import java.util.ArrayList;

public abstract class Piece{ // ENTIRE class coded by Aman Modi

private int row;

private int col;

private String color = "";

private String pieceType;

public Piece[][] array = Board.board1;

private ArrayList<String> capturedWhite ;

private ArrayList<String> capturedBlack;

private boolean enPassant;

private boolean moved;private int p1Score;

private int p2Score;

private Multiplayer multi = new Multiplayer();

private boolean twoMovement = true;

public Piece(int r, int c, String s, String pt){

pieceType = pt;

row = r;

col = c;

color = s;

enPassant = false;

moved = false;

p1Score = 0;

p2Score = 0;

capturedWhite = new ArrayList<String>();

capturedBlack = new ArrayList<String>();

}

public boolean getTwoMovement(){

return twoMovement;

}

public void setTwoMovement(boolean tm){

twoMovement = tm;

}

public boolean getMoved(){

return moved;

}

public void setMoved(boolean m){

moved = m;

}

public boolean getEnPassant(){

return enPassant;

}

public void setEnPassant(boolean ep){

enPassant = ep;

}

public int getRow(){

return row;

}

public void setRow(int r){

row = r;

}

public int getCol(){

return col;

}

public void setCol(int c){

col = c;

}

public String getColor(){

return color;

}

public void setColor(String c){

color = c;

}

public String getPieceType(){

return pieceType;

}

public void setPieceType(String pt){

pieceType = pt;

}

public void changePosition(int r, int c)

{

int newr = getRow();

int newc = getCol();

array[r][c] = array[getRow()][getCol()];

if(newr%2==0 && newc%2==0 || newr%2 != 0 && newc%2!=0)

array[newr][newc] = new DefaultWhite(newr,newc,"#");

else

array[newr][newc] = new DefaultBlack(newr,newc,"/");

setRow(r);

setCol(c);

}

public boolean capturePiece(int r, int c) //Start of Ayaan

{

if(!(array[r][c].getPieceType().equals("#") || array[r][c].getPieceType().equals("/")) && !array[r][c].getColor().equals(getColor()))

return true;

else

return false;

} //End of Ayaan

public boolean checkBounds(int r, int c)

{

if((r<8 && r>=0) && (c<8 && c>=0))

return true;

else

return false;

}

public boolean checkOpen(int r, int c)

{

if(array[r][c].getPieceType().equals("#") || array[r][c].getPieceType().equals("/"))

return true;

else

return false;

}

public boolean checkForCheck(int r, int c, String co)

{

int origR = getRow();//Stores all data of current space

int origC = getCol();//Stores all data of current space

String color1 = array[r][c].getColor();//Stores all data of current space

String pt1 = array[r][c].getPieceType();//Stores all data of current space

boolean moved1 = array[r][c].getMoved();//Stores all data of current space

boolean twoMovement1 = array[r][c].getTwoMovement();

changePosition(r,c);//changes position momentarily

if(array[r][c].getPieceType().equals("k"))

{

Multiplayer.bKingr=r;

Multiplayer.bKingc=c;

}

else if(array[r][c].getPieceType().equals("K"))

{

Multiplayer.wKingr=r;

Multiplayer.wKingc=c;

}

if(inCheck(co))//checks if in check in modified position

{

array[origR][origC] = array[r][c];//sets everything back to the way it was

setRow(origR);//sets everything back to the way it was

setCol(origC);//sets everything back to the way it was

switch(pt1)//sets everything back to the way it was

{

case "/": array[r][c] = new DefaultBlack(r,c,"/");break;

case "#": array[r][c] = new DefaultWhite(r,c,"#");break;

case "b": array[r][c] = new Bishop(r,c,color1,pt1);break;

case "B": array[r][c] = new Bishop(r,c,color1,pt1);break;

case "P": array[r][c] = new Pawn(r,c,color1,pt1);break;

case "p": array[r][c] = new Pawn(r,c,color1,pt1);break;

case "Q": array[r][c] = new Queen(r,c,color1,pt1);break;

case "q": array[r][c] = new Queen(r,c,color1,pt1);break;

case "R": array[r][c] = new Rook(r,c,color1,pt1);break;

case "r": array[r][c] = new Rook(r,c,color1,pt1);break;

case "n": array[r][c] = new Knight(r,c,color1,pt1);break;

case "N": array[r][c] = new Knight(r,c,color1,pt1);break;

case "K": array[r][c] = new King(r,c,color1,pt1);break;

case "k": array[r][c] = new King(r,c,color1,pt1);break;

