OS2 :Make a square

Project Guidelines:

Make a square with size 4X4 by using 4 or 5 pieces. The pieces can be rotated and all pieces should be used to form a square. Example sets of pieces.

There may be more than one possible solution for a set of pieces, and not every arrangement will work even with a set for which a solution can be found. Examples using the above set of pieces... Rotate piece D 90 degree then flip horizontal {R 90 + F H}

Input:

The first line contains number of pieces. Each piece is then specified by listing a single line with two integers, the number of rows and columns in the piece, followed by one or more lines which specify the shape of the piece. The shape specification consists of 0 or 1 characters, with the 1 character indicating the solid shape of the puzzle (the 0 characters are merely placeholders). For example, piece A above would be specified as follows:

2 3

111

101

Output:

Your program should report all solution, in the format shown by the examples below. A 4-row by 4-column square should be created, with each piece occupying its location in the solution. The solid portions of piece #1 should be replaced with `1' characters, of piece #2 with `2' characters.

Sample output that represents the figure above could be:

1112

1412

3422

3442

For cases which have no possible solution simply report "No solution possible".

Description:

We made a 2D array to specify each piece, we used object oriented programming (classes,inheritance,..etc) and used java multithreading to implement the project.

Our idea : we have number of thread that assumed as the sum of all pieces' threads . if we have 10 thread then we have 10 solutions. All threads running in parallel so that when one of the threads find the solution it interrupts the rest of the threads.

Each piece rotate 90 degrees in threads and put all input pieces together in the board and try all possible solutions . All pieces represented by a binary numbers (0,1).

Team members:

Mohammed Mossad

Karim Amr

Mennatullah Essam

Yara Mohammed

Nadia Talaat

Mina Nady

Code :

1-Pieces:

1.1 Piece class and constructor :

Each piece is defined by the number of rows and columns of a 2D array which are passed to the constructor :

```java
package make_a_square;
public class Piece {
    private int row , col;
    private int[][] piece;

    public Piece(int[][] piece){
        this.row = piece.length;
        this.col = piece[0].length;
        this.piece = piece;
    }

    public int[][] getPiece(){
        return this.piece.clone();
    }
```

1.2 Rotation Methods:

We made 4 private methods to rotate each piece 90 ,180,270 deg or keep it as original state :

```java
    private int[][] rotation0() {
        return this.piece.clone();
    }
```

```java
private int[][] rotation1() {
    int id = this.row - 1;
    int[][] newPiece = new int[col][row];
    for (int i = 0; i < this.row; i++) {
        for (int j = 0; j < this.col; j++) {
            newPiece[j][id] = this.piece[i][j];
        }
        id--;
    }
    return newPiece;
}
private int[][] rotation2() {
    int[][] newPiece = new int[row][col];
    for (int i = 0; i < this.row; i++) {
        for (int j = 0; j < this.col; j++) {
            int t = this.col - (j + 1);
            int r = this.row - (i + 1);
            newPiece[r][t] = this.piece[i][j];
        }
    }
    return newPiece;
}
private int[][] rotation3() {
    int[][] newPiece = new int[col][row];
    for (int i = 0; i < this.row; i++) {
        for (int j = 0; j < this.col; j++) {
            int t = this.col - (j + 1);
            newPiece[t][i] = this.piece[i][j];
        }
    }
    return newPiece;
}
```

1.3 getPieceAfterRotation():

this method takes rotation id and then get the new piece

```java
public int[][] getPieceAfterRotation(int rotationID) {
    switch (rotationID) {
        case 0:
            return this.rotation0();
        case 1:
            return this.rotation1();
        case 2:
            return this.rotation2();
        case 3:
            return this.rotation3();
    }
    throw new IndexOutOfBoundsException();
}
```

2- Board :

2.1 Board class and constructor:

We created a board class and initialized our board (grid), specified its dimensions and we created a HashMap for pieces on the board

```
package make_a_square;
import java.util.HashMap;
import java.util.Map;
public class Board {
    private final int[][] grid = new int[GameContent.gridRows][GameContent.gridCols];
    public final int sizeX = GameContent.gridRows;
    public final int sizeY = GameContent.gridCols;
    Map<Integer, int[][]> pieces = new HashMap<>();

    public Board(Map<Integer, int[][]> pieces) {
        this.pieces = pieces;
        for(int i = 0; i < GameContent.gridRows; i++){
            for(int j = 0; j < GameContent.gridCols; j++){
                grid[i][j] = -1;
            }
        }
    }
```

