Project 5: Make a square

description:

The task is to construct a 4x4 square using either 4 or 5 given pieces. Each piece can be rotated or flipped, and all pieces must be used to form the square. There might be multiple solutions for a set of pieces, and not all arrangements are valid. The pieces are represented as binary shapes. The input specifies the number of pieces and their shapes, while the output should present all possible solutions. In case no solution exists, "No solution possible" should be reported.

What we have actually did:

1. **Problem Statement:**
    - The goal is to create a 4x4 square using 4 or 5 pieces.
    - The pieces can be rotated or flipped.
    - All pieces must be used to form the square.
    - Multiple solutions may exist for a set of pieces.
2. **Implementation Approach:**
    - Used a 2D array to represent the pieces and the grid (square).
    - Utilized object-oriented programming (OOP) principles by creating a **Shapes** class to manage the pieces and rotations.
    - Implemented Java threads (**Multithreading** class) to handle the concurrent exploration of different possibilities.
    - The program divides the number of solutions among four threads running in parallel.

3. **Handling Possible Combinations:**
    - The code generates all possible combinations of pieces that can be used.

Team members roles:

- **Board** يوسف احمد - عمرو اشرف
- **Pieces:** يوسف احمد - مصطفى حسنى
- **Documentation:** نورهان محمد - هاجر محمد
- **Gui:** محمد عادل - احمد ممتاز - نورهان محمد
- **Testing the code:** هاجر محمد- احمد ممتاز - محمد عادل

Code documentation:

1. Shapes Class

- Represents the different Tetris shapes and their rotations.
- Contains a 4x4 grid (**arr**) to define the shapes.
- Provides a **pre()** method to initialize the grid with specific values.

```
public class Shapes {
    int[][][][] arr = new int[4][4][4][2];// num of shape, rotate, valid (x,y)
    int[][] grid = new int[4][4];

    public void pre() {
        for (int i = 0; i < 4; i++) {...}
        /* Shape 0 rotate 0 */
        arr[0][0][0][0] = 0;
        arr[0][0][0][1] = 0;
        arr[0][0][1][0] = 1;
        arr[0][0][1][1] = 0;
        arr[0][0][2][0] = 2;
        arr[0][0][2][1] = 0;
        arr[0][0][3][0] = 2;
        arr[0][0][3][1] = -1;

        /* Shape 0 rotate 1 */
        arr[0][1][0][0] = 0;
        arr[0][1][0][1] = 0;
        arr[0][1][1][0] = 1;
        arr[0][1][1][1] = 0;
        arr[0][1][2][0] = 1;
        arr[0][1][2][1] = 1;
        arr[0][1][3][0] = 1;
```

2. outer Class

- Contains a static nested class **Multithreading** that implements the **Runnable** interface.
- Manages a **Semaphore** (**sem**) and a **boolean** (**check**) to coordinate multithreading.

```
import java.util.concurrent.Semaphore;

class outer {
    static boolean check = false;
    static Semaphore sem = new Semaphore( permits: 1);
    static class Multithreading implements Runnable {
        boolean f = true;
        int num_of_shape;
        int[][] grid;
        int[] choose;
        int[][][][] arr;
        int rotate;
        int id;

        public Multithreading(int num_of_shape, int rotate, int[][] grid, int[] choose, int[][][][] arr, int id) {...}

        boolean valid(int x, int y, int i, int k) {...}

        void put(int i, int r) throws InterruptedException {...}

        void remove(int[][] loc) {...}

        public void run() {...}
    }
}
```

```
void put(int i, int r) throws InterruptedException { // shape
    if (!f) {
        return;
    }
    int tetris = choose[i];
    int[][] loc = new int[4][2];

    for (int x = 0; x < 4; x++) {
        for (int y = 0; y < 4; y++) {
            if (valid(x, y, tetris, r)) {
                for (int j = 0; j < 4; j++) {
                    int pos_x = x + arr[tetris][r][j][0], pos_y = y + arr[tetris][r][j][1];
                    grid[pos_x][pos_y] = i;
                    loc[j][0] = pos_x;
                    loc[j][1] = pos_y;
                }
                if (i == 3) {...} else {
                    for (int k = 0; (choose[i + 1] <= 1 && k < 4) || k < 2; k++) {
                        put( i: i + 1, k);
                        if (!f) return;
                    }
                }
                remove(loc);
```