}

array[r][c].setMoved(moved1);//sets everything back to the way it was

array[r][c].setTwoMovement(twoMovement1);

return true;//returns not in check

}

else

{

array[origR][origC] = array[r][c];//sets everything back to the way it was

setRow(origR);//sets everything back to the way it was

setCol(origC);//sets everything back to the way it was

switch(pt1)//sets everything back to the way it was

{

case "/": array[r][c] = new DefaultBlack(r,c,"/");break;

case "#": array[r][c] = new DefaultWhite(r,c,"#");break;

case "b": array[r][c] = new Bishop(r,c,color1,pt1);break;

case "B": array[r][c] = new Bishop(r,c,color1,pt1);break;

case "P": array[r][c] = new Pawn(r,c,color1,pt1);break;

case "p": array[r][c] = new Pawn(r,c,color1,pt1);break;

case "Q": array[r][c] = new Queen(r,c,color1,pt1);break;

case "q": array[r][c] = new Queen(r,c,color1,pt1);break;

case "R": array[r][c] = new Rook(r,c,color1,pt1);break;

case "r": array[r][c] = new Rook(r,c,color1,pt1);break;

case "n": array[r][c] = new Knight(r,c,color1,pt1);break;

case "N": array[r][c] = new Knight(r,c,color1,pt1);break;

case "K": array[r][c] = new King(r,c,color1,pt1);break;

case "k": array[r][c] = new King(r,c,color1,pt1);break;

}

array[r][c].setMoved(moved1);//sets everything back to the way it was

array[r][c].setTwoMovement(twoMovement1);

return false;//return that they would be in check

}

}

public boolean inCheck(String co)

{

int q = 0;

//wKingr,wKingc,bKingr, and bKingc are the positions of the black and white kings. These are updated via the king code and are stored in Multiplayer

if(co.equals("w"))

{

int c = Multiplayer.wKingc;//preps column

int r = Multiplayer.wKingr;//preps row

for(int k = 0;k<2;k++)//checks Rook directions where k dictates horizontal or vertical, i dictates left/right or up/down, and j is distance from origin

for(int i = -1;i<2;i+=2)

for(int j = 1;j<7;j++)

{

if(checkBounds(r+(j\*i\*k),c+(j\*i\*(1-k))))

if(!checkOpen(r+(j\*i\*k),c+(j\*i\*(1-k)))){

{

if(array[r+(j\*i\*k)][c+(j\*i\*(1-k))].getPieceType().equals("r") || array[r+(j\*i\*k)][c+(j\*i\*(1-k))].getPieceType().equals("q"))

return true;

else if((Math.abs(j\*i\*k)==1 || Math.abs(j\*i\*(1-k)) == 1) && array[r+(j\*i\*k)][c+(j\*i\*(1-k))].getPieceType().equals("k"))

return true;

j=7;

}}

else

j = 7;

}

for(int k = -1;k<2;k+=2)//checks Bishop directions where k and i dictate either (+,+), (+,-), (-,+), or (-,-), and j is distance from origin

for(int i = -1;i<2;i+=2)

for(int j = 1;j<7;j++)

{

if(checkBounds(r+(k\*j),c+(j\*i))){

if(!checkOpen(r+(k\*j),c+(j\*i)))

{

if(array[r+(k\*j)][c+(j\*i)].getPieceType().equals("b") || array[r+(k\*j)][c+(j\*i)].getPieceType().equals("q"))

return true;

else if((Math.abs(j\*k)==1 || Math.abs(j\*i) == 1) && array[r+(k\*j)][c+(j\*i)].getPieceType().equals("k"))

return true;

else if((Math.abs(j\*i) == 1 && (j\*k) == 1) && array[r+(k\*j)][c+(j\*i)].getPieceType().equals("p"))

return true;

j=7;

}}

else

j = 7;

}

for(int i = -1;i<2;i+=2)//checks Knight directions where i is 1 left/right, j is 1 up/down from there, and k is the final disantce to a knight's directions (1 up/down or left/right)

for(int j = -1;j<2;j+=2)

for(int k = 0;k<2;k++)