2.2 getPieces() :

get pieces method returns 2D array that specifies the id and the move of each piece used in a specific combination.

```
    private int[][] getPieces(int numericState) {
        int[][] arr2D = new int[pieces.size()][2];
        StringBuilder sB = convertDecToBinary( num: numericState , pieces: pieces.size());
           for (int i = 0; i < pieces.size(); i++) {
               try{
               arr2D[i][1] = Integer.parseInt( s: sB.substring(i * 5, i * 5 + 2), radix: 2); //moves
               arr2D[i][0] = Integer.parseInt( s: sB.substring(i * 5 + 2, i * 5 + 5), radix: 2); //id
               }catch (Exception exception)
               {
                   System.out.println( x: exception.toString());
               }
           }
        return arr2D;
    }
```

2.3 IsValidSeq() :

We used isValidSeq method to check if the sequence given to the board is valid by checking if the given sequence satisfy the main conditions:

1- No two pieces have the same id
2- Id of the move is less than number of moves
3- Pieces is less than number of pieces we have

```
private boolean IsValidSeq(int[][] sequence) {
    /*
    to be a valid sequence we should
    1 - no two pieces are equal in id.
    2 - id of the move < numberOfMoves
    3 - pieces < numberOfPieces we have
    */
    for (int i = 0; i < sequence.length; i++) {
        boolean pieceValid = (sequence[i][0] < pieces.size());
        boolean moveIValid = sequence[i][1] < GameContent.numberOfRotation;

        for (int j = i + 1; j < sequence.length; j++) {
            boolean pieceIUnique = sequence[i][0] != sequence[j][0];
            if(!pieceIUnique)
                return false;
            }

        if(!pieceValid || !moveIValid)
            return false;
    }
    return true;
}
```

2.4 Rotation():

Returns piece after rotation.

```
private int[][] Rotation(int moveId, int[][] piece) {
    Piece piec = new Piece(piece);
    return piec.getPieceAfterRotation( rotationID: moveId);
}
```

2.5 FirstEmptyCeilInBoard() :

returns the index of first empty cell in the grid.

```
private int[] FirstEmptyCeilInBoard(int[][] ceils) {
    int[] indx = {-1 , -1};
    // give me the indx of the firt empty ceil
    for (int r = 0; r < ceils.length; r++) {
        boolean b = false;
        for (int c = 0; c < ceils[0].length; c++) {
            if (ceils[r][c] == -1) {
                indx[0] = r;
                indx[1] = c;
                b = true;
                break;
            }
        }
        if (b) {
            break;
        }
    }
    return indx;
}
```

