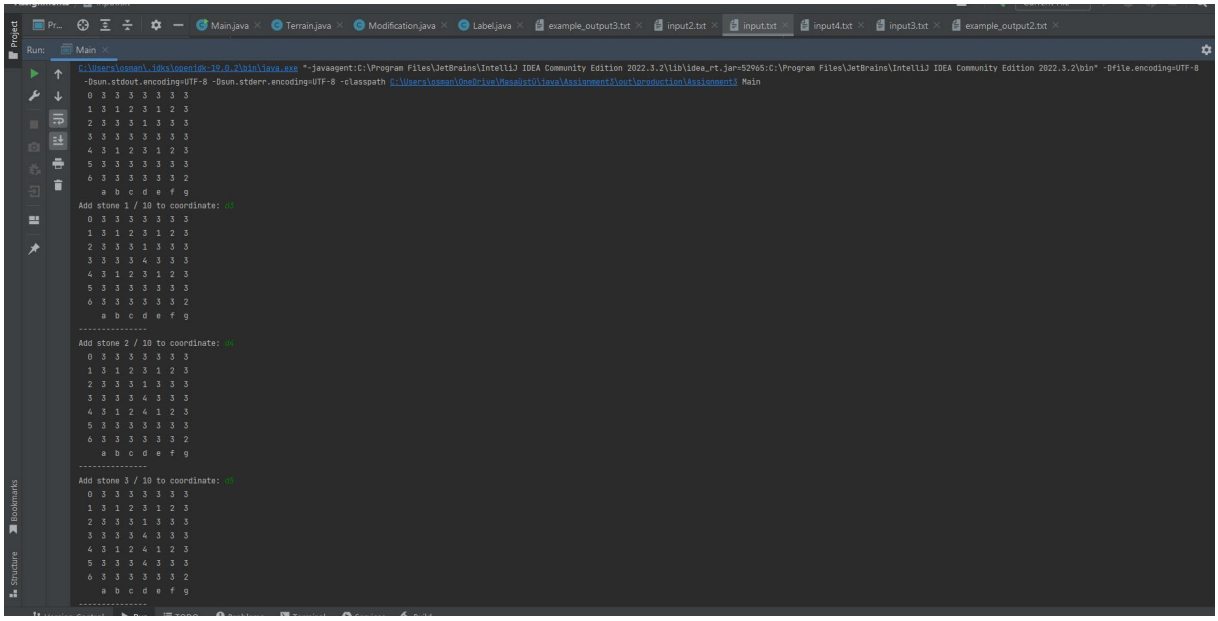
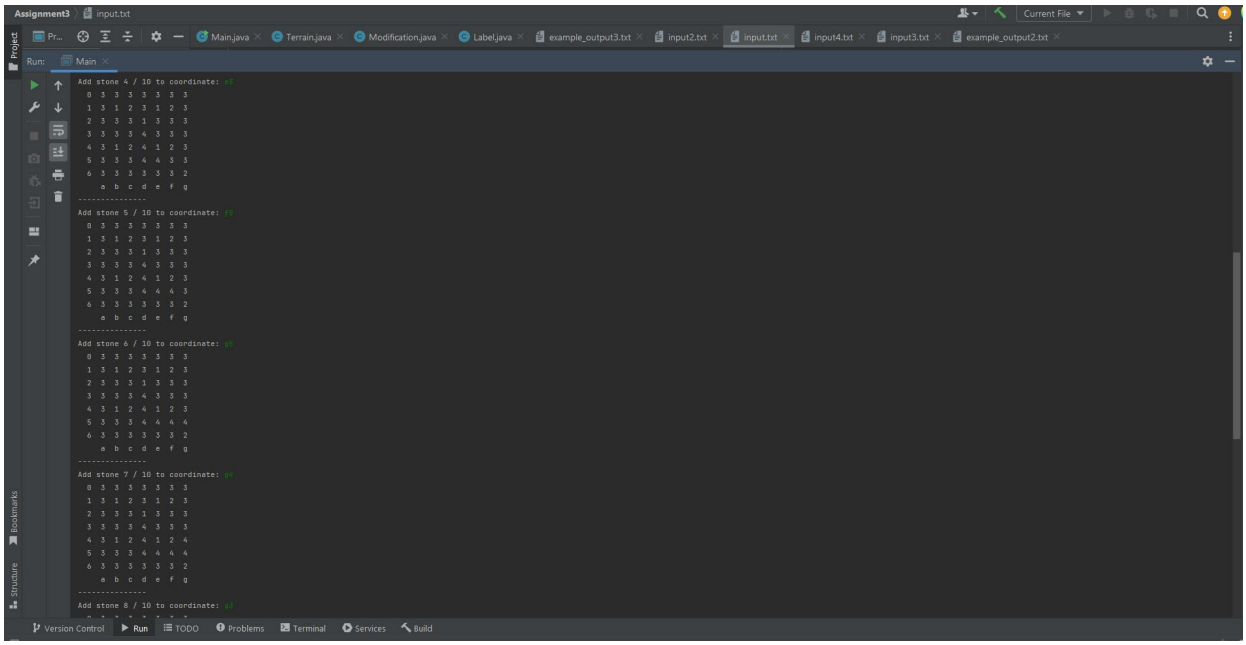
This is a brief explanation of all the classes:
Main class: This class's function is to read the input file, call other methods when needed, create the terrain, label objects and run the program
Terrain class: This class represents the terrain of the game. It contains the matrix representing the game terrain, visited list to check while running the dfs method, height list and the inner and outer sizes.
It also contains the methods to print the matrix, calculate the score and modify the matrix.
Label class: This class represents the labels of the game. It contains the label list, the label count and the label size.It also contains the methods to traverse the matrix and find the elements to be labeled, label the elements that are of the same lake
Modification class: This class is used to keep the methods to check whether the input is valid or not. For the sake of OOP and readability, it is created as a separate class.

