Project Number: 5

Project Name: Make a square

Team Members:

|  |  |  |
| --- | --- | --- |
| Grade | Team member name (**in Arabic**) | Team Member ID |
|  | طارق مصطفى اسماعيل محمد | 201900387 |
|  | سهيله محسن سعد احمد | 201900357 |
|  | عبد الرحمن عماد عبدالله السيد | 201900425 |
|  | احمد سمير حشمت حافظ | 201900040 |
|  | احمد محمد احمد عثمان | 201900085 |
|  |  |  |

**Evaluation Criteria**

General Criteria

|  |  |  |
| --- | --- | --- |
| Criteria | | Grade |
| **Multithreading (5)** | No multithreading ( 2 out of 5 ) | **…………………………………………………………………..** |
| Threads in serial ( 3 out of 5)  Correct usage of threads, and synchronization mechanisms |
| Multithreading (4 or 5 out of 5)  Correct usage of threads, and synchronization mechanisms |
| **GUI (2)** | No GUI (0 out of 2) | **…………………………………………………………………..** |
| GUI without thread communication or realtime update (1 out of 2) |
| GUI with correct I/O and Thread communication or realtime update   (2 out of 2) |
| **Documentation (1)** |  |  |
| **Understanding (2)** |  |  |

**OS2 Project 5 Documentation**

1. **Project Description:**

**(2) What you do (explanation of the all logic)**

Main logic:

Our solution depends on problem-solving techniques.

The main logic is we try all possible shapes of the piece and try those shapes in the square.

And we consider each trying as thread

The generated by rotating the original piece to 90˚ for the right side and taking the four shapes (original shape and three times rotated) and try to put each shape in the square if it putted successfully then we break trying and move to the next piece.

If all shapes cannot take place in the square we will flip the original shape and rotate it three times and keep trying if then shapes cannot take place then That is no solution

The logic of trying:  
the most important logic in project that take each shape and try to put it in square

First it search in square for empty place when this index founded, let’s imagine we cut from this index the size of shape and try to paste this shape in this cutting part if we able to do that we will return true and keep tring with other pieces.

------------------------------------------------------------------------------------

(3) Team member Rules:

Tarek Mostafa Esmail (201900387)

-Rotate & flip function

Abdelrahman Emad Abdallah (201900425)

-solve class

Sohaila Mohsen Saad (201900357)

-Run method

Ahmed Samir Hishmat (201900040)

-GUI

Ahmed Mohammed Ahmed (201900085)

-GUI

-------------------------------------------------------------------------

**(4) Code** **Doc (Explanation of code) :**

1)Samples class:

This class take the number of the sample that choosed by the user and append pieces in the list created at the start of the class then it returns this list.

2)Solve class:

This class is the main class it takes a square and the pieces list which choosed by the user

It also declares Boolean variable called test initially = false

Then it will check for some constraints:

The first constrain is to check the solid in the pieces if it did not equal 16( square size 4x4) then there is no solution

Then it will loop on the pieces list and check another constrain :

If the size of any piece > 4x4

Then there is no solution.

If the piece is fair to try it :

The class will rotate it three times (to create the different possible shapes of the piece)

And then will create a copy of the square

And start to separate the shapes into threads and start trying each shape to put it into the square, if we can’t put the first shape, we will try with the second shape and so on.

If all shapes return false, we will flip the original shape and rotate again and make those steps again.

If we flip and it doesn’t make difference, then there is no solution.

(if any shape fit in the square we will convert the test variable to true in RUN method and the converting will be checked at the end of each thread and if it is true we will break and move to the next piece “we don’t need to try all shapes if one of them fit” ).

At the end of all trying, we will return the square.

3)Rotating\_Fliping class:

In this class, we implement the rotate and flip functions.

1)Rotate function:

This function rotates the matrix (piece) to 90**˚** to the right side

It first makes a temp matrix by rotating the original one denominational (if the original 2x3 the temp will be 3x2)

And start to make each column in original to row in temp to the right side

Text, letter

Description automatically generated

That is what exactly the function does.

2)flip function:

This function also flips the matrix horizontally to the right side.

Making the temp matrix will same of original denominational and start to put the first column in original on the last one in the temp.

Text

Description automatically generated

The result of function must be exactly like that.

4)th1 class

This class is the thread class that implements the runnable interface.

And contain the main logic of the problem.

This class have two setters to set the piece and square and one getter to return the Boolean variable if the piece fits or not.

And contain the run method

The run method takes the piece and square and loop on the square until it finds an index containing ZERO and start from this index to paste the piece in a square

We can imagine we cut from this index the piece size and trying to put each index’s content in the piece in its equivalent to it in this cutting part

It checks if the value in the square’s index is not empty and the value in the piece’s index is also not empty then return false.

And check if the value in the square’s index is empty and the value in the piece’s index is also not empty then paste the value in this index and move to the next index.

It also checks if the square denominational is still in the square size range (<4), if not, it returns false.

If the function fit the piece in square successfully it will set the Boolean variable to true and return it

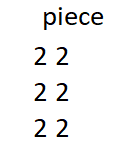


Fig 1

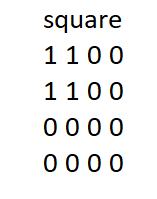


Fig 2

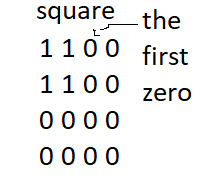


Fig 3

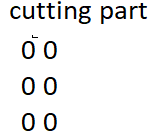


Fig 4

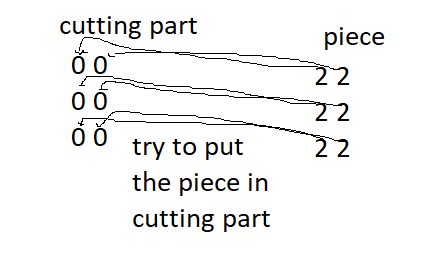


Fig 5

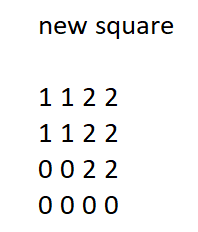


Fig 6