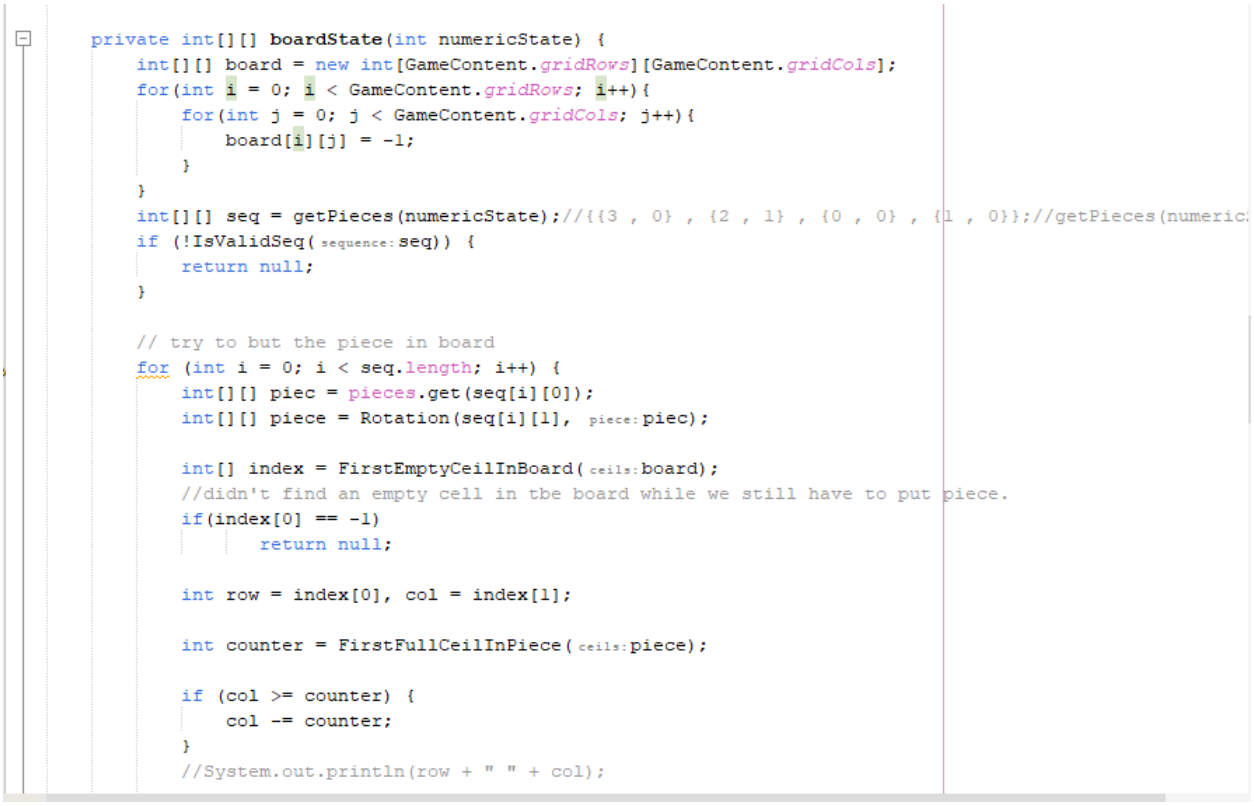
2.6 FirstFullCeilInPiece() :

returns the location of the first solid part in the piece that consists of 1 .

```
private int FirstFullCeilInPiece(int[][] ceils) {
    int counter = 0;
    for (int j = 0; j < ceils[0].length; j++) {
        if (ceils[0][j] == 1) {
            break;
        }
        counter++;
    }
    return counter;
}
```

2.7 boardState():

This is the method where all other methods glued together, it takes the sequence of pieces by getPieces(), make sure it is valid by IsValidSeq() and try to put those pieces on the board by trying the possible rotations of the piece by Rotation(), it finds the first empty cell In board in the upper right corner to place the piece in by findFirstEmptyCeilInBoard() and make sure we find empty cells while we still have pieces to place, find first solid part of the piece by findFirstFullCeilInPiece() and at last it puts the pieces on the board and returns the board



```
private int[][] boardState(int numericState) {
    int[][] board = new int[GameContent.gridRows][GameContent.gridCols];
    for(int i = 0; i < GameContent.gridRows; i++){
        for(int j = 0; j < GameContent.gridCols; j++){
            board[i][j] = -1;
        }
    }
    int[][] seq = getPieces(numericState);//{{3 , 0} , {2 , 1} , {0 , 0} , {1 , 0}};//getPieces(numeric
    if (!IsValidSeq( sequence: seq)) {
        return null;
    }

    // try to but the piece in board
    for (int i = 0; i < seq.length; i++) {
        int[][] piec = pieces.get(seq[i][0]);
        int[][] piece = Rotation(seq[i][1],  piece: piec);

        int[] index = FirstEmptyCeilInBoard( ceils: board);
        //didn't find an empty cell in tbe board while we still have to put piece.
        if(index[0] == -1)
            return null;

        int row = index[0], col = index[1];

        int counter = FirstFullCeilInPiece( ceils: piece);

        if (col >= counter) {
            col -= counter;
        }
        //System.out.println(row + " " + col);
```

```
            for (int r = 0; r < piece.length; r++) {
                for (int c = 0; c < piece[0].length; c++) {
                    int ro = r + row, co = c + col;
                    if (ro >= 4 || co >= 4) {
                        return null;
                    } else if (board[ro][co] > -1 && piece[r][c] == 1) {
                        return null;
                    } else if (piece[r][c] == 1) {
                        board[ro][co] = seq[i][0];
                    }

                }
            }
        }
        return board;
    }
```