{

if(checkBounds(r+(i\*(k+1)),c+(j\*(2-k))))

if(array[r+(i\*(k+1))][c+(j\*(2-k))].getPieceType().equals("n"))

return true;

}

}

else//everything here is copy and pasted (except for the changes in color)

{

int c = Multiplayer.bKingc;//preps column

int r = Multiplayer.bKingr;//preps row

for(int k = 0;k<2;k++)//checks Rook directions

for(int i = -1;i<2;i+=2)

for(int j = 1;j<7;j++)

{

if(checkBounds(r+(j\*i\*k),c+(j\*i\*(1-k)))){

if(!checkOpen(r+(j\*i\*k),c+(j\*i\*(1-k))))

{

if(array[r+(j\*i\*k)][c+(j\*i\*(1-k))].getPieceType().equals("R") || array[r+(j\*i\*k)][c+(j\*i\*(1-k))].getPieceType().equals("Q"))

return true;

else if((Math.abs(j\*i\*k)==1 || Math.abs(j\*i\*(1-k)) == 1) && array[r+(j\*i\*k)][c+(j\*i\*(1-k))].getPieceType().equals("K"))

return true;

j=7;

}}

else

j = 7;

}

for(int k = -1;k<2;k+=2)//check Bishop directions

for(int i = -1;i<2;i+=2)

for(int j = 1;j<7;j++)

{

if(checkBounds(r+(k\*j),c+(j\*i))){

if(!checkOpen(r+(k\*j),c+(j\*i)))

{

if(array[r+(k\*j)][c+(j\*i)].getPieceType().equals("B") || array[r+(k\*j)][c+(j\*i)].getPieceType().equals("Q"))

return true;

else if((Math.abs(j\*k)==1 || Math.abs(j\*i) == 1) && array[r+(k\*j)][c+(j\*i)].getPieceType().equals("K"))

return true;

else if((Math.abs(j\*i) == 1 && (j\*k) == 1) && array[r+(k\*j)][c+(j\*i)].getPieceType().equals("P"))

return true;

j=7;

}}

else

j = 7;

}

for(int i = -1;i<2;i+=2)//checks Knight direction

for(int j = -1;j<2;j+=2)

for(int k = 0;k<2;k++)

{

if(checkBounds(r+(i\*(k+1)),c+(j\*(2-k))))

if(array[r+(i\*(k+1))][c+(j\*(2-k))].getPieceType().equals("N"))

return true;

}

}

return false;

}

public abstract boolean movePiece(int r, int c, String co);

public boolean checkForStalemate(String co)

{

for(int r = 0; r<array.length; r++)

for(int c = 0; c<array[r].length; c++)

if(!(checkOpen(r,c) || array[r][c].getColor().equals(co)))

{

if(array[r][c].getPieceType().equalsIgnoreCase("r") || array[r][c].getPieceType().equalsIgnoreCase("q"))

{

for(int k = 0;k<2;k++)//checks Rook directions

for(int i = -1;i<2;i+=2)

for(int j = 1;j<7;j++)

if(checkBounds(r+(j\*i\*k),c+(j\*i\*(1-k))))

return false;

else

j=7;

}

else if(array[r][c].getPieceType().equalsIgnoreCase("b") || array[r][c].getPieceType().equalsIgnoreCase("q"))

{

for(int k = -1;k<2;k+=2)//check Bishop directions

for(int i = -1;i<2;i+=2)

for(int j = 1;j<7;j++)

if(checkBounds(r+(k\*j),c+(j\*i)))

return false;

else

j=7;

}

else if(array[r][c].getPieceType().equalsIgnoreCase("n"))

{

for(int i = -1;i<2;i+=2)//checks Knight direction

for(int j = -1;j<2;j+=2)

for(int k = 0;k<2;k++)

if(checkBounds(r+(i\*(k+1)),c+(j\*(2-k))))

return false;

}

else if(array[r][c].getPieceType().equals("p") && co.equals("b"))

{

for(int i = -1; i<2; i+=2)

if(!checkOpen(r+1,c+i))

return false;

if(checkOpen(r+1,c))

return false;

}

else if(array[r][c].getPieceType().equals("P") && co.equals("w"))

