/\*FALL 2021 CS 2300 004

\* Brett Ford

\* Assignment 2

\*

\*/

import java.io.\*;

import java.util.\*;

import java.math.\*;

public class CS2300P2BrettFord {

public static void main(String[] args) throws FileNotFoundException {

String location = "p2-2.txt";//the name of the file

//these lines prepare to open the file

File fileName = new File(location);

Scanner readFile = new Scanner(fileName);

int size=readFile.nextInt();//these lines read the size of the playboard and the number of previous turns to compare against

int K=readFile.nextInt();

readFile.nextLine();

int sR,sC,eR,eC;//these briefly hold the values while the text is being read

int player = 1;//this maintains what player is being read

Matrix myBoard=new Matrix(size,K);//this structure holds the gameboard

/\*the game must meet three criteria to continue playing

\* 1. there are moves in the file to be played

\* 2. there have not been two consecutive invalid moves

\* 3. the board has not been filled(win)

\*/

//this loop reads the text document

while((readFile.hasNextLine())&&(Matrix.getInvalidMoves()<2)&&(Matrix.hasWon()==false)) {

System.out.println("player "+player+", take your move.");

//these read the file, 1 is subtracted to make them index to zero

sR=readFile.nextInt()-1;

sC=readFile.nextInt()-1;

System.out.print(sR+","+sC+" to ");

eR=readFile.nextInt()-1;

eC=readFile.nextInt()-1;

System.out.println(eR+","+eC);

readFile.nextLine();

//this calls the methods that will play the game

myBoard.takeMove(player,sR,sC,eR,eC);

//this toggles the player turns

if(player==1) {

player=2;

}else {

player=1;

}

}

//this closes the file

readFile.close();

//if there are two invalid moves in a row, the game ends

if (Matrix.getInvalidMoves()>=2) {

System.out.println("that is two invalid moves in a row");

}

//at the end of the game, these two lines print the score

System.out.println("player 1 scored: "+Matrix.calculatePoints(1)+" points");

System.out.println("player 2 scored: "+Matrix.calculatePoints(2)+" points");

}//end of main

}//end of class

class Matrix{

static char[][] gameboard;//this matrix stores the game

private static int N;//this stores the size of the gameboard

private int K;//this determines how long a record of previous moves should be kept

static int invalidMoves=0;//this stores how many invalid moves have been played. if it reaches 2 the game should end.

ArrayList<TurnHistory> record=new ArrayList<TurnHistory>();//this arraylist stores a record of previous moves as a TrunHistory object

//this generates a game board to play the game on, and serves as a constructor..

public Matrix(int N,int K) {

Matrix.gameboard = new char[N][N];

Matrix.N=N;

this.K=K;

for(int c = 0 ; c<N ; c++) {

for(int r=0 ; r<N ; r++) {

//the gameboard is initially filled with underscores to represent empty squares

gameboard[r][c]='\_';

}

System.out.print("\n");

}

System.out.print("game is ready\n");

}//end of constructor

//this prints the entire gameboard

public void printBoard() {

for(int c = 0 ; c<N ; c++) {

for(int r=0 ; r<N ; r++) {

System.out.print(gameboard[r][c]+" ");

}

System.out.print("\n");

}

System.out.print("\n");

}//end of printBoard();

//this method is used for each move,it calls other functions to do most of its work

public void takeMove(int player, int sCol,int sRow,int eCol, int eRow){

record.add(new TurnHistory(sCol,sRow,eCol,eRow));

//if the play is valid, add the line to the board

if(checkValid()==true) {

makeLine(player);

}

//this maintaines the list of moves, It only examines moves that are k turns ago.

if(record.size()>K) {

record.remove(0);

}

//the slope and the current gameboard are displayed each turn

System.out.println("your slope was "+record.get(record.size()-1).getAngle());

printBoard();

}//end of takeMove()

/\*this method checks if a move is valid, to be valid it must meet three criteria

\* 1. it must not begin or end on a space that another line began or ended on in the past k turns

\* 2. it can't begin and end on the same square

\* 3. it must not be perpendicular to a pre-existing line in the past k turns\*/

public Boolean checkValid() {

int size = record.size();

//does it represent a line

if(size>1) {

if((record.get(size-1).getsRow()==record.get(size-1).geteRow())&&(record.get(size-1).getsCol()==record.get(size-1).geteCol())){

System.out.println("your line begins and ends at the same point");

return false;

}

for(int i = 0;i<record.size()-2;i++) {

//does it begin or end on the same square as a pre-existing line

if((record.get(size-1).getsRow()==record.get(i).getsRow())&&(record.get(size-1).getsCol()==record.get(i).getsCol())){

System.out.println("Your line begins at the same location as another line began");

invalidMoves++;

return false;