2.8 decompose():

It takes the numeric state of the boards, it returns the valid board and returns NULL if the board state is not valid

```
    public int[][] decompose(int numericState) {
        int[][] returnedBoard = boardState(numericState);
        if(returnedBoard == null)
            return null;
        if(isValidBoard( grid: returnedBoard))
            return returnedBoard;
        return null;
    }
```

2.9 convertDecToBinary():

It is used to represent the numeric state of the current combination by a binary number as a string to be used later to figure out the board state

```
public StringBuilder convertDecToBinary(int num,int pieces) {
    StringBuilder s = new StringBuilder();
    while (num != 0) {
        if (num % 2 == 0) {
            s.append( c: '0');
        } else {
            s.append( c: '1');
        }
        num /= 2;
    }
    int sz = s.length();
    for (int i = 0; i < 25 - sz; i++) {
        s.append( c: '0'); // 25 - 5
         }
    s.reverse();
    return s;
}
```

2.10 isValidBoard():

This method makes sure that the state board passed to it doesn't cross the boundaries of the grid, the grid must always be 4x4 square.

```
public boolean isValidBoard(int[][] grid) {
    for (int row = 0; row < sizeX; row++) {
        for (int column = 0; column < sizeY; column++) {
            if (grid[row][column] == -1) {
                return false;
            }
        }
    }

    return true;
}
```

2.11 setGrid():

For each position in the grid, it assigns the value from the corresponding position in the copyGrid array.

```
public void setGrid(int[][] copyGrid) {
    for (int row = 0; row < sizeX; row++) {
        for (int column = 0; column < sizeY; column++) {
            grid[row][column] = copyGrid[row][column];
        }
    }
}
```

3-Multithreading :

We used java runnable which is an interface to execute code on a concurrent thread, this interface is implemented by any class if we want the instances of that class to be executed by a thread.

We also used a ReentrantLock class which implements the lock interface and provides synchronization to methods while accessing shared resources. The code which manipulates the final board is surrounded by calls to lock and unlock method.

```java
package make_a_square;
import java.util.Map;
import java.util.concurrent.locks.ReentrantLock;

public class Multithreading implements Runnable {

    static boolean foundBoard;
    private final ReentrantLock lock;
    public static int[][] finallyBoard;
    static public Map<Integer, int[][]> allPieces;

    public Multithreading(){
        lock = new ReentrantLock();
    }

    @Override
    public void run() {
        int[][] finalBoard;
        int threadID = Integer.parseInt( s:Thread.currentThread().getName());

        int from = threadID * GameContent.sectionSize;
        int to = Math.min(from + GameContent.sectionSize - 1 , GameContent.numberOfBoards - 1);
        if(threadID == GameContent.numberOfThreads - 1)
            to = GameContent.numberOfBoards - 1;

        //last thread must complete to the end of the states.
        for(int mask = from; mask <= to; mask++){
            Board slaveBoard = new Board( pieces: allPieces);
//              slaveBoard.decompose(mask);
            finalBoard = slaveBoard.decompose( numericState: mask);
```

```
                if(foundBoard)
                    break;

                if(finalBoard != null){
                    lock.lock();
                    foundBoard = true;
                    finallyBoard = finalBoard;
                    lock.unlock();
                }
            }
        }
    }
```

4- Forms of pieces:

It performs each piece in binary in 2D array

```
package make_a_square;
public class formOfPieces {

    private int id;
    private int [][] matrix;
    private int count;
    private String textField;
    private static int [][][] pieces;
    //static
    public formOfPieces()
    {
        pieces = new int[7][][];
        pieces[0] = new int[][]{{1, 0}, {1, 0}, {1, 1}};
        pieces[1] = new int[][]{{1, 1, 0}, {0, 1, 1}};
        pieces[2] = new int[][]{{1}, {1}, {1}, {1}};
        pieces[3] = new int[][]{{0, 1}, {0, 1}, {1, 1}};
        pieces[4] = new int[][]{{1, 1, 1}, {0, 1, 0}, {0, 1, 0}};
        pieces[5] = new int[][]{{0, 1, 1}, {0, 1, 0}, {1, 1, 0}};
        pieces[6] = new int[][]{{1, 1}, {1, 1}};
    }
    int [][] retrievePiece(Character character)
    {
        if(character=='L') return pieces[0].clone();
        else  if(character=='Z') return pieces[1].clone();
        else  if(character=='I') return pieces[2].clone();
        else  if(character=='J') return pieces[3].clone();
        else  if(character=='T') return pieces[4].clone();
        else  if(character=='S') return pieces[5].clone();
        else  if(character=='O') return pieces[6].clone();
        throw new IndexOutOfBoundsException();
    }
}
```