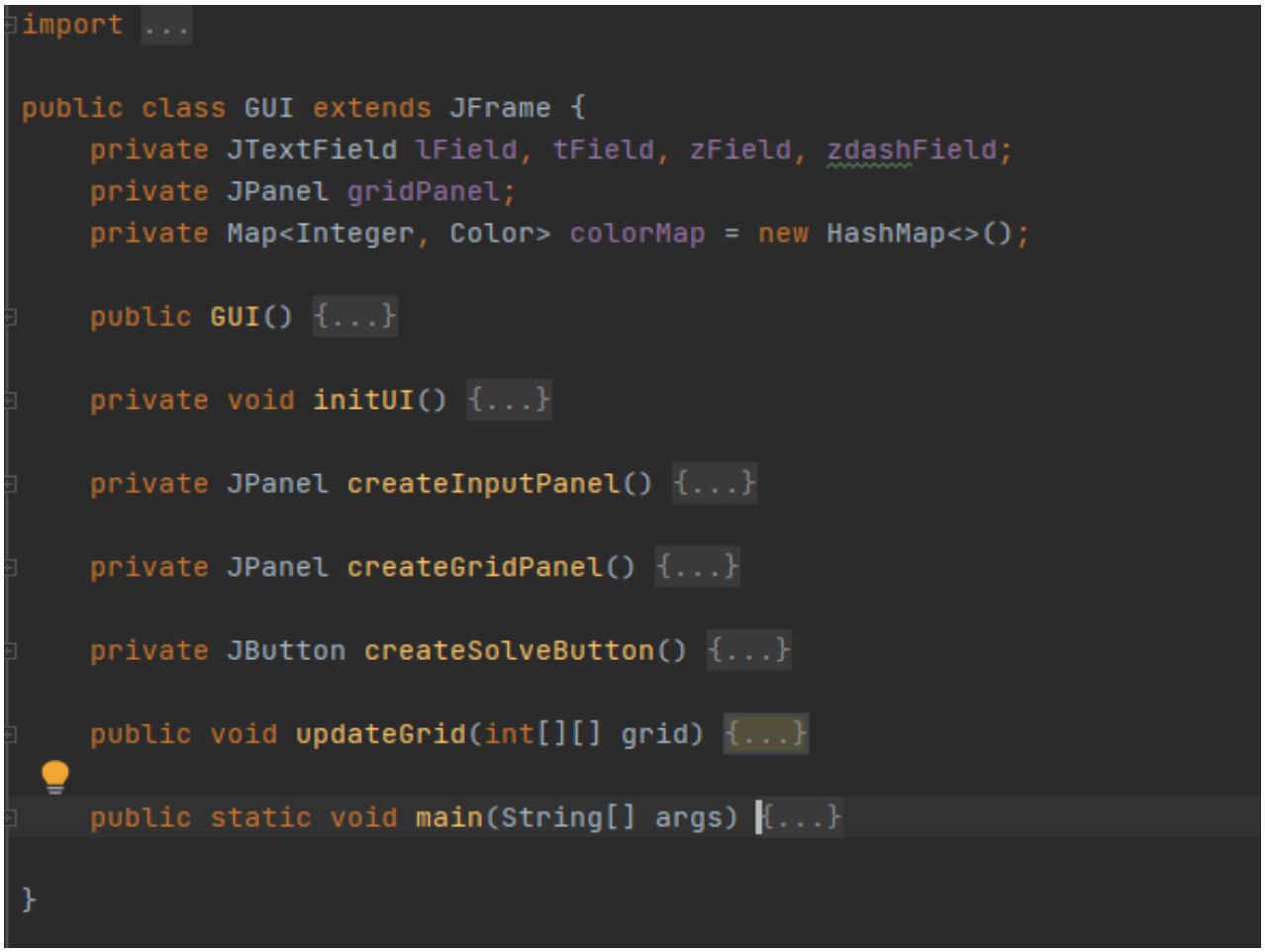
```
void remove(int[][] loc) {
    for (int i = 0; i < 4; i++) {
        grid[loc[i][0]][loc[i][1]] = -1;
    }
}
```

3. Multithreading Class

- Implements the **Runnable** interface to define a thread for handling Tetris shapes.
- Uses a recursive backtracking approach to try placing Tetris shapes on the grid.
- Acquires and releases a semaphore to ensure thread safety when updating the grid.
- Prints the final grid if the solution is found.

4. GUI Class

- Represents the graphical user interface for the game.
- Includes input fields for specifying the quantity of each Tetris shape.
- Displays a grid of buttons to represent the game board.
- Provides a "Solve" button to start the solving process using multithreading.
- Uses Swing for creating the GUI elements.