}else if((record.get(size-1).geteRow()==record.get(i).geteRow())&&(record.get(size-1).geteCol()==record.get(i).geteCol())) {

System.out.println("Your line ends at the same location as another line ends");

invalidMoves++;

return false;

}else if((record.get(size-1).getsRow()==record.get(i).getsRow())&&(record.get(size-1).geteCol()==record.get(i).geteCol())) {

System.out.println("Your line begins at the same location as another ends ends");

invalidMoves++;

return false;

}else if((record.get(size-1).geteRow()==record.get(i).geteRow())&&(record.get(size-1).getsCol()==record.get(i).getsCol())) {

System.out.println("Your ends at the same location as another line began");

invalidMoves++;

return false;

//is it perpendicular to another line

}else if(Math.round( (record.get(size-1).getAngle()-record.get(i).getInverseAngle() )\*10 )==0) {

//this if checks if two slopes are perpendicular,

System.out.println("That line is perpindicular to an already existing line. ");

System.out.println("your lines slope was "+record.get(size-1).getAngle()+", your opponents was "+record.get(i).getAngle());

invalidMoves++;

return false;

}

}

}//end of if

//if the move is legal, the invalid turn counter is reset and the method returns true so the move may be played

invalidMoves=0;

return true;

}//end of checkValue

//this method returns true if the game has been won.

public static boolean hasWon() {

for(int c = 0 ; c<N ; c++) {

for(int r=0 ; r<N ; r++) {

//the game is won if there are no underscores( blank spaces) on the board

if(Character.compare(gameboard[c][r],'\_')==0) {

return false;

}

}

}

return true;

}//end of hasWon()

//this getter is used to keep track of how many invalid moves were made

public static int getInvalidMoves() {

return invalidMoves;

}

//this method determines how many points are on the board

public static int calculatePoints(int player) {

int playerscore =0;

char check;

//it looks for different symbold based on its arguments

if (player==1)

check='X';

else

check='O';

for(int c = 0 ; c<N ; c++) {

for(int r=0 ; r<N ; r++) {

//the game is won if there are no underscores( blank spaces) on the board

if(Character.compare(gameboard[c][r],check)==0) {

playerscore++;

}

}

}

return playerscore;

}//end of calculate points

//this method adds symbols to the playerboard.

private void placeOnBoard(int player,int sRow,int sCol){

if(player==1){

Matrix.gameboard[sRow][sCol]='X';

}else if (player==2) {

Matrix.gameboard[sRow][sCol]='O';

}

}//end of placeOnBoard();

//this function calculates the distance of each square on the gameboard, and places a mark if the line passes through it.

public void makeLine(int player) {

int size = record.size();

double StartingRow=record.get(size-1).getsRow();

double StartingCol=record.get(size-1).getsCol();

double endingRow=record.get(size-1).geteRow();

double endingCol=record.get(size-1).geteCol();

//these equation can be found in the textbook under the section 3.3

double a=-(endingCol-StartingCol);

double b=endingRow-StartingRow;

double c=-a\*StartingRow-b\*StartingCol;

double magnitudeA=Math.sqrt(a\*a+b\*b);

double f;

//each square in the compared is tested for its distance from the line, if it is closer than (sqrt2)/2, it is added

for(int row = 0 ; row<N ; row++) {

for(int col=0; col<N ; col++) {

f=a\*row+b\*col+c;

if(Math.abs(f/magnitudeA)<.707) {

placeOnBoard(player,col,row);

}

}

}

}//end of makeLine()

}

//this class records a record of the past k turns to verify if new moves are legal. It is used heavily in the checkValid(). It records 6 pieces of data for each turn, the starting and ending column, the angle of the line, and the inverse angle.

class TurnHistory{

int sRow,sCol,eRow,eCol;

double angle,inverseAngle;

public TurnHistory(int sRow, int sCol, int eRow, int eCol) {

this.sRow = sRow;

this.sCol = sCol;

this.eRow = eRow;

this.eCol = eCol;

//if the line is vertical

if(eCol==sCol) {

this.angle=10000000;//infinite angles are set to this number, they need to be rounded later, so this works fine

this.inverseAngle=0;

//if the line is horizontal

}else if(eRow==sRow){

this.angle=0;

this.inverseAngle=10000000;

//if the line is not orthogonal on the board

}else {

this.angle = (((double)eRow-(double)sRow)/((double)eCol-(double)sCol));

this.inverseAngle=-(((double)eCol-(double)sCol)/((double)eRow-(double)sRow));

}

}//end of constructor

public int getsRow() {

return sRow;

}

public int getsCol() {

return sCol;

}

public int geteRow() {

return eRow;

}

public int geteCol() {

return eCol;

}

public double getAngle() {

return angle;

}

public double getInverseAngle() {

return inverseAngle;

}

}