5-Number of threads :

This function calculates the number of threads that will run in parallel.

```
public class makeNumberOfThreads {
    public static int[][] Square(HashMap<Integer, int[][]> sendPieces) throws InterruptedException {


        //pass pieces to the slaves area...
        Multithreading.allPieces = sendPieces;
        Multithreading.foundBoard = false;


        //preparing the slaves.
        Multithreading masterSlave = new Multithreading();
        ArrayList<Thread> slaves = new ArrayList<Thread>();
        for(int i = 0; i < GameContent.numberOfThreads; i++){
            Thread tmp = new Thread( r:masterSlave , string:Integer.toString(i));
            slaves.add( e:tmp);
        }

        //go slaves....
        for(int i = 0; i < GameContent.numberOfThreads; i++){
            slaves.get( index:i).start();
        }
        //wait slaves to finish...
        for(int i = 0; i < GameContent.numberOfThreads; i++){
            slaves.get( index:i).join();
        }

        if(Multithreading.foundBoard) return Multithreading.finallyBoard;
        else return null;

    }
}
```

6-Gui important functions:

6.1 solveProblem():

This function execute full code to get the result , It creates a new HashMap named sendPieces to store pieces retrieved from a piecesModel., iterates through each entry in the hashMap, retrieves the count of pieces for each character, and adds the corresponding pieces to sendPieces with unique IDs , calculates the total number of rotations needed based on the pieces in sendPieces , It uses a method (makeNumberOfThreads.Square) to generate a square board based on the pieces in sendPieces and check if the generated board is null, If so, it shows an information alert indicating that no solution was found. Otherwise, it calls the paintButton method.

```
int[][] finalBoard;
public void solveProblem() {
    setHashMap();
    HashMap<Integer, int[][]> sendPieces = new HashMap<>();
    int id = 0;
    for (Map.Entry<Character, Integer> set : hashMap.entrySet()) {
        int cnt = set.getValue();
        while (cnt > 0) {
            sendPieces.put(id++, value: piecesModel.retrievePiece( character: set.getKey()));
            cnt--;
        }
    }
    int totalRotations = calculateTotalRotations( userPieces: sendPieces);
    // Create an instance of GameContent and pass totalRotations to the constructor
    GameContent constants = new GameContent(totalRotations);
    try {
        finalBoard = makeNumberOfThreads.Square(sendPieces);
        if (finalBoard == null) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Message Here...");
            alert.setHeaderText("Make A Square");
            alert.setContentText("No Solution Founded");
            alert.showAndWait().ifPresent(rs -> {
                if (rs == ButtonType.OK) {
                    System.out.println( x: "Pressed OK.");
                }
            });
        } else {
            paintButton();
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
```