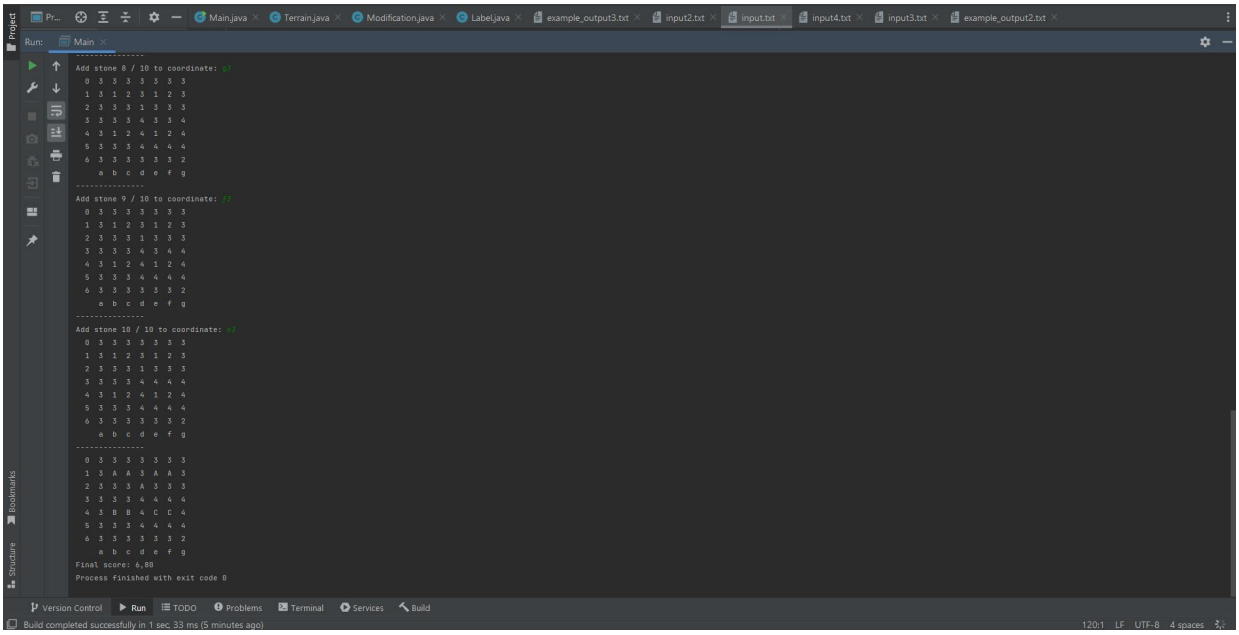
Algorithm and approach:
In this problem, the most challenging part is to identify the lakes and find the elements that are of the same lake. To do this, I used a dfs algorithm. I created a visited list to check whether a coordinate has been visited or not. I created a height list to keep track of the lake heights. Mainly, the dfs algorithm finds the lakes like this: 1. Check if the coordinate is valid 2. Check if the coordinate has been visited.
If those two conditions are met, then the algorithm will check if the coordinate is a lake or not.
To check whether a lake can exist in a coordinate, the algorithm tries to find a path from the coordinate to the edge of the matrix by simulating the water flow with a max value.
But here is the tricky part: while calling the dfs method in the main, we initially assign the max value to the height of the coordinate. Thanks to that, if the coordinate is not a lake, the method will return false and the program will know that the coordinate is not a lake.
On the other hand, if the coordinate is a lake, the method will return true and the program will know that the coordinate is a lake. But it is not enough, the program should also find the height of the lake by calling the dfs method again, until the method returns false. At this point, the program will know that the height of the lake is the (max value) - (height of the coordinate).
After identifying the lakes and their heights, the program will label the elements that are of the same lake. To do this, the program will again traverse the matrix and find the elements that are of the same lake by dfs. To find the elements that are of the same lake, the program will use the height list. If the height of the element in the height list is 0, then the element is not a lake.
If the height of the element in the height list is not 0, then the element is a lake. But we should label all the elements in the same lake. So, we basically do the following: If the height of the element in the height list is 0, program does not label this coordinate. Otherwise, the program will label the element using the label count and check adjacent coordinates because if the current coordinate is a label, its adjacent may be a label too. So, we call the dfs method again with the SAME COUNTER variable.
Thanks to this, the coordinates with same label will be checked with same counter and program will know that they are of the same lake. After labeling one lake, we increment the counter by 1 to label next lake. label count is the order of a lake in the matrix. While printing the matrix, we transform the label count to a character by adding 65 to it. So, the first lake will be labeled as A, the second lake will be labeled as B and so on.

