tile can be safe because one piece blocks the threatening piece from view. Here are some examples:







Thus, in the first example tiles (5,5), (6,6), and (7,7) are not threatened by the Queen even though they are on the same diagonal. This is because the Knight blocks the threat.

The Method `qrbThreat`

You are to complete the method `qrbThreat` in the class `chess` so that it performs as specified. After this is done and you run the main method in `P7B.java` the rightmost example above will be displayed. Submit a listing of chess that includes all your contributions and a copy of the output that it produces. The necessary Java files are on the website.

Start by reviewing the class `chess`. Observe that an instance of this class has two fields. We represent the state of the chess board with an 8-by-8 array of strings. If a Queen is located at (4,3), then `pieceBoard[4][3]` has the value “`Q`”. The boolean 8-by-8 array `threatBoard` is used to mark which tiles are threatened. Thus, if (2,7) is threatened by any piece on the board, then `threatBoard[2][7]` is true. Note how the constructor initializes these two fields and that it uses the methods `kingThreat`, `knightThreat`, and `qrbThreat.`There are two “get methods” : `getPiece` and `getThreat`. Observe how they are used in `P7B.java`. The method `onBoard` can be used to check if a given integer pair (r,c) is on the chessboard. Appreciate its value by looking at the `kingThreat` and `knightThreat` implementations.

Finally we come to `qrbThreat.` (So that the given programs run, we have implemented `qrbThreat` so that it always returns false.) Some observations about checking whether or not (r,c) is threatened by a Queen, Rook, or Bishop:

- Look left along row r. If no piece is found then there is no threat from this direction. If the first piece encountered in this direction is a Rook or Queen, then `qrbThreat` should return `true.` The same process needs to be repeated when looking right along row r, down along column c, and up along column c.
- Look up along the northeast diagonal. If no piece is found then there is no threat from this direction. If the first piece encountered in this direction is a Bishop or Queen, then `qrbThreat` should return `true`. The same process needs to be repeated when searching along the southeast diagonal, the southwest diagonal, and the northwest diagonal.
- Use a `while` loop for these searches and make effective use of `onBoard` so that your searching doesn’t carry you off the board. As soon as `qrbThreat` discovers that (r,c) is threatened by a Queen, Rook, or Bishop, it should return immediately with the value true. Appreciate that there is no need to continue the serach once this happens.
- As we have discussed things, it looks like you’ll need eight while-loop fragments to do the searching. However, the only way these searches differ is in how you step from tile to tile: (row_step,col_step) = (0,1) for searching right along a row, , (0,-1) for searching left along a row, (-1, 0) for searching up along a column, (1,0) for searching down along a column, (-1,1) for searching northeast, (1,1) for searching southeast, (1,-1) for searching southwest, and (-1,-1) for searching northwest. Exploitation of this greatly reduces the amount of code you have to write. (Hint: use nested for loops to carry you through all the `row_step` and `col_step` possibilities.
- Start by implementing one of the eight searches to see what is going on. Then either add the other seven to your implementation explicitly or try to follow the preceding hint.
- You are free to add additional methods to `chess.` But do not change any of the given methods (except `qrbThreat`) and do not add additional fields.

