Hi, my version of the problem.

The project is made on the basis of Node.js

To establish dependencies (for Unit testing) unzip the file, go to it, open the console and write the command:

**npm install**

To start the project, write to the console:

**node main.js**

OR

**npm run start**

After that, if everything is done correctly, the project will start.

You will see logs for counting, and a conclusion as to whether you can build a wall or not

To enter your data, open the main.js file in the editor.

Create a new instance of the Wall class (width, height, matrix, countTypeBricks, combinations)

-width - the maximum number of elements in the width of the matrix

-height - the maximum number of elements in the height of the matrix

-matrix - a plan of what the wall should look like. It consists of 1 or 0. The height and width of the matrix must be equal to the width and height entered above.

-countTypeBricks - the number of types of bricks

-combinations - an array of brick combinations, consisting of 3 elements. The first and second element is the width and height of the brick, the third element is the quantity in the warehouse The number of items must not exceed countTypeBricks

After filling in the data write in the console the command for start.

To run the tests, write the command:  
 **npm run test**

The tests are based on the Jest framework. He has 8 tests for all methods in the Wall class.

The project directory has a wall folder. If you go to it you can see two files Wall.js and Wall.test.js

If you open the file Wall.js in the editor you can see the class Wall with a constructor in which our data from the file main.js is transferred

Methods in this class can be divided into two parts.

The first is data validation:

These include the following methods: validationCombinations (), validationMatrix (), validationCountTypeBricks (), validationPlace ().They are needed to monitor your data and to ensure that it is correct.

The second is data processing:

- sortCombinations (array) - accepts and returns an array of brick combinations. And sorts their sizes from bigger to smaller

-searchFullLines (matrix) - takes a matrix and looks for rows without 0. And returns an array with indexes of these rows

-calculateBricks - returns an array of bricks that go in size for lines

-minusLines - takes data from searchFullLines and calculateBricks methods and calculates which brick to put, and subtracts them from the warehouse. Returns true or false. Which indicates whether it is possible to build complete lines

-searchBlank - finds empty spaces and returns an array with a distance in the number of elements between spaces

-calculateBlank - Counts enough bricks to fill lines without spaces. Returns true or false. Which indicates whether it is possible to build a wall

A special method to **build()** - collects all the methods, and prints the output to the screen.