Input 1: input.txt file

Output:



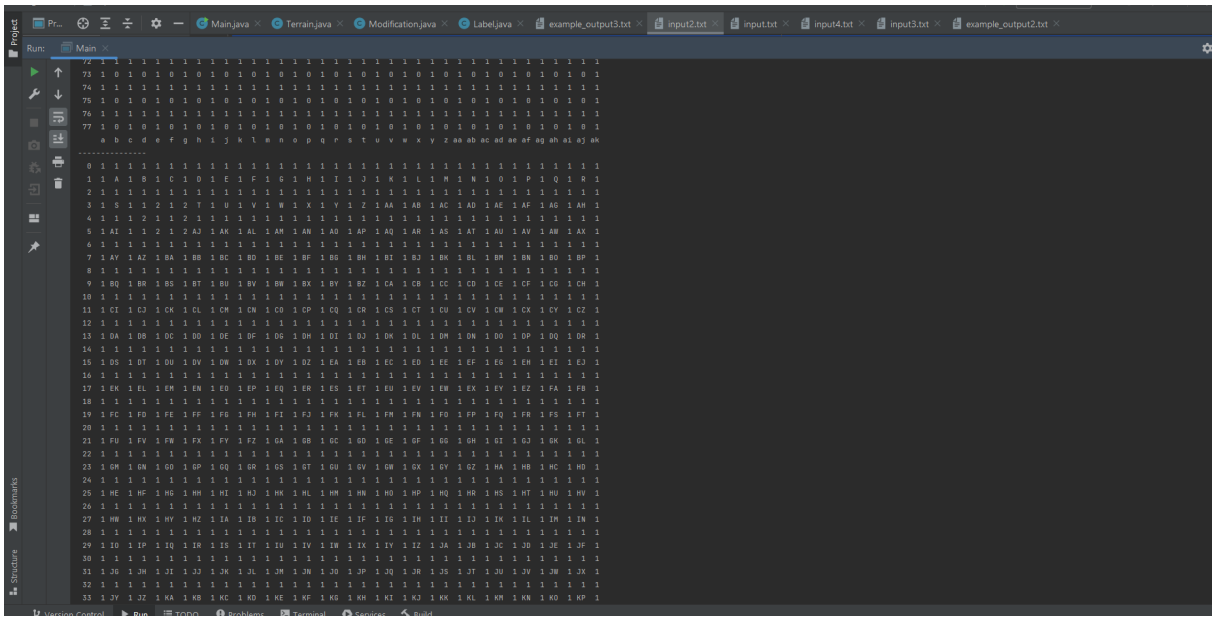


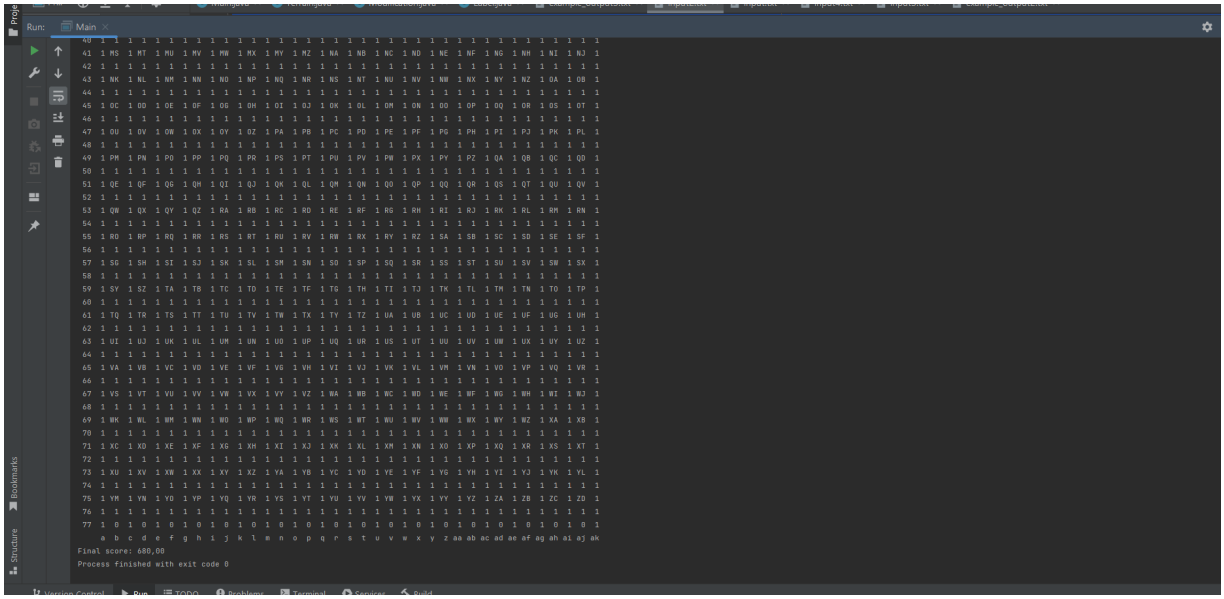


Input 2:

Provided by input2.txt

Output:

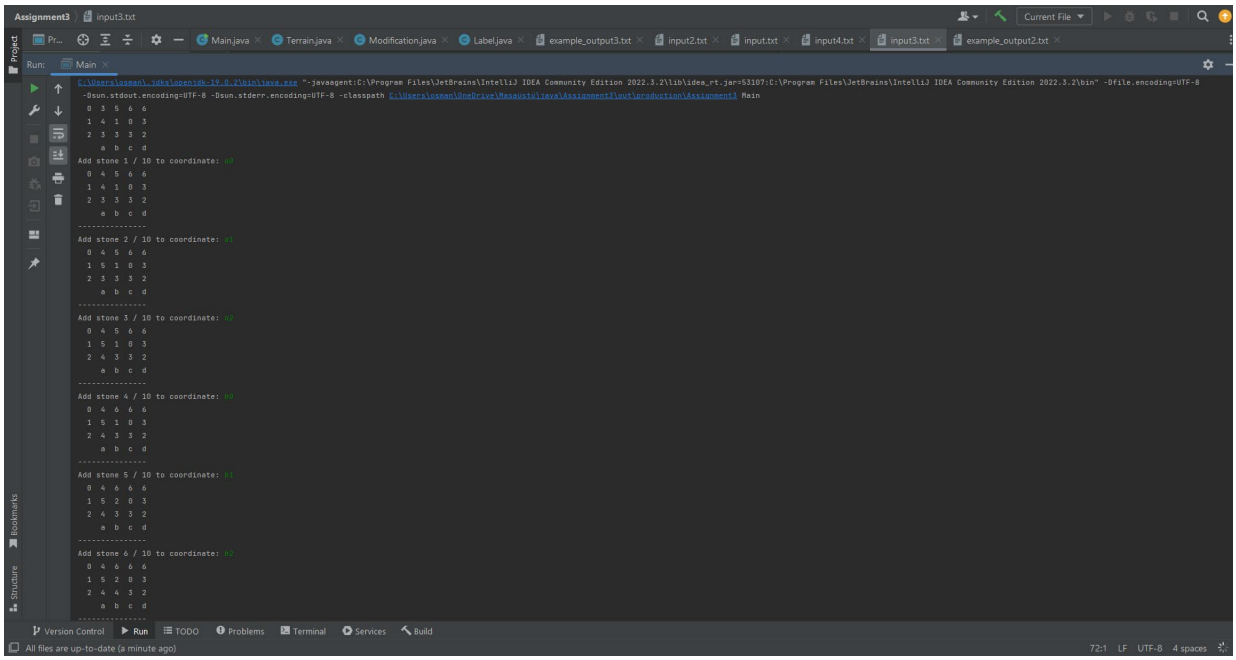


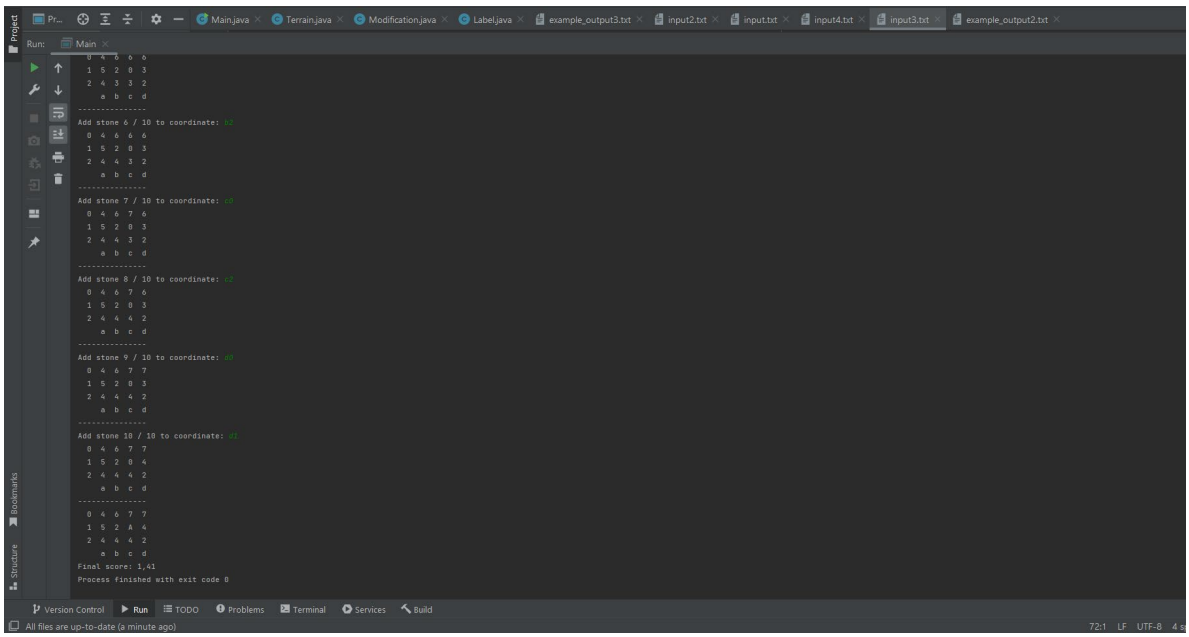


Input 3:

Provided by input3.txt

Output:


Assignment3
input3.txt
Current File
Main.java
Terrain.java
Modification.java
Label.java
example_output3.txt
input2.txt
input.txt
input4.txt
input3.txt
example_output2.txt
Run:
Main
C:\Users\osmen\.jdks\openjdk-18.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.3.2\lib\idea_rt.jar=53107:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.3.2\bin" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath C:\Users\osmen\OneDrive\Masaüstü\java\Assignment3\out\production\Assignment3 Main
0 3 5 6 6
1 4 1 8 3
2 3 3 3 2
a b c d
Add stone 1 / 10 to coordinate: a0
0 4 5 6 6
1 4 1 8 3
2 3 3 3 2
a b c d
Add stone 2 / 10 to coordinate: a1
0 4 5 6 6
1 5 1 8 3
2 3 3 3 2
a b c d
Add stone 3 / 10 to coordinate: b2
0 4 5 6 6
1 5 1 8 3
2 4 3 3 2
a b c d
Add stone 4 / 10 to coordinate: b0
0 4 6 6 6
1 5 1 8 3
2 4 3 3 2
a b c d
Add stone 5 / 10 to coordinate: b1
0 4 6 6 6
1 5 2 8 3
2 4 3 3 2
a b c d
Add stone 6 / 10 to coordinate: b2
0 4 6 6 6
1 5 2 8 3
2 4 4 3 2
a b c d
Version Control
Run
TODO
Problems
Terminal
Services
Build
All files are up-to-date (a minute ago)
72:1
LF
UTF-8
4 spaces


Main.java
Terrain.java
Modification.java
Label.java
example_output3.txt
input2.txt
input.txt
input4.txt
input3.txt
example_output2.txt
Run:
Main
Add stone 6 / 10 to coordinate: b2
Add stone 7 / 10 to coordinate: c0
Add stone 8 / 10 to coordinate: c2
Add stone 9 / 10 to coordinate: d3
Add stone 10 / 10 to coordinate: d1
Final score: 1,41
Process finished with exit code 0
Version Control
Run
TODO
Problems
Terminal
Services
Build
All files are up-to-date (a minute ago)
72:1
LF
UTF-8