Chess.java

```
public class chess{
    private String[][]  pieceBoard;
    private boolean[][] threatBoard;

    // An instance of this class is chessboard with the piece positions indicated by pieceBoard
    // and the threatened positions indicated by threatBoard. These are both 8-by-8 arrays.

    // pieceBoard[i][j] = "K", "Q", "R", "B", "Kn", or "" according to whether there is a King,
    // Queen, Rook, Bishop, Knight, or nothing at position (i,j).

    // threatBoard[i][j] is true if position (i,j) is threatened and is false otherwise.

    // Constructor. B must be 8-by-8 with entries equal to "K", "Q", "R", "B", "Kn" or "".
    // Thus, B specifies the location of the pieces.
    public chess(String[][] B)
    {
       pieceBoard  = new String[8][8];
       threatBoard = new boolean[8][8];

       for(int r=0;r<=7;r++)
          for(int c=0;c<=7;c++)
             pieceBoard[r][c] = new String(B[r][c]);
       for(int r=0;r<=7;r++)
          for(int c=0;c<=7;c++)
             // the (r,c) position is threatened if it is threatened by a King, Knight, or
             // a Queen, Rook, or Bishop.
             threatBoard[r][c] =  kingThreat(r,c)||knightThreat(r,c)||qrbThreat(r,c);
    }

    // Yields true if position (r,c) is threatened.
    public boolean getThreat(int r, int c) { return threatBoard[r][c];}

    // Yields the piece situated at position (r,c).
    public String getPiece(int r, int c) {return new String(pieceBoard[r][c]);}

    // Yields true if (r,c) is a valid board position.
   public static boolean onBoard(int r, int c){return 0<=r && r<=7 && 0<=c && c<=7;}

   // Yields true if position (r,c) is threatened by a King.
   public boolean kingThreat(int r, int c)
   {
      if        (onBoard(r-1,c-1) && pieceBoard[r-1][c-1].equals("K")) {return true;}
       else if  (onBoard(r-1,c )  && pieceBoard[r-1][c  ].equals("K")) {return true;}
       else if  (onBoard(r-1,c+1) && pieceBoard[r-1][c+1].equals("K")) {return true;}
       else if  (onBoard(r  ,c-1) && pieceBoard[r  ][c-1].equals("K")) {return true;}
       else if  (onBoard(r  ,c+1) && pieceBoard[r  ][c+1].equals("K")) {return true;}
       else if  (onBoard(r+1,c-1) && pieceBoard[r+1][c-1].equals("K")) {return true;}
       else if  (onBoard(r+1,c  ) && pieceBoard[r+1][c  ].equals("K")) {return true;}
       else if  (onBoard(r+1,c+1) && pieceBoard[r+1][c+1].equals("K")) {return true;}
       else {return false;}
    }

    // Yields true if position (r,c) is threatened by a Knight.
    public boolean knightThreat(int r, int c)
   {
      if        (onBoard(r-2,c-1) && pieceBoard[r-2][c-1].equals("Kn")) {return true;}
       else if  (onBoard(r-2,c+1) && pieceBoard[r-2][c+1].equals("Kn")) {return true;}
       else if  (onBoard(r-1,c-2) && pieceBoard[r-1][c-2].equals("Kn")) {return true;}
       else if  (onBoard(r-1,c+2) && pieceBoard[r-1][c+2].equals("Kn")) {return true;}
       else if  (onBoard(r+1,c-2) && pieceBoard[r+1][c-2].equals("Kn")) {return true;}
       else if  (onBoard(r+1,c+2) && pieceBoard[r+1][c+2].equals("Kn")) {return true;}
       else if  (onBoard(r+2,c-1) && pieceBoard[r+2][c-1].equals("Kn")) {return true;}
       else if  (onBoard(r+2,c+1) && pieceBoard[r+2][c+1].equals("Kn")) {return true;}
       else return false;
    }

    // Yields true if position (r,c) is threatened by a Queen, a Rook, or a Bishop.
    public boolean qrbThreat(int r, int c)
    {

        return false;
    }
}
```

P7B.java

```
import java.io.*;
import java.awt.*;

public class ShowP7b extends Frame
{
    // Displays the chessboard showing threatened tiles.
    public void showThreats(Graphics g, chess C)
    {
       int L = 50;
       int T = 50;
       int s = 60;
       String piece;
       g.setFont(new Font("TimesRoman",Font.PLAIN, 18));
       for(int r=0;r<=7;r++)
          for(int c=0;c<=7;c++)
          {
             if(C.getThreat(r,c))
                g.setColor(Color.magenta);
             else
                g.setColor(Color.white);
             g.fillRect(L+c*s,T+r*s,s,s);
             g.setColor(Color.black);
             g.drawRect(L+c*s,T+r*s,s,s);
             g.drawString(C.getPiece(r,c),25+L+c*s,35+T+r*s);
          }
    }

   public void paint(Graphics g)
   {
      String[][] B = {
                        {"" , ""   , "" , ""  , "" , ""  , "" , "" },
                        {"" , "Kn" , "" , ""  , "" , ""  , "" , "" },
                        {"" , ""   , "" , "K" , "" , ""  , "" , "B"},
                        {"" , "R"  , "" , ""  , "" , "B" , "" , "" },
                        {"" , "Kn" , "" , ""  , "" , ""  , "" , "" },
                        {"" , ""   , "" , ""  , "" , ""  , "" , "" },
                        {"" , ""   , "" , ""  , "" , ""  , "" , "" },
                        {"" , ""   , "" , "R" , "" , ""  , "" , "Q"}
                      };

      chess C = new chess(B);

      showThreats(g,C);

   }

}

public class P7B
{
   public static void main(String args[])
   {
      ShowP7b d = new ShowP7b();
      d.resize(800,600);
      d.move(0,75);
      d.setTitle("Chess");
      d.show();
      d.toFront();
   }
}
```