{

for(int i = -1; i<2; i+=2)

if(!checkOpen(r-1,c+i))

return false;

if(checkOpen(r-1,c))

return false;

}

}

return true;

}

public boolean checkmate(String co)

{

for(int r = 0; r<array.length; r++)

for(int c = 0; c<array[r].length; c++)

if(!(checkOpen(r,c) || array[r][c].getColor().equals(co)))

{

if(array[r][c].getPieceType().equalsIgnoreCase("r") || array[r][c].getPieceType().equalsIgnoreCase("q"))

{

for(int k = 0;k<2;k++)//checks Rook directions

for(int i = -1;i<2;i+=2)

for(int j = 1;j<7;j++)

if(checkBounds(r+(j\*i\*k),c+(j\*i\*(1-k))))

{

if(!checkOpen(r+(j\*i\*k),c+(j\*i\*(1-k))))

j=7;

if(!array[r][c].checkForCheck(r+(j\*i\*k),c+(j\*i\*(1-k)), co))

return false;

}

else

j=7;

}

else if(array[r][c].getPieceType().equalsIgnoreCase("b") || array[r][c].getPieceType().equalsIgnoreCase("q"))

{

for(int k = -1;k<2;k+=2)//check Bishop directions

for(int i = -1;i<2;i+=2)

for(int j = 1;j<7;j++)

if(checkBounds(r+(k\*j),c+(j\*i)))

{

if(!checkOpen(r+(j\*i\*k),c+(j\*i\*(1-k))))

j=7;

if(!array[r][c].checkForCheck(r+(k\*j),c+(j\*i),co))

return false;

}

else

j=7;

}

else if(array[r][c].getPieceType().equalsIgnoreCase("n"))

{

for(int i = -1;i<2;i+=2)//checks Knight direction

for(int j = -1;j<2;j+=2)

for(int k = 0;k<2;k++)

if(checkBounds(r+(i\*(k+1)),c+(j\*(2-k))))

if(!array[r][c].checkForCheck(r+(i\*(k+1)),c+(j\*(2-k)),co))

return false;

}

else if(array[r][c].getPieceType().equals("p") && co.equals("b"))

{

for(int i = -1; i<2; i+=2)

if(!(checkOpen(r+1,c+i) || array[r][c].checkForCheck(r+1,c+i,co)))

return false;

if(checkOpen(r+1,c) && !array[r][c].checkForCheck(r+1,c,co))

return false;

}

else if(array[r][c].getPieceType().equals("P") && co.equals("w"))

{

for(int i = -1; i<2; i+=2)

if(!(checkOpen(r-1,c+i) || array[r][c].checkForCheck(r-1,c+i,co)))

return false;

if(checkOpen(r-1,c) && !array[r][c].checkForCheck(r-1,c,co))

return false;

}

}

return true;

}

// -----------------------------------------------------------------------------------------------------------------------------

//Methods for points

public void score(Piece piece) //Jon and Sherwin code

{ //score(board[r][c])

//determine whether player1 is white or black

if(Multiplayer.player1Color.equals("white"))//playerOne is white

{

switch(piece.getPieceType())

{

//Capture White by Player 2

case "P": p2Score++; break;

case "N": p2Score += 3; break;

case "B": p2Score += 3; break;

case "R": p2Score += 5; break;

case "Q": p2Score += 9; break;

//Capture Black by Player 1

case "p": p1Score++; break;

case "n": p1Score += 3; break;

case "b": p1Score += 3; break;

case "r": p1Score += 5; break;

case "q": p1Score += 9; break;

default: p1Score += 0;

}

}

else //playerOne is black

{

switch(piece.getPieceType())

{

//Capture White by Player 1

case "P": p1Score++; break;

case "N": p1Score += 3; break;

case "B": p1Score += 3; break;

case "R": p1Score += 5; break;

case "Q": p1Score += 9; break;

//Capture Black by Player 2

case "p": p2Score++; break;

case "n": p2Score += 3; break;

case "b": p2Score += 3; break;

case "r": p2Score += 5; break;

case "q": p2Score += 9; break;

default: p1Score += 0;

}

}

}

public int getPlayerOneScores()

{

return p1Score;

}

public int getPlayerTwoScores()

{

return p2Score;

}

}