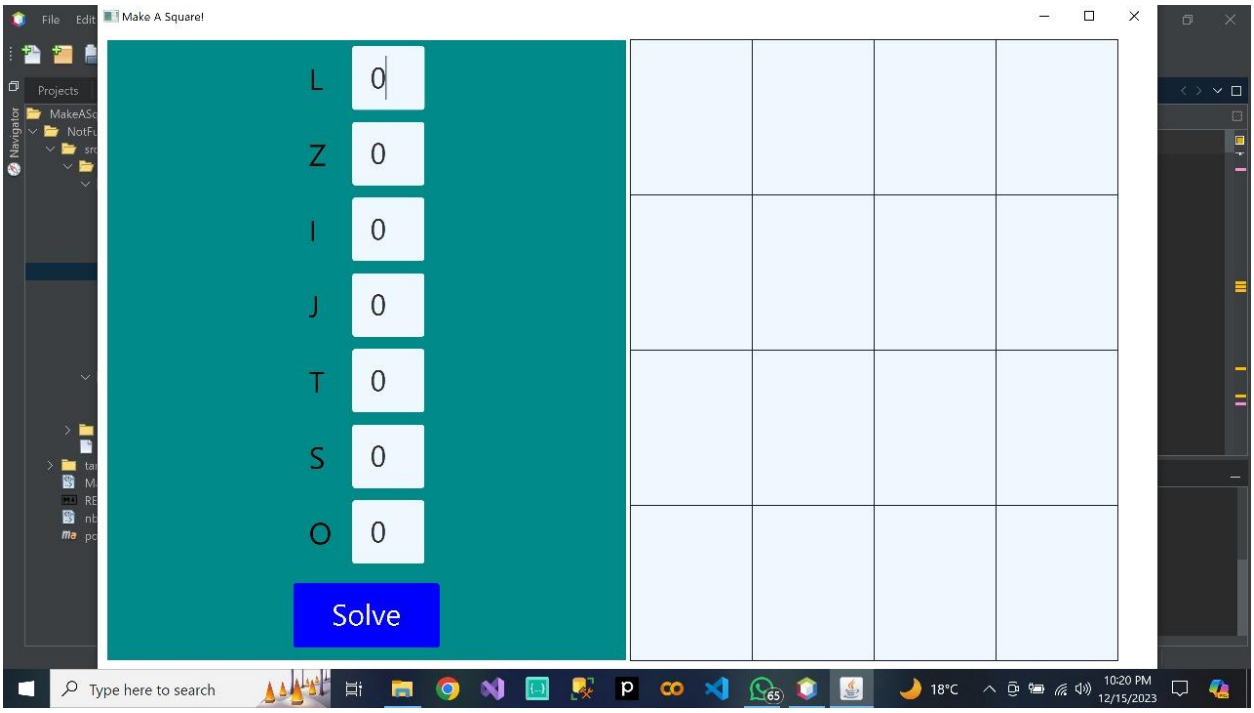
6.2 calculateTotalRotations() :

It Calculates the total number of rotations for the user input pieces

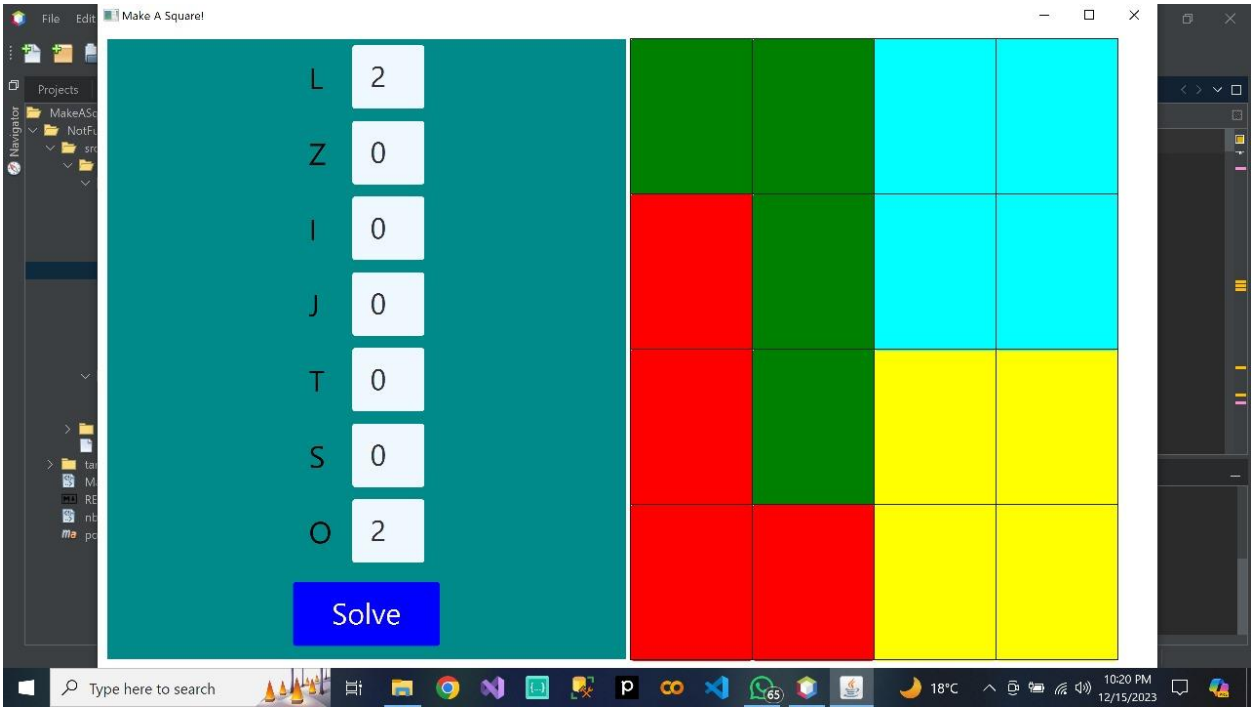
```
// Calculate the total number of rotations for the user input pieces
private int calculateTotalRotations(Map<Integer, int[][]> userPieces) {
    int totalRotations = 0;

    for (int[][] piece : userPieces.values()) {
        totalRotations += piece.length; // Assuming each piece has an array of rotations
    }

    return totalRotations;
}
```