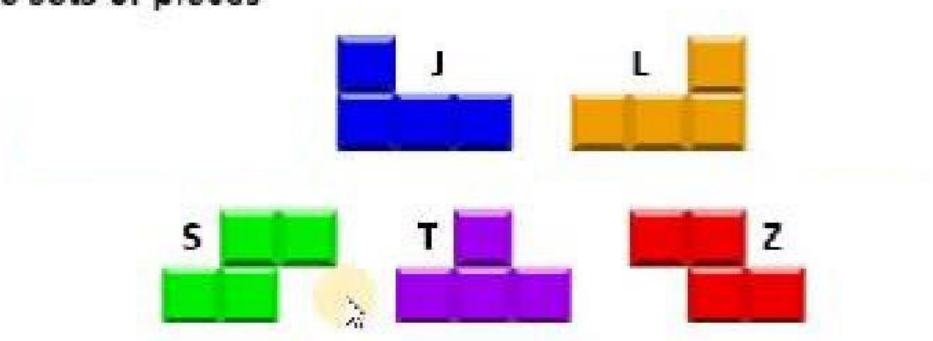
5. Main Class

- Takes user input for the quantity of each Tetris shape.
- Creates instances of **Shapes**, **Multithreading**, and **outer** classes.
- Starts multiple threads to find a solution using the **Multithreading** class.
- Prints the solution if found; otherwise, displays a message.

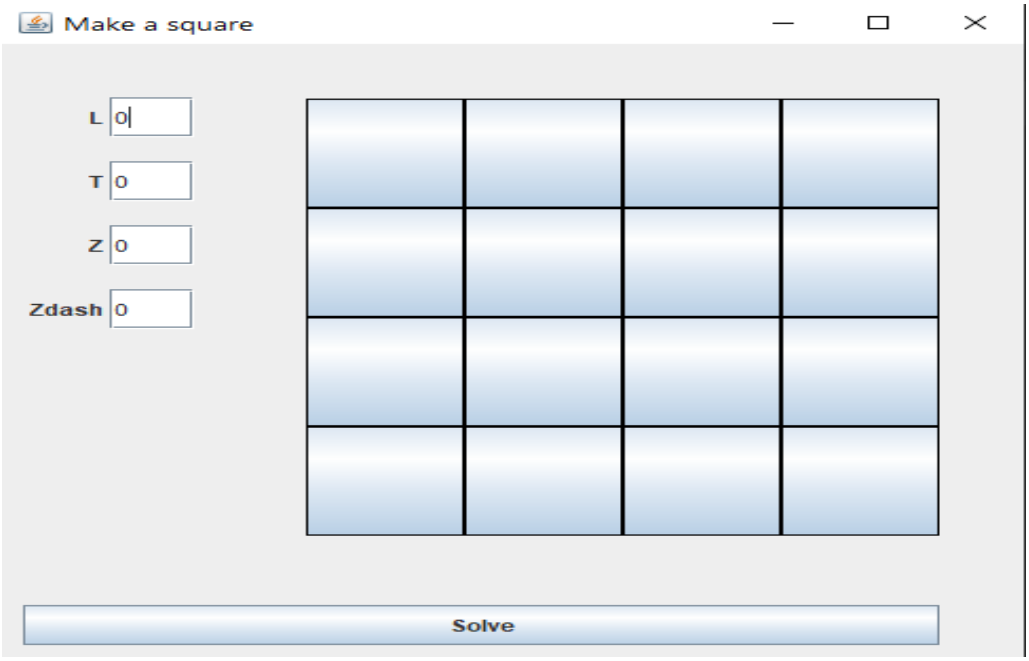
```java
import java.util.Scanner;
import java.util.concurrent.Semaphore;

public class Main {
    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);
        System.out.print("Choose 4 shapes : ");
        int[] arr = new int[4];
        for(int i=0;i<4;i++){...}
        int n;
        if(arr[0] <= 1) n=4;
        else n=2;
        outer.Multithreading[] thread = new outer.Multithreading[n];
        Thread[] proc = new Thread[n];
        for(int i=0;i<n;i++){...}
        for(int i=0;i<n;i++){...}
        if(!outer.check) System.out.println("NO RESULT");
    }
}
```

Graphical user interface:

- Example sets of pieces



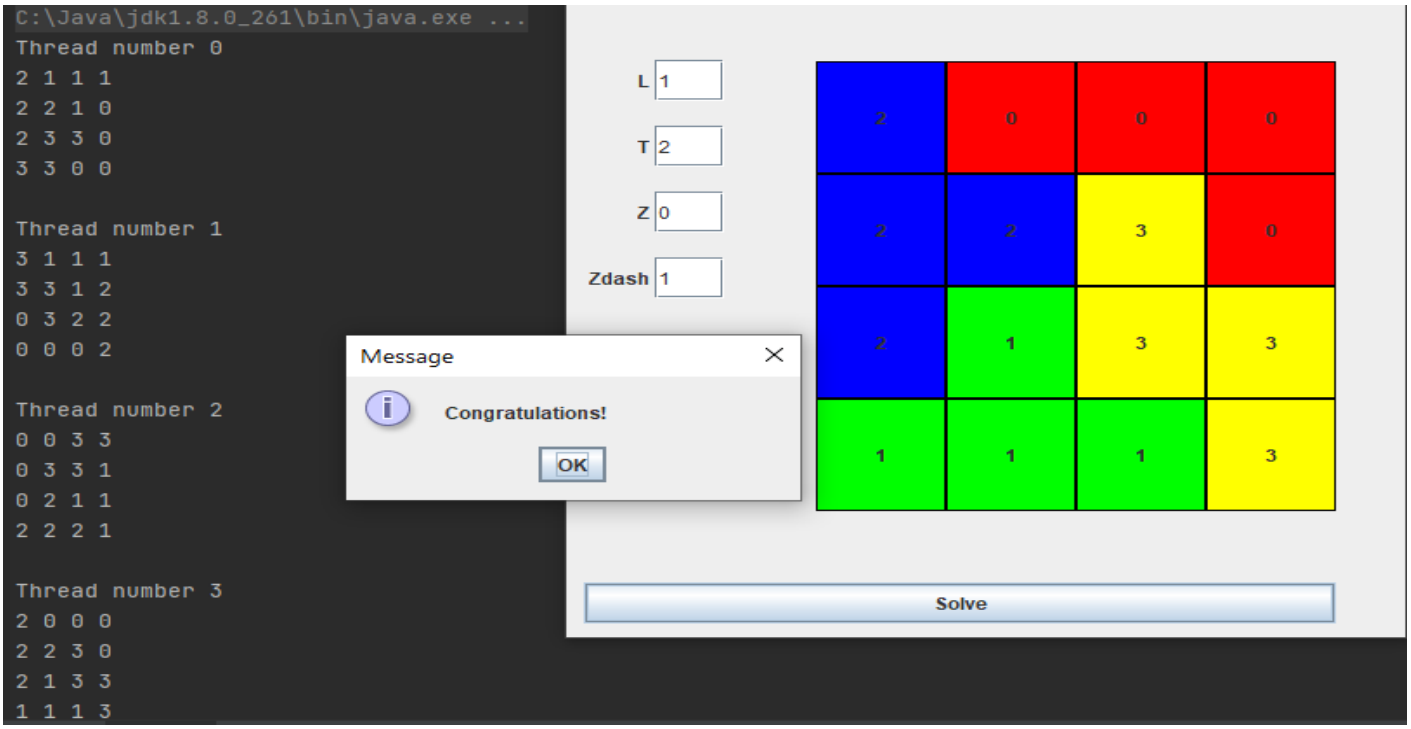
This is an example of pieces we have used in our project, each piece is given to the program by writing how many pieces of the same shape we need in the text box beside the shape of the piece in the GUI.



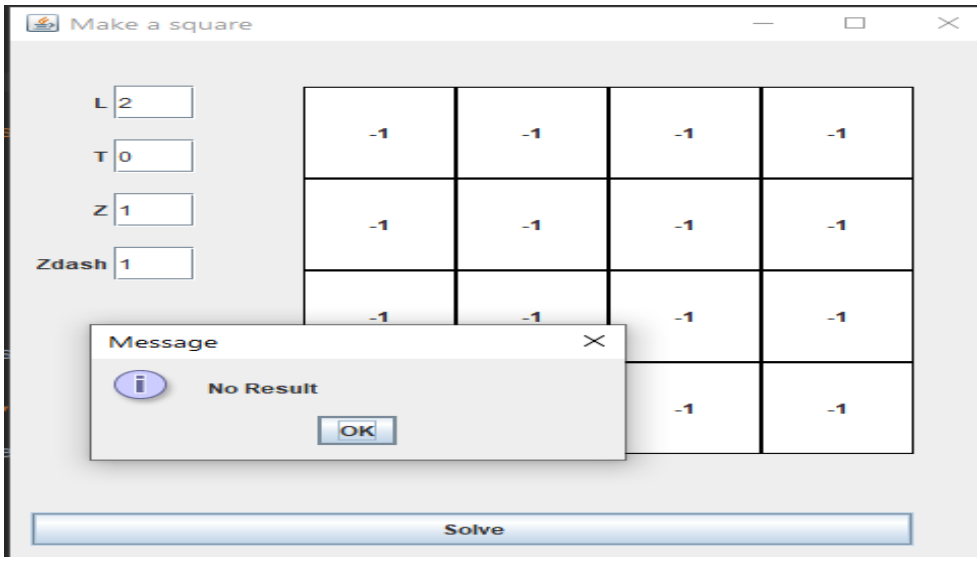
We tried many samples in the project, by giving it different pieces and it finds the way to combine those pieces.
if there's a possible combination to the pieces, it will be drawn on the gui as we have shown above.
if the program doesn't find a way to combine the pieces, it prints no solution found!

Sample Input and Output:

ENTER **1L 2T 1 Zdash**



Faild solution: print No Result.



Thanks