Example:

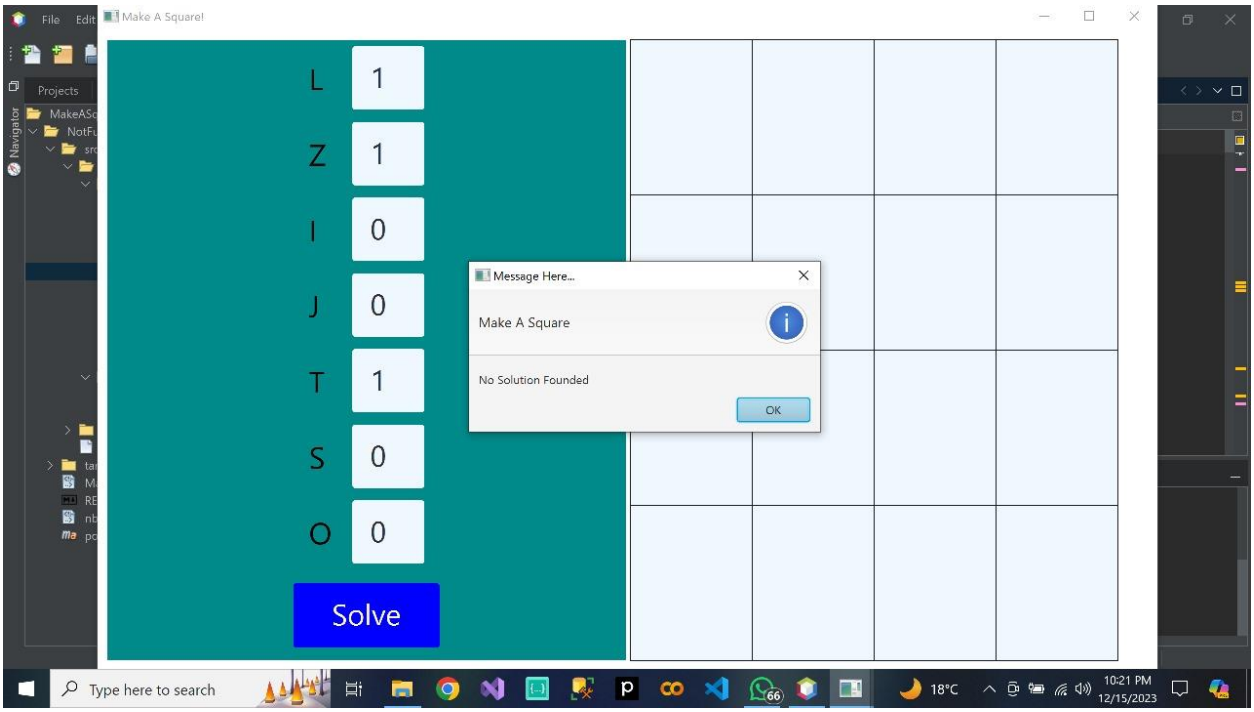
Before put input:



After:



If the program doesn’t find a way to combine the pieces, it prints no solution found!



Resources :

https://en.wikipedia.org/wiki/Tetromino

https://www.spoj.com/problems/